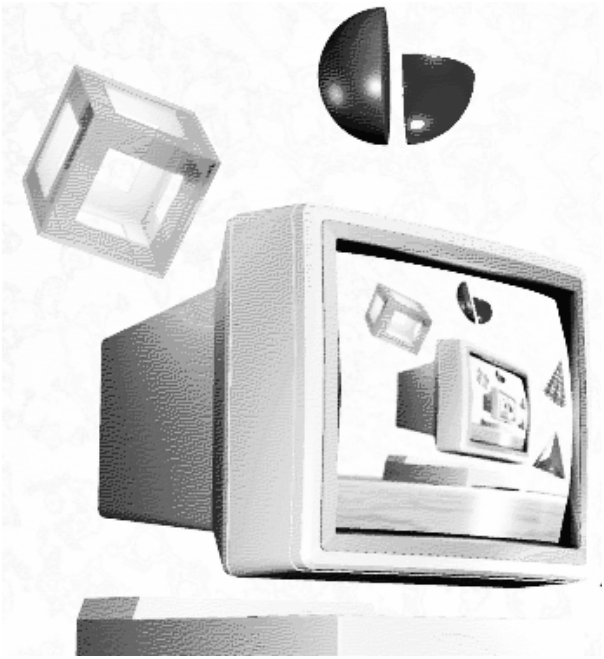
ТЭРГҮҮН БҮЛЭГ



-Үндсэн ойлголт
-Идентификатор
-Тогтмол болон хувьсагч
-Санах ой
-Анхны программ

Тэргүүн бүлэг

Компьютер бол Монгол орон төдийгүй дэлхий дахины хувьд хамгийн залуу гэж болох шинжлэх ухаан билээ. Гэхдээ орчин цагийн ямар ч үйл ажиллагааг компьютергүйгээр төсөөлөхөд тун хүнд болжээ. Учир нь гэвэл, компьютер нь өнөө үед дэлхий дахины олон жилийн турш шийдэж чадахгүй байсан асуудлаас эхлээд айл гэрийн хамгийн энгийн хэрэгцээг ч хүртэл хангаж чадахуйц тийм өргөн хүрээг хамран ашиглагдаж байгаа юм. Компьютер нэгэнт ийм чухал байгаа өнөө үед түүнийг программчилж, өөрийн хүссэн үйлдлийг гүйцэтгүүлэх нь хүн бүрийн хүсэл төдийгүй хэрэгцээ юм. Программчлал гэдэг нь хүнээс багагүй хэмжээний хичээл зүтгэл, бие даасан оролдлогыг шаарддаг ажил билээ. Өнөөгийн манай нийгэмд хэрэглээний программыг хагас дутуу ашигладаг хүнийг л программист гэж үздэг муу үзэл газар авсан байна. Энэ бол тун ч буруу ойлголт юм. Нийгэмд байгаа ийм гаж буруу ойлголтуудыг засаж залруулахад өөрсдийн өчүүхэн хувь нэмрийг оруулах зорилгыг агуулан энэхүү номыг хэвлэн гаргаж байгааг минь уншигч авгай та хүлээн авна уу. Уг номд элдэв ташаа, гэм буруу, дутагдан хаягдсан зүйлс байх аваас уншигч та 313682, 323956 утсуудаар зохиогчтой холбоо барьж болох юм. Таны санал хүсэлтийг хүлээж авахдаа би туйлын баяртай байх болно.

Таны эрдмийн их аянд амжилт хүсье.

Үндсэн ойлголт

Компьютерийн хэрэглээ болоод судалгааны аливаа асуудал нь техник хангамж эсвэл программ хангамжтай холбогддог. Түүнчлэн эдгээрийн аль алинтай ч холбогдох тохиолдол олон байдаг.

Техник хангамж гэдэгт компьютерийн гадна байдлаас харагдаж байгаа бүх техник тоног төхөөрөмж, компьютертэй холбогдсон гадаад байгууламжуудыг ойлгож болно. Тухайлбал: Процессор, дэлгэц, гараас эхлээд хулгана, принтер, скайнер, хатуу болон уян диск, тэдгээрийн төхөөрөмжүүд, сүлжээний болон дууны, дэлгэцийн картууд гээд тун их зүйлийг энд нэрлэж болно. Мөн дээр дурдсан байгууламжуудын үйлдвэрлэл болоод засварын ажиллагаа ч энэ хэсэгт хамрагдана.

Дээр дурдсан техник хангамжуудыг ашиглан, тодорхой удирдлагын доор дэс дараалан биелэх бүлэг үйлдэл болон мэдээллийг боловсруулах боломжоор хангаж өгсөн өгөгдлийн бүтцийг программ хангамж гэж нэрлэнэ. Түүнчлэн программ хангамжид баримт бичгийг зайлшгүй дагалдуулах ёстой.

Компьютерийн программ хангамж нь ямар ч зориулалттай байж болно. Тэрхүү программ хангамжийг хэн хүссэн хүн өөрийн хэрэгцээнд ашиглах бүрэн боломжтой. Харин тэрхүү программ хангамжийг шинээр зохиох гэдэг дээр дурдсанаар танаас багагүй хэмжээний хичээл зүтгэлийг шаардана.

Нэгэнт зохиогдон ашиглагдахад бэлэн болсон программ хангамжийг хэрэглээний программ гэдэг. Хэрэглээний программуудыг хэрхэн ашиглахыг төрөл бүрийн дамжаануудаар суралцаж болно. Microsoft Word, Excel, Access, PowerPoint, PageMaker төдийгүй Microsoft Windows ч өөрөө хэрэглээний программ юм. Харин тэрхүү хэрэглээний программыг зохиохын тулд бид программчлалын хэлийг ашиглана. Хэдийгээр компьютер үүссэн цагаас эхлэн нэлээн тооны программчлалын хэлүүд зохиогдож байсан боловч өнөө үед өргөн хэрэглэгддэг цөөхөн хэдэн хэл байдаг. Эдгээрээс бид C++ гэсэн программчлалын хэлийг үзэх гэж байна.

Программчлалын хэлийг ашиглан программ бичихэд доорх алхмуудыг гүйцэтгэх ёстой. Үүнд:

1.Алгоритм зохиох (Designing)
2.Алгоримтын дагуу программын кодыг бичих (Coding)
3.Хөрвүүлэлт хийх (Compilnig)
4.Программыг шалгах (Testing)
5.Сайжруулах (Developing)

1. Бичих гэж буй программын үндсэн **алгоритмыг зохиох**. Бид энэ алхамтай төстэй үйлдлүүдийг бодит амьдрал дээр маш их гүйцэтгэдэг. Гэрээс гараад хаагуур явбал зорьсон газраа хурдан хүрэх, ийм ажил бүтсэн тохиолдолд ямар ажлыг хийх, бүтээгүй тохиолдолд ямар ажлыг хийх гэх мэтийн шийдвэрүүдийг гаргах бүрд бид богино хэмжээний алгоритм зохиож байгаа гэсэн үг юм. Тэгвэл бид бодлого бодохдоо бүр нарийн алгоритм зохиоход хүрнэ. Эхлээд энэ утгыг олоод, дараа нь харин энэ утгаасаа 2 дахь утгыг олно гэх мэтчилэн. Алгоритмын хамгийн энгийн жишээг үзэцгээе. Бид 2 тооны ихийг нь олдог программ зохиох шаардлагатай болжээ. Юуны өмнө бид хэрэглэгчээс хооронд нь жиших 2 тоог авах хэрэгтэй болно. Өгсөн 2 тоог хооронд нь жишээд 1 дүгээр тоо их байвал

программын үр дүн нь 1 дүгээр тоо болно, харин эсрэг тохиолдолд программын үр дүн нь 2 дахь тоо болох юм. Энэ тохиолдолд программын алгоритм нь тоо оруулах, жиших, үр дүнгээ харуулах гэсэн 3-хан үйлдээс тогтож байна.

2. **Алгоритмын дагуу программын кодыг бичих**. Алгоритм нэгэнт тодорхой болсон тохиолдолд уг программд хэдэн ширхэг хувьсагч, тогтмол ашиглагдахыг шийдэж тэдгээрийг тодорхойлж өгнө. Хувьсагч болон тогтмол утгууд, тэдгээрийг тодорхойлох тухай доор дэлгэрэнгүй үзэх болно. Уг хувьсагчууд, тогтмолууд болон өмнөх алхамд зохиосон алгоритмаа ашиглан программын үйлдлүүдийг C++-ийн текст засварлагчийн тусламжтайгаар бичиж өгнө.

3. **Хөрвүүлэлт хийх**. Программаа бичиж дуусаад хөрвүүлэлт хийхийн тулд Compile гэсэн менюний Run коммандыг ашиглана. Ингээд программ зохиогчийн бичиж өгсөн коммандуудыг хөрвүүлэн биелэх файл үүсгэнэ. Энэ ажиллагааг компиляци буюу хөрвүүлэлт хийх гэж нэрлэх бөгөөд хөрвүүлэлт хийдэг системийг компилятор гэж нэрлэдэг. Компилятор программд бичигдсэн бүх коммандуудыг шалгаж үзээд, хэрэв үл ойлгогдох бичлэг ч юмуу, ер нь ямар нэгэн алдаа гарсан тохиолдолд хэрэглэгчид мэдээлж, компиляцийг зогсооно. Энэ тохиолдолд эргэж 2-р алхамдаа шилжих бөгөөд программын коммандууд дотор байгаа алдаануудыг засч дахин хөрвүүлэлт хийх шаардлагатай болдог. Программ ганц ч алдаагүй болсон тохиолдолд л биелэх файл буюу ашиглахад бэлэн болсон хэрэглээний программ үүснэ.

4. **Программыг шалгах**. Хөрвүүлсний дараа программаа сайтар шалгах нь зүйтэй. Хэдийгээр программ огт алдаагүй хөрвөсөн ч гэсэн таны хүссэн маягаар ажиллахгүй, хэрэгцээтэй зөв үр дүнг өгөхгүй байж магадгүй. Энэ тохиолдолд алдааг компилятор бус, харин та өөрөө олох хэрэгтэй болно.

5. **Сайжруулах**. Зохиосон программ нэгэнт ном ёсоороо ажиллаж байгаа бол одоо уг программд байгаа ашиглагдаагүй хувьсагч, тогтмол, функц, макро, илүү болон давхар хийгдэж байгаа үйлдлүүд гэх мэтчилэн программын хурд ба хэмжээнд сөргөөр нөлөөлж болох зүйлсийг олж засах хэрэгтэй. Ингэснээр программ санах ой болон дискэн дээр эзлэх хэмжээ нь багасах, үр дүнд нь программын ачаалах болон биелэх хурд ихсэх зэрэг олон талын сайн үр дүнд хүрэх боломжтой юм.

Идентификатор, түүнийг ашиглах тухай

Программд ашиглагдаж байгаа нэр томъёонуудыг ерөнхийд нь **идентификатор** гэж нэрлэдэг. Идентификатор нь дотроо нөөц үг буюу хэрэглэгчийн тодорхойлсон гэж ангилагдана. Нөөц үг гэдэг нь уг программчлалын хэлэнд стандартаар ашиглагддаг, хэрэглэгч өөрийн хүссэнээр өөрчилж болохгүй нэр томъёонууд юм. C++-д доорх нөөц үгүүд байдаг.

asm	auto	break	case
catch	char	class	const
continue	default	delete	do
double	else	enum	extern
float	for	friend	goto
if	inline	int	long
new	operator	private	protected
public	register	return	short
signed	sizeof	static	struct
switch	template	this	throw
try	typedef	union	unsigned
virtual	void	volatile	while

Харин хэрэглэгч программдаа ашиглагдах хувьсагч, тогтмол, функц, макро зэрэгт өгсөн нэрүүдийг хэрэглэгчийн тодорхойлсон идентификатор гэдэг. Хэрэглэгчийн тодорхойлсон идентификаторт доорх шаардлагууд тавигдана. Үүнд:

Идентификатор нь 1 болон хэд хэдэн тэмдэгтээс тогтоно.

Тоо болон латин үсэг аль аль нь орж болох боловч заавал үсгээр эхэлсэн байх ёстой.

Зөвхөн доогуур зураас ашиглана. Ө.х. +, -, *, /, ?, !, `, ~, ", ', ;, :, @, #, $, %, ^, &, (,), =, \, {, }, [,], <, > тэмдэгүүд болон цэг, таслал оруулж болохгүй.

Нөөц үгтэй ижилхэн байж болохгүй.

Кирилл үсэг оруулж болохгүй.

Идентификаторт орж байгаа том, жижиг үсгүүд хоорондоо ялгаатай гэдгийг бүү март. Өөрөөр хэлбэл, Fvar ба fvar гэсэн идентификаторууд зэрэг тодорхойлогдож болох бөгөөд эдгээр нь хоорондоо ялгаатай байна.

Нөөц үгийг программын аль нэг хэсэгт тодорхойлох шаардлагагүй байдаг бол харин хэрэглэгч өөрийн идентификаторуудыг программын шаардлагатай газруудад тодорхойлж байх хэрэгтэй.

Тогтмолыг ашиглах нь

Өмнө үзсэн хэрэглэгчийн тодорхойлсон идентификаторын тухай арай дэлгэрэнгүй тайлбарлая.

Тогтмол гэдэг нь программын явцад огт өөрчлөгдөхгүй утгуудыг хэлнэ. Шууд бодоход программын явцад өөрчлөгдөхгүй юм бол тийм идентификатор тодорхойлох шаардлагагүй мэт санагдаж болох юм. Гэхдээ тогтмол утга тун чухал нэгэн үүрэгтэй бөгөөд үүнийг сүүлд жишээн дээр тайлбарлаж үзэх болно. Программ дотор тогтмол утга тодорхойлохдоо DEFINE гэсэн нөөц үгийг ашиглана.

#define <тогтмолын нэр> <тогтмол утга>

Тогтмол утга тодорхойлж байгаа хэдэн жишээ үзье.

```
#define month 12          // эерэг бүхэл тогтмол
#define degree –30        // сөрөг бүхэл тогтмол
#define pi 3.1412         // бутархай тогтмол
#define name “П.Амарбат” // тэмдэгт мөр тогтмол
#define question “Sure?” // тэмдэгт мөр тогтмол
```

Мөр бүрийн төгсгөлд ‘//’ тэмдгээр эхэлсэн кирилл текстийг оруулсан байна. Эдгээр текст программын үйл ажиллагаанд зөв буруу ямар нэгэн нөлөө үзүүлэхгүй. Учир нь компилятор ийм тэмдгээс хойш уг мөрөнд байгаа текстийг ердийн тайлбар гэж ойлгодог юм.

Тогтмол тодорхойлохын гол ашиг тусыг та бүхэн дараах жишээнээс ойлгон авч болно. Бидний программд нэгэн тогтмол тоо буюу 23 гэсэн тоо маш олон дахин, тодруулбал 100 дахин ашиглагддаг гэж үзье. Бид тэгвэл программдаа 100 удаа 23 гэж бичиж өгөх болно. Нэг удаа бичээд өнгөрөхөд энэ нь тийм ч төвөгтэй үйлдэл биш байж болох юм. Гэтэл тэрхүү тоо 25 болж өөрчлөгдөх шаардлага гарчээ. Тэгвэл энэ удаад би программдаа байгаа бүх 23 гэсэн тоог 25 болгож өөрчлөх шаардлагатай болж байна. Энэ тохиолдолд хийх ажил тийм ч хялбар биш болж ирэх

нь ойлгомжтой. Харин бид 23 гэсэн тоог программдаа тогтмол болгон тодорхойлсон бол үүнийг өөрчлөх тун ч амархан. Өөрөөр хэлбэл,

#define num 23

гэж бичээд, уг тоотой ажиллах бүх газар num гэсэн тогтмолоо ашиглах хэрэгтэй юм. Тэгвэл дээрх шаардлага гарахад зөвхөн энэ мөрийг л

#define num 25

болгон өөрчилөхөд л хангалттай юм. Ингэвэл программ дахь 23 гэсэн тоон дээр хийж байсан бүх үйлдлүүд 25 гэсэн тоон дээр хийгдэх болно.

‘#’ тэмдгээр эхэлсэн нөөц үгийг компиляторын директив гэж нэрлэх бөгөөд #define нь ч гэсэн директив болно. Энэхүү #define директив нь энд үзсэнээс өөр, илүү өргөн боломжтой бөгөөд энэ тухай та 3-р бүлгээс дэлгэрүүлэн үзээрэй.

Хувьсагчийг ашиглах нь

Программын явцад харгалзах утгууд нь өөрчлөгдөж байдаг идентификаторыг **хувьсагч** гэнэ. Өөрөөр хэлбэл, хувьсагч нь ганц тогтмол утга биш, харин утгуудын тодорхой мужаас аль нэгийг нь авах боломжтой юм. Хувьсагч нь программд их чухал үүрэгтэй бөгөөд хэдэн хувьсагч ашиглагдах, тэдгээр нь ямар ямар төрөлтэй байхыг шийдэх нь программ зохиох гол алхмуудын нэг гэж дээр дурдсан билээ. Хувсагчийн төрөл гэдэг нь түүний авч болох утгуудын мужийг хэлж байгаа юм. C++-д дараах стандарт төрлүүд байдаг.

Төрөл	Утгын муж
char	‘0’..’9’,’A’..’Z’,’a’..’z’,*/+- Ў@#$%^&*(){}[];:.,’”?`~
short	-128..127
unsigned short	0..255
int	-32768..32767
unsigned int	0..65535
long	-2147483648.. 2147483647
unsigned long	0..4294967295

float	1.17549435e-38..3.40282347e+38
double	2.2250738585072014e-308 .. 1.7976931348623157e+308

Эдгээр стандарт төрлөөс гадна хэрэглэгч өөрийн хэрэгцээнд таарсан шинэ төрлийн үүсгэж болох бөгөөд энэ тухай дараа бүлгүүдэд тун дэлгэрэнгүй үзэх болно.

Программд ашиглагдах хувьсагчийг тодорхойлохдоо доорх бичиглэлийг ашиглана.

<хувьсагчийн төрөл> <хувьсагчийн нэр><=анхны утга>

Тодорхойлж буй хувьсагч ямар утга авах, ямар нэртэй байхыг зааж өгөх төдийгүй, уг хувьсагч анхнаасаа ямар утгатай байхыг зааж өгч болж байна. Хувьсагч тодорхойлж байгаа хэд хэдэн жишээ авч үзэцгээе.

```
int             ivar1;    //бүхэл утга авах хувьсагч
int             ivar2=-10;      //Анхны утга
float       fvar1=2.57;       //Бутархай хувьсагч
unsigned int    ivar3=500;        //Эерэг бүхэл хувьсагч
char        ch1,ch2='*',ch3;          //Тэмдэгт хувьсагчууд
```

Хувьсагчийн тухайд доорх хэдэн санамжийг мартаж болохгүй.

Бид программын дурын хэсэгт хувьсагчийн утгыг өөрчилж болох боловч хувьсагчид өгч байгаа утга нь уг хувьсагчийн төрлийнх байх ёстой.

Нэг программ дотор ижил нэртэй 2 хувьсагч байж болохгүй.

Хувьсагчийн утгатай холбоотой гарч болох алдааг үргэлж тооцож, боловсруулж байх нь зүйтэй.

Шаардлагагүй тохиолдолд их том хэмжээний утгын мужтай хувьсагч тодорхойлох нь программыг их нүсэр, болхи болгодог.

Санах ой, түүний хаяг

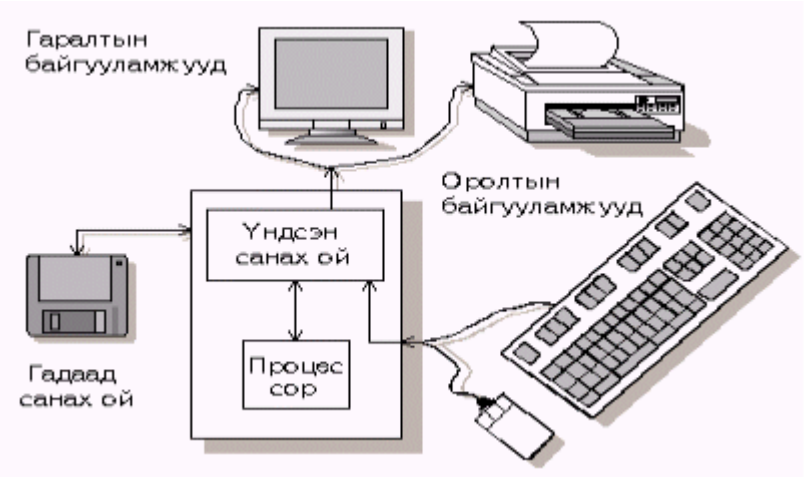
Программ ажиллахын тулд уг программын бүх биелэх коммандууд болон программд шаардлагатай өгөгдлүүдийг

бүгдийг нь компьютерийн дотоод санах ойд байрлуулдаг. Санах ой нь нэг удаад нэг тэмдэгт болон тоог санах боломжтой нүднүүдээс тогтоно. Нэг нүдэнд нэг **байт** өгөгдөл хадгалагдана. Ө.х. нэг тэмдэг буюу тоо нь санах ойд нэг байтыг эзэлдэг гэсэн үг юм. Санах ойн багтаамж гэдэг нь энэхүү нүднүүдийн тоог хэлж байгаа юм. Тухайлбал, 640000 байтын багтаамжтай санах ой гэвэл түүнд 640000 нүд байгаа бөгөөд нийтдээ 640000 тэмдгээс тогтсон өгөгдлийг уг санах ойд хадгалах боломжтой гэж ойлгож болно. Санах ойн нүд бүр өөрийн хаягтай байдаг. Ө.х 640000 байтын санах ойн эхний нүдний хаяг 0, сүүлийн нүдний хаяг нь 639999 байна.

Программ дотор хувьсагч тодорхойлно гэдэг нь санах ойн нэг болон хэд хэдэн нүдийг уг хувьсагчид зориулан нөөцлөн авч, тэр нүднийхээ хаягийг хувьсагчид сануулж өгдөг байна. Уг хувьсагчтай ажиллана гэдэг нь тэрхүү нөөцөлсөн нүдэнд байгаа өгөгдөлтэй ажиллана гэсэн үг юм. Харин уг хувьсагчид хэдэн нүдийг нөөцлөх вэ гэдэг нь тэрхүү хувьсагчийн төрлөөс хамаардаг.

Төрөл	Санах ойд эзлэх байт
char	1
short	1
unsigned short	1
signed int	2
unsigned int	2
long	4
unsigned long	4

Ингээд санах ойд хадгалагдаж байгаа программын кодуудыг төв процессор нэг бүрчлэн уншиж уг коммандуудыг дэс дараалан биелүүлдэг юм. **Гэхдээ санах ойд байгаа өгөгдлийг программын код, хувьсагч, тогтмол аль нь болохыг процессор огт мэдэхгүй бөгөөд хэрэв программыг оновчгүй зохион байгуулвал процессор буруу комманд биелүүлэн, системийг гацахад хүргэж болзошгүй.**



Компьютерийн санах ойд өгөгдлийг (физик талаас нь үзвэл) ямар зарчмаар хадгалдаг вэ? гэдгийг сонирхоё. Өгөгдлийг санах ойд соронзон зарчмаар хадгалдаг юм. Соронзон нь ердөө хоёр төлөвт л орох чадалтай байдаг. Өөрөөр хэлбэл, хоёр л утга (0 эсвэл 1) илэрхийлнэ гэсэн үг. Хоёр төлөвт орох чадалтай, энэхүү хамгийн бага нэгжийг **бит** гэж нэрлэнэ. Тэгвэл санах ойн нэг байт буюу нэг нүд нь ийм 8 битээс тогтдог. Эндээс үзвэл санах ойн нэг байт нь 2^8=256 янзын утга л илэрхийлэх чадалтай юм. Ер нь бидний сайн мэддэг тооллын системд бүх тоог 10 янзын цифрээр л дүрсэлдэг. Ийм учраас 10-тын тоолол гэж нэрлэдэг билээ. Харин бүх тоог зөвхөн 0 ба 1 гэсэн 2-хон цифрээр дүрсэлдэг тооллын системийг 2-тын тоолол гэдэг. Одоо 10-тын тооллын дурын тоонд харгалзах 2-тын тооллын тоог олцгооё.

10-тын тооллоор 156 гэсэн тоо өгөгджээ. Уг тоог

$$\mathbf{156}=\mathbf{1}*10^2+\mathbf{5}*10^1+\mathbf{6}*10^0$$

гэж 10-тын зэргүүдэд задалж болно. Харин 2-тын зэргүүдэд задалбал :

$$156=1*2^7+1*2^4+1*2^3+1*2^2$$

буюу

$$156=\mathbf{1}*2^7+\mathbf{0}*2^6+\mathbf{0}*2^{5+}\mathbf{1}*2^4+\mathbf{1}*2^3+\mathbf{1}*2^2+\mathbf{0}*2^1+\mathbf{0}*2^0$$

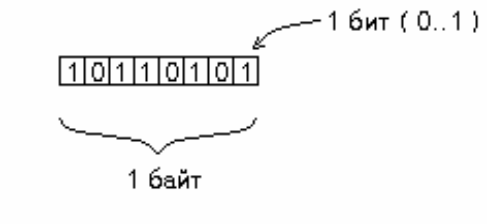
болно. Иймээс харгалзах 2-тын тоо маань **10011100** болж байна.

Ямарваа нэгэн тоог 10-тын зэргүүдэд задлахад гарч буй коэффициентүүд уг тооны 10-тын тоолол дахь утгын цифрүүд нь байдаг бол 2-тын зэрэгт задлахад гардаг коэффициентүүд нь түүний 2-тын тоолол дахь утгын цифрүүд болдог ажээ.

10-тын тооллоор	2-тын тооллоор
0	00000000
1	00000001
2	00000010
3	00000011
4	00000100
5	00000101
6	00000110
7	00000111
8	00001000

. . . . . .

255	11111111



Санах ойтой холбоотой доорх зүйлсийг үргэлж санах байх хэрэгтэй.

Санах ойн нүд бүр өөр өөрийн хаягтай байдаг.

Санах ойн нэг нүд нь нэг байт өгөгдлийг хадгалах бөгөөд нэг байт нь 8 битээс тогтоно.

Санах ойн нүд нь заавал ямар нэгэн өгөгдөл агуулж байх бөгөөд хоосон байна гэж үгүй. Харин уг нүдэнд зөв өгөгдөл байгаа эсэхийг сайтар анхаарч байх нь зүйтэй.

Санах ойн нүдэнд өгөгдөл бичихэд тэнд байсан хуучин өгөгдөл устаж алга болдог.

Ямар нэгэн программ ажиллахын тулд заавал санах ойд ачаалагдсан байх ёстой. Хэрэв санах ой өгөгдлөөр дүүрсэн байвал шинээр программ ачаалагдан ажиллах боломжгүй болно.

CONST нөөц үгийн тухай

Харин одоо тогтмолыг тодорхойлох бас нэгэн аргыг товч тайлбарлая. Бид тогтмолыг тодорхойлохын тулд мөн **const** гэсэн нөөц үгийг ашиглаж болдог юм. Бичлэгийн хувьд үндсэндээ #define директивтэй ижил гэж үзэж болно.

#define <төрөл> <нэр>=<тогтмол утга>

Жишээ нь:

```
const int month=12
const int degree=–30
const double pi=3.1412
const char name[]=“П.Амарбат”
const char question[]=“Sure?”
```

Зөвхөн тогтмолын нэр ба утгын хооронд тэнцүүгийн тэмдэг тавьдагаараа өмнөх директивээс ялгарах юм. Бичлэгийн хувьд ижил боловч компиляторын хувьд энэ хоёр нь маш их ялгаатай тодорхойлолтууд юм. Учир нь const нөөц үгийг ашиглан тодорхойлсон тогтмол нь программын явцад огт өөрчлөгдөхгүй боловч хувьсагч шиг программын туршид санах ойд нэг болон хэд хэдэн байтыг эзлэн оршсоор байдаг. Гэтэл #define директивээр тодорхойлсон тогтмол утга гэдэг маань жинхэнэ биелэх программын хаана нь ч байдаггүй юм. Харин программ хөрвөх үед уг тогтмолыг ашиглаж байгаа газар бүрт уг утгыг нь шууд орлуулан бичдэг байна. Жишээн дээрээ тайлбарлая.

```
a = pi * 2
```

гэсэн комманд байг. Хэрэв pi тогтмолыг const нөөц үгээр тодорхойлсон бол компилятор pi - г яг л хувьсагч шиг хөрвүүлнэ. Өөрөөр хэлбэл, энэ комманд биелэхийн тулд уг тогтмолд зориулсан санах ойгоос очиж өгөгдлийг унших болно.

Харин #define пирективээр тодорхойлсон бол pi гэсэн тогтмолд зориулсан санах ой гэж хаана ч байхгүй юм. Харин программ дотор дээрх коммандын оронд ердөө л

a = 3.1412 * 2

гэсэн комманд бичигдсэн байх юм.

Толгой файлыг ашиглах нь

C++ -ийг ашиглан программ бичиж байгаа тохиолдолд өөр файлд тодорхойлогдсон байгаа функц, тогтмол, макро зэргийг ашиглах бүрэн боломжтой байдаг. Хэрэв ийм боломжгүй бол ялангуяа том хэмжээний программ бичиж байгаа тохиолдолд маш их хэмжээний тогтмол болон функцүүдийг нэг файлдаа тодорхойлох нь файлын хэмжээ болон программын бүтцийн хувьд тун эмх замбараагүй болох бөлгөө. Харин тогтмол болон макронуудаа нэг файлд ялгаж тодорхойлоод түүнийгээ өөрийн файлдаа холбож өгвөл программын бүтэц энгийн болоод ойлгомжтой болох юм. Ийм файлыг **толгой файл** гэх бөгөөд толгой файл нь ‘ *.h ’ өргөтгөлтэй байна.

```
//defs.h
void fun1()
        {
        }

//my.cpp
#include “defs.h”

void main()
{
        . . . . .

        fun1();
        . . . . .
}
```

Толгой файлыг зааж өгөхдөө #include нөөц үгийг ашиглах бөгөөд түүний ард файлын нэрийг (“ “) тэмдгээр хашиж бичнэ.

Энэ нөөц үгийг заавал мөрийн эхлэлээс бичих ёстой. C++ өөрийн гэсэн стандарт толгой файлуудтай байдаг. Энэ файлуудад C++-д нийтлэг ашиглагдах тогтмолууд, функцүүд болон макронуудыг бичиж өгсөн байдаг юм. Энэхүү стандарт толгой файлууд нь Borland C++-ийн library гэсэн дэд каталогид байрлах ёстой. Ийм стандарт толгой файлуудыг тодорхойлох тохиолдолд #include директивийн бичлэгт ялимгүй өөрчлөлт орно. Нөөц үгийн ард файлын нэрийг бичихдээ (“ “) биш, (< >) хаалт хэрэглэх ёстой. Жишээ нь, математик болон логикийн функц, тогтмол, макронууд нь math.h гэсэн толгой файлд агуулагдаж байдаг. Хэрэв та программынхаа эхэнд

```
#include <math.h>
```

гэж зааж өгсөн бол программдаа математикийн функцүүдийг ашиглах боломжтой боллоо гэсэн үг юм.

C++ программын ерөнхий бүтэц

Программ нь ерөнхийдөө бэлтгэл хэсэг болон дэд ба үндсэн программаас тогтдог. Программын бэлтгэл хэсэгт уг программын бүхий л хүрээнд ашиглагдаж болох глобаль хувьсагч, тогтмолууд болон уг программд ашиглагдах стандарт функцүүдийн байрлаж буй толгой файлуудыг тодорхойлж өгдөг юм.

Одоо хамгийн анхны жишээ программаа бичиж үзэцгээе.

```
//hello.cpp
#include <iostream.h>

#define greeting = ”Сайн байна уу?”;

void main()
{

cout << greeting;

}
```

Программыг бичиж дуусаад өмнө ярьснаар Compile гэсэн менюний Run коммандыг өгөх хэрэгтэй. Энэ программ дэлгэц дээр

Сайн байна уу?

гэсэн ганц үг хэвлэж гаргана. Одоо программыг тайлбарлая.

Программын хамгийн эхний мөрөнд тайлбар оруулсан байна. Ийм тайлбар нь программын үйл ажиллагаанд ямар ч нөлөөгүй гэдгийг дээр өгүүлсэн билээ. Харин 2-р мөрөнд iostream.h гэсэн толгой файлыг тодорхойлж өгсөн байна. Энэ нь бид программдаа уг толгой файлд байгаа ямар нэгэн өгөгдлийг ашиглана гэдгээ хэлж байгаа юм. Түүний дараа greeting гэсэн нэртэй тэмдэгт мөр тогтмол тодорхойлсон байна. Харин түүний дараа байгаа

void main()

гэсэн мөр нь эндээс бидний программын биелэх кодууд эхэлж байгааг зааж өгч байна. C++-д нэгэн бүлэг үйлдлүүдийг гоё хаалтаар ({}) хашиж бичдэг. Өөрөөр хэлбэл ийм хаалтанд бичигдсэн үйлдлүүд нь нэг функц, процедур, давталт зэрэгт хамрагдана гэсэн үг юм. Үүний нэгэн адил бид энэ программынхаа биелэх кодуудыг гоё хаалтаар хашиж бичсэн байна. Дээрх программын тохиолдолд ердөө ганцхан үйлдэл байна. Энэхүү комманд нь урьдчилан тодорхойлсон байгаа тогтмолыг дэлгэцэнд хэвлэн гаргаж байна.

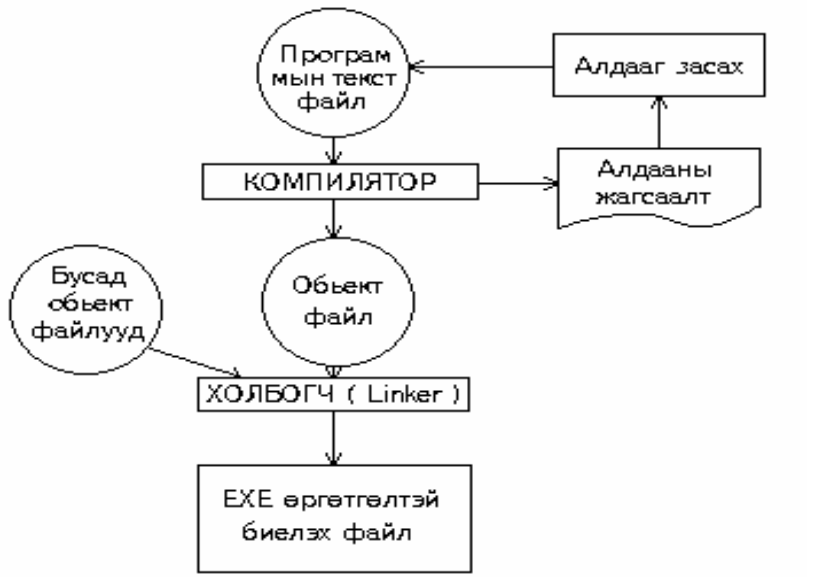
cout << greeting;

cout гэсэн энэ комманд дээр тодорхойлсон iostream.h гэсэн толгой файлд байрладаг юм.

Compile гэсэн менюний Run коммандыг өгөхөд зурагт үзүүлсэн процесс явагддаг юм.

Юуны өмнө программын текстийг компилятор хүлээн авч, хөрвүүлэлт хийж үзнэ. Хөрвүүлж дуусаад, хэрэв алдаа гарсан байвал гарсан алдаануудын тухай мэдээллүүдийг өгөөд цааших ажиллагааг зогсооно. Та программын текстэд байгаа алдаануудыг засаад Run коммандыг дахин өгөх хэрэгтэй. Харин ямар ч алдаагүй хөрвөж дууссан бол уг текстэд харгалзах обьект файлыг (*.OBJ өргөтгөл бүхий) дискэнд үүсгэж, ажиллагаа холбогчид шилждэг. Нэг программ хэд хэдэн унит буюу хэсгээс

тогтож болох бөгөөд холбогч нь тэдгээр хэсэг бүрийн обьект файлуудыг нэгтгэн дискэнд биелэх файлыг (*.EXE өргөтгөлтэй) үүсгэдэг юм. Энэхүү биелэх файл санах ойд ачаалагдсанаар уг программ ажилладаг юм.



Бүлгийн дүгнэлт

Номынхоо оршил бүлэгт программчлалын хэлний хамгийн энгийн буюу үндсэн ойлголтуудын тухай товч дурдлаа. Дараа дараагийн бүлгүүдэд энэ ойлголтуудыг гүнзгийрүүлэх төдийгүй маш олон шинэ ойлголтуудыг олж авах болно.

Программчлалын хэл гэдэг бол компьютер гэдэг ухааны чухал нэгэн салшгүй хэсэг юм. Бид программ хангамж зохиохын тулд программчлалын хэлийг ашиглана.

Программ бичихийн тулд уг программынхаа үндсэн санааг урьдчилан зохиосон байх ёстой. Уг программд ямар ямар функц, процедурууд ашиглагдах, ямар төрлийн хэдэн хувьсагч, тогтмол байхыг нэг бүрчлэн тооцсон байх нь цаашид хийх үйл ажиллагааг тань тун их хэмжээгээр хөнгөвчлөх юм.

Программд ашиглагдаж байгаа нэр томъёонуудыг идентификатор гэж нэрлэх бөгөөд идентификатор нь нөөц үг ба хэрэглэгчийн тодорхойлсон гэсэн 2 хэсэгт хуваагдана. Нөөц үг нь C++-д урьдчилан тодорхойлогдсон байдаг бөгөөд хэрэглэгчийн тодорхойлж байгаа идентификатор нь тогтмол ба хувьсагч гэсэн 2 ангилалтай байна.

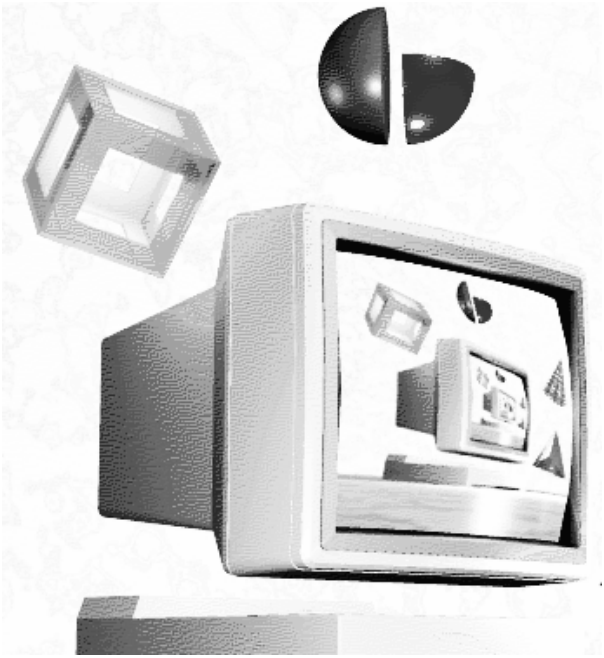
Тогтмол гэдэг нь программын явцад утга нь огт өөрчлөгдөхгүй идентификаторыг хэлж байгаа юм. Нэг утга программд олон давтан ашиглагдаж байгаа тохиолдолд түүнийг тогтмолоор тодорхойлох нь цаашид программд өөрчлөлт оруулах ажиллагааг их хэмжээгээр хөнгөвчилдөг. Тогтмолыг #define болон const гэсэн нөөц үгүүдийг ашиглан тодорхойлно.

Хувьсагчийг тодорхойлохдоо түүний авч болох утгуудын муж буюу төрлийг зааж өгдөг. Программын явцад хувьсагчийн утгыг өөрчлөх боломжтой. Хувьсагчийг тодорхойлохдоо түүний анхны утгыг зааж өгч болно.

Тогтмол, хувьсагч болон программын биелэх кодууд бүгд компьютерийн санах ойд байрлана. Санах ой нь тус бүрдээ 1 байт өгөгдөл хадгалах зориулалт бүхий нүднүүдээс тогтоно. Тэрхүү нүдний тоогоор санах ойн багтаамж хэмжигдэнэ. 1 байт нь 8 битээс тогтох бөгөөд нийтдээ 256 янзын утгыг санах чадвартай байдаг.

Өөр файлд тодорхойлогдсон функц, процедур, коммандуудыг ашиглахын тулд бид программдаа толгой файлыг тодорхойлж өгдөг. Толгой файлуудыг зааж өгөхийн тулд #include гэсэн нөөц үгийг ашигладаг.

2-р бүлэг



-Харьцуулах үйлдлүүд
-Давталтууд
-Нөхцөлт программ
-Логик үйлдлүүд
-Үйлдлийн эрэмбэ

2-р бүлэг

Бодит амьдрал дээр бүх кодууд нь эхнээсээ төгсгөлөө хүртэл шугаман дараалаар биелэдэг программ ховор байдаг. Ихэнх программ ямар нэгэн нөхцлөөс хамааруулан хийх үйлдлээ сонгоход (яг л хүн шиг) хүрдэг. Ө.х. программ доторх нөхцлөөс хамааран үйлдлийн удирдлага программын нэг хэсгээс нөгөө хэсэг рүү үргэлж “ үсэрч “ байдаг. Программ дотор ийм үсрэлт хийлгэдэг командуудыг удирдлагын коммандууд гэнэ. Удирдлагын комманд нь давталтын бол нөхцөл шалгалтын гэсэн 2 янз байдаг.

Давталтын тоо болон нөхцөл шалгалт нь өгсөн үйлдэл үнэн, худал эсэхийг тогтоодог. Тэрхүү үйлдэл нь харьцуулах үйлдлийн дагуу хийгдэнэ. Иймд дээрх 2 төрлийн коммандуудыг үзэхийн өмнө харьцуулах үйлдлийн тухай товч ярилцъя.

Харьцуулах үйлдэл

Харьцуулах үйлдэл нь хоёр утгыг хооронд нь жишдэг юм. Утгууд нь C++ ийн стандарт төрлүүд (char, int, float г.м) болон хэрэглэгчийн тодорхойлсон ангийн (энэ тухай хойно үзэх болно) хувьсагчид байж болно. Жишилт нь *тэнцүү, их тэнцүү, бага тэнцүү* гэх мэтчилэн байх юм. Харин үйлдлийн үр дүн нь үнэн ба худал байна. Жишээ нь : 2 утгын тэнцүү эсэхийг шалгасны үр дүн нь тэдгээр нь тэнцүү бол үнэн, ялгаатай бол худал гэсэн үг.

Бидний анхны программ RELAT нь бүхэл тоон хувьсагчийн утгыг тогтмол тоотой тэнцэж байгаа эсэхийг шалгаж үр дүнг харуулах юм.

```
//relat.cpp
#include <iostream.h>

void main()
```

```
{
int numb;

cout << “Тоогоо оруулна уу : ”;
cin >> numb;

cout << “numb<10 нь “ << (numb<10) << endl;
cout << “numb>10 нь “ << (numb>10) << endl;
cout << “numb==10 нь “ << (numb==10) << endl;
}
```

Энэ программд орсон cin гэсэн комманд бидний өмнө тайлбарласан cout коммандын яг эсрэг комманд юм. Ө.х. хэрэглэгчийн гараас оруулж өгсөн өгөгдлийг заасан хувьсагчид оноож өгдөг юм.

Энэ программ нь хэрэглэгчийн оруулсан тоо болон 10 гэсэн тогтмол тоонуудыг 3 янзаар жишиж байна. Хэрэглэгч 20 гэж оруулсан тохиолдолд бидний программ доорх үр дүнг өгнө.

```
Тоогоо оруулна уу: 20
numb<10 нь 0
numb>10 нь 1
numb=10 нь 0
```

Хэрэв numb нь 10 аас бага бол эхний үйлдэл үнэн болно. Харин numb нь 10 аас их бол 2 дахь үйлдэл үнэн, numb нь 10-тай тэнцэж байвал 3 дахь үйлдэл тус тус үнэн болно. Дээрх үр дүнгээс харахад C++ үнэнийг 1-ээр, худалыг 0–оор төлөөлөн ойлгодог байна. Зарим хэлнүүд зөвхөн үнэн, худал гэсэн 2 л утга авдаг Boolean гэсэн төрөлтэй байдаг боловч C++ д байдаггүй. Харин энгийн бүхэл тоон төрлийг ашигладаг байна.

C++-ийн харьцуулах үйлдлүүдийг жагсаавал:

Оператор	Утга
>	Эрс их

<	Эрс бага
==	Тэнцүү
!=	Тэнцүү биш
>=	Их тэнцүү
<=	Бага тэнцүү

Одоо харьцуулах үйлдлүүдийг ашиглах жишээ болон тэдгээрийн утгыг авч үзье. Энд эхний 2 мөр нь утга олгох үйлдлүүд бөгөөд Harry, Jane нарын анхны утгыг тогтоож байна.

Jane=44	Утга оноож байна
Harry=12	Утга оноож байна
(jane=harry)	Худал
(harry<=12)	Үнэн
(jane>harry)	Үнэн
(Jane>=44)	Үнэн
(harry!=12)	Худал
(7<harry)	Үнэн
(0)	Худал
(44)	Үнэн

Тэнцүүг шалгахдаа 2 ширхэг (=) тэмдэг тавина. Нэг (=) бол утга оноох үйлдэл юм. Утга оноох үйлдлийг харьцуулах үйлдлийн оронд хэрэглэх нь C++ анхан суралцагчдын хувьд нийтлэг гаргадаг алдаа юм. Энэ тохиолдолд компилятор ямар ч алдаа өгөхгүй бөгөөд харин программ буруу ажиллах болно.

Үнэн хэрэгтээ C++ зөвхөн 0-ийг л худал гэж үзэх бөгөөд 0-ээс ялгаатай бусад бүх утгуудыг үнэн гэж ойлгоно. Тийм учраас дээрх жишээний сүүлийн мөрөнд үнэн гэсэн үр дүн гарсан байна. За одоо энэхүү үйлдлүүд маань бүлгийн эхэнд дурдсан 2 төрлийн коммандуудад хэрхэн ашиглагдахыг үзэх болно. Давталтын тухай эхлээд үзье.

Давталт

Давталт нь программын нэг хэсгийг тодорхой тоогоор давтаж биелүүлэх зорилгоор ашиглагдана. Нөхцлийн үр дүн үнэн байгаа тохиолдолд давталт биелэгдсээр байх болно. Харин нөхцөл худлаа болоход давталт зогсож удирдлага давталтын дараагийн коммандад шилжих болно. C++ нь for, while, do гэсэн 3 төрлийн давталттай байна.

FOR Давталт

Энэ давталт нь C++-ийн давталтын тухай ойлголт авахад хамгийн дөхөм, энгийн давталт юм. Ер нь for давталт нь программчлалын бараг бүх хэлүүдэд байх бөгөөд жишээ нь BASIC хэлний хувьд энэ нь удаан хугацааны туршид цорын ганц давталт байсаар ирсэн юм. Энэ давталт нь хэсэг коммандуудыг тогтмол тоогоор давтан биелүүлнэ. Ө.х тухайн бүлэг үйлдэл хэдэн удаа давтагдан биелэхийг мэдэж байгаа тохиолдолд энэ давталтыг хэрэглэх юм.

Доор 0-оос 14 хүртэлх тооны квадратуудыг дэлгэцэнд харуулдаг FORDEMO гэсэн жишээ байна.

```
//Fordemo.cpp
#include <iostream.h>

void main()
{
int j;
for (j=0; j<=15; ++j)
        cout << j*j << " ";
}
```

Үр дүн нь доорх байдлаар байна.

0 1 4 9 16 25 36 49 64 81 100 121 144 169 196

Энэ программ яаж ажиллаж байна вэ? For давталт нь for гэсэн үг ба хаалт бүхий 3 үйлдлээс тогтжээ. Эдгээр нь харгалзан *анхны утга олгох, нөхцөл шалгах, утгыг өөрчлөх* үйлдлүүд юм. Эдгээр үйлдлүүд нь ихэвчлэн (гэхдээ үргэлж биш) *давталтын хувьсагч* гэж нэрлэгдэх ганц л хувьсагчтай холбогддог. Бидний жишээний хувьд давталтын хувьсагч нь j юм. *Давталтын бие* гэдэг нь давталт бүрт биелэгдэх бүлэг үйлдүүдээ хэлж байгаа юм. Бидний жишээний хувьд бие нь cout << j*j <<" " гэсэн ганцхан коммандаас бүрдсэн байна. Энэ комманд нь j-ийн квадрат тоог зайгаар тусгаарлан дэлгэцэнд хэвлэх юм. Давталт биелэхэд j нь 0, 1, 2, 3 гэх мэтчилэн утгуудыг дараалан авах бөгөөд тэр бүрд квадратууд нь хэвлэгдсээр дээр харуулсан үр дүнг хэвлэж гаргана.

For гэсэн мөрийнхөө төгсгөлд (;) тавьдаггүй. Учир нь энэ мөр ба давталтын бие нь нэг бүхэл комманд гэсэн утгыг илэрхийлэх юм. Хэрэв уг тэмдгийг тавьбал компилятор ямар нэгэн алдаа өгөхгүй бөгөөд харин давталтын биеийг хоосон гэж үзэх болно.

Одоо For давталтын гурван үйлдлийн тухай дэлгэрэнгүй үзье.

Анхны утга оноох

Энэ үйлдэл нь зөвхөн давталт эхлэх үед ганцхан удаа биелэгдэх бөгөөд давталтын хувьсагчид хамгийн анх авах утгыг оноох үүрэгтэй юм. Дээрх жишээнд j = 0 болно.

Нөхцөл шалгах

Энэ үйлдэл нь ихэвчлэн харьцуулах операторыг ашиглана. Энэ үйлдэл нь давталт бүрийн өмнө биелэх бөгөөд давталт дахин хийгдэх эсэхийг тогтоох үүрэгтэй. Нөхцлийн үр дүн үнэн бол давталт дахин биелэгдэх бөгөөд эсрэг тохиолдолд давталт дуусч удирдлага давталтын дараагийн коммандад очно. Дээрх жишээнд давталтын дараа комманд байхгүй тул программ шууд дуусна гэсэн үг юм.

Утга өөрчлөх

Энэ үйлдэл давталтын хувьсагчийн утгыг өөрчлөх бөгөөд ихэнх тохиолдолд нэмэгдүүлэх үйлдэл байдаг. Энэ нь давталт бүрийн дараа биелэх бөгөөд дээрх жишээнд j-ийн утга 1-ээр нэмэгдэнэ. Харин энд ашиглагдаж байгаа ++j гэсэн үйлдэл нь өөр ямар ч программчлалын хэл дээр байдаггүй үйлдэл юм. Энэ нь j-ийн утгыг нэгээр нэмэгдүүлнэ гэсэн үг. Мөн үүнтэй адил хэд хэдэн үйлдэл байдаг. Хэрэв --j гэж бичвэл j-ийн утга нэгээр хорогдоно гэсэн үг. Программчлалын бусад хэл дээр энэ үйлдлийг

```
j=j+1 эсвэл j=j-1
```

гэж гүйцэтгэж болно. Энэ хоёр үйлдэл нь хувьсагчийн утгыг зөвхөн нэгээр л өөрчилдөг байна. Хэрэв утгыг нь нэгээс ялгаатай тоогоор өөрчилье гэвэл += болон -= гэсэн үйлдлүүдийг ашиглаж болно. Үүнийг доорх байдлаар бичнэ.

```
j+=3
j-=5
```

Энэ нь j-ийн утгыг 3-аар нэмэгдүүлье (эсвэл 5-аар хорогдуулъя) гэж байгаа хэрэг юм. Энэ нь бусад хэлний дараах үйлдэлтэй мөн адил болно.

```
j=j+3
j=j-5
```

Хэдэн удаа давтах вэ?

Бидний жишээнд байгаа давталт яг 15 удаа биелэх ёстой. Анхны утга оноох үйлдлээр j-д 0 гэсэн утга өгсөн бөгөөд j нь 14 утга автлаа биелэх ёстой. Үүнийг шалгаж буй утгаас мэдэж болно. Харин j = 15 болоход давталт зогсоно. Яг энэ утга дээр биелэхгүй. Эндээс ажиглахад тогтмол тооны давталтыг 0-ээс эхлүүлээд, бага тэнцүү болон эрс бага үйлдлүүдийг ашиглан итерацийн тоог тогтоож, давталт бүрийн төгсгөлд нэмэгдүүлэхээр зохион байгуулдаг байна.

Давталтын бие дэх бүлэг үйлдлүүд

Программчлах явцад давталтын биед нэг биш хэд хэдэн үйлдэл хийх шаардлага их гардаг. Ийм үйлдлүүдийг функцуудийн нэгэн адил гоё хаалтанд ({ }) хашиж бичиж өгнө. **Харин хааж байгаа хаалтны дараа цэгтэй таслалыг тавих шаардлагагүй гэдгийг мэдэж авах нь зүйтэй.**

Cubelist гэсэн бидний дараагийн жишээнд давталтын биед гурван комманд агуулагдсан байна. Энэ жишээ нь 1-ээс 10 хүртэлх тоонуудын кубуудыг дэлгэцэнд хоёр багана ашиглан хэвлэнэ.

```
//cubelist.cpp
#include <iostream.h>
```

```
#include <iomanip.h>

void main()
{
int numb;
for (numb=1; numb<=10; numb++)
        {
        cout << setw(4) << numb;
        int cube=numb*numb*numb;
        cout << setw(6) << cube << endl;
        }
}
```

Программын үр дүн доорх байдалтай байна.

```
1       1
2       8
3       27
4       64
5       125
6       216
7       343
8       512
9       729
10      1000
```

Бидний жишээнд орсон өөр нэг өөрчлөлт бол давталтын хувьсагчийн анхны утга нь 1 бөгөөд харин давталт 10 хүртэл (9 биш) хийгдэнэ. Учир нь энд “эрс бага” биш “бага тэнцүү” гэсэн оператор ашиглагдсан байна. Давталтын тоо 10 хэвээрээ боловч үйлдлүүд 0-ээс 9 биш, 1-ээс 10 хүртэл тоонууд дээр хийгдэх болно.

Блок ба Хувьсагчийн “нөлөөлөх хүрээний” тухай

Хэд хэдэн коммандуудыг агуулсан давталтын биеийг коммандуудын блок гэж нэрлэж болно. Ийм блок дотор тодорхойлогдсон хувьсагч нь уг блокынхоо гадна талд “нөлөөлдөггүй” тухай анхаарч авах хэрэгтэй. Сүүлийн жишээнд int төрлийн cube гэсэн хувьсагч тодорхойлогдсон билээ.

```
int cube=numb*numb*numb;
```

Энэ хувьсагчид та уг блокийн гадна талд утга олгож болохгүй. Өөрөөр хэлбэл та давталтын дараа

```
cube=10;
```

гэсэн үйлдэл хийвэл компилятор cube гэсэн хувьсагч тодорхойлогдоогүй байгаа тухай алдаа өгнө гэсэн үг.

Харин нэмж зөвлөхөд өөр өөр блокууд дотор ижил нэртэй хувьсагчууд байж болно. Энэ тухай жишээг сүүлд үзэх болно.

Давталтын бичлэгийн хэлбэрийн тухай

Нийтлэг ашиглагдах бичлэгийн хэлбэрт давталтын бие нь догол мөр оруулсан байдалтай буюу ө.х. давталтын бие нь давталтын эхлэлээс баруун тийш шилжсэн байдалтай бичигддэг юм. For жишээнд 1 мөр, cubelist жишээнд давталтын блок бүхлээрээ шилжиж байрласан байна. Ийм бичиглэл нь давталтын блокийн эхлэл болон төгсгөлийг программчлагчид тодорхой харуулж байх гол зорилготой юм. Харин компиляторт бол ингэж бичсэн эсэх нь огт хамаагүй.

Мөн үүнээс гадна нийтлэг ашиглагдах нэгэн бичлэгийг доор үзүүллээ.

```
for (numb=1; numb<=10; numb++) {
        cout << setw(4) << numb;
        int cube=numb*numb*numb;
        cout << setw(6) << cube << endl; }
```

Энэ бичлэг нь мөр хэмнэдэг талтай ч уншихад хэцүү, нээж байгаа хаалтыг олж харахад төвөгтэй, мөн олон хаалтны тухайд харгалзах хаалтуудыг олоход хүндрэлтэй болно.

C++ -ийн зүгшрүүлэгчийг ашиглах нь

Давталтын үйлдлийн алхам бүрийг хянаж харахын тулд та C++ -ийн зүгшрүүлэгчийг ашиглаж болно. Үүний тулд програмаа засварлаж дуусаад та F8 товчийг дарах хэрэгтэй (Энэ үед шаардлагатай бол программыг дахин хөрвүүлнэ). Ингэсний дараагаар программын хамгийн эхний биелэх мөр гэрэлтэж харагдах болно (Хувьсагчийг тодорхойлох мөрүүд биелэх мөрт тооцогдохгүй). Одоо биелэх гэж байгаа мөр гэрэлтэж харагддаг. Ө.х. F8 товчийг дарах бүрд гэрэлтэлт шилжинэ гэсэн үг. Үүнийг ашиглан давталт хэрхэн биелэгдэхийг хянаж болно.

Хувьсагчдын утгыг харуулах цонх

Дээр дурдсан зүгшрүүлэгчийг ашиглах явцад бидний тодорхойлсон хувьсагчууд ямар утга авч байгааг харж болно. Үүний тулд Debug менюний Watch гэсэн коммандаас Add Watch гэсэн дэд коммандыг сонгох шаардлагатай. Гарч ирсэн харилцах цонхонд харахыг хүсч байгаа хувьсагчийнхаа нэрийг бичиж өгнө. Сүүлийн жишээний хувьд numb гэж бичиж өгье. Дэлгэцийн

доод хэсэгт Watch гэсэн нэртэй шинэ цонх үүсч, өгсөн хувьсагчийн утга уг цонхонд харагдах болно. Зүгшрүүлэлтийн алхам бүрд энэхүү цонхонд гарсан утга өөрчлөгдөхийг та үзэж болно. Та шаардлагатай бүх хувьсагчдынхаа утгыг энэ цонхыг ашиглан үзэж болно.

Зүгшрүүлэгч болон Watch цонх бол программд гарч байгаа алдааг илрүүлэх хамгийн найдвартай багаж юм. Таны программ хүссэний дагуу ажиллахгүй тохиолдолд эдгээр багажуудыг ашиглах нь зүйтэй.

FOR Давталтын хувилбарууд

Давталтын утга өөрчлөх үйлдэл нь зөвхөн утгыг нэмэгдүүлээд зогсохгүй, дурын үйлдэл байж болно. Бидний доорх жишээнд давталтын утга буурч байна. Factor гэсэн энэ жишээ нь хэрэглэгчийн оруулж өгсөн тооны факториалыг тооцож гаргах юм(Бүхэл тооны факториал гэдэг нь уг тооноос бага болон тэнцүү бүх бүхэл тоонуудын үржвэрийг хэлнэ). Жишээ нь: 5-ийн факториал нь 5*4*3*2*1=120 юм.

```
//factor.cpp

void main()
{
unsigned int numb;
unsigned long fact=1;

cout << “Тоо оруулна уу: ”;
cin >> numb;

for (int j=numb; j>0; j--)
          fact*=j;

cout << “ Факториал нь ” << fact;
}
```

Энэ жишээнд давталтын хувьсагчийн анхны утга нь numb, давталтын эцсийн утга 1 (numb > 0), давталт бүрд давталтын утга 1-ээр буурч байна. Тооны факториал гэдэг харьцангуй асар их тоо гардаг учраас бид энд unsigned long төрлийн хувьсагч ашиглаж байна. Программын үр дүнгийн нэг жишээ нь:

```
Тоо оруулна уу: 10
Факториал нь 3628800
```

FOR давталтын бичлэгт хувьсагч тодорхойлох нь

Энэ жишээнд бас нэгэн шинэ бичиглэл оржээ. Давталтын хувьсагч j давталтын бичиглэл дотор шууд тодорхойлогдсон байна.

```
for (int j=numb; j>0; j--)
```

Ийм бичиглэл C++-д нэлээн өргөн ашиглагдана. Бичиглэлийн энэ хэлбэр нь хувьсагчийг тодорхойлж байгаа хамгийн тодорхой бөгөөд энгийн хэлбэр юм. Ийм хувьсагч нь тодорхойлсон цагаасаа эхлэн бүхэл программын хүрээнд ашиглагдаж болох бөгөөд блок дотор тодорхойлогдсон хувьсагч шиг ашиглагдах хүрээ нь хязгаарлагдахгүй.

Хэд хэдэн анхны утга оноох ба нөхцөл шалгах

For давталтын бичиглэлд хэд хэдэн утга оноох ба нөхцөл шалгах үйлдлүүдийг нэг зэрэг хийж болно. Үүний тулд таслал (,) ыг ашиглана. Эсвэл утга

өөрчлөх үйлдлийг ч хэд хэдээр нь хийж болно. Доорх жишээнд энэ тухай үзүүлж байна.

```
For (j=0, alpha=100; j<50; j++, beta--)
```

Энэ жишээнд j гэсэн энгийн давталтын хувьсагч байгаа боловч мөн alpha хувьсагчид анхны утга оноож, beta хувьсагчийн утгыг өөрчилж байна. Энд байгаа j болон alpha, beta хувьсагчууд заавал бие биетэйгээ холбоотой буюу харилцаж байх шаардлагагүй.

WHILE Давталт

Өмнө үзсэн for давталт нь нэгэн бүлэг тодорхой тоогоор давтан биелүүлдэг. Гэтэл давталт хэдэн удаа биелэх нь огт мэдэгдэхгүй байх тохиолдолд яах вэ?

EndOn0 гэх дараагийн жишээнд хэрэглэгчээс тоо оруулах бөгөөд хэрэглэгч 0 гэсэн утга өгтөл давталтыг үргэлжлүүлэх юм. Энэ программын хувьд хэрэглэгч хэдэн удаа тэгээс ялгаатай утга өгөх нь тодорхой биш юм.

```
//EndOn0.cpp
#include <iostream.h>

void main()
{
int n=99;
while (n!=0) cin>>n;
}
```

Энэ программын үр дүнгийн нэгэн жишээ дор өгөдсөн байна. Энд үзүүлсэн мөрийн тоо хэд ч байж болно.

```
1
```

```
27
33
144
9
0
```

While давталт нь for давталтын хялбар хувилбар нь гэж ойлгож болох юм. Учир нь энэ давталтанд шалгах нөхцлөөс өөр ямар ч анхны утга оноох ба утга өөрчлөх үйлдлүүд байхгүй юм.

Нөхцөл л үнэн байвал давталт үргэлжилсээр л байх болно. Дээрх жишээнд хэрэглэгч 0 гэсэн утга оруултал n!=0 гэсэн нөхцөл үнэн хэвээрээ байх болно.

Анхааруулга: Давталтын хувьсагчийн анхны утга давталт эхлэхээс өмнө өгөгдөх бөгөөд харин давталт дотор давталтын хувьсагчийн утгыг өөрчлөх ямар нэгэн үйлдэл хийж өгөх ёстой бөгөөд ингээгүй тохиолдолд энэ нь үл дуусах давталт болно.

Дээрх жишээнд давталтын утга өөрчлөх үйлдэл нь cin >> n юм.

While давталтаар бүлэг үйлдэл хийх

While4 гэсэн дараагийн жишээ нь while давталтанд бүлэг үйлдэл хэрхэн хийхийг харуулж байна. Энэ нь Cubelist жишээний өргөтгөсөн хэлбэр бөгөөд Cubelist нь тооны кубыг жагсаадаг бол энэ жишээ нь 4 зэргийг харуулах юм. Бидэнд программын үр дүнг ердөө 4 тэмдэгтийн зайнд хэвлэх шаардлага гарчээ гэж үзвэл жагсаах тоонууд маань 4 зэрэг нь 9999-ээс хэтрэхгүй байх ёстой болно. Тэгвэл урьдчилан тооцолгүйгээр ямар тооны 4 зэрэг үүнээс хэтрэхгүй байхыг мэдэх боломжгүй юм. Иймд доорх маягийн программ бичих хэрэгтэй.

```
//while4.cpp
#include <iostream.h>
#include <iomanip.h>

void main()
{
int pow=1;
int numb=1;
while (pow<9999)
        {
        cout << setw(2) << numb;
        cout << setw(5) << pow << endl;
        ++numb;
pow=numb*numb*numb*numb;
        }
}
```

Pow хувьсагч нь Numb-ийн 4 зэргийг хадгалах бөгөөд давталтын нөхцөлд numb-ийг биш, pow хувьсагчийг шалгаж байна. Үр дүн нь:

```
1       1
2       16
3       81
4       256
5       625
6       1296
7       2401
8       4096
9       6561
```

Дараагийн утга нь 10000 гарах бөгөөд энэ нь бидний нөхцөлд харшлах тул программ дуусч байна.

Арифметик үйлдэл болон харьцуулах үйлдлийн биелэх эрэмбийн тухайд

Дараагийн жишээ маань үйлдлийн биелэх эрэмбийг харуулна. Энэ жишээ нь Фибаноччийн цувааг дэлгэцэнд хэвлэн гаргах юм.

1 1 2 3 5 8 13 21 34 55 г.м.

Цувааны гишүүн бүр өмнөх хоёр гишүүний нийлбэрээс үүсч байгаа нь жишээнээс харагдаж байна. Фибаноччийн цуваа нь компютерийн эрэмбэлэх аргуудаас эхлэн наранцэцэг дэх спиралийн тоо хүртэлх шинжлэх ухааны олон салбаруудад өргөн хүрээтэй ашиглагддаг юм. Эртний Грекийн сүм, ер нь уран барилга болон уран зурагт өргөн ашиглагддаг алтан харьцаанд тасралтгүй дөхдөг нь энэ цувааны хамгийн гайхалтай чанар юм.

```
//fibo.cpp
#include <iostream.h>

void main()
{
const unsigned long limit=4294967295;
unsigned long next=0;
unsigned long last=0;
while (next<limit/2)
        {
        cout << last << " ";
        long sum=next+last;
        next=last;
        last=sum;
        }
}
```

Программын үр дүнг үзэцгээе.

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181
6765 10946 17711 28657 46368 75025 121393 196418 317811
514229 832040 1346269 2178309 3524578 5702887 9227465
14930352 24157817 39088169 63245986 102334155 165580141
```

267914296 433494437 701408733 1134903170 1836311903
2971295073

Энэ жишээнд хамгийн том бүхэл тоон утга авдаг unsigned long төрлийн хувьсагч ашиглаж байна. Харин while давталтын нөхцөлд цувааны дараагийн гишүүн маань энэ төрлийн утгаас илүү гарах эсэхийг шалгаж байгаа юм. Энэ хязгаар нэгэнт өөрчлөгдөхгүй учраас бид Limit хувьсагчийг тогтмолоор тодорхойлж өгөх боломжтой. Одоо давталтын нөхцлөө эргэж нэг харцгаая.

```
(next<limit/2)
```

Энд бид next-ийн утгыг limit/2-ийн утгатай жишиж байгаа билээ. Уг нь бид үйлдлийг зөв хийлгэхийн тулд

```
(next<(limit/2))
```

гэж бичих ёстой. Гэтэл дээрх жишээнд яагаад ингэж бичилгүйгээр зөв үр дүнд хүрсэн бэ? Хариулт нь их хялбархан. Учир нь арифметик үйлдэл нь биелэх эрэмбээрээ харьцуулах үйлдлээс өндөр байдагт байгаа юм. Биелэх эрэмбийн хүснэгтийг бүлгийн төгсгөлд логик операторуудтай цуг үзнэ.

DO Давталт

Өмнөх while давталтын хувьд давталтын нөхцлийг эхэнд нь шалгаж байна. Хэрвээ давталт эхлэхэд давталтын нөхцөл худал утгатай байвал уг давталт огт биелэхгүй юм. Гэтэл зарим тохиолдолд нөхцөл ямар байхаас хамааралгүйгээр давталтын бие ядаж нэг удаа биелэх шаардлага гарч болно. Энэ

тохиолдолд та нөхцлөө төгсгөлд нь шалгадаг do гэсэн давталтыг ашиглах нь илүү зохимжтой.

Divdo гэсэн дараагийн жишээнд хэрэглэгчийн оруулж өгсөн эхний тоог хоёр дахь тоонд хувааж, ногдвор болон үлдэгдлийг нь дэлгэцэнд хэвлэж байна.

```
//divdo.cpp
#include <iostream.h>

void main()
{
long dividend, divisor;
char ch;
do
        {
        cout << “Хуваагдагч: ”; cin >> dividend;
        cout << “Хуваагч: ”; cin >> divisor;
        cout << “Бүхэл нь ” << dividend/divisor;
        cout << “, үлдэгдэл нь ” << dividend%divisor;
        cout << “\nYргэлжүүлэх үү? (Y/N): ”;
        cin >> ch;
        }
while (ch!=’n’);
}
```

Энэ программ ердөө ганцхан do давталтаас тогтож байна. Do гэсэн үг давталтын эхлэлийг зааж, давталтын бие {} хаалтаар хашигдсан байна. Харин давталтын төгсгөлийг while мөр илэрхийлж байгаа юм. **Энд байгаа while мөр нь while давталтын толгойтой үндсэндээ адилхан бөгөөд харин давталтын төгсгөлд бичигдсэн тул хойно нь (;) тавих ёстойг мартаж болохгүй.**

Давталт бүрийн төгсгөлд хэрэглэгчээс дахин ажиллах эсэхээ лавлах бөгөөд хэрэв хэрэглэгч ‘n’-ээс ялгаатай үсэг дарвал нөхцөл үнэн утга авч давталт

дахин биелэх юм. Доор программын үр дүнгийн нэгэн жишээг харуулъя.

```
Хуваагдагч: 11
Хуваагч: 3
Бүхэл нь 3, үлдэгдэл нь 2
Үргэлжлүүлэх үү? (Y/N) y
Хуваагдагч: 222
Хуваагч: 17
Бүхэл нь 13, үлдэгдэл нь 1
Үргэлжлүүлэх үү? (Y/N) n
```

Давталтуудыг хэрхэн ашиглавал зохимжтой вэ?

Өмнөх бүлгүүдэд давталтын талаар ерөнхий ойлголтыг та бүхэнд өглөө. For давталт нь уг блок хэдэн удаа биелэх шаардлагатайг мэдэж байгаа тохиолдолд ашиглахад зохимжтой юм. Харин үлдсэн хоёр давталтыг давталт хэдийд дуусах нь тодорхойгүй тохиолдолд ашиглах бөгөөд дор хаяж нэг удаа биелэх шаардлагатай бол do давталтыг харин эсрэг тохиолдолд while давталтыг ашиглах нь зүйтэй. Гэхдээ эдгээр нь зөвхөн зөвлөгөө бөгөөд программчлагч өөрөө зохицуулж чадвал аль ч тохиолдолд ямар ч давталтыг нь ашиглаж болох юм.

Нөхцөл шалгах

Дээр үзсэн давталтын оператор нь цаг үргэлж ганц л асуудалтай тулгарч байна. Давталтыг дахин хийх үү, эсвэл зогсоох уу?

Бодит амьдрал дээр ч хүн үргэлж сонголт хийж байдаг. Ажлаа дахин хийх үү, болих уу; цамц ногоон өнгөтэйг авах уу, улааныг авах уу гэхчилэн эсвэл амрах уу, болих уу; амарвал ууланд амрах уу, далайн эрэг дээр амрах уу г.м.

Программ зохиоход ч гэсэн цаг үргэлж сонголттой учирдаг. Ямарваа нэгэн нөхцлийг шалгаад түүнийхээ үр дүнгээс хамааран программын өөр өөр хэсгүүд рүү ‘ харайж ’ очих болно. C++-д нөхцөл шалгах хэд хэдэн арга бий. Эдгээрээс хамгийн нийтлэг ашиглагдах нь If…else гэсэн оператор юм. (Мөн энэ комманд нь else -гүй ч ашиглагдаж болно.) Энэ оператор нь зөвхөн 2 сонголт хийдэг бол switch гэх дараагийн оператор нь хэд хэдэн сонголтоор үйлдлийг хийх бололцоотой болох юм. Хэрэв нөхцөлт операторуудыг ашиглавал илүү нарийн үйлдлүүдийг ч хийх боломжтой болно. Энэ бүгдийн тухай дэлгэрүүлэн үзье.

IF Оператор

IF оператор бол хамгийн энгийн нөхцөл шалгах оператор юм. Ifdemo программаас харахад ойлгомжтой болно.

```
//ifdemo.cpp
#include <iostream.h>

void main()
{
int x;
cout << “Тоо оруулна уу : ”;
cin >> x;
if (x>100) cout << “Энэ тоо 100-аас их байна\n”
}
```

If гэсэн нөөц үгийн ард шалгах нөхцлийг хаалтанд хийж бичиж өгсөн байна. Эндээс харахад If –ийн бичлэг нь While давталтын бичлэгтэй тун төстэй байна. Ганц гол ялгаа нь If –ийн ард байгаа үйлдлүүд зөвхөн ганц биелэнэ, харин While -ийн ард байгаа үйлдлүүд давтагдан биелэнэ. Программын үр дүн доорх байдалтай харагдана.

```
Тоо оруулна уу : 2000
Энэ тоо 100-аас их байна
```

Харин оруулсан тоо 100-аас бага байвал энэ программ ямар ч үйлдэл хийхгүй.

IF операторт бүлэг үйлдэл хийх

Давталтын нэгэн адилаар энэхүү нөхцөл шалгах оператор нь өөрийн биедээ нэг болон хэд хэдэн бүлэг үйлдэл (блок) хийх боломжтой байна.

```
//if2.cpp
#include <iostream.h>

void main()
{
int x;
cout << “Тоо оруулна уу : ”;
cin >> x;
if (x>100)
        {
        cout << x << “нь “;
        cout << “100-аас их байна\n”
        }
}
```

Үр дүн нь :

```
Тоо оруулна уу : 12345
12345 нь 100-аас их байна
```

Давталтын дотор нөхцөл шалгах нь

Бидний үзэж байгаа давталт болон нөхцөлт шалгалт нь нэг нь нөгөөгөө агуулсан хэлбэртэй байж болдог. Ө.х. нөхцөл нь нөхцөл дотроо, нөхцөл нь давталт дотроо, давталт нь давталт дотроо, давталт нь нөхцөл дотроо тут тус агуулагдаж болно. Доор үзүүлсэн жишээ нь prime-д үүнийг тод харуулжээ. Энэ жишээ нь хэрэглэгчийн өгсөн тоог анхных эсэхийг нь тогтоох юм.(Өөрөөсөө гадна 1 ээс өөр хуваагчгүй тоог анхны тоо гэнэ.)

```
//prime.cpp
#include <iostream.h>
#include <process.h>

void main()
{
unsigned long n, j;
cout << “Тоо оруулна уу: ”;
cin >> n;
for (j=2; j<=n/2; j++)
          if (n%j==0)
          {
          cout << “Энэ тоо ” << j
        << “-д хуваагдана.” << endl;
          exit(0);
          }
cout << “Энэ бол анхны тоо\n”;
}
```

Хэрэглэгчийн оруулсан утгыг N хувьсагчид авч, 2-оос N/2 хүртэл давталт хийж байна. Хуваагч нь j хувьсагч юм. Хэрэв j–ийн ямар нэгэн утганд N нь хуваагдаж байвал энэ нь анхны тоо биш болно.

Хуваагдаж байгаа эсэхийг үлдэгдэл нь 0-тэй тэнцэж байгаа эсэхээр нь мэдэж авна. Хэрэв анхных биш бол энэ тухай мэдээлээд программаас шууд гарч байна. Үр дүнгийн жишээ:

```
Тоо оруулна уу: 13
Энэ бол анхны тоо
Тоо оруулна уу: 22229
Энэ бол анхны тоо
Тоо оруулна уу: 22231
Энэ тоо 11-д хуваагдана.
```

(Хэрэв жишээнд үзүүлснээс их тоо өгвөл программ уг тоог шалгахын тулд мэдэгдэхүйц хугацаа зарцуулах болно.)

Дээрх жишээнд for давталтын биед нь {} хаалт хийгээгүй. Учир нь If оператор нь биеийнхээ хамтаар нэг үйлдэл гэж тооцогдож байгаа юм.

Exit () функцийн тухай товчхон

Дээрх жишээ уг тоо анхных биш гэдгийг тогтоосноос хойш давталтыг цааш үргэлжлүүлэх шаардлагагүй. Тийм учраас мэдээллийг өгөөд exit функцийг ашиглан шууд гарч байна. Энэ функц нь программын хаана байгаагаас үл хамааран программыг шууд дуусгаж байгаа юм. Энэ функцийн ганц параметр нь уг программ яаж дууссан тухай мэдээллийг үйлдлийн системд өгдөг. Бидний жишээнд 0 утга өгч байгаа нь уг программ хэвийн дууссаныг илэрхийлж байна. (Энэ код нь пакет файлд (*.bat) өргөн ашиглагдах бөгөөд errorlevel гэсэн функцээр дээрх утгыг авч болно.)

IF … ELSE оператор

Өмнөх if оператор нь ямар нэгэн нөхцөл үнэн утгатай байвал бүлэг үйлдлүүдийг хийх боломжоор хангаж өгсөн. Харин худлаа байвал удирдлага дараагийн коммандад шилжинэ. Харин үнэн байвал нэг хэсэг үйлдэл, харин худлаа байвал өөр нэг бүлэг үйлдэл хийх шаардлага гарвал if…else гэсэн хэлбэрийн операторыг ашиглах юм. Энэ оператор нь

if (нөхцөл) {үйлдлийн блок} else {үйлдлийн блок}

гэсэн хэлбэртэй байна. Доор ifelse.cpp гэсэн жишээ өгөгджээ.

```
//ifelse.cpp
#include <iostream.h>

void main()
{
int x;
cout << “Тоо оруулна уу : ”;
cin >> x;
if (x>100)
        cout << “Энэ тоо 100-аас их\n”;
else
        cout << “Энэ тоо 100-аас бага\n”;
}
```

Хэрвээ шалгасан нөхцөл үнэн бол программ нэг мэдэгдэл гаргана, харин худлаа байвал өөр мэдэгдэл гаргах юм.

GETCHE () хэмээх функцийн тухай

Дараагийн жишээнд while дотор агуулагдсан if…else операторыг харуулжээ. Мөн энэ жишээнд getche гэсэн шинэ функцийг ашигласан байна. Chcount гэсэн нэртэй энэ программ нь хэрэглэгчийн бичсэн мөрөн дэх үсэг болон үгийг тоолж гаргана.

```
//chcount.cpp
#include <iostream.h>
#include <conio.h>

void main()
{
int chcount=0;
int wdcount=1;
char ch='a';
while (ch!='\r')
        {
        ch=getche();
        if (ch=' ')
                wdcount++;
        else
                chcount++;
        }
cout << "\nYr=" << wdcount << endl
        << "Ycэг=" << chcount << endl;
}
```

Үүнээс өмнө бид гарнаас мэдээлэл авахдаа зөвхөн CIN урсгалыг ашиглаж байсан билээ. Энэ урсгалыг ашиглахад хэрэглэгч ENTER товч дарахаас наана бид мэдээллийг авч чадахгүй юм. Ө.х. бид хэрэглэгчийн дарсан товч бүрийг биш, ENTER дартал оруулсан бүтэн мөрийг л авдаг. Гэтэл бидэнд хэрэглэгчийн дарсан товч бүрийг авах шаардлага гарч байна. Харин getche() функц нь энэ шаардлагад нийцэх юм. Энэ функц нь гарнаас нэг л товч дарахыг хүлээж байдаг байна. Энэ функц ямар нэгэн аргумент шаардахгүй, харин conio.h гэсэн толгой файлыг тодорхойлж өгсөн байх ёстой. Дээрх жишээнд getche

–ийн үр дүн ch гэсэн хувьсагчид ирж байна. (Энэ функцээс гадна getch() гэсэн функц байх бөгөөд getche() нь хэрэглэгчийн оруулсан товчийг дэлгэцэнд харуулдагаараа ялгаатай юм.)

Уг жишээнд хэрэглэгчийн оруулсан тэмдэгт space (хоосон зай) байвал wdcount хувьсагчийн утга, үүнээс ялгаатай тэмдэгт байвал chcount хувьсагчийн утга нэгээр нэмэгдэж байна. Ө.х. үгнүүдийг хооронд нь хоосон зайгаар тусгаарласан байгаа гэж үзэж байна. Харин дараалсан хоосон зайнууд оруулвал тэр бүрийг үг гэж тоолно. Программын үр дүнгийн жишээ :

```
This is an example
Үг=4
Үсэг=15
```

Хэрэглэгч мөрийг оруулж дуусаад ENTER дарахад getche() функц ‘\r‘ гэсэн утга өгч программ дуусна.

Утга оноох үйлдэл

```
//chcnt2.cpp
#include <iostream.h>
#include <conio.h>

void main()
{
int chcount=0;
int wdcount=1;
char ch;
while ((ch=getche())!='\r')
        {
        if (ch=' ')
                wdcount++;
        else
                chcount++;
```

```
        }
cout << "\nYr=" << wdcount << endl
        << "Yсэг=" << chcount << endl;
}
```

Энэ жишээ нь өмнөх жишээтэй бараг адилхан харагдах боловч тун ялихгүй нэгэн ялгаа бий. getche() функцийн утгыг ch хувьсагчид оноож байгаа мөр нь while давталтын нөхцөл шалгах хэсэгт зөөгдсөн байна. Утга оноох үйлдлийн үр дүн нь уг оноож байгаа утга өөрөө гардгийг ашиглан ингэж мөр хэмнэсэн юм. Ө.х. getche() гэсэн функц 'a' гэсэн утга өгсөн бол ch=getche() гэсэн үйлдийн үр дүн нь тэрхүү 'a' өөрөө байдаг ажээ. Ингээд тэрхүү үр дүнг шууд '\r'-тэй жишиж байна. Үүнээс үндэслэн утга оноох үйлдэл доорх хэлбэрээр ч бичигдэж болно.

```
x=y=z=0
```

Эхлээд Z нь 0 утга авна. Энэ үйлдлийн дүн нь мөн 0 учраас Y нь 0 утга авч, үүгээр дамжуулан X нь мөн 0 утгатай болно.

Харин утга оноох үйлдэл нь харьцуулах үйлдлээс бага эрэмбэтэй учраас (ch = getche()) үйлдэлд хаалт хийхгүй байж болохгүй. Энэ хаалтгүй бол харьцуулах оператор эхэлж биелээд түүний үр дүн ch хувьсагчид оноогдоход хүрнэ.

Тэгэхээр chcnt2 жишээн дэх while оператор нь тун бага зайнд маш олон үйлдэл багтааж байгаа юм. Үүнийг анхлан харж байгаа хүн уншихад их төвөгтэй байдаг.

Давталтын дотор if…else – ийг хэрэглэх

Одоо adifelse гэсэн жижиг хэмжээний адал явдалт тоглоом зохиоцгооё.

```
//adifelse.cpp
#include <iostream.h>
#include <conio.h>

void main()
{
char dir='a';
int x=10, y=10;
cout << "Гарахыг хүсвэл Enter товч дарна уу\n"
while (dir!='\r')
        {
        cout << "Байрлал : " << x <<"," << y;
        cout << "\nТовч дарна уу(n, s, e, w):";
        dir=getche();
        if (dir=='n')
                y--;
        else
                if (dir=='s')
                        y++;
                else
                        if (dir=='e')
                                x++;
                        else
                                if (dir=='w')
                                        x--;
        }
}
```

‘ Тоглоом ’ эхлэхэд та хоосон талд зогсож байх юм. Таны байрлал 10, 10 гэсэн координат дээр эхлэх бөгөөд заасан чиглэлийн товчийг дарснаар та дөрвөн зүг рүү нэг нэг алхмаар явах юм. Ингээд ENTER товч дартал та хоосон талаар зугаацаж болно. Одоохондоо энэ жишээ маань таны байрлалыг харуулахгүй, харин байрлаж байгаа координатыг дэлгэцэнд хэвлэх юм. Харин цаашдаа энэ

программаа өргөтгөн, боловсронгуй болгох болно. Программ доор байдлаар ажиллана.

```
Байрлал : 10,10
Товч дарна уу(n, s, e, w): n
Байрлал : 10,9
Товч дарна уу(n, s, e, w): e
Байрлал : 11,9
Товч дарна уу(n, s, e, w):
```

Энэ программ нь жинхэнэ тоглоом шиг санагдахгүй ч гэсэн “ юм ганцаас эхэлдэгийг “ санах хэрэгтэй.

Энэ жишээнд нөхцөл шалгах операторууд давталт дотор агуулагдсан байна. Түүгээр ч зогсохгүй нөхцөл шалгах оператор нь нөхцөл шалгах оператор дотроо агуулагдсан байна. Хэрвээ эхний нөхцөл худлаа бол дараагийн нөхцлийг гэх мэтчилэн бүх шалгалтууд хийгдэх болно. Харин хэрэв аль нэг нөхцөл биелэх аваас харгалзах үйлдлийг хийж шууд нөхцөл шалгалтаас гарна. Ингэж байрласан нөхцөл шалгалтуудыг *шийдвэрийн мод* ч гэж нэрлэсэн байдаг.

Гэтэл if…else операторыг ашиглах үед амаргүй нэгэн асуудал гардаг. Энэ тухай доорх жишээ дээр үзье.

```
//badelse.cpp
#include <iostream.h>

void main()
{
int a, b, c;
cout << “a, b, c тоонуудыг оруулна уу:”;
cin >> a >> b >> c;
if (a==b)
        if (b==c)
```

```
                cout << “Энэ тоонууд тэнцүү байна\n”;
else
                cout << “a, b нь ялгаатай”;
}
```

a, b, c-д харгалзах утгуудыг өгье. Тэдгээр нь хоорндоо ялгаатай байг. Эхний нөхцөл биелэхгүй учраас else хэсгээр дамжин “a, b нь ялгаатай” гэсэн текст хэвлэгдэх ёстой. Гэтэл үнэн хэрэгтээ юу ч хэвлэгдэхгүй. Учир нь гэвэл бид else-ийг эхний if-тэй харгалзуулан бичсэн боловч жинхэнээрээ энэ нь хоёр дахь if-д харгалзаж байгаа юм. **If…else-үүд нь биедээ багтсан байдалтай байхаас биш, огтлолцож байрлахыг зөвшөөрөхгүй.** Тийм учраас хоёр дахь if–ийн else-ийг бичилгүйгээр нэг дэх if-ийн else-ийг бичиж болохгүй юм. Харин доорх байдлаар бичих ёстой.

```
if (a==b)
        if (b==c)
                cout << “a, b, c тоонууд тэнцүү”;
        else
                cout << “b, c нь ялгаатай”;

else
        cout << “a, b нь ялгаатай”;
```

Хэрэв та заавал ганц else ашиглах шаардлагатай байгаа бол доорх хэлбэрээр бичиж бас болно.

```
if (a==b)
{
        if (b==c)
                cout << “a, b, c тоонууд тэнцүү”;
}
else
        cout << “a, b нь ялгаатай”;
```

SWITCH Оператор

Хэрвээ таны “шийдвэрийн мод” дэндүү томдоод байгаа, тэгээд бүх нөхцлийн хамаарч байгаа утга нь ижил байвал та энэхүү switch операторыг ашиглах нь зүйтэй болно. Доорх жишээг авч үзэцгээе. Энэ программ нь хэрэглэгчийн өгсөн утгаас хамааран гурван төрлийн мэдэгдэл гаргадаг.

```
//platters.cpp
#include <iostream.h>

void main()
{
int speed;
cout << “\n33, 45, 78-ийн аль нэгийг оруул : ”;
cin >> speed;
switch(speed)
        {
        case 33:
                cout << “МУИС\n”;
                break;
        case 45:
                cout << “УБИС\n”;
                break;
        case 78:
                cout << “ТИС\n”;
                break;
        }
}
```

Switch-ийн ард хаалтанд бичигдсэн хувьсагчийг сонголтын хувьсагч гэнэ. Case гэсэн үг бүрийн ард байгаа тогтмол утгууд нь сонголтын хувьсагчийн төрөлтэй таарч байх ёстой. Switch рүү орохын өмнө сонголтын хувьсагчид утга оноогдсон байх ёстой бөгөөд тэрхүү утга нь аль нэг case-ийн утгатай таарч байвал түүнд харгалзах үйлдлүүд нь break оператор

хүртэл (эсвэл сонголтын операторын төгсгөл хүртэл) биелэх юм. Программын үр дүн:

```
33, 45, 78-ийн аль нэгийг оруул : 33
МУИС
```

Break оператор

Дээр үзсэн жишээнд case бүрийн төгсгөлд break гэсэн оператор бичигджээ. Энэ оператор нь сонголтын операторoос (программаас биш!) гарахыг зааж байгаа юм. Хэрвээ энэ коммандыг орхивол программ шаардлагатай үйлдлүүдийг хийж дуусаад, огт шаардлагагүй буюу тодруулбал дараагийн case – ийн үйлдлүүдийг гүйцэтгэдэг юм. Харин зарим тохиолдолд break – ийг орхих шаардлага гарч болох юм.

DEFAULT гэсэн түлхүүр үг

```
//adswitch.cpp
#include <iostream.h>
#include <conio.h>

void main()
{
char dir='a';
int x=10, y=10;

while (dir!='\r')
        {
        cout << "Байрлал нь " << x <<"," << y;
        cout << "\nТовч дарна уу(n, s, e, w):";
        dir=getche();
        switch(dir)
                {
```

```
                case 'n' : y--; break;
                case 's' : y++; break;
                case 'e' : x++; break;
                case 'w' : x--; break;
                case '\r' : cout "Программ дууслаа…\n";
        break;
                default : cout "Буруу товч\n";
                break;
                }
        }
}
```

Adswitch гэсэн дээрх жишээнд switch операторын хамгийн сүүлийн мөрөнд Default гэсэн түлхүүр үг бичигджээ. Хэрвээ сонголтын хувьсагчийн утга нь аль ч case-ийн утгатай тохирохгүй байвал энэхүү default гэсэн түлхүүр үгийн хойно байгаа коммандууд биелэдэг байна. Тийм учраас энэ тохиолдолд "Try again" гэж хэвлэж байна. Нэгэнт сонголтын хамгийн төгсгөлд бичигдэх учраас break коммандыг бичих шаардлагагүй.

Switch ба If…else

Хэдийд switch операторыг, if…else операторыг ашиглавал зохимжтой вэ? Энэ асуултад хариулахын тулд доорх программын хэсгийг анхаарцгаая.

```
If (SteamPressure*Factor>56)
        . . . . .
else if (VoltageIn+VoltageOut<23000)
                . . . . .
        else if (day==Thursday)
                        . . . . .
```

Энэ жишээнд үзүүлсэн тохиолдолд if…else–ээс өөр аргыг хэрэглэж болохгүй. Учир нь энд орсон хувьсагчуудаас алийг нь ч switch-ийн сонголтын хувьсагч болгох боломжгүй юм. Мөн шалгаж байгаа нөхцөл нь (a<3) гэсэн хэлбэртэй байвал case (a<3) гэж бичиж болохгүй учраас мөн if…else –ийг ашиглахад хүрнэ.

Харин case ('a'+32) гэсэн хэлбэрийн бичиглэл байхыг switch операторт зөвшөөрнө.

Энэ бүхнээс дүгнэхэд switch операторыг ашиглахад доорх нөхцлүүд хангагдсан байх шаардлагатай байна.

- **Бүх нөхцөл ганцхан хувьсагчаас хамаарч байх**
- **Case-ийн утгууд нь тогтмол буюу тогтмол дээр хийгдсэн үйлдэл байх**
- **Сонголт нь 3 ба түүнээс дээш байх**

Нөхцөлт үйлдэл

Программ бичих явцад шалгасан нөхцөл үнэн байвал ямар нэгэн хувьсагч нэг утга авдаг, худал байвал өөр утга авдаг байх тохиолдол тун их таарна. Доорх товч жишээнд ийм тохиолдлыг харуулсан байна.

```
If (alpha<beta)
        Min=alpha;
Else
        Min=beta;
```

C++-ийг зохиогчид үүнийг хэмнэх нэгэн хялбар бичиглэл зохиосон байна. Бусад бүх үйлдэл (+, - г.м.)

ганц тэмдгээс тогтдог байхад энэ үйлдэл ганцаараа 2 тэмдгээс тогтдог.

```
Min=(Alpha<Beta) ? alpha:beta
```

? ба : гэсэн хоёр тэмдэг энэхүү нөхцөлт үйлдлийг үүсгэж байна. Хэрвээ шалгаж буй нөхцөл үнэн байвал асуултын тэмдгийн ард байгаа утга, эсрэг тохиолдолд (:) тэмдгийн ард байгаа утга нь энэхүү нөхцөлт үйлдлийн үр дүн нь болно. Шалгах нөхцлийн гадуур байгаа хаалт нь үнэндээ илүү бөгөөд энд зөвхөн уншихад ойлгомжтой болгох зорилгоор ашиглагдаж байна. Энэ нөхцөлт үйлдэл нь утга оноохоос эхлээд бүх төрлийн үйлдэлд ашиглагдаж болно. Дээрх жишээнд min гэсэн хувьсагчид утга оноож байна. Энэ үйлдлийг ашиглан тооны абсолют хэмжигдэхүүнийг доорх маягаар олж болно.

```
Absvalue=(n<0) ? –n:n;
```

Одоо дэлгэцийн 8 дахь багана бүр дээр (*) хэвлэдэг condi.cpp жишээг үзье.

```
//condi.cpp
#include <iostream.h>

void main()
        {
        for (int j=0; j<80; j++)
                {
                char ch=(j%8) ? ’ ’:’*’ ;
                cout << ch;
                }
        }
```

j-ийн утгыг 8 –д хувааж, үлдэгдлийг нь шалгахад зөвхөн 8-д хуваагдах тоонууд дээр 0 (худал) гэсэн утга, бусад тохиолдолд 0-ээс ялгаатай (үнэн) гэсэн

утга өгнө. Ийм учраас зөвхөн 8-д хуваагдах дугаартай баганууд дээр од, бусад багана дээр хоосон зай хэвлэх юм. Давталтын биед байгаа хоёр мөрийг доорх аргаар нэгтгэж бас болно.

```
cout << ((j%8) ? ' ':'*');
```

Программын мөрүүдийг ингэж шахах нь хэмнэлттэй боловч бичлэгийг улам төвөгтэй болгодог муу талтай. Зарим тохиолдолд программчлагч өөрөө ч ойлгоход төвөгтэй бичиглэл үүсч болох юм.

Логик операторууд

Үүний өмнөх хичээлүүдээр бид хоёр бүлэг үйлдлийн тухай авч үзлээ. (Арифметик үйлдлүүд +, -, *, /, % ба харьцуулах үйлдлүүд <, >, <=, >=, ==, !=)

Одоо логик гэгдэх гурав дах бүлэг үйлдлүүдийг үзье. Ийм үйлдлүүд нь Boolean (үнэн/худал гэсэн хоёрхон утгыг Булийн утга гэнэ) утгуудыг хооронд нь холбох зорилгоор ашиглагддаг. Энгийн математик биш жишээн дээр авч үзье. “ Маргааш дулаахан өдөр болно ” гэсэн өгүүлбэр нэг бол үнэн байна, эсвэл худлаа байна. Өөр нэг өгүүлбэр “ Тэр машинтайгаа ирнэ ”. Мөн л Булийн өгүүлбэр. Одоо харин эдгээр өгүүлбэрүүдийг холбоё. “ Маргааш дулаахан өдөр байгаад *бас* тэр машинтайгаа ирвэл бид зугаалганд явна. ” Энд 2 Булийн өгүүлбэр холбогдохдоо “ бас ” (and) гэсэн холбоос хэрэглэжээ. Энэхүү and холбоосыг хэрэглэснээр “ хоёр өгүүлбэр хоёулаа үнэн байгаа тохиолдолд бид зугаалганд явна “ гэсэн утга санааг илэрхийлж байна.

AND үйлдэл

Одоо C++ дээр Булийн үйлдлүүд хэрхэн холбогдохыг сонирхоё. Advenand.cpp гэсэн дараагийн жишээ бол Adswitch программын сайжруулсан хэлбэр гэж үзэж болно. (7,11) гэсэн координат дээр нэг “ эрдэнэс ” байрлуулаад, тоглогчоор түүнийгээ олуулъя.

```
//advenand.cpp
#include <iostream.h>
#include <process.h>
#include <conio.h>

void main()
{
char dir='a';
int x=10, y=10;
while (dir!='\r')
        {
        cout << "Байрлал нь " << x <<"," << y;
        cout << "\nТовч дарна уу(n, s, e, w):";
dir=getche();
switch(dir)
        {
                case 'n': y--; break;
        case 's': y++; break;
        case 'e': x++; break;
        case 'w': x--; break;
        }
if (x==7 && y==11)
                {
        cout << "Та оллоо!\n";
        exit(0);
                }
        }

}
```

If операторын үр дүн зөвхөн x нь 7-той, y нь 11-тэй тэнцүү үед л үнэн байна. Тэгэхээр логик ‘and’ үйлдлийг ‘&&’ гэсэн тэмдгээр илэрхийлдэг байна. Энд ‘x==7’, ‘y==11’ үйлдлүүдэд тус бүрд нь хаалт хийх шаардлагагүй. Учир нь харьцуулах оператор нь логик операторoос өндөр эрэмбэтэй юм. C++-д 3 логик оператор байна.

&&	Логик AND
||	Логик OR
!	Логик NOT

OR үйлдэл

Бидний тоглоомд хэтэрхий зүүн буюу баруун тал руу явбал “луу” дэр очдог гэж саная. Тийм учраас тоглогчийг луунд ойртвол анхааруулах шаардлагатай болно. Advenor.cpp гэсэн дараагийн жишээг үзье.

```
//advenor.cpp
#include <iostream.h>
#include <process.h>
#include <conio.h>

void main()
{
char dir='a';
int x=10, y=10;
while (dir!='\r')
        {
        cout << “Байрлал нь ” << x <<”,” << y;
        if (x<5 || x>15)
        cout << “Энэ орчинд луу байгаа шүү!”;
        cout << “\n Товч дарна уу(n, s, e, w):”;
dir=getche();
switch(dir)
        {
```

```
            case ‘n’: y--; break;
            case ‘s’: y++; break;
            case ‘e’: x++; break;
            case ‘w’: x--; break;
            }
}
}
```

If оператор нь x нь 15-аас бага юм уу, эсвэл x нь 15-аас их болсон тохиолдолд үнэн утга авна. Харин хоёр нөхцөл аль ч биелэхгүй тохиолдолд худал утгатай байна.

NOT үйлдэл

Логик not үйлдэл нь ганц параметртэй үйлдэл юм. Ийм үйлдлийг унар үйлдэл гэнэ (Бидний өмнө үзсэн үйлдлүүд нь ихэвчлэн 2 параметр авдаг бинар үйлдэл, нөхцөлт үйлдэл харин 3 параметр авдаг тернар үйлдэл юм). Энэ үйлдэл нь ганц параметрийнхээ утгыг эсрэгээр нь болгодог. Үнэн утгын өмнө ‘!’ тэмдэг тавибал худал, худал утгын өмнө тавибал үнэн болно. Жишээлбэл: x нь 7 гэсэн утгатай байвал (x==7) гэсэн нөхцөл үнэн, харин !(x==7) нөхцөл худал утгатай болно.

Бүхэл тоо ба Булийн утга

Одоо бидэнд бүхэл тоог Булийн утга болгон ашиглаж болох уу гэсэн асуулт тавигдана. Хэрэв ингэж ашиглаж болохгүй бол бүхэл тоог Булийн утга болгохын тулд бас нэгэн үйлдэл хийгдэх шаардлагатай болно. Гэвч аз таарахад ингэх шаардлага огт байхгүй. C++ нь бүхэл тоог 0 утгатай

байвал худал утга, 0-ээс ялгаатай байвал үнэн утга гэж ойлгодог байна.

Үүнийг тоглоомдоо ашиглаж тоглоомынхоо 7-д хуваагдах дугаар бүхий мөр, багана бүр дээр мөөг тавьцгаая. Мөн үлдэгдлийг нь авдаг өмнөх аргаар 7-д хуваагдах дугааруудыг олно.

```
//advenor.cpp
#include <iostream.h>
#include <process.h>
#include <conio.h>

void main()
{
char dir='a';
int x=10, y=10;
while (dir!='\r')
        {
        cout << "Байрлал " << x <<"," << y;
        if (!(x%7) && !(y%7))
                cout << "Энд мөөг байна!\n";
        cout << "\nТовч дарна уу(n, s, e, w):";
        dir=getche();
        switch(dir)
                {
                case 'n': y--; break;
                case 's': y++; break;
                case 'e': x++; break;
                case 'w': x--; break;
                }
        }
}
```

Энд бичигдсэн (! (x%7) && ! (y%7)) гэсэн нөхцөл нь (x%7 ==0 && y%7 ==0) гэсэнтэй үр дүнгийн хувьд яг адилхан болно. Харин x%7, y%7 гэсэн үйлдлүүдэд хаалт хийсэн байгааг анхаарах хэрэгтэй. Учир нь унар үйлдэл нь хамгийн өндөр эрэмбэтэй байдаг байна.

Үйлдлийн эрэмбэ

Одоо нэгэнт бүх үйлдлүүдийг үзэж дуусгасан хойно тэдгээрийн эрэмбийг тогтооцгооё. Хүснэгтэд дээр бичигдсэн үйлдлүүд нь доор бичигдсэнээсээ өндөр эрэмбэтэй (түрүүлж биелэнэ) байна. Нэг түвшинд бичигдсэн үйлдүүд ижил эрэмбэтэй (урд бичигдсэн нь түрүүлж биелэнэ) байна.

Эрэмбэ	Үйлдэл
Унар	
I	!, ++, --, -
Арифметик	
II	*, /, %
III	+, -
Харьцуулах	
IV	<, >, <=, >=
V	==, !=
Логик	
VI	&&
VII	||
Нөхцөлт	
VIII	? :
Утга оноох	
IX	=, +=, -=, *=, /=, %=

Бусад удирдах коммандууд

Одоо C++-д байдаг бусад удирдах коммандуудын тухай үзэцгээе. Бид switch дотор хэрэглэгдсэн break операторыг үзэж байсан. Энэ оператор нь өөр газар ч ашиглагдаж болох тухай өгүүлэх болно. Мөн түүнчлэн conitinue, goto гэсэн 2 коммандыг шинээр үзэх болно.

BREAK оператор

Бидний өмнө үзсэнээр энэ оператор нь сонголтын оператороос гардаг билээ. Энэ операторын өөр нэгэн үүрэг нь ажиллаж байгаа давталтыг шууд дуусгах зорилгоор мөн ашиглагдана. Энэ тухай Showprim гэх дараагийн жишээнд харуулжээ.

```
//showprim.cpp
#include <iostream.h>
#include <conio.h>

void main()
{
const unsigned char WHITE=219;
const unsigned char GRAY=176;
unsigned char ch;

for (int count=0; count<80*25-1; count++)
        {
        ch=WHITE;
        for (int j=2; j<count; j++)
                if (count%j==0)
                {
                ch=GRAY; break;
                }
        cout << ch;
        }
getche();
}
```

Дэлгэцийн бүх координатыг 0-ээс 1999 хүртэл дугаарлаж, тэр дугаарыг нь анхны тоо эсэхийг шалгаж байна. Хэрэв анхны тоо байвал уг координат дээр цагаан өнгө, биш саарал өнгө хэвлэгдэнэ. Дотор талын давталт нь хэрэв уг тоо анхных биш гэдгийг

мэдвэл ch-ийн утгыг GRAY болгож, давталтыг шууд дуусгаж байна. Бидэнд программаас биш, зөвхөн дотор талын давталтаас л гарахад хангалттай. Break оператор нь зөвхөн хамгийн дотор талын давталтаас л гаргана.

Continue оператор

Өмнөх break оператор нь удирдлагыг давталт болон сонголтын гадна талд авчирдаг. Гэтэл зарим тохиолдолд давталтыг дуусгах биш, харин удирдлагыг давталтын эхлэлд авчирах шаардлага гарч болно. Ө.х. давталтын зөвхөн энэ удаагийн алхмыг л алгасуулна гэсэн үг. Ийм үед continue оператор ашиглах хэрэгтэй.

Бидний өмнө үзсэн divdo гэсэн жишээ бий. Энэ жишээ өгсөн тоонуудыг хувааж бүхэл болон үлдэгдлийг олдог билээ. Гэхдээ энд нэг алдаа бий. Хэрэв хэрэглэгч хуваагчийг 0 гэж өгвөл тоо 0-д хуваагдаж, программ ‘Division by zero’ гэсэн алдаатайгаар тасрахад хүрнэ. Харин энэ алдааг засаж, divdo2 гэсэн жишээ үзье.

```
//divdo2.cpp
#include <iostream.h>

void main()
{
long dividend, divisor;
char ch;
do
        {
        cout << “Хуваагдагч:”; cin >> dividend;
        cout << “Хуваагч:”; cin >> divisor;
        if (divisor==0)
                {
```

```
                cout << “0-д хуваагдаж болохгүй\n ”;
                conitnue;
                }
        cout << “Бүхэл хэсэг нь ” << dividend/divisor;
        cout << “,үлдэгдэл нь ” << dividend%divisor;
        cout << “\nYргэлжлүүлэх үү?(Y/N):”;
        cin >> ch;
        }
while (ch!=’n’);
}
```

Хэрэв хэрэглэгч 0 утга оруулвал программ “ Illegal divisor ” гэсэн алдааны мэдэгдэл өгч, удирдлага дахин давталтын эхэнд очих юм.

GOTO оператор

Goto оператор удирдлагыг заасан газарт шууд аваачдаг. Энэ операторын тухай өгүүлэхийн өмнө goto операторыг ашиглана гэдэг тийм ч зөв сонголт биш гэдгийг анхааруулах гэсэн юм. Goto оператор ашигласан программ нь унших болон зүгшрүүлэлт хийхэд төвөгтэй болдог. Ер нь goto ашиглалгүйгээр ямар ч үйлдлийг хийх боломжтой.

Гэхдээ энэ операторын тухай товчхон танилцуулъя. Юуны өмнө программын очвол зохих газарт нэг LABEL тавих хэрэгтэй. Label-ийн ард ‘:’ тэмдэг тавьдагийг мартаж болохгүй. Goto операторын ард очих label-ийн нэрийг бичиж өгнө. Энэ тухай доорх жишээнээс үзэж болно.

```
. . . . .
Goto SystemCrash;
. . . . .
SystemCrash :
//Программ эндээс үргэлжлэн ажиллах болно.
```

Бүлгийн дүгнэлт

Харьцуулах үйлдлүүд хоёр утгуудыг хоорондоо жишиж тэнцүү, эсвэл аль нэг нь их эсэхийг тогтоодог. Үр дүн нь Булийн утга байна. Худал утга нь 0-тэй тэнцүү, үнэн утга нь 0-ээс ялгаатай байдаг.

С++-д 3 төрлийн давталт байна. Давталт хэдэн удаа хийгдэхийг мэдэж байгаа тохиолдолд та for давталтыг ашиглаж болно. Харин давталтын тоо тодорхойгүй үед while болон do давталтуудыг ашиглах нь зүйтэй.

Давталтын бие нь ганц комманд ч байж болно, эсвэл гоё хаалтаар хашигдсан үйлдлүүд буюу блок байж болно. Блок дотор тодорхойлогдсон хувьсагч зөвхөн блок дотроо л хүчинтэй байна.

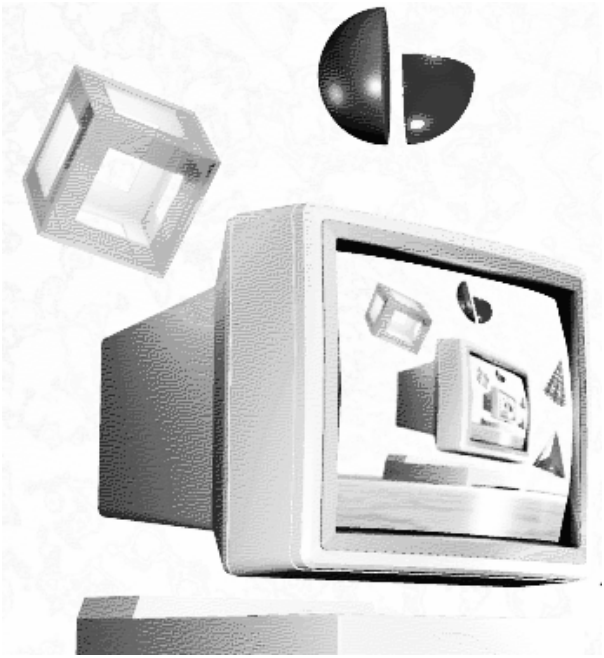
Нөхцөл шалгах 4 төрлийн үйлдэл байна. If нь шалгасан нөхцөл үнэн байвал ямар нэгэн үйлдэл хийхэд ашиглагдана. Харин if…else оператор нь нөхцөл үнэн байвал нэг үйлдэл, худал байвал өөр үйлдэл хийнэ. Харин switch оператор нь нэг хувьсагчийн утгаас олон сонголт хийхэд зориулагдсан юм. Харин нөхцөлт үйлдэл нь нөхцлийн үр дүнгээс хамаарч, 2 утга буцаадаг байна.

Логик and болон or үйлдлүүд хоёр Булийн утгуудыг нэгтгэж нэг Булийн утга үүсгэхэд, харин not оператор нь Булийн утгыг эсрэгээр нь болгоход тус тус хэрэглэгдэнэ.

break оператор удирдлагыг давталт болон сонголт дотроос гаргахад, харин continue оператор давталтын нэг алхмыг алгасуулж, удирдлагыг давталтын эхэнд авчирахад ашиглагдана. Харин goto оператор удирдлагыг шууд заасан байрлалд аваачдаг.

Үйлдлүүд дараах дарааллаар биелэгддэг. Үүнд : унар, арифметик, харьцуулах, логик, нөхцөлт, утга оноох.

3-р бүлэг



-Нийлмэл төрөл
-Бүтэц төрөл
-Тоочсон төрөл

3-р бүлэг

Бид энгийн бүтэцтэй float, int, char гэх мэтийн төрлүүдийг үзсэн. Ийм төрлийн хувьсагч нь зөвхөн ганц л мэдээллийг агуулна. Нэг өндөр, эсвэл нэг хэмжээ, нэг жин гэх мэт. Гэтэл олон өндөг нэг сагсанд байрладаг, олон ажилчид нэг хэсэгт харьяалагддаг шиг олон энгийн мэдээллүүд нэг бүхэл юманд хамаарагдах шаардлага гарна. Ийм шаардлагын үүднээс C++-д бүтэц гэсэн төрөл тодорхойлогддог.

Бүтцийн тухай

Бүтэц гэдэг нь олон хувьсагчуудын цуглуулга юм. Бүтцэд байгаа хувьсагчууд өөр өөр төрлийнх байж болно. Нэг нь бүхэл тоо байхад нөгөө нь тэмдэгт мөр байх жишээтэй. Бүтэц дотор байгаа эдгээр хувьсагчуудыг бүтцийн талбар гэж нэрлэнэ.

C++-ийн хувьд бүтэц бол обьект болон ангийн тухай ойлголт авах үндсэн хэсэг юм. Бүтцийг судалснаар OOP-г судлах үндэс суурь нь бий болох ёстой. Бүтэц нь Basic болон Pascal –ийн бичлэг (record) төрөлтэй адилхан гэж үзэж болно.

Энгийн бүтэц

Хоёр бүхэл болон нэг бутархай тоон талбараас тогтох бүтцийг эхэлж үзье. Үүнийгээ ямар нэгэн үйлдвэрийн барааны тухай мэдээлэл гэж үзвэл, эхний тоо нь уг барааны дугаар, дараагийн талбар нь уг барааны загварын дугаар, сүүлийн бутархай тоон

талбар нь түүний үнэ гэж үзье. Доорх жишээ нь ийм нэгэн барааны мэдээллийг үүсгэж, түүнийгээ дэлгэцэнд хэвлэдэг.

```
//parts.cpp
#include <iostream.h>

struct part
        {
        int partnum;
        int modelnum;
        float cost;
        };

void main()
{
part part1;

part1.partnum=6244;
part1.modelnum=373;
part1.cost=217.55;

cout << "\nДугаар " << part1.partnum ;
cout << ", Загвар " << part1.modelnum ;
cout << ", Үнэ " << part1.cost ;
}
```

Программын үр дүн:

```
Дугаар 6244, загвар 373, үнэ 217.55
```

Программ 3 үндсэн алхмуудыг дамжиж байна. 1-рт, бүтцийг тодорхойлох, 2-рт, бүтэц төрлийн хувьсагч тодорхойлох, эцэст нь, бүтцийн талбаруудтай ажиллах.

Бүтэц тодорхойлох

```
struct part
        {
        int partnum;
        int modelnum;
        float cost;
        };
```

Энэ бол бүтцийг тодорхойлж байгаа хэрэг юм. Struct гэдэг нь бүтэц эхэлж байгааг заасан нөөц үг. Энэ үгийн ард байгаа нь бүтцийн нэр юм. Хаалтан дотор бүтцийн 3 талбарыг энгийн хувьсагч тодорхойлохын адилаар тодорхойлж өгсөн байна. Хаалтны төгсгөлд цэг таслал (;) тавьсан нь энэхүү тодорхойлолт дуусч байгааг илэрхийлнэ. Үүгээрээ бүтэц тодорхойлох нь энгийн коммандын блокоос ялгаатай юм (коммандын блокын төгсгөлд цэг таслал тавьдаггүй.).

Бүтэц төрлийн хувьсагч тодорхойлох

Үндсэн бие дэх анхны мөр нь:

```
part part1;
```

Энэ мөр нь part төрөлтэй part1 гэсэн хувьсагч тодорхойлж байна. Ингэснээрээ санах ойд энэ хувьсагчид зориулсан муж авна. Харин ямар хэмжээтэй муж авах вэ? Бүх 3 хувьсагчид зориулсан санах ой авах нь мэдээж. Бидний жишээнд бол бүхэл хувьсагч бүрт 2 байт, харин бутархай хувьсагчид 4 байт, нийт 8 байтыг авах болно.

Өмнөх үеийн C хэл дээр хувьсагчийг тодорхойлохдоо struct гэсэн нөөц үгийг ашигладаг байсан бол C++-д ингэх шаардлагагүй юм. Ө.х. struct part part1 гэж тодорхойлдог байсан байна.

Бүтцийн талбаруудад хандах

Бүтэц болон түүний хувьсагчийг тодорхойлсны дараа тэдгээрийн талбаруудтай ажиллана. Ингэж ажиллахдаа цэг операторыг ашигладаг.

```
part1.partnum=6244;
```

Талбарт хандах бичлэг нь бүтэц төрлийн хувьсагчийн нэр, цэг оператор (.), талбарын нэр гэсэн 3 хэсэгтэй. Энэ нь part1-ийн partnum талбар гэсэн үг юм. Цэг операторын өмнө талд бүтэц төрлийн нэр биш, харин ийм төрлийн хувьсагчийн нэр байхыг андуурч болохгүй. Ингэж хандсанаар бүтцийн талбар бүрт өөр өөр хувьсагчууд мэт энгийн үйлдлүүд гүйцэтгэж болно. Өмнөх жишээнд утга олгох оператор ашигласан байна.

Бүтцийн тухай нэмж өгүүлэхэд

Өмнө үзсэн жишээнд бид бүтцийг тодорхойлох болон хувьсагч тодорхойлох үйлдлүүдийг тус тусд нь гүйцэтгэсэн билээ. Харин энэ үйлдлүүдийг доорх маягаар нэгтгэж болдог юм.

```
//partscom.cpp
#include <iostream.h>

struct
        {
        int partnum;
        int modelnum;
        float cost;
```

```
        } part1;

void main()
{
part1.partnum=6244;
part1.modelnum=373;
part1.cost=217.55;

cout << "\nДугаар " << part1.partnum ;
cout << ", Загвар " << part1.modelnum ;
cout << ", Үнэ " << part1.cost ;
}
```

Өөр хувьсагч тодорхойлохгүй бол бичлэгийг ингэж хялбарчилах нь бичлэг болон санах ойн нэлээд хэмнэлттэй юм.

Энгийн хувьсагчдын нэгэн адилаар бүтэц төрлийн хувьсагч тодорхойлохдоо түүний талбаруудын анхны утгыг шууд оноон өгч болно.

```
//partinit.cpp
#include <iostream.h>

struct part
        {
        int partnum;
        int modelnum;
        float cost;
        };

void main()
{
part part1={ 6244, 373, 217.55 };
part part2;

cout << "\nДугаар " << part1.partnum ;
cout << ", Загвар " << part1.modelnum ;
cout << ", Үнэ " << part1.cost ;

part2=part1;
```

```
cout << “\nДугаар ” << part2.partnum ;
cout << “, Загвар ” << part2.modelnum ;
cout << “, Үнэ ” << part2.cost ;
}
```

Энэ программ бүтэц төрлийн 2 хувьсагч тодорхойлж, эхний хувьсагчийн анхны утгуудыг оноож өгч байна. Доорх үр дүн гарах болно.

```
Дугаар 6244, загвар 373, үнэ 217.55
Дугаар 6244, загвар 373, үнэ 217.55
```

Ингэж анхны утга онооход хаалтанд хашиж, таслалаар зааглан бичсэн байна. Эхний утга нь эхний талбарт, дараагийн утга нь 2 дахь талбарт гэх мэтчилэн оноогдох болно. Сүүлийн хэсэгт бүтэц төрлийн хувьсагчуудыг утга оноож өгч байна. Ингэж утга онооход талбаруудын утгууд харгалзан оноогддог.

Бүтцийн жишээ

Англи, Америкт уртын хэмжээг feet болон inch – ээр хэмждэгийг та мэдэх биз. Inch-ийг (‘) тэмдгээр, feet-ийг (“) тэмдгээр тэмдэглэнэ. 12’ – 8” гэсэн байвал 12 inch 8 feet гэж ойлгох хэрэгтэй. Дунд нь байгаа тэмдэг бол хасах тэмдэг биш, харин inch болон feet-ийг хооронд нь зааглаж байгаа тэмдэг юм.

Ийм нэгжээр уртыг хэмжих шаардлагатай программ бичвэл, уртын хэмжээг хадгалахын тулд inch болон feet – ийг хадгалах бүтэц тодорхойлох шаардлага гарна.

```
//englstrc.cpp
#include <iostream.h>
```

```
struct Distance
        {
        int feet;
        float inches;
        };

void main()
{
Distance d1, d3;
Distance d2=(11, 6.25);

cout << "\nФут: "; cin >> d1.feet;
cout << "Инч: "; cin >> d1.inches;

d3.inches=d1.inches+d2.inches;
d3.feet=0;
if (d3.inches >=12.0)
        {
        d3.inches-=12.0; d3.feet++;
        }
d3.feet+=d1.feet+d2.feet;

cout << d1.feet << "\'-" << d1.inches << "\"+";
cout << d2.feet << "\'-" << d2.inches << "\"=";
cout << d3.feet << "\'-" << d3.inches << "\"\n";
}
```

Distance бүтэц нь feet ба inches гэсэн 2 талбартай. Feet нь бүхэл тоон, inches нь бутархай тоон төрөлтэй байна. d1 ба d3 хувьсагчуудыг шууд, харин d2 хувьсагчийг анхны утгыг нь оноож тодорхойлсон байна. Хэрэглэгчээс d1-ийн талбарын утгуудыг авч, энэ утгуудыг d2 дээр нэмж, хариуг d3-т бичиж байна. Ингээд гарсан үр дүн нь:

```
Фут: 10
Инч: 6.75
10'-6.75"+11'-6.25"=22'-1"
```

Яагаад бид шууд d3 = d1 + d2 гэсэн хэлбэрээр нэмж болохгүй байгааг тайлбарлая. Int, float мэтийн стандарт хувьсагчуудын нэмэх үйлдлүүдийг стандарт header файл дотор тодорхойлсон байдаг юм. Харин бидний бүтцэд ийм нэмэх үйлдэл тодорхойлогдоогүй байгаа билээ.

Бүтэц доторх бүтэц

Та бүтэц дотор бүтэц төрлийн хувьсагч тодорхойлж болно. Одоогийн жишээнд өрөө гэсэн бүтэц тодорхойлох бөгөөд энэ бүтэц нь өргөн, өндөр гэсэн 2 талбартай, талбар тус бүр distance төрлийнх байх юм.

```
//englarea.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

struct Room
        {
        Distance length;
        Distance width;
        };

void main()
{
Room dining;

Dining.length.feet=13;
Dining.length.inches=6.5;
Dining.width.feet=10;
```

```
Dining.width.inches=0.0;

Float l=Dining.length.feet+Dining.length.inches/12;
Float w=Dining.width.feet+Dining.width.inches/12;

cout << “\nГал тогооны өрөөний хэмжээ ” << l*w
    << “ квадрат фут”;
}
```

Бүтэц дотор агуулагдсан бүтцийн талбарт хандахын тулд цэг операторыг давхарлан ашиглах шаардлагатай болно.

```
Dining.length.feet=13;
```

Энд dining гэсэн бүтцийн length талбарын feet талбарт 13 гэсэн утга оноож байгаа юм. Утгуудыг оноосны дараагаар уг өрөөний талбайг тооцон гаргаж, дэлгэцэнд хэвлэж байна. Үр дүн нь:

```
Гал тогооны өрөөний хэмжээ 135.416672 квадрат фут
```

Зарчмын хувьд хэдэн ч давхар бүтэц байж болно. Жишээ нь:

```
Apartment1.laundry.washing_machine.width.feet
```

Бүтэц доторх бүтцэд анхны утга оноох

Өөртөө бүтэц багтаасан бүтцийн анхны утгуудыг тодорхойлох үед шууд өгч болно. Энэ үед доорх маягийн бичлэг ашигладаг юм.

```
Room dining={{13, 6.5}, {10, 0.0}};
```

Room дотор байрлаж байгаа distance бүтэц бүрийг хаалтаар зааглан бичнэ. Хаалт бүрийн доторх утгууд хоорондоо таслалаар тусгаарлагдах болно. Эхний distance төрлийн хувьсагч {13, 6.5} гэсэн утга, дараагийн distance төрлийн хувьсагч {10, 0.0} гэсэн утгуудыг тус тус авах юм.

Хөзрийн тоглоомны тухай

Өөр нэгэн жишээ үзэцгээе. Энэ жишээ нь хөзрийн нэгэн тоглоомыг та бүхэнд танилцуулах болно. Cardsharp хэмээх энэ тоглоом нь дэлгэцэнд 3 хөзөр харуулах ба дараа нь тэдгээрийг буруу харуулаад, хэд хэдэн удаа холино. Хэрвээ та сонгосон хөзрөө хаана байгааг тааж чадвал хожих болно. Программд доорх бүтэц ашиглагдана.

```
struct card
{
        int number;
        int suit;
}
```

Эхний талбар нь хөзрийн дугаар бөгөөд 2-оос 14 хүртэл утга авна. Энд 11-ээс 14 хүртэл харгалзан боол, хатан, ноён, тамга байх юм. Дараагийн талбар нь 0-ээс 3 хүртэл утга авна. Эдгээр нь мөн харгалзан цэцэг, дөрвөлжин, бунд, гэл гэсэн үг.

```
const int clubs=0;                  //цэцэг
const int diamonds=1;   //дөрвөлжин
const int hearts=2;         //бунд
const int spades=3;         //гэл

const int jack=11;          //боол
const int queen=12;         //хатан
```

```
const int king=13;          //ноён
const int ace=14;                    //тамга

struct card
          {
          int number;
          int suit;
          }

void main()
{
card temp, chosen, prize;
int position;

card card1={7, clubs};
cout << “Эхний хөзөр нь цэцгийн 7 байна\n”;

card card2={jack, hearts};
cout << “2 дахь хөзөр бундангийн боол\n”;

card card3={ace, spades};
cout << “3 дахь хөзөр гэлийн тамга\n”;

prize=card3;

cout << “1-р хөзрийг 3-р хөзөртэй солилоо\n”;
temp=card3; card3=card1; card1=temp;

cout << “2-р хөзрийг 3-р хөзөртэй солилоо\n”;
temp=card3; card3=card2; card2=temp;

cout << “1-р хөзрийг 2-р хөзөртэй солилоо\n”;
temp=card2; card2=card1; card1=temp;

cout << “Гэлийн тамга нь хаана“
    << “орсон бэ? (1,2,3) \n”;
cin >> position;

switch (position)
          {
          case 1: chosen=card1; break;
          case 2: chosen=card2; break;
          case 3: chosen=card3; break;
```

```
        }
if (chosen.number==prize.number &&
   chosen.suit==prize.suit)
                cout << “Маш зөв! Та яллаа!\n”;
        else
                cout << “Уучлаарай. Та алдлаа.\n”;
}
```

Программ ажилласны дараа дэлгэц дээр дараах мэдээллүүд хэвлэгдсэн байх болно.

```
Эхний хөзөр нь цэцгийн 7 байна
2 дахь хөзөр бундангийн боол
3 дахь хөзөр гэлийн тамга
1-р хөзрийг 3-р хөзөртэй солилоо
2-р хөзрийг 3-р хөзөртэй солилоо
1-р хөзрийг 2-р хөзөртэй солилоо
Гэлийн тамга нь хаана орсон бэ? (1,2,3) 3
Уучлаарай. Та алдлаа.
```

Программын эхэнд хэдэн тогтмол хувьсагч тодорхойлж байна. Дараа нь анхны утгыг олгосон болон олгоогүй хэд хэдэн “хөзөр“ тодорхойлж байна. Тэрхүү анхны утгуудыг энэ жишээнд автоматаар оноосон байна. Ерөнхий тохиолдолд эдгээр утгуудыг санмсаргүйгээр оноох нь зүйтэй. Ингээд аль нэг хөзрийг санаж аваад, хөзрүүдийг хольж байгаа юм. Холихдоо 1-рийг 3-ртай, 3-рыг 2-ртай, 2-рыг 1-ртэй сольж байна. Эцэст нь программ хэрэглэгчээс сонгосон хөзөр хаана байгааг асууж, хэрэглэчийн сонголтыг авах бөгөөд тэр сонголтыг шалгаад, тохирох хариултыг өгч байна. Бүтцүүдийг шууд нэмж болдоггүйн адил тэдгээрийг шууд харьцуулж болохгүй учраас талбар бүрээр нь жишиж байна.

Тоочсон төрөл

Бидний өмнө үзсэн бүтэц төрөл нь хэрэглэгчийн тодорхойлдог төрөл билээ. Одоо хэрэглэгчийн тодорхойлдог өөр нэг төрөл **тоочсон төрлийг** үзэх гэж байна. Энэ төрөл обьект хандалтат программчлалд нэг их чухал төрөл биш боловч хэрэглэгчийн тодорхойлдог төрлийн тухай ойлголт авахад их нөлөөтэй. Pascal –д ч мөн ийм төрөл байдаг.

Долоо хоногийн гаригууд

Хэрэв та уг төрлийн бүх утгуудыг мэддэг байгаад тэр нь тоочин бичиж болохуйц цөөн бол тоочсон төрлийг үүсгэж болно. Доорх жишээг анхааралтай үзэцгээ.

```
//dayenum.cpp
#include <iostream.h>

enum days_of_week {Sun,Mon,Tue,Wed,Thu,Fri,Sat};

void main()
{
days_of_week day1, day2;

day1=Mon;
day2=Thu;

int diff=day2 – day1;
cout << “Өдрийн зөрөө=” << diff << endl;

if (day1 < day2)
        cout << “1-р өдөр 2-р өдрөөс өмнө байна.\n”;
}
```

Тоочсон төрлийг enum гэсэн нөөц үгээр эхлүүлэн тодорхойлно. Ард нь төрлийн нэрийг, хойноос нь уг төрлийн авч болох утгуудыг цувуулан бичиж байна. Эдгээр утгуудыг гишүүн гэж нэрлэнэ. Бидний тодорхойлсон төрөл 7 гишүүнтэй байна. Ийм төрөл нь тодорхой тооны утгууд авдаг байхад жишээ нь бүхэл тоон төрлийн утгуудыг тооны мужаар л зааж өгөхөөс өөр арга байхгүй.

Нэгэнт төрлийг тодорхойлсны дараагаар ийм төрлийн хувьсагчийг тодорхойлж өгнө. Дээрх жишээнд day1, day2 гэсэн хувьсагчууд тодорхойлогдсон байна. Тоочсон төрлийн хувьсагч нь тэрхүү тоочсон утгуудын аль нэгийг авна. Day1 хувьсагч Mon гэсэн утга, day2 нь Thu утга авчээ. Тоочсон төрлийн утга дээр дурын арифметик үйлдэл хийж болно. Дээрх жишээнд хоёр утгыг хооронд нь хассан байна. Программын үр дүн нь:

```
Өдрийн зөрөө=3
1-р өдөр 2-р өдрөөс өмнө байна.
```

Тоочсон төрөл нь компиляторт энгийн бүхэл тоонууд гэж ойлгогдоно. Ийм учраас арифметик болон харьцуулах үйлдлүүд хийж болдог байна. Уг төрөл дахь эхний гишүүнийг 0, дараагийнхийг 1 гэх мэтчилэн тоогоор төлөөлүүлдэг. Өмнөх жишээнд Sun нь 0, Sat нь 6 гэсэн утгатай ажээ. Өөрөөр хэлбэл та day1 = 5 гэсэн үйлдэл хийж болно. Гэхдээ ингэж утга оноохдоо тун болгоомжтой байх нь зүйтэй.

Булийн төрөл үүсгэх

Өмнөх бүлэгт дурдсанаар C++-д үнэн ба худал утга авдаг Булийн төрөл байдаггүй билээ. Нэгэнт

тоочсон төрлийг үзсэн учраас өөрсдөө Булийн төрөл үүсгэж болно.

Та бүхэн өмнөх бүлгийн chcount гэсэн жишээг санаж байгаа биз. Тэр программын гол дутагдал нь олон space (хоосон зай) цувуулан бичсэн тохиолдолд тэр бүрийг үг гэж тоолдогт байгаа юм. Харин одоо энэ алдааг залруулъя.

```
//wdcount.cpp
#include <iostream.h>
#include <conio.h>

enum boolean {false,true};

void main()
{
boolean isWord=false;

char ch='a';
int wordcount=0;

do
        {
        ch=getche();
        if (ch==' ' || ch=='\r')
                {
                if (isWord)
                        {
                        wordcount++;
                        isWord=false;
                        }
                }
        else
                if (!isWord)
                        isWord=true;
        }
while (ch!='\r');
cout << "\n—Үгийн тоо " << wordcount << "--\n";
}
```

Хэрэглэгч товч дарах бүрд дарсан товчийг шалгаж, хоосон зай иртэл үсэгнүүдийг алгасах бөгөөд харин дараа нь үсэг иртэл хоосон зайнуудыг алгасах юм. Ингэсэн тохиолдолд дараалан орж ирсэн хоосон зайнуудыг үг гэж тоолохгүй. Программын эхэнд isWord хувьсагч худал утгатай байх бөгөөд анхны үсэг дарагдахад л энэ хувьсагч үнэн утгаа авна. Үүнээс хойш хоосон зай дарагдахад анхны үгийг тоолох бөгөөд isWord хувьсагч дахиж худал утгатай болно. Хоёр дахь if операторыг хаалтанд хийсэн нь түүний дараа байгаа else операторыг энэ нөхцөл шалгалтын биш, өмнөх шалгалтын else гэдгийг тодруулж өгч байгаа юм.

Дахиад л хөзрийн тухай

Өмнөх хөзрийн тоглоомдоо тодорхойлж байсан олон тогтмол хувьсагчдыг тоочсон төрөл болгон тодорхойлъё.

```
//cardenum.cpp
#include <iostream.h>

const int jack=11;          //боол
const int queen=12;         //хатан
const int king=13;          //ноён
const int ace=14;                       //тамга

enum Suit {clubs,diamonds,hearts,spades};

struct card
        {
        int number;
        Suit suit;
        }

void main()
```

```
{
card temp, chosen, prize;
int position;

card card1={7,clubs};
cout << “1-р хөзөр нь цэцгийн 7 байна.\n”;

card card2={jack,hearts};
cout << “2-р хөзөр нь бундангийн боол байна.\n”;

card card3={ace,spades};
cout << “3-р хөзөр нь гэлийн тамга байна.\n”;

prize=card3;

cout << “1-р хөзрийг 3-р хөзөртэй сольж байна.\n”;
temp=card3; card3=card1; card1=temp;

cout << “2-р хөзрийг 3-р хөзөртэй сольж байна.\n”;
temp=card3; card3=card2; card2=temp;

cout << “1-р хөзрийг 2-р хөзөртэй сольж байна.\n”;
temp=card2; card2=card1; card1=temp;

cout << “Гэлийн тамга хаана“
    << “орсон бэ. (1,2,3) \n”;
cin position;

switch (position)
        {
        case 1: chosen=card1; break;
        case 2: chosen=card2; break;
        case 3: chosen=card3; break;
        }
if (chosen.number==prize.number &&
   chosen.suit==prize.suit)
                cout << “Маш зөв! Та яллаа!\n”;
        else
                cout << “Уучлаарай, та алдлаа.\n”;
}
```

Тоочсон төрлийн хамгийн эхний утга нь 0 гэсэн бүхэл тоо болдог тухай дээр дурдсан. Нэмж сонирхуулахад, эхний утгыг 0-ээс өөр тоо болгож болно.

```
Enum Suit {clubs=1,diamonds,hearts,spades};
```

Энэ тохиолдолд diamonds = 2, hearts = 3, spades = 4 гэсэн утгуудыг авах болно. Бас нэгэн зүйлийг нэмье. Тоочсон төрлийн хувьсагчийн утгыг хэвлэхэд утгын нэр нь биш, тоон утга нь хэвлэгддэг юм шүү. Жишээ нь:

```
day1.suit=hearts;
cout << day1.suit;
```

гэвэл hearts гэсэн үг биш, 3 гэсэн тоо хэвлэгдэх болно.

Тоочсон төрлийн хэд хэдэн жишээг доор үзүүллээ.

```
enum months {Jan,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};

enum lightswitch {off,on};

enum chess {pawn,knight,bishop,rook,queen,king};
```

Бүлгийн дүгнэлт

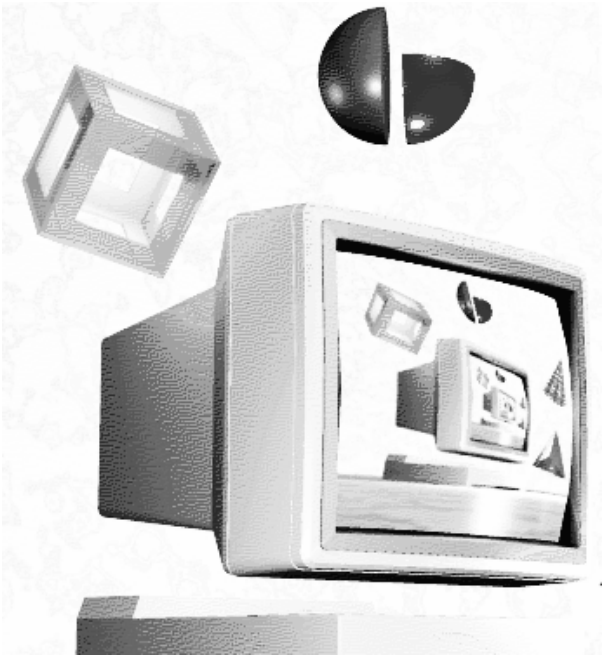
Бид энэ бүлэгт бүтэц ба тоочсон гэсэн 2 төрлийн тухай үзлээ.

Бүтэц гэдэг бол C++-ийн хувьд нэлээн чухал, бичиглэлийн хувьд ангитай тун төстэй төрөл юм. Бүтэц нь өөртөө хэд хэдэн энгийн төрлийн хувьсагчийг агуулсан хэрэглэгчийн тодорхойлдог

нийлмэл төрөл билээ. Зарим талаас нь авч үзвэл бүтэц нь нэг бүхэл төрөл шиг боловч, зарим тохиолдолд тэдгэрийн талбар тус бүртэй ажиллахаас өөр аргагүйд хүрдэг. Бүтцийн талбарт хандахын тулд цэг операторыг ашиглана.

Тоочсон төрөл бол тодорхой тооны, тогтмол утгаас тогтсон төрөл байх юм. Тоочсон төрлийн элементүүдийг компилятор бүхэл тоо болгон ойлгодог.

4-р бүлэг



-Функц, процедур
-Давхардсан функцүүд
-Дотоод функцүүд
-Стандарт утга
-Рекурсив функц
-Макро

4-р бүлэг

Хэд хэдэн үйлдлийг нэгэн хэсэг болгон бүлэглэж, нэр өгснийг функц гэж нэрлэдэг. Ийм бүлэг үйлдлийг программын өөр хэсгүүдээс дуудаж болно.

Программын бүтцийг цэгцтэй болгохын тулд функцийг үүсгэж ашигладаг. Өөр нэгэн зорилго бол программын хэмжээг багасгах явдал юм. Программд олон дахин ашиглагддаг үйлдлүүдийг функц болгон хурааснаар программын хэмжээ нэлээн хэмжээгээр багасах боломжтой.

Функц нь Basic-ийн дэд программ, Pascal-ийн функц болон процедуртай адил ойлголт болно.

Энгийн функцүүд

Функцийн жишээ болгосон доорх программ дэлгэцэнд хүснэгт үүсгэхдээ, хүснэгтийн хүрээг од (*) ашиглан зурах юм.

```
//table.cpp
#include <iostream.h>

void starline();

void main()
{
        starline();
        cout << “Төрөл          Муж” << endl;
        starline();
        cout    << “char       -128 to 127” << endl
        << “int         -32768 to 32767” << endl
        << “double    -2147483648 to 2147483647”
        << endl;
        starline ();
```

```
}

void starline()
        {
        for (int j=0; j<45; j++)
                cout << '*';
        cout << endl;
        }
```

Программын үр дүн нь:

```
*********************************************
Төрөл         Муж
*********************************************
char                    -128 to 127
int                     -32768 to 32767
double          -2147483648 to 2147483647
*********************************************
```

Бидний өмнө үзсэн программаас функц бүхий энэ программд ямар шинэ бичлэг орсныг тайлбарлая.

1-рт, Функцийг урьдчилан тодорхойлох

2-рт, Үндсэн программаас функцийг дуудаж ажиллуулах

3-рт, Функцийг биеийг тодорхойлох.

Функц бичиж, тодорхойлох

Ямар нэг хувьсагчийг урьдчилан тодорхойлолгүйгээр ашиглаж болохгүйн нэгэн адил функцийг ч мөн урьдчилан тодорхойлох шаардлагатай. Функцийг программын эхэнд тодорхойлж өгнө.

```
void starline();
```

Void гэсэн нөөц үг нь уг функц ямар нэгэн утга буцаахгүйг, хоосон хаалт нь уг функц ямар нэгэн аргумент авахгүйг илэрхийлж байгаа юм. Аргумент авахгүй гэдгийг илэрхийлэхийн тулд хаалт дотор мөн void гэж бичиж өгч болдог. Аргумент болон буцах утгын тухай дараа дэлгэрэнгүй тайлбарлах болно. Функцийг урьдчилан тодорхойлсон мөрийн төгсгөлд (;) тавьж өгнө. Функцийг урьдчилан тодорхойлохыг prototype (прототип) гэж нэрлэнэ.

Бидний тодорхойлсон функц үндсэн программаас 3 удаа дуудагджээ. Дуудах бүрд доорх бичлэг ашиглагдсан байна.

```
starline();
```

Эхэнд нь функцийн нэр, араас нь хоосон хаалтыг бичиж өгсөн байна. Коммандын төгсгөлд (;) тэмдэг тавьж өгнө. Ингэж дуудсанаар функцийг ажиллуулах бөгөөд ө.х. программын удирдлага функц доторх коммандуудад очих юм. Функцийн коммандууд биелэгдэж дуусаад, удирдлага функц дуудсан газар буцаж ирнэ.

Үндсэн программын дараа бид функцийн жинхэнэ биеийг тодорхойлж байна. Функцийн биед ажиллах коммандуудыг үндсэн программын нэгэн адилаар тодорхойлж өгнө.

```
void starline()
        {
        for (int j=0; j<45; j++)
                cout << '*';
        cout << endl;
        }
```

Функцийг урьдчилан тодорхойлохын адилаар бичиж өгөөд, функцийн биеийг гоё хаалтаар хашиж бичдэг. Харин ингэхдээ эхний мөрийн ард цэг

таслалгүй бичдэг нь тодорхойлолт дуусалгүй цааш үргэлжилж байгааг илэрхийлж байна. Буцах утга, аргумент, функцийн нэр зэрэг нь урьдчилан тодорхойлсноос зөрж болохгүй.

Харин функцийн биеийг тодорхойлох хэсэг үндсэн программаас өмнө байрлаж байгаа бол функцийг урьдчилан тодорхойлох шаардлагагүй. Программыг доорх байдлаар өөрчилж болно.

```
//table2.cpp
#include <iostream.h>

void starline()
        {
        for (int j=0; j<45; j++)
                cout << '*';
        cout << endl;
        }

void main()
{
        starline();
        cout << "Төрөл          Муж" << endl;
        starline();
cout    << "char       -128 to 127" << endl
        << "int        -32768 to 32767" << endl
        << "double     -2147483648 to 2147483647"
        << endl;
        starline ();
}
```

Энэ программ нь хэмжээний хувьд өмнөхөөс бага байна. Гэхдээ зарим тохиолдолд ингэж бичих боломжгүй ч байдгийг анхаарч авбал зүйтэй.

Функцэд аргумент өгөх

Аргумент гэдэг нь дуудсан программаас дуудагдаж буй функцэд дамжиж байгаа өгөгдлийг хэлдэг. Аргумент нь функцийг үндсэн программд байгаа хувьсагч дээр буюу тийм хувьсагчийн утгыг үндэслэсэн үйлдэл хийх боломжтой болгож өгнө.

Аргументэд тогтмол утга дамжуулах

Бидний хамгийн сүүлд үзсэн жишээнд байгаа starline () функцийг бага зэрэг боловсронгуй болгох нь зүйтэй байх. Тухайлбал дурын тэмдгийг дурын тоогоор хэвлэдэг болгож болох юм.

```
//tablearg.cpp
#include <iostream.h>

void repchar(char, int);

void main()
{
repchar('-', 43);
cout << "Төрөл            Муж" << endl;
repchar ("=", 23);
cout      << "char          -128 to 127" << endl
          << "int       -32768 to 32767" << endl
          << "double  -2147483648 to 2147483647"
          << endl;
          repchar ('-', 43);
}

void repchar(char ch, int n)
          {
          for (int j=0; j<n; j++)
                    cout << ch;
          cout << endl;
          }
```

Функцийг урьдчилан тодорхойлох хэсэгт байгаа бичлэгийг харцгаая. Функцийн нэрийн хойно байгаа хаалтанд бичигдсэн төрлүүд нь уг функцийн аргументэд очих хувьсагчдын төрлүүд юм. Энэ жишээнд байгаа repchar () функц нь тэмдэг болон бүхэл тоон төрлийн 2 аргумент авна гэсэн үг. Функцийг дуудахдаа аргументуудад нь харгалзах утгуудыг өгч байна.

```
repchar('-', 43);
```

Эхний аргумент нь тэмдэг төрлийн тогтмол, дараагийнх нь бүхэл тоо төрлийн тогтмол утгууд байна. Программ доорх үр дүнг өгнө.

```
-----------------------------------------
Төрөл          Муж
=========================================
char                    -128 to 127
int                     -32768 to 32767
double          -2147483648 to 2147483647
-----------------------------------------
```

Функцийг биеийг тодорхойлох үедээ фунцкийн аргументүүдийн төрлөөс гадна, нэрийг нь бас зааж өгөх шаардлагатай.

```
void repchar(char, int);
```

Дуудаж байгаа программ нь функцэд утгууд өгч, түүнийг нь функц ch ба n гэсэн хувьсагчуудын анхны утгууд болгон авч байгаа юм. Аргументэд тодорхойлогдсон ийм хувьсагчууд нь функц дотор энгийн тодорхойлогдсон хувьсагчуудтай яг ижил ашиглагдана. Өөрөөр хэлбэл ийм хувьсагчуудын утгыг уншиж, бичиж, өөрчилж болно.

Аргументэд хувьсагч дамжуулах

Бидний өмнөх жишээнд функцэд өгч байгаа аргументүүд нь тогтмол утгууд байсан билээ. Үүний нэгэн адилаар функцийн аргументэд хувьсагчийн утгыг дамжуулж болдог. Одоогийн үзэх жишээнд функцийн аргументэд өгөх утгуудыг хэрэглэгч тодорхойлж өгөх болно.

```
//vararg.cpp
#include <iostream.h>

void repchar(char, int);

void main()
{
char chin;
int nin;

cout << “Дурын тэмдэгт оруул : ”;
cin >> chin;
cout << “Уг тэмдэгтийн давтагдах тоог өгнө үү:”;
cin >> nin;
repchar(chin, nin);
}

void repchar(char ch, int n)
        {
        for (int j=0; j<n; j++)
                cout << ch;
        cout << endl;
        }
```

Программын ажиллаж байгаа нэгэн жишээ нь:

```
Дурын тэмдэгт оруул : +
Уг тэмдэгтийн давтагдах тоог өгнө үү: 20
++++++++++++++++++++
```

Аргументэд өгч байгаа хувьсагчийн төрөл нь функцийн аргументэд очих ёстой утгатай ижил төрөлтэй байх ёстой.

Аргументэд бүтэц дамжуулах

Энгийн хувьсагч төдийгүй бүхэл бүтцийг ч аргумент болгон дамжуулж болно. Өмнөх бүлэгт үзсэн distance бүтцийг аргумент болгон дамжуулъя.

```
//engldisp.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

void engldisp(Distance);

void main()
{
        Distance d1, d2;

        Cout << “\nФут : ”; cin >> d1.feet;
        Cout << “Инч : ”; cin >> d1.inches;

        Cout << “\nФут : ”; cin >> d2.feet;
        Cout << “Инч : ”; cin >> d2.inches;

        Cout << “\nd1=”;
        Engldisp(d1);

        Cout << “\nd2=”;
        Engldisp(d2);
}

void engldisp(Distance dd);
```

```
{
cout << dd.feet << "\'-" << dd.inches << "\"";
}
```

Энэ программ хэрэглэгчийн оруулж өгсөн өгөгдлүүдийг Distance бүтцэд авч, тэдгээрийгээ стандарт хэлбэрээр хэвлэн харуулах юм. Функцийн аргументэд бүтэц төрлийн хувьсагч өгч байна. Программ доорх хэлбэрийн үр дүн өгнө.

```
Фут : 6
Инч : 4

Фут : 5
Инч : 4.25

d1=6'-4"
d2=5'-4.25"
```

Бүтцийг аргументэд дамжуулахад энгийн төрлүүдийг дамжуулахаас ямар ч ялгаагүй бөгөөд функцийг урьдчилан тодорхойлох, функцийн биеийг тодорхойлох, функцийг дуудах зэрэг үйлдлүүд нь яг ижил бичигдэнэ. Энгийн хувьсагчдын нэгэн адил функц доторх dd гэсэн хувьсагчийн утгыг өөрчилж болох бөгөөд ингэж өөрчилөх нь үндсэн программ дахь d1, d2 хувьсагчуудад нөлөөлөхгүй.

Функцийн урьдчилан тодорхойлох хэсэгт аргументүүдийн нэрийг бичих, бичихгүй байх нь компиляторын хувьд огтын ялгаагүй хэрэг бөгөөд зөвхөн программ зохиогчид аргументүүдийн тухай ойлгомжтой болгохын тулд бичиж өгч болох юм. Ө.х. доорх бичлэгүүд ялгаагүй гэж ойлгож болно.

```
void display_point(int, int)
void display_point(int horiz, int vert)
```

Тогтмол утга болон хувьсагчийн аль нь ч аргументэд ирсэн үед, тэдгээр утгуудыг авах шинэ

хувьсагчдыг тодорхойлж (дээрх жишээнд char төрлийн ch гэсэн хувьсагч, int төрлийн n гэсэн хувьсагч), харгалзах утгуудыг оноодог. Ингээд уг хувьсагчийг функц дотор шууд тодорхойлогдсон хувьсагчийн нэгэн адил ашиглаж болдог билээ. Аргументүүдийг ингэж дамжуулахыг утгаар дамжуулах гэж нэрлэдэг юм. Харин аргументүүдийг хуурамч нэрээр буюу хаягаар дамжуулах тухай дараа авч үзэх юм.

Функцээс утга буцаах нь

Функц ажиллаж дууссаныхаа дараагаар дуудсан программдаа ямар нэгэн утга буцаадаг байж болно. Ийм утга нь голчлон функцийн шийдсэн асуудлын үр дүн байдаг юм. Дараагийн жишээнд фунтээр өгсөн жинг килограммд шилжүүлдэг функцийг үзье.

```
//convert.cpp
#include <iostream.h>

float lbstokg(float);

void main()
{
        float lbs, kgs;
        cout << "\nЖингээ оруулаарай (фунт) : ";
        cin >> lbs;
        kgs=lbstokg(lbs);
        cout << "Таны жин (килограмм) " << kgs;
}

float lbstokg(float pounds);
        {
        float kilograms=0.453592 * pounds;
        return kilograms;
        }
```

Программын үр дүн нь:

```
Жингээ оруулаарай (фунт) : 182
Таны жин (килограмм) 82.553741
```

Хэрвээ функц утга буцаадаг байвал уг функцийг тодорхойлохдоо буцах утгын төрлийг зааж өгөх ёстой. Энэхүү төрлийг функцийн нэрийн өмнө бичиж өгдөг. Өмнө үзсэн жишээнүүдэд утга буцаадаггүй функц учраас буцах утгын төрлийн оронд void гэсэн түлхүүр үгийг ашиглаж байсан билээ. Энэ жишээнд буй lbstokg функц нь бодит тоон төрлийн нэг аргумент авах бөгөөд мөн бодит тоон утга буцаадаг байна.

Функц утга буцааж байгаа тохиолдолд түүний утгыг авахын тулд энгийн илэрхийлэлтэй ажилладаг аргыг ашиглах нь зүйтэй. Жишээ нь, өмнөх программд бид функцийг утга оноох үйлдэлд ашиглаж байна.

```
kgs=lbstokg(lbs);
```

lbstokg функцийн буцаасан утга kgs хувьсагчид очих болно.

return нөөц үг

Өмнөх жишээнд буй lbstokg функц нь pounds гэсэн нэртэй нэг аргумент авдаг билээ. Энэ хувьсагч нь фунтийг илэрхийлсэн бодит тоон утга бөгөөд үүнийг тогтмол тоогоор үржүүлэн гарсан хариуг kilograms гэсэн хувьсагчид өгч байна. Харин энэ хувьсагчийн утгыг үндсэн программд буцаахын тулд return гэсэн нөөц үгийг ашиглаж байна.

```
return kilograms;
```

Функцээс буцсан утга нь үндсэн программын kgs гэсэн хувьсагчид дамжиж байна. Энд анхааруулахад, функцэд хичнээн ч аргумент дамжуулж болдог боловч харин цорын ганцхан утга буцааж болдог. Харин олон утга буцаах шаардлагатай болсон тохиолдолд хэрэглэдэг аргуудыг сүүлд танилцуулах болно. Хэрэв функцийн тодорхойлолтод буцах утгыг зааж өгөөгүй бол (void нөөц үгийг ч бичээгүй) уг функцийг int төрлийн утга буцаана гэж ойлгоно. Жишээ нь:

```
Somefunc();
```

Гэхдээ функц int төрлийн утга буцаадаг байсан ч тодорхойлолтод нь бичиж өгөх нь зүйтэй бөгөөд энэ нь программыг ойлгомжтой бөгөөд уншихад эвтэйхэн болгодог юм.

Өмнөх convert программд үндсэндээ шаардлагагүй байж болох зарим нэгэн хувьсагч байгаа бөгөөд эдгээрийг хэрхэн “цэгцлэх”-ийг дараагийн convert2 гэсэн жишээнээс үзье.

```
//convert2.cpp
#include <iostream.h>

float lbstokg(float);

void main()
{
float lbs;
cout << “ \nТа жингээ оруулна уу(фунт) : ”;
cin >> lbs;
cout << “Таны жин (килограмм) ”
<< lbstokg(lbs);
}

float lbstokg(float pounds);
          {
```

```
return 0.453592*pounds;
}
```

Үндсэн программд байсан kgs хувьсагчийг ашиглахгүй функцийн утгыг шууд хэвлэх коммандад өгсөн байна. Мөн lbstokg функц дэх kilograms хувьсагчийг хэрэглэхгүй болгож, үржих үйлдлийн үр дүнг шууд буцаасан байна. Программд ийм хэмнэлтүүдийг оруулснаар түүний хийх үйлдлүүд өөрчлөгдөхгүй боловч программын ашиглах санах ойг бага хэмжээгээр ч болов хэмнэх болно.

Бүтэц төрлийн утга буцаах

Одоогийн үзэх жишээнээс бүтэц нь функцэд аргумент болон дамжиж болдгийн адилаар функцийн утга болон буцаж болохыг мэдэж болно.

```
//retstrc.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

Distance addeng(Distance, Distance);
void engldisp (Distance);

void main()
{
        Distance d1, d2, d3;

        Cout << “\nФут : ”; cin >> d1.feet;
        Cout << “Инч : ”; cin >> d1.inches;

        Cout << “\nФут : ”; cin >> d2.feet;
```

```
        Cout << “Инч : ”; cin >> d2.inches;

        d3=addeng(d1, d2);

        Engldisp(d1); cout << “+”;
        Engldisp(d2); cout << “=”;
        Engldisp(d3); cout << “ \n ”;
}

Distance addeng(Distance dd1, Distance dd2);
        {
        Distance dd3;

        dd3.inches=dd1.inches+dd2.inches;
        dd3.feet=0;
        if (dd3.inches>=12.0)
                {
                dd3.inches-=12.0;
                dd3.feet++;
        }
dd3.feet+=dd1.feet+dd2.feet;
return dd3;
        }

void engldisp(Distance dd);
{
cout << dd.feet << “\’-” << dd.inches << “\””;
}
```

Программ доорх загварын үр дүн үзүүлнэ.

```
Фут : 4
Инч : 5.5

Фут : 5
Инч : 6.5

4’-5.5”+5’-6.5”=10’-0”
```

Хэрэглэгчийн өгсөн утгуудыг бүтцэд авч, addengl гэсэн функцээр нэмээд, гарсан хариуг функцийн утга

болгон буцааж байна. Бүтцийг хооронд нь шууд нэмж болохгүй учраас dd3 гэсэн шинэ хувьсагчийг тодорхойлон ашигласан байна. Энэ жишээнд main()-ийг оролцуулбал 3 функц тодорхойлогдон ашиглагдаж байгаа бөгөөд ер нь программд хэдэн ч функцийг ямар ч дарааллаар тодорхойлон ашиглаж болно. Харин функц дуудагдахаас өмнө урьдчилан тодорхойлогдсон байх ёстойг л мартаж болохгүй.

Аргументийг хуурамч нэрээр дамжуулах нь

Бидний өмнө үзсэн жишээнүүдэд функцийн аргументийг утгаар дамжуулж байгаа тухай дээр дурдсан билээ. Аргумент ингэж дамжих үед функц шинээр хувьсагч тодорхойлж, аргументэд өгсөн утгыг тэрхүү шинэ хувьсагчид оноодог билээ. Өөрөөр хэлбэл, функц параметрт өгөгдсөн хувьсагчтай ажилладаггүй бөгөөд харин түүнээс утга дамжуулж авсан дотоод хувьсагчтайгаа ажилладаг гэсэн үг юм. Аргументэд ингэж утга дамжуулах нь функц үндсэн программын хувьсагчуудын утгуудыг өөрчлөх шаардлагагүй үед хэрэгтэй юм.

Харин аргументийг хуурамч нэрээр дамжуулах гэдэг нь дээрхээс бүр өөр хэрэг юм. Энэ тохиолдолд аргументэд ирсэн хувьсагчийн зөвхөн утга нь биш, харин тэрхүү хувьсагчийг өөрийг нь дамжуулдаг юм. Ингэсний үр дүнд функц дотроос үндсэн программд байгаа хувьсагчийн утгыг шууд өөрчлөх боломжтой болох юм. Мөн ийм аргаар үндсэн функц рүү нэг биш хэд хэдэн утгыг буцаах боломжтой болно.

Энгийн төрлүүдийг хуурамч нэрээр дамжуулах

Одоо нэгэн жишээ үзэцгээе.

```
//ref.cpp
#include <iostream.h>

void main()
{
void intfrac(float, float &, float &);
float number, intpart, fracpart;
do
        {
        cout << “\nБодит тоо оруулна уу : ”;
        cin >> number;
        intfrac(number, intpart, fracpart);
        cout << “Бүхэл хэсэг нь ” << intpart
        << “, бутархай нь “ << fracpart;
        }
while (number!=0);
}
void intfrac(float n, float &ipart, float &fpart);
{
        ipart=float(long(n));
        fpart=n–ipart;
}
```

Хэрэглэгчийн оруулж өгсөн бодит тоог бүхэл болон бутархай хэсэг болгон хуваж өгч байна. Ингэж хуваахын тулд үндсэн программ intfrac() гэсэн функцийг ашиглаж байгаа юм. Жишээ нь:

```
Бодит тоо оруулна уу : 99.44
Бүхэл хэсэг нь 99, бутархай нь 0.44
```

Уг функц нь параметрт ирсэн тоог long төрөлд хувиргах аргаар бүхэл хэсгийг ялган авч, эхний тооноос уг бүхэл хэсгийг хасах замаар бутархай

хэсгийг нь олж байна. Харин тэрхүү олсон утгуудаа үндсэн программд хэрхэн дамжуулж байгааг одоо судалцгаая. Return операторыг ашиглавал хоёр биш зөвхөн ганц утга буцаах болно. Харин энэ функцэд дээр дурдсанаар хуурамч нэрийг ашиглажээ. Утгаар бус, хуурамч нэрээр дамжих хувьсагчийг тодорхойлохдоо төрлийн нэрийн дараа (&) тэмдэг тавьж өгдөг байна. Ийм тэмдэг тавигдсанаар функцэд ийм хувьсагч шинээр тодорхойлогдох бус, харин энэ нь функцийн аргументэд ирсэн хувьсагчид зориулагдсан хоёрдогч буюу хуурамч нэр гэдгийг илэрхийлэх болно. Ө.х. функц доторх fpart гэсэн хувьсагчид хандана гэдэг нь үнэн хэрэгтээ үндсэн программын fracpart хувьсагчид шууд хандаж байгаа хэрэг юм. fpart болон fracpart гэдэг нь санах ойн нэг ижил мужууд руу заасан хоёр нэр юм. Функцийн урьдчилсан тодорхойлолт болон биеийг тодорхойлох хэсгийн аль алинд нь (&) тэмдгийг орхиж болохгүй. Харин функцийг дуудах үед уг тэмдгийг тавихгүй. Өөрөөр хэлбэл, функцийг дуудаж байгаа хэсгээс аргументүүд нь утгаар дамжих, хуурамч нэрээр дамжихыг шууд хэлэх боломжгүй гэсэн үг юм. Харин n болон number гэдэг нь хоёр өөр хувьсагчууд болно. Учир нь, функцийн эхний параметр нь утгаар дамжиж байгаа билээ.

Хуурамч нэрээр дамжуулах өөр нэгэн жишээ

Энд өөр нэгэн жишээ авч үзье. Программд байгаа хос тоонуудыг жишиж, бага тоог нь ихийнх нь өмнө тавих шаардлага гарсан байг. Үүнийг хийхийн тулд order гэсэн функцийг ашиглана.

```
//reforder.cpp
#include <iostream.h>

void main()
{
void order(int &,int &);

int n1=99, n2=11;
int n3=22, n4=88;

order(n1, n2);
order(n3, n4);

cout << endl << “n1=” << n1;
cout << endl << “n2=” << n2;
cout << endl << “n3=” << n3;
cout << endl << “n4=” << n4;
}

void order(int & numb1, int & numb2);
        {
        if (numb1>numb2)
                {
                int temp=numb1;
                numb1=numb2;
                numb2=temp;
                }
        }
```

Программд байгаа хоёр хос тоонуудыг тус бүрд нь эрэмбэлсний дараа бүх тоог дэлгэцэнд хэвлэн харуулж байна. Үр дүн нь:

```
n1=11
n2=99
n3=22
n4=88
```

Энд функцэд байгаа numb1, numb2 гэсэн нэрнүүд бол аргументэд ирсэн хувьсагчуудын хуурамч нэр гэдгийг сайн ойлгох хэрэгтэй. Функцийг

эхлээд дуудахад numb1 нь n1-ийн хоёрдогч нэр, numb2 нь n2-ийн хоёрдогч нэр болох бөгөөд харин дараагийн удаа дуудахад numb1 нь n3-ийн хуурамч нэр, numb2 нь n4-ийнх тус тус болно. Энд хуурамч нэрээр дамжуулахын хамгийн гол ашиг тус ажиглагдаж байгаа юм. Үндсэн программ ямар хувьсагч дээр үйлдлийг хйихийг заагаад өгнө, харин функц уг хувьсагчуудын жинхэнэ нэрийг мэдэхгүй атлаа тэр хувьсагчууд дээр үйлдлийг чөлөөтэй хийж байна. Яг л алсын удирдлага шиг ажиллаж байгаа юм. Хувьсагчийг ингэж хуурамч нэрээр дамжуулах нь утгаар дамжуулах болон огт дамжуулалгүй, шууд хандаж ажиллахаас аль алинаас нь өөр арга юм.

Бүтцийг хуурамч нэрээр дамжуулах

Доорх жишээнд бидний өмнө үзсэн Distance бүтцийг тодорхой тоогоор үржүүлэх (масштаблах) үйлдлийг хийнэ. Жишээ нь: 6’ – 8” –ийг 0.5 дахин масштаблавал 4’ – 3” болно.

```
//referst.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

void scale(Distance &, float);
void engldisp(Distance);

void main()
{
        Distance d1={12, 6.5},
        Distance d2={10, 5.5},
```

```
        Cout << "\nd1="; Engldisp(d1);
        Cout << "\nd2="; Engldisp(d2);

        Scale(d1, 0.5);
        Scale(d2, 0.25);

        Cout << "\nd1=";        Engldisp(d1);
        Cout << "\nd2=";        Engldisp(d2);
}

void scale(Distance &dd, float factor)
        {
        float inches=(dd.feet*12+dd.inches)*factor;
        dd.feet=inches/12;
        dd.inches=inches–dd.feet*12;
        }

void engldisp(Distance dd);
{
cout << dd.feet << "\'-" << dd.inches << "\"";
}
```

Программ Distance төрлийн хоёр хувьсагч тодорхойлж, тэдгээрийн анхны утгуудыг дэлгэцэнд хэвлэн харуулж байна. Харин дараа нь d1-ийг 0.5 дахин, d2-ийг 0.25 дахин масштаблаж, түүнийгээ дэлгэцэнд хэвлэн харуулах болно. Программын үр дүн нь доорх байдлаар гарна.

```
d1=12'–6.5"
d2=10'–5.5"
d1=6'–3.25"
d1=2'–7.375"
```

Функц ямар нэгэн үр дүн буцаахгүй, харин шууд үндсэн программын бүтцүүдэд өөрчлөлтийг хийдэг юм.

Аргументийг ингэж дамжуулах арга нь Pascal болон Basic хэлэнд мөн ашиглагддаг. Аргументийг

дамжуулах 3 дахь арга нь заагчийг ашиглах явдал юм. Энэ тухай бид хойно үзэх болно.

Давхардсан функцүүд

Зарим тохиолдолд нэгэн ижил функцэд өөр өөр аргументүүд өгөх шаардлага гарч болно. Өөрөөр хэлбэл, бидний функц нэгэн төрлийн өгөгдөлтэй нэг бүлэг үйлдэл, харин өөр төрлийн өгөгдлүүдтэй бол өөр бүлэг үйлдэл хийнэ гэсэн үг юм.

Өөр тоотой аргументүүд

Өмнөх бүлгүүдэд бид (*) тэмдгийг 45 удаа хэвлэдэг starline() гэсэн функц, харин өгсөн тэмдгийг өгсөн тоогоор хэвлэдэг repchar() гэсэн функцүүдийг үзэж байсан билээ. Харин одоо өгсөн тэмдгийг 45 удаа хэвлэдэг charline() гэсэн функцийг шинээр бичих шаардлага гарлаа гэж үзье. Энэ 3 функц тун төстэй үйлдлүүд хийдэг атлаа өөр өөр нэртэй. Программчлагчийн хувьд эдгээр функцүүдийг ашиглахын тулд 3 өөр нэрийг тогтоох, мөн help ашиглахын тулд 3 өөр нэр хайх хэрэгтэй болно. Харин өөр аргументүүд авдаг ижил нэртэй функц байвал дээрх асуудлууд их хэмжээгээр хөнгөрөх нь ойлгомжтой.

```
//overload.cpp
#include <iostream.h>

void repchar();
void repchar(char);
void repchar(char, int);
```

```
void main()
{
        repchar();
        repchar('=');
        repchar('+', 30);
}

void repchar();
        {
        for (int j=0; <45; j++)
                cout << '*';
        cout << endl;
        }
void repchar(char ch);
        {
        for (int j=0; <45; j++)
                cout << ch;
        cout << endl;
        }
void repchar(char ch, int n);
        {
        for (int j=0; <n; j++)
                cout << ch;
        cout << endl;
        }
```

Программ доорх үр дүнг харуулна.

```
*********************************************
=============================================
++++++++++++++++++++++++++++++
```

Эхний хоёр мөрөнд 45 тэмдэгт, сүүлийн мөрөнд 30 тэмдэгт хэвлэгдсэн байна. Энэ программд ижил нэртэй 3 функцийг тодорхойлсон бөгөөд тийм нэртэй функцийг 3 дахин дуудсан байна. Тэгвэл эдгээр функцүүдийг хооронд нь юу ялгаж өгч байна вэ? Ердөө л тэдгээрийн аргументийн тоо болон аргументүүдийн төрлүүд. Өөрөөр хэлбэл, функцийг

дуудах үед тэдгээрийн аль нь дуудагдаж байгааг компилятор аргументийн тоо ба тэдгээрийн төрлөөр нь ялгана гэж ойлгох хэрэгтэй.

Өөр төрөлтэй аргументүүд

Тооны хувьд ижил ч аргументүүдийнхээ төрлүүдээр ялгаатай функцүүдийг ч давхардуулан тодорхойлж болно. Доорх жишээнд байгаа функцийн аргументэд нэг бол Distance төрлийн бүтэц, эсвэл float төрлийн хувьсагчийг өгнө.

```
//overengl.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

void engldisp(Distance);
void engldisp(float);

void main()
{
        Distance d1;
        float d2;

        cout << “\nФут:”; cin >> d1.feet;
        cout << “\nИнч:”; cin >> d1.inches;

cout << “\nНийт зай (инчээр):”; cin >> d2;

        cout << “\nd1=”; engldisp(d1);
        cout << “\nd2=”; engldisp(d2);
}
```

```
void engldisp(Distance);
        {
cout << dd.feet << "\'-"
<< dd.inches << "\"";
        }

void engldisp(float dd);
        {
        int feet=dd/12;
        float inches=dd–feet*12;
cout << feet << "\'-" << inches << "\"";
        }
```

Энд ашиглагдаж байгаа engldisp () функцүүд нь хоёулаа адилхан, нэг нэг аргумент авдаг боловч авч байгаа тэр нэг аргумент нь эхний функцийн хувьд Distance төрөлтэй, харин дараагийн функцэд float төрөлтэй байна. Үр дүн нь:

```
Фут:5
Инч:10.5
Нийт зай (инчээр):76.5
d1=5'–10.5"
d2=6'–4.5"
```

Давхардсан функц нь программистыг олон тооны функцийн нэр тогтоохоос хөнгөвчилдөг юм. Давхардсан функц ашиглаагүй байхад асуудал яаж хүндэрэхийг доорх тайлбараас үзэж болно. С – д функцийг давхардуулан тодорхойлох гэсэн ойлголт байхгүй учраас тооны абсолют утгыг өгдөг 4 функц байдаг. Бүхэл тооны хувьд abs(), комплекс тооны хувьд cabs(), бутархай тооны хувьд fabs(), харин их бүхэл тооны хувьд labs() гэсэн функцүүд тус тус байдаг. Харин С++ - д бол ердөө ганцхан abs() гэсэн функцээр бүх төрлийн абсолют утгыг олж болно.

Дотоод функцүүд

Функцийн амин чухал үүргүүдийн нэг нь санах ойг хэмнэх явдал хэмээн өмнө бид дурдсан билээ. Өөрөөр хэлбэл функцийг дуудах бүрийд программын удирдлага нэг л газар очдог бөгөөд функц дуудагдах тоогоор программд давтагдан бичигдэхгүй гэсэн үг юм. Функц дуудах коммандад хүрмэгц удирдлага функцийн эхлэлийн комманд дээр харайж очих бөгөөд, харин функц дуудахад удирдлага буцаж дуудсан газраа ирдэг юм.

Функц хэдийгээр санах ойг хэмнэх сайн талтай ч маш олон дуудагдах тохиолдолд программын хурдад муу нөлөө үзүүлж болох юм. Өөрөөр хэлбэл, тэр бүрийд шаардлагатай регистрүүдийг стект хийх, аргументүүдийг мөн хадгалах, удирдлагыг шилжүүлэх, харин буцах бүрийд регистрүүдийг сэргээх, удирлага буцаах зэрэг үйлдлүүд хийгдэнэ.

Харин биелэлтийн хурдыг нэмэгдүүлэхийн тулд функцийг дотоод функц болгож болох юм. Ийм функц нь программыг хөрвүүлэх үед функцийг дуудсан газар бүрт удирдлага шилжүүлэх коммандыг биш харин уг функцийн коммандуудыг шууд орлуулан бичдэг юм. Том хэмжээний функцүүдийн хувьд энгийн аргаар зохион байгуулах нь илүү байж болох юм. Харин цөөн тооны коммандаас тогтсон функцүүдийг дотоод биш болгон зохион байгуулах нь санах ойн хувьд ч бараг хэмнэлтгүй (бүр алдагдалтай ч байж болно), хурдны хувьд ч алдагдалтай байж магадгүй. Нэг программд тун олон удаа давтагдан орж байгаа бага хэмжээний бүлэг үйлдлийг функц болголгүй дахин давтан бичих нь программын хурдыг хэмнэх боловч, программ дэндүү нүсэр бүтэцтэй, ойлгомж муутай болох аюултай. Ийм л тохиолдолд дотоод функцийг

ашиглах нь зүйтэй. Доорх жишээнд дотоод функцийг ашиглаж үзүүлсэн байна.

```
//inline.cpp
#include <iostream.h>

inline float lbstokg(float pounds)
        {
        return 0.453592 * pounds;
        }

void main()
{
float lbs;

cout << “\nЖингээ оруулна уу(фунтээр): ”;
cin >> lbs;
cout << “Таны жин(килограммаар) ”
<< lbstokg(lbs);
}
```

Дотоод функцийг тодорхойлохын тулд inline гэсэн түлхүүр үгийг функцийн тодорхойлолтод хэрэглэхэд л хангалттай. Харин функцийн урьдчилсан тодорхойлолт биш, жинхэнэ биеийг нь үндсэн программаас өмнө тодорхойлж өгсөн байх ёстой. Уг функцийн биеийг программ дотор шууд орлуулан ашиглах ёстойгоос ийм шаардлага бий болж байна. Ийм тохиолдолд функцийн урьдчилсан тодорхойлолт шаардагдахгүй.

Стандарт утгууд

Функцийг дуудах үед бүх аргументэд утга өгөхгүй байж болно. Харин утга өгөхгүй байж болох

аргументүүдэд стандарт утгууд оноож өгсөн байх шаардлагатай.

Доор үзэх жишээ нь өмнө үзсэн overload функцтэй үр дүнгийн хувьд яг ижил боловч программын хувьд бага зэрэг ялгаатай болохыг та анзаарах болно.

```
        }
void repchar(char ch, int n);
        {
        for (int j=0; <n; j++)
                cout << ch;
        cout << endl;
        }
```

Программ доорх үр дүнг харуулна.

```
**********************************************
==============================================
++++++++++++++++++++++++++++++
```

Эхний хоёр мөрөнд 45 тэмдэгт, сүүлийн мөрөнд 30 тэмдэгт хэвлэгдсэн байна. Энэ программд ижил нэртэй 3 функцийг тодорхойлсон бөгөөд тийм нэртэй функцийг 3 дахин дуудсан байна. Тэгвэл эдгээр функцүүдийг хооронд нь юу ялгаж өгч байна вэ? Ердөө л тэдгээрийн аргументийн тоо болон

аргументүүдийн төрлүүд. Өөрөөр хэлбэл, функцийг дуудах үед тэдгээрийн аль нь дуудагдаж байгааг компилятор аргументийн тоо ба тэдгээрийн төрлөөр нь ялгана гэж ойлгох хэрэгтэй.

Өөр төрөлтэй аргументүүд

Тооны хувьд ижил ч аргументүүдийнхээ төрлүүдээр ялгаатай функцүүдийг ч давхардуулан тодорхойлж болно. Доорх жишээнд байгаа функцийн аргументэд нэг бол Distance төрлийн бүтэц, эсвэл float төрлийн хувьсагчийг өгнө.

```
//overengl.cpp
#include <iostream.h>

struct Distance
        {
        int feet;
        float inches;
        };

void engldisp(Distance);
void engldisp(float);

void main()
{
        Distance d1;
        float d2;

        cout << "\nФут:"; cin >> d1.feet;
        cout << "\nИнч:"; cin >> d1.inches;

cout << "\nНийт зай (инчээр):"; cin >> d2;

        cout << "\nd1="; engldisp(d1);
        cout << "\nd2="; engldisp(d2);
}
```

```
void engldisp(Distance);
        {
cout << dd.feet << "\'-"
<< dd.inches << "\"";
        }

void engldisp(float dd);
        {
        int feet=dd/12;
        float inches=dd–feet*12;
cout << feet << "\'-" << inches << "\"";
        }
```

Энд ашиглагдаж байгаа engldisp () функцүүд нь хоёулаа адилхан, нэг нэг аргумент авдаг боловч авч байгаа тэр нэг аргумент нь эхний функцийн хувьд Distance төрөлтэй, харин дараагийн функцэд float төрөлтэй байна. Үр дүн нь:

```
Фут:5
Инч:10.5
Нийт зай (инчээр):76.5
d1=5'–10.5"
d2=6'–4.5"
```

Давхардсан функц нь программистыг олон тооны функцийн нэр тогтоохоос хөнгөвчилдөг юм. Давхардсан функц ашиглаагүй байхад асуудал яаж хүндэрэхийг доорх тайлбараас үзэж болно. С – д функцийг давхардуулан тодорхойлох гэсэн ойлголт байхгүй учраас тооны абсолют утгыг өгдөг 4 функц байдаг. Бүхэл тооны хувьд abs(), комплекс тооны хувьд cabs(), бутархай тооны хувьд fabs(), харин их бүхэл тооны хувьд labs() гэсэн функцүүд тус тус байдаг. Харин С++ - д бол ердөө ганцхан abs() гэсэн функцээр бүх төрлийн абсолют утгыг олж болно.

Дотоод функцүүд

Функцийн амин чухал үүргүүдийн нэг нь санах ойг хэмнэх явдал хэмээн өмнө бид дурдсан билээ. Өөрөөр хэлбэл функцийг дуудах бүрийд программын удирдлага нэг л газар очдог бөгөөд функц дуудагдах тоогоор программд давтагдан бичигдэхгүй гэсэн үг юм. Функц дуудах коммандад хүрмэгц удирдлага функцийн эхлэлийн комманд дээр харайж очих бөгөөд, харин функц дуудахад удирдлага буцаж дуудсан газраа ирдэг юм.

Функц хэдийгээр санах ойг хэмнэх сайн талтай ч маш олон дуудагдах тохиолдолд программын хурдад муу нөлөө үзүүлж болох юм. Өөрөөр хэлбэл, тэр бүрийд шаардлагатай регистрүүдийг стект хийх, аргументүүдийг мөн хадгалах, удирдлагыг шилжүүлэх, харин буцах бүрийд регистрүүдийг сэргээх, удирлага буцаах зэрэг үйлдлүүд хийгдэнэ.

Харин биелэлтийн хурдыг нэмэгдүүлэхийн тулд функцийг дотоод функц болгож болох юм. Ийм функц нь программыг хөрвүүлэх үед функцийг дуудсан газар бүрт удирдлага шилжүүлэх коммандыг биш харин уг функцийн коммандуудыг шууд орлуулан бичдэг юм. Том хэмжээний функцүүдийн хувьд энгийн аргаар зохион байгуулах нь илүү байж болох юм. Харин цөөн тооны коммандаас тогтсон функцүүдийг дотоод биш болгон зохион байгуулах нь санах ойн хувьд ч бараг хэмнэлтгүй (бүр алдагдалтай ч байж болно), хурдны хувьд ч алдагдалтай байж магадгүй. Нэг программд тун олон удаа давтагдан орж байгаа бага хэмжээний бүлэг үйлдлийг функц болголгүй дахин давтан бичих нь программын хурдыг хэмнэх боловч, программ дэндүү нүсэр бүтэцтэй, ойлгомж муутай болох аюултай. Ийм л тохиолдолд дотоод функцийг

ашиглах нь зүйтэй. Доорх жишээнд дотоод функцийг ашиглаж үзүүлсэн байна.

```
//inline.cpp
#include <iostream.h>

inline float lbstokg(float pounds)
        {
        return 0.453592 * pounds;
        }

void main()
{
float lbs;

cout << “\nЖингээ оруулна уу(фунтээр): ”;
cin >> lbs;
cout << “Таны жин(килограммаар) ”
<< lbstokg(lbs);
}
```

Дотоод функцийг тодорхойлохын тулд inline гэсэн түлхүүр үгийг функцийн тодорхойлолтод хэрэглэхэд л хангалттай. Харин функцийн урьдчилсан тодорхойлолт биш, жинхэнэ биеийг нь үндсэн программаас өмнө тодорхойлж өгсөн байх ёстой. Уг функцийн биеийг программ дотор шууд орлуулан ашиглах ёстойгоос ийм шаардлага бий болж байна. Ийм тохиолдолд функцийн урьдчилсан тодорхойлолт шаардагдахгүй.

Стандарт утгууд

Функцийг дуудах үед бүх аргументэд утга өгөхгүй байж болно. Харин утга өгөхгүй байж болох

аргументүүдэд стандарт утгууд оноож өгсөн байх шаардлагатай.

Доор үзэх жишээ нь өмнө үзсэн overload функцтэй үр дүнгийн хувьд яг ижил боловч программын хувьд бага зэрэг ялгаатай болохыг та анзаарах болно.

```
//missarg.cpp
#include <iostream.h>

void repchar(char='*', int=45);

void main()
{
        repchar();
        repchar('=');
        repchar('+',30);
}

void repchar(char ch,int n);
        {
        for (int j=0; j<n; j++)
                cout << ch;
        cout << endl;
        }
```

Энэ программд repchar гэсэн функцийг 3 удаа дуудаж байна. Аргументийнх нь тоо зөрж байхад яагаад эхний 2 удаа дуудахад ямар нэгэн алдаа гаргахгүй байна вэ? Үүний учир нь уг функцийг аргументүүд нь стандарт утгуудтай байгаад оршиж байгаа юм. Дуудаж буй программ уг аргументэд утга өгөхгүй бол функц уг аргументийн утгыг стандарт утгаар нь авдаг. Стандарт утгуудыг функцийн урьдчилсан тодорхойлолтод зааж өгсөн байна.

```
void repchar(char='*', int=45);
```

Ингэж тодорхойлох үед мөн хувьсагчдын нэрийг зааж өгч болно.

```
void repchar(char ch='*', int n=45);
```

Харин анхааруулахад, функцийн зөвхөн сүүлийн аргументүүдийн утгыг л орхиж болно. Ө.х. энэ функцийг дуудахдаа эхний аргументийн утгыг орхиж, repchar(30) гэж дуудвал компилятор төрөл тохирохгүй байгаа тухай алдаа өгөх болно. Мэдээж хэрэг энэ тохиолдолд компилятор аргументийн утгыг 2 дугаар аргументийнх гэж мэдэх боломжгүй хэрэг. Харин дундаас нь утга орхих шаардлагатай болвол орхисон утгынхаа байранд хоосон таслал үлдээх ёстой.

```
repchar(, 30)
```

Стандарт утгыг ашиглах нь аргументэд тохиромжгүй буюу буруу утга өгөх аюулыг багасгадаг юм (Гэхдээ 100 хувь биш). Мөн программчлагч тухайн үед шаардагдахгүй ч яваандаа хэрэг болж магадгүй аргументүүдийг ийм байдлаар тодорхойлж өгч болох юм.

Хувьсагчдын тухай

Одоо функц болон үндсэн программд байгаа хувьсагчуудын товчхон тайлбар үзье. Хувьсагчийг дотор нь автомат, глобаль, статик гэж 3 ангилдаг.

Автомат хувьсагчууд

Бидний одоог хүртэл үзсэн бүх хувьсагчид функц болон программынхаа дотор талд тодорхойлогдож байсан билээ.

```
void somefunc()
        {
        int somevar;
        float othervar;

        //коммандууд

        }
```

Ингэж функцийнхээ бие дотор тодорхойлогдсон хувьсагчдыг автомат хувьсагчууд гэж нэрлэнэ. Жинхэнэ бичлэгээрээ бол автомат хувьсагчдыг тодорхойлохдоо төрлийнх нь өмнө auto гэсэн үгийг бичих ёстой.

```
void somefunc()
        {
        auto int somevar;
        auto float othervar;

        //коммандууд

        }
```

Гэхдээ энэ нь хувьсагчийн стандарт хэлбэр учраас ямар нэгэн түлхүүр үг ашиглаагүй тодорхойлсон хувьсагчийг автомат хувьсагч гэж авдаг. Одоо автомат хувьсагчдын шинж чанаруудыг авч үзье.

Хүчинтэй хугацаа

Автомат хувьсагч нь түүнийг тодорхойлсон функцийг дуудтал санах ойд үүсээгүй байдаг (Ямар нэгэн блок доторх хувьсагчид уг блокт хандах үед л үүсдэгтэй нэг адил). Дээрх жишээнд буй somevar болон othervar хувьсагчууд нь somefunc функцийг дуудагдтал хаана ч байхгүй байна гэсэн үг. Тэдгээрийн тодорхойлогдсон блок нь санах ойд үүсээгүй байгаа учраас л тэр. Функц дуудагдангуут л уг хувьсагчуудад зориулагдсан санах ой бий болох бөгөөд харин функц ажиллаж дуусаад, удирдлага буцах үед мөн хувьсагчид санах ойгоос арчигдах болно. Ингэж автоматаар үүсч, устдаг учраас ийм хувьсагчдыг автомат гэж нэрлэсэн байна. Хувьсагч үүссэнээс устах хүртэл хугацааг түүний хүчинтэй хугацаа гэнэ. Ингэж хүчинтэй хугацааг хязгаарлагдсанаар санах ойн хэмнэлт үүсдэг юм. Функц дуусч, хувьсагч арчигдсанаар тэдгээрт зориулагдаж байсан санах ой өөр функц, хувьсагчид ашиглагдах боломжтой болж чөлөөлөгдөх болно.

Хүчинтэй хүрээ

Хувьсагчийн хүчинтэй хүрээ гэдэг нь энэ хувьсагчид хандаж болох функц болон программуудыг хэлж байгаа юм. Өөрөөр хэлбэл, уг хувьсагчид хандахад unknown variable (тодорхойлогдоогүй хувьсагч) гэсэн алдаа гарахгүй хүрээг хэлнэ. Автомат хувьсагчид бол зөвхөн түүнийг тодорхойлсон байгаа функц дотроо л хүчинтэй байдаг юм.

```
void somefunc()
        {
        int somevar;
        float othervar;
```

```
        somevar=10;
        othervar=123.45;
        nextvar=15;                 //буруу
        }

void main()
{
int nextvar;
somevar=20;             //буруу
othervar=987.65;        //буруу
nextvar=35;
}
```

nextvar гэсэн хувьсагч эхний функцэд хүчинтэй биш тул ингэж утга оноовол дээр дурдсан алдааг өгөх болно. Үүний нэгэн адилаар somevar болон othervar хувьсагчид үндсэн программд хүчингүй юм. Хувьсагчийн хүчинтэй хүрээ ингэж хязгаарлагдах нь уг хувьсагчийн утгыг мэдэхгүй өөр функц дотроос өөрчлөх боломжийг хааж өгөх болно. Ер нь энэхүү хүчинтэй хүрээ гэсэн ойлголт нь обьект хандлагат программчлалын хувьд тун өргөн ашиглагдах болно.

Анхны утга

Автомат хувьсагч тодорхойлогдох үед түүнд ямар ч анхны утга өгөгдөхгүй. Түүний утга нь 0 ч байх албагүй. Харин ямар нэгэн утгагүй тоо байх болно. Ийм учраас хувьсагчтай анхны утга оноолгүйгээр ажиллах нь маш том, ойлгомжгүй алдаанд хүргэж болох аюултай. Харин анхны утгыг шууд тодорхойлох үедээ оноох нь хамгийн зөв арга юм.

```
int n=33;
```

Автомат хувьсагчдыг зарим тохиолдолд локаль ч гэж нэрлэдэг.

Глобаль хувьсагчууд

Хувьсагчийн өөр нэгэн хэлбэр бол глобаль хувьсагч билээ. Автомат хувьсагч ямар нэгэн функцийн дотор тодорхойлогддог бол глобаль хувьсагчид нь бүх функцийн гадна талд тодорхойлогддог юм. Тэрээр бүх функцэд ашиглагддаг болохоор түүнийг глобаль гэж нэрлэдэг бас болно.

```
//extern.cpp
#include <iostream.h>
#include <conio.h>

char ch='a';

void getachar();
void putachar();

void main()
{
while (ch!='\r')
        {
        getachar(); putachar();
        }
}

void getachar();
        {
        ch=getch();
        }

void putachar();
        {
        cout << ch;
```

```
}
```

Дээрх программд байгаа getachar функц гарнаас нэг тэмдэг хүлээж авах ба харин энэ тэмдэгээ дэлгэцэнд харуулахгүй. Харин хүлээж авсан тэрхүү тэмдгийг putachar гэсэн функцээр дэлгэцэнд хэвлэж байгаа юм. Программын үр дүнд нь таны дарсан товч бүр дэлгэцэнд доорх байдлаар хэвлэгдэн гарах болно.

```
I`m typing
```

Хамгийн гол анхаарах зүйл гэвэл ch гэсэн хувьсагчийг аль ч функц дотор тодорхойлоогүй боловч бүх функц, үндсэн программ хүртэл түүнийг ашиглаж байна. Глобаль хувьсагчийг тодорхойлсноос хойш тодорхойлогдсон бүх функц дотор түүнийг ашиглаж болдог. Глобаль хувьсагчийн хүчинтэй хүрээ нь бүхэл программ болно.

Гадаад хувьсагчийн гол үүрэг

Хэд хэдэн функцэд ашиглагдах хувьсагчийг глобаль болгон тодорхойлох нь зүйтэй. Ер нь глобаль хувьсагчууд нь уг программынхаа хувьд нэлээд чухал хувьсагчид байдаг. Харин шаардлагагүй хувьсагчийг глобаль болгох нь тийм ч зөв арга биш. Учир нь глобаль хувьсагчид бүх функц ханддаг учраас түүний утга нь санаандгүй байдлаар өөрчлөгдөж, буруудах магадлал ихтэй юм.

Анхны утга

Глобаль хувьсагчид автомат хувьсагчийн нэгэн адил тодорхойлох үед нь анхны утга оноож болно. Программ дуудагдаж, санах ойд ачаалагдах үед глобаль хувьсагчууд шууд үүсч анхны утга нь оноогдоно. Гэхдээ автомат хувьсагчаас давуу тал нь хэрэв түүнд анхны утга оноогоогүй байвал шууд 0 утга олгогддог явдал юм.

Хүчинтэй хугацаа ба хүрээ

Глобаль хувьсагч бүхий л программын туршид хүчинтэй байдаг. Программ санах ойд ачаалагдах үед л глобаль хувьсагчууд үүсэх бөгөөд, уг программыг дуустал санах ойд байрлах юм. Глобаль хувьсагчид тодорхойлогдсоноос хойш тодорхойлогдсон бүх функцэд ашиглагдаж болно. Хэрэв ch хувьсагчийг үндсэн программын дараа тодорхойлсон бол түүнийг бусад функцүүд нь ашиглаж болох боловч, үндсэн программ ашиглаж чадахгүй.

Статик хувьсагчид

Одоо хувьсагчийн өөр нэгэн төрлийг үзье. Статик глобаль хувьсагчууд болон статик автомат хувьсагчууд гэж байна. Статик гадаад хувьсагчийн тухай 15 дугаар бүлэгт үзэх болно.

Статик автомат хувьсагч нь локаль хувьсагч шиг хүрээтэй атлаа глобаль хувьсагч шиг хугацаанд хүчинтэй байдаг. Өөрөөр хэлбэл, зөвхөн тодорхойлогдсон функц дотроо ашиглагдах боловч, программ (функц биш) эхлэхээс дуусах хүртэл санах ойгоос арчигдахгүй байна гэсэн үг.

Статик автомат хувьсагч нь функцэд шаарлагатай ямар нэгэн хувьсагчийн утгыг функцийг дахин дуудах хооронд өөрчлөхгүй байлгах шаардлагатай тохиолдолд ашиглагдах ёстой юм. Одоо getavg() гэх дараагийн жишээг үзэцгээе.

```
//static.cpp
#include <iostream.h>

float getavg(float);

void main()
{
        float data=1, avg;

        while (data!=0)
                {
                cout << " Тоо оруулна уу : ";
                cin >> data;
                avg=getavg(data);
                cout << "Дундаж утга нь " << avg << endl;
                }
}

float getavg(float newdata)
        {
        static float total=0;
        static int count=0;

        count++;
        total+=newdata;
        return total / count;
        }
```

Энэ программ хэрэглэгчийн оруулж өгсөн тоонуудын дунджийг гаргах юм. Дунджийг тооцохдоо getavg функцийг ашиглаж байна. Энд өмнө оруулсан тоонуудын дунджийг уг функц дуудагдахын хооронд хадгалж байх шаарлага гарч байна. Хэрэв total ба count гэсэн хувьсагчууд нь статик биш байвал уг

функц нэг дуудагдаад буцах үед хувьсагчид устаж тэдгээрийн утгууд үгүй болно. Ингээд дахин дуудагдах үед уг хувьсагчид дахин тодорхойлогдож, шинээр 0 утга авах болно. Харин статикаар тодорхойлогдсон учраас функц дахин дуудагдах үед өмнөх total, count хувьсагчуудын утга огт өөрчлөгдөөгүй байх болно.

Анхны утга

Статик хувьсагчид бусад хувьсагчдын нэгэн адил тодорхойлох үед нь анхны утгыг оноож өгч болно. Гэхдээ энэ анхны утга нь программ эхлэх үед цорын ганц удаа олгогдох юм (Харин автомат хувьсагчид функц дуудагдах бүрд анхны утга оноогддог билээ).

ТӨРӨЛ	АВТОМАТ	СТАТИК АВТО.	ГЛОБАЛЬ
ХҮРЭЭ	Функц	Функц	Бүхэл программ
ХУГАЦАА	Функц	Бүхэл программ	Бүхэл программ
АНХНЫ УТГА	Тодорхойгүй	0	0
ЗОРИЛГО	Функцийн энгийн хувьсагчид	Автомат боловч функц дуусахад утгаа	Хэд хэдэн функц дунд ашиглагдах

Хуурамч нэрээр утга буцаах нь

Аргументийг хуурамч нэрээр дамжуулж болдгийн нэг адил буцах утгыг нь ч хуурамч нэрээр дамжуулж болно. Утгыг хуурамч нэрээр буцааснаар бид

функцийг утга оноох үйлдлийн зүүн талд ашиглах боломжтой болох юм.

```
//retref.cpp
#include <iostream.h>

int x;
int& setx();

void main()
{
setx()=92;
cout << “\nx=” << x;
}

int& setx();
        {
        return x;
        }
```

Энэ программд setx функцийн буцах утгыг хуурамч нэрээр дамжуулна гэж зааж өгсөн байна. Уг функц x гэсэн глобаль хувьсагчийг буцаах болно. Нэгэнт энэ нь хуурамч нэр учраас уг функц зүгээр нэг утга бус, бүтэн хувьсагч буцааж байгаа хэрэг юм. Тийм учраас функцийн үр дүн болон буцаж байгаа хувьсагчид шууд утга оноож болно гэсэн үг. Өөрөөр хэлбэл, функцэд утга оноохыг түүний буцааж буй хувьсагчид утга оноож байна гэж ойлгож болно. Үр дүн нь:

```
x=92
```

Функцийн заагч

Функц гэдэг нь санах ойд агуулагдаж байгаа хэлбэрээрээ үнэндээ хувьсагчаас нэг их гойдын ялгаагүй юм. Учир нь гэвэл, хувьсагч гэдэг нь ямар нэгэн өгөгдлийг хадгалж байгаа санах ойн хаягийг агуулж байдаг бол функц гэдэг нь мөн л ямар нэгэн бүлэг үйлдлийг агуулж байгаа санах ойн хаягийг зааж байдаг ажээ. Хувсагчид заагч үүсгэж болдгийн нэг адилаар бид заагч төрлийн хувьсагч тодорхойлоод, түүгээрээ функцийн биелэх кодуудын эхлэх хаяг руу (ө.х. функц рүү) заалгаж болно. Ингэж функцийн заагч тодорхойлохын нэг гол хэрэглээ бол функцийн аргументэд функц өгөх асуудал юм. Жишээ болгож int төрлийн утга буцаадаг, аргумент авдаггүй фнкцийн заагчийг тодорхойлъё.

```
int ( *ptr ) ()
```

Энд бичгдсэн хаалтуудаас алийг нь ч орхиж болохгүй. Хэрэв

```
int ( *ptr )
```

гэж тодорхойлвол энэ нь функцийн биш энгийн нэгэн int төрлийн заагч болно. Хэрэв

```
int *ptr ()
```

гэж тодорхойлвол энэ нь заагч хувьсагч биш, харин int төрлийн заагч утга буцаадаг функцийг тодорхойлж байгаа хэрэг болно.

Рекурсийн тухай

Өөрийгөө дууддаг функцийг рекурсив функц гэнэ. Рекурс бол программ зохиогчийн хувьд маш хүчирхэг бөгөөд ойлгомжтой зэвсэг бөгөөд рекурсийг ашиглан маш олон асуудлыг шийдэж болдог. Тодруулж хэлбэл, рекурсийг эх бодлоготойгоо адил чанарын дэд бодлогуудыг шийдэхэд ашигладаг юм. Дэд бодлого бүр тодорхой үр дүнг гаргахын тулд өөрийн дэд бодлогоо шийдэх ёстой. Ямар нэгэн шатанд дэд бодлого энгийн бодлогод шилжиж, үр дүн гаргасан цагт бодолт түүний эх бодлого руу шилжинэ.

Рекурсийн хамгийн ойрын жишээ бол факториал юм. 1-ээс N хүртэл тоонуудын үржвэрийг N-ийн факториал гэнэ. Жишээ нь, 1-ийн факториал нь 1, 2-ийн факториал нь 2, 3-ийн факториал нь 6, харин 7-ийн факториал нь 5040 байх жишээтэй. Харин 0-ийн факториалийг 1 гэж үзнэ. Факториалийг N! гэсэн хэлбэртэй бичнэ.

```
1! = 1
2! = 1*2
3! = 1*2*3
…
10! = 1*2*3*4*5*6*7*8*9*10
N! = 1*2*3*…*(N-1)*N
```

Тэгвэл дээрх факториалын томъёонд бяцхан хувиргалт хийе.

```
N! = 1*2*3*…*(N-1)*N = N*(N-1)*…*3*2*1 = N*(N-1)!
```

Эндээс харвал N-ийн факториал гэдэг бол (N-1)-ийн факториалийг N-ээр үржсэнтэй тэнцэж байгаа юм. Ингэхээр өмнө өгүүлсэн рекурсийн шинж тэмдэг илэрч байгаа биз. N-ийн факториалийг олохын тулд (N-1)-ийн факториалийг олох шаардлагатай байна. Харин (N-1)-ийн факториалийг олохын тулд (N-2)-ийн факториалийг олох хэрэгтэй гэх мэтчилэн

үргэлжилнэ. Харин хамгийн эцсийн шатанд 1-ийн факториалийг олох буюу энэ нь 1 байна. Энэ бодлогын алхмуудыг дараах хүснэгтээр харуулъя.

Бодлого	Дэд бодлого
5!	5*4!
4!	4*3!
3!	3*2!
2!	2*1!
1!	1

Хамгийн сүүлийн алхамд бодлого энгийн болж, үр дүн гаргасан байна. Ингээд бодолт эх бодлоготоо шилжих ёстой юм.

Бодлого	Үр дүн
1!	1
2!	2*1! = 2*1 = 2
3!	3*2! = 3*2 = 6
4!	4*3! = 4*6 = 24
5!	5*4! = 5*24 = 120

Ингээд факториал тооцож гаргах жишээ программаа үзэцгээе.

```
void main()
{
        int num;

        cout << “Тоо оруулна уу : ”д
```

```
        cin >> num;

        if (num<0)
                cout << “Та натурал тоо оруулах ёстой!”;
        else
                fact(num);
}

int fact(num)
        {

        if (num<=1)
                return (1);
        else
                return (num*fact(num-1));

        }
```

Программ ажиллаж байх явцыг доорх хүснэгтээр үзүүлэв.

Эх функц	Дуудагдсан дэд функц	Дэд функцэд өгөх аргумент	Дэд функцээс буцах үр дүн
main	fact (1-р удаа)	5	5 * fact(4)
fact (1-р удаа)	fact (2-р удаа)	4	4 * fact(3)
fact (2-р удаа)	fact (3-р удаа)	3	3 * fact(2)
fact (3-р	fact (4-р	2	2 * fact(1)

удаа)	удаа)		
fact (4-р удаа)	fact (5-р удаа)	1	1

Эхний удаад main функц fact функцийг 5 гэсэн аргументттэй дуудна. Функц аргументийг шалгахад 1-ээс их учраас функц 5*fact(4) гэсэн үр дүн буцаах болно. Энэ үр дүнг буцаахын тулд fact функцийг 4 аргументтэй дуудна. Шинэ дуудагдсан функц аргументийг шалгахад мөн 1-ээс байна. Уг функц 4*fact(3) гэсэн утга буцаахын тулд fact функцийг 3 аргументтэй дуудна. Үүнчлэн явсаар fact функц 1 аргументтэй дуудагдах бөгөөд харин энэ удаад дахиж функц дуудалгүйгээр 1 гэсэн тоон утгыг 4 дэх удаа дуудагдсан fact функцэд буцаах юм. Харин тэр нь 2*1 буюу 2 гэсэн утгыг 3 дах fact функцэд, тэр нь 3*2=6 гэсэн утгыг 2 дахь fact функцэд, тэр нь 4*6=24 утгыг 1 дэх fact функцэд, харин эндээс 5*24=120 гэсэн утга үндсэн main функцэд хүрч ирэх болно.

Программыг рекурсив маягаар зохион байгуулахад дараах хоёр зүйлийг сайтар анхаарах хэрэгтэй. Нэгдүгээрт, рекурсивээр ажиллах функц дотор өөрийгөө рекурсив дуудахгүй нэг нөхцөл заавал байх шаардлагатай. Эс тэгвээс функц тасралтгүйгээр өөрийгөө рекурсив дуудаж, программ тодорхой үр дүн өгөлгүй гацахад хүрнэ. Бидний жишээнд энэ нөхцөл нь (num<=1) юм. Энэ нөхцөл биелэсэн тохиолдолд функц дахиж өөрийгөө дуудалгүй, шууд үр дүн буцаах болно. Дараагийн нэг анхаарах зүйл бол функцийг рекурсээр дуудах үед түүнд өгч байгаа аргумент нь өөрчлөгдөж байх ёстой. Жишээ нь бид өмнөх программд доорх алдааг гаргасан бол функц мөн л тасралтгүй рекурс болж хувирна.

```
if (num<=1)
        return (1);
else
        return (num*fact(num));     // АЛДАА!!!
```

Учир нь функцэд ирж байгаа аргумент нь үргэлж тогтмол байгаа учраас (num<=1) нөхцөл хэзээ ч биелэхгүй.

#DEFINE директивийн тухай

Энэ удаа C++-д тогтмол болон макрог хэрхэн тодорхойлох тухай товч тайлбарлах гэж байна. Тогтмолыг тодорхойлохын ач тусыг доорх нэгэн жишээн дээр тайлбарлая.

C++ дээр программ бичиж байх үед эхлэн суралцагсдын гаргадаг нийтлэг нэгэн алдаа бол = ба == үйлдлүүдийг андуурах явдал юм. Хэдийгээр = бол утга оноох үйлдэл, харин == бол хоёр утгын тэнцүү эсэхийг шалгадаг Булийн үйлдэл гэдгийг мэдэж байвч санамсаргүйгээр энэ үйлдлүүдийг сольсноор программыг үл ойлгогдох буруу үйлдэл хийхэд хүргэдэг. Тэгвэл энэ асуудлыг доорх байдлаар шийдэж болно.

```
#define EQUALS       ==
#define GETS_VALUE_OF  =
```

Ингэж тогтмол тодорхойлсон цагт бид программаа доорх байдлаар өөрчилж болно гэсэн үг.

```
x=y
```

гэхийн оронд

```
x GETS_VALUE_OF y;
```

мөн

```
if (x==y)
```

гэхийн оронд

```
if (x EQUALS y)
```

гэж болох юм.

Тэмдэгтэн тогтмолыг тодорхойлохын тулд бид номын эхний бүлэгт товч дурдсан #define гэсэн нөөц үгийг ашиглаж байна.

```
#define <тогтмолын нэр> <тогтмолын утга>
```

Нөөц үгийг заавал мөрийн эхлэлээс бичих ёстойг үргэлж санаж байх хэрэгтэй. Хэрэв тогтмол тодорхойлж буй мөр дэндүү урт болвол түүнийг хэд хэдэн мөрөнд хувааж болох бөгөөд ингэхдээ тасарсан мөрийн төгсгөлд ‘ \ ’ тэмдгийг тавьж өгөх хэрэгтэй. Ө.х. энэ тэмдэг нь энэ мөрийн үргэлжлэл дараагийн мөрөнд байгаа гэж хэлж өгч байгаа юм.

```
#define GETS_VALUE_OF  \
                       =
```

Тогтмолын нэрийг ерөнхийдөө том үсэгнүүдээр өгдөг. Хэдийгээр заавал ингэх шаардлагагүй боловч ингэж тодорхойлох нь тогтмол утгыг хувьсагчуудаас ялгаж харах боломжтой болгодог юм.

Тэмдэгтэн тогтмол нь уг файл дотроо түүнээс хойш тодорхойлогдсон бүх функцийн хувьд хүчинтэй

байна. Нэг #define-ийг ашиглан нэг л тогтмолыг тодорхойлж өгнө. Нэг тогтмол нь өөр тогтмолын утга болж болно.

```
#define EQUALS   ==
#define EQ       EQUALS
```

Ингэснээр нэг утгагтай хоёр тогтмол тодорхойлогдож байна. Тэмдэгтэн тогтмолын нэг адил тоон тогтмолыг тодорхойлж болно.

```
#define NINE     9
```

#define бол ямар нэгэн биелэх комманд биш бөгөөд энэ нь зөвхөн программ зохиогчийн ажиллагааг хялбарчлах зориулалттай юм. Компилятор программыг хөрвүүлэх үед тогтмол утга бүрийг түүний жинхэнэ утгаар орлуулж тавьдаг бөгөөд ө.х. программ хөрвүүлэгдээд, биелэх файл үүссэн хойно тогтмол утгууд огт байхгүй болдог юм. Жишээ нь, дээрх мөрүүдийг агуулсан программ хөрвөх үед EQUALS гэсэн үг бүрийн оронд == тэмдэг солигдон бичигддэг байна.

Харин одоо #define директивийн өөр нэгэн чухал үүргийг авч үзье. Энэхүү нөөц үгийг ашиглан бид макро гэж нэрлэгдэх жижиг хэмжээний бүлэг үйлдлүүд тодорхойлж болдог юм. Бидний программд “Дурын товч дарна уу” гэсэн мөрийг хэвлэх үйлдэл олон дахин давтагддаг байна гэж саная. Тэгвэл бид тэр үйлдлийг макро болгон тодорхойлж болох юм.

```
#define PRESS_KEY \
    cout << “Дурын товч дарна уу”
```

Харин макрогоо программ дотроос доорх байдлаар шууд дуудаж ажиллуулна.

```
void main()
{

        . . . . .

        PRESS_KEY;

        . . . . .

}
```

Макро тодорхойлох нь бичлэгийн тэмдэгтэн тогтмол тодорхойлохтой ижил бөгөөд өөрөөр хэлбэл

```
#define <макро нэр> <бүлэг үйлдлүүд>
```

Урт мөрийг хуваахдаа ‘ \ ‘ тэмдгийг ашиглана. Макро нь мөн тэмдэгтэн тогтмолын адил хөрвүүлэх үед жинхэнэ утгаараа солигдон бичигдэнэ. Мөн макрогийн нэрийг том үсгүүдээр бичих нь программ дотроос ялгаж харах боломжийг өгдөг. Макро нь уг файл дотроо түүнээс хойш тодорхойлогдсон бүх функцэд хүчинтэй байна.

С++-ийн макро нь аргумент авах боломжтой байдаг. Функцийн аргументтай нэгэн адилаар макрог тодорхойлохдоо аргументийг тодорхойлж өгнө. Жишээ нь, бичиж буй программ нь хүний нэр болон насыг дахин дахин хэвлэх шаардлагатай байж болно.

```
#define PRINT(name, age) \
    cout << name << ‘-’ << age

void main()
{

        . . . . .

        PRINT(‘Амарбат’, 25);
```

```
        PRINT('Отгонбаяр', 28);

        . . . . .

}
```

Нэмэлт болгон энд нэг тун сонин жишээ үзүүлье.

```
#define MUL2(num)  2*num

void main()
{
        int a1,a2;
        a1=MUL2(5);
        a2=MUL2(2+3);
}
```

Энэ программ ажиллахад a1-ийн утга 10 гарах нь ойлгомжтой, Харин a2-ийн утга мөн 10 гарах мэт санагдаж байгаа боловч үнэндээ бол 7 гарна. Учир нь 2*2+3=7 шүү дээ. Өмнө хэлсэн ёсоор программ хөрвөх үед MUL2(2+3) гэсний оронд 2*2+3 гэсэн бичлэг орж ирнэ шүү дээ. Нэгэнт үржих үйлдэл нэмэх үйлдлээс өндөр эрэмбэтэй тул шууд түрүүлж биелэнэ. Ийм учраас 7 гэсэн хариу гарч байгаа юм. Харин бид иймэрхүү бэрхшээлээс гарахын тулд макрогоо доорх байдлаар тодорхойлох боломжтой.

```
#define MUL2(num)      2 * (num)
```

Ингэсэн тохиолдолд программ хөрвүүлэх үед 2*2+3 биш, харин 2*(2+3) гэсэн комманд болж, 10 гэсэн үр дүн буцаана.

Бүлгийн дүгнэлт

Функц гэдэг нь программыг боловсронгуй, ойлгомжтой болгох, түүний хэмжээг багасгах зориулалттай нэр бүхий бүлэг үйлдэл юм. Функцийн урьдчилсан тодорхойлолт нь уг функц ямар ямар мэдээллийг дамжуулан авах шаардлагатайг харуулсан бичиглэл юм.

Аргумент нь утгаараа дамжиж болно. Энэ тохиолдолд функц аргументэд ирсэн хувьсагчтай биш, түүний утгыг хувилж авсан өөрийн хувьсагчидтайгаа ажилладаг юм. Аргумент мөн хуурамч нэрээр дамжиж болно. Харин ингэсэн үед, функц жинхэнэ аргументэд ирсэн хувьсагчтай өөртэй нь ажиллах болно.

Функц нь хэд хэдэн аргумент авч болох боловч цорын ганцхан утга буцаах боломжтой. Утгыг мөн хуурамч нэрээр дамжуулж болно. Аргумент болон буцах утгуудын аль аль нь энгийн стандарт төрөл төдийгүй нийлмэл бүтцүүд хүртэл байж болдог.

Давхардсан функцүүд гэдэг нь ижил нэртэй бүлэг функцүүд юм. Функцийг дуудахад өгсөн аргументүүдийн төрөл болон тооноос хамааран давхардсан функцүүдийн аль нэг нь ажилладаг.

Программыг хөрвүүлэх үед функцийг дуудах коммандын оронд бүтэн бие нь тэр чигээрээ бичигддэг функцийг дотоод функц гэж нэрлэдэг. Функцийг дотоод болгон зохион байгуулах нь программын хурдыг нэмэгдүүлж болох боловч программын хэмжээг нэмэгдүүлж, ингэснээрээ диск болон санах ойд их хэмжээний зай эзлэхэд хүрч болзошгүй.

Хэрэв функцийн аргументүүдэд стандарт утгуудыг тодорхойлж өгсөн бол функцийг дуудах үед тийм аргументийн утгыг өгөхгүй байж болно.

Хувьсагчдын хамгийн өргөн ашиглагддаг хэлбэр бол автомат юм. Ийм хувьсагч нь зөвхөн

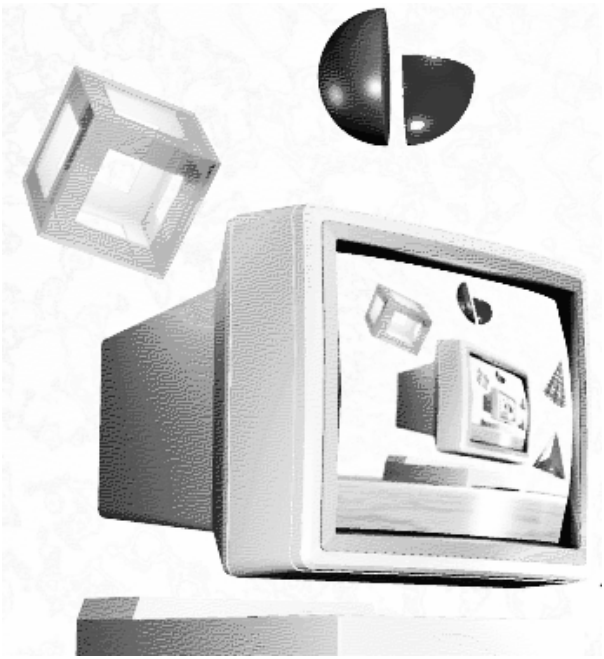
тодорхойлогдсон функц дотроо л ашиглагдах бөгөөд уг функцийг дуудагдах үед үүсч, дуусах үед нь автоматаар санах ойгоос арчигддаг байна. Харин глобаль хувьсагчууд бүхэл программын туршид санах ойд байрлах бөгөөд программд байгаа бүх функцэд ашиглагдах боломжтой. Харин статик автомат хувьсагчид нь мөн программын ажиллах туршид санах ойд байх боловч зөвхөн өөрийн функц дотроос л ашиглагдаж болдог.

Функц нь санах ойд үүсэх байдлаараа үндсэндээ хувьсагчтай адил байдаг бөгөөд хувьсагчийн нэгэн адил функцэд заагч үүсгэж болдог. Функцийг заагчийг ашиглан функцийн аргументэд функц дамжуулах боломжтой болно.

Эх бодлоготойгоо адил чанарын дэд бодлогууд шийдэхийн тулд функцийг рекурсивээр зохион байгуулах хэрэгтэй. Рекурсив функц зохион байгуулж байгаа тохиолдолд анхааралгүй хандваас функц тасралтгүй рекурсэд орж, программ гацахад хүрч болох аюултайг санаж байх хэрэгтэй.

Жижиг хэмжээний бүлэг үйлдлүүдийг функц болголгүйгээр макро болгон ашиглаж болно. Макрог #define гэсэн нөөц үгийн тусламжтайгаар тодорхойлдог. Макро нь энгийн функцээс ялгагдах гол зүйл нь биелэх программд макро чигээрээ ордоггүй явдал юм. Өөрөөр хэлбэл, программ хөрвөх үед макро дуудагдаж байгаа газар бүрт түүний жинхэнэ үйлдлүүд шууд солигдон бичигддэг юм.

5-р бүлэг



-Обьектийн тухай ойлголт
-Оператор тодорхойлох
-Суурь ба үүссэн анги
-Өгөгдлийн төрөл хөрвүүлэх
-Виртуаль ба найз ангиуд
-this заагчийн тухай

5-р бүлэг

Энэ бүлэгтээ бид орчин цагийн программчлалын хамгийн хүчирхэг хэрэгсэл болох обьект хандалтат программчлалын тухай үзэх гэж байна.

Обьект Хандалтат Программчлалыг (Object Oriented Programming) агуулсан С хэлийг С++ гэж анх нэрлэсэн ба энэ нь анх 1983 онд New Jersey хотын Murray Hill их сургуулийн Bell Laboratories-д хийгдсэн байна. Түүнээс хойш одоо энэ хэл нь маш хүчирхэг хэл болж хөгжсөн байна. С++ хэлд проект үүсгэн олон дэд программуудыг зохион байгуулан тэдгээрийг бие биед нь хэрэглэх гэх мэтээр ямар ч том хэмжээний программыг үүсгэн хөгжүүлэх боломжтой байдаг.

Өмнө үеийн С хэл болон бусад ихэнх дээд төвшний хэлүүд программ бичихдээ загварчилан программчилж байгаа обьектийн бүх мэдээллийг тодорхойлдог. Жишээ нь: Их сургуулийн үйл ажиллагааг загварчилан программчлахын тулд багш, ажилтан, оюутан, өрөө, тасалгаа, сургалтын төлөвлөгөө, хичээлийн хуваарь, гэх мэт мэдээллүүдийг тодорхойлно.

Ингээд энэ обьектод хийгдэх бүх үйлдлүүд, процедуруудыг бичих ёстой. Жишээ нь: Элсэх оюутныг бүртгэх, элсүүлж сургах, шалгалт шүүлэг авах, хичээллэх гэх мэт бүх үйлдлийн процедуруудийг бичих ёстой.

Эцэст программын зохиогчийн хийх ажил нь тодорхойлсон мэдээллүүдийг загварчлах, бүх процессуудыг программчлах явдал байдаг.

Ийм ч учраас программчлалын энэ аргыг процедур хандлагат программчлал (procedure oriented programming) гэдэг. Бас программчлал үүсэж хөгжсөнөөс хойш 40 гаруй жил хэрэглэгдэж байсан учир уламжлалт арга гэж мөн хэлдэг. Энэ аргыг баримталсан программчлалын хэлүүдийг процедур хандлагат хэл гэж нэрлэдэг.

Обьектийн тухай ерөнхий ойлголт

Өмнө дурдсан уламжлалт аргын хувьд программын мэдээлэл болон код хоёр нь хоёр тусдаа байдаг. Өөрөөр хэлбэл нягт холбоотой биш байдаг. Иймд зөв мэдээллийг буруу

үйлдлээр боловсруулах, эсвэл зөв программаар буруу мэдээллийг боловсруулах аюул байнга гарч байдаг. Тэгэхээр зөв программаар зөв мэдээллийг боловсруулах үүргийг программ зохиогч өөрөө дааж хариуцаж байдаг. Процедур хандлагат программчлалын хэлний хөрвүүлэгч (compiler) нь сайндаа л тохирохгүй байгаа хувьсагчийг хэлдэг.

Дэлхий дээр байгаа дурын обьектийн хувьд мэдээлэл болоод обьектийн үйлдлүүд нь хамт байсан цагт л сая төгс төгөлдөр бүтэн обьектийг үүсгэж байдаг. Жишээ нь: Их сургуулийн багш, оюутан, хичээлийн хуваарь гэх мэт мэдээлэл нь элсэлт, сургалт, төгсөлт гэх мэт үйлдлүүдтэйгээ салшгүй холбоотой. Иймд обьектийн мэдээлэл болон үйлдлийг хамтад нь тодорхойлж программчлах ёстой гэсэн санаа анх 1960-аад оны дундуур гарч ирсэн. Тэгэхээр обьект бол мэдээлэл ба үйлдлүүдийн нэгдэл гэж тодорхойлж программчлах аргыг обьект хандалтат программчлал (ОХП буюу Object Oriented Programming-OOP) гэж нэрлэдэг. 1967 оны Simula-67 хэлийг анх зохиосон боловч өргөн хэрэглэгдэж чадаагүй. Энэ нь үндсэндээ хүмүүс өөрийн уламжлалт аргаасаа салах дургүй байсантай холбоотой. 1980-аад оны эхэнд ОХП-ын Smalltalk хэл зохиож хэрэглэж эхэлснээр программчлалын энэ чиглэл эримтэй хөгжиж ирсэн. Үүний C, Pascal хэлүүдэд ОХП-ын аргыг оруулсан C++, Pascal 5.5, Object Pascal хэлүүд бий болсон байна.

Дэлхий дээр олон обьектууд байдаг боловч эдгээр нь ижил анги (Class) төрлийн обьектууд олон байдаг. Жишээлбэл шоргоолж гэдэг амьтныг авч үзвэл маш олон тооны шоргоолж дэлхий дээр байгаа. Тэгвэл шоргоолж гэсэн энэ төрөл ангийг шоргоолжийн анги гэж нэрлээд ямар нэг тодорхой нэгэн шоргоолжийг энэ ангийн нэг обьект гэж нэрлэдэг.

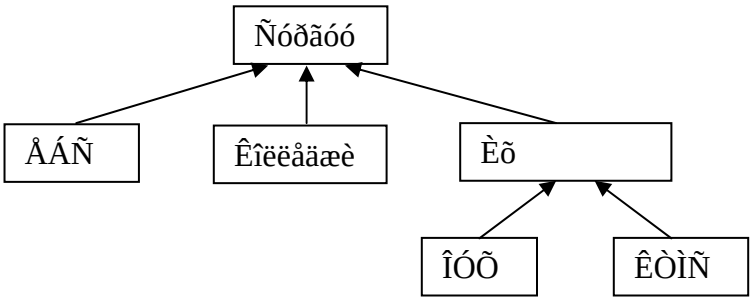
Дэлхий дээр байгаа обьектын ангиуд бусад ангиудтайгаа салшгүй холбоотой байдаг. Жишээ нь: Шавьжийг авч үзье. Энэ обьект нь дотроо далавчтай, далавчгүй гэсэн 2 бүлэгт хуваагдана. Тэгвэл далавчтай шавьж гэсэн анги нь шавьж болохынхоо хувьд шавьжын бүх шинжийг хадгалсан байна.

Иймд цаашид обьект, обьектийн анги гэж ярих нь төстэй боловч ялгаатай ойлголтууд болохыг анхаарах хэрэгтэй.

Ургийн мод

Нэг обьект нь өөр нэг обьектоос үүсч болох бөгөөд энэ обьект нь үүссэн эх обьектынхоо бүх шинжийг агуулсан байх шинжийг удамших шинж гэнэ. (inheritance - удамших)

Бид цаашид үүссэн обьектыг удамшсан обьект буюу child обьект гэнэ. Эх обьектыг parent гэнэ. Обьектуудын удамшлыг харуулсан схем буюу мод маягийн зургийг **ургийн мод** гэнэ. Хэрэв ямар нэг ургийн мод зохиогдож байвал түүнийг хялбархан



өргөтгөх боломжтой байдаг. Жишээ нь : Сургуулиудын хувьд ургийн модыг авч үзье.

Тэгвэл ямар нэг шинэ их сургууль үүссэн тохиолдолд түүнийг тохирох байранд нь тавих замаар өргөтгөж болно. Программчлалд бол обьект гэдэг нь өгөгдөл түүнийг боловсруулах (code) функцүүдийн битүүмжилсэн нэгдэл юм. Энэ битүү обьект дотор код нь мэдээллийнхээ хэмжээ хэлбэрийг удирдан боловсруулах, мэдээлэл нь кодын урсгал буюу үйлдлийг тодорхойлно. Ийм битүү обьектын хувьд ямар нэг мэдээлэл авахын тулд тэр мэдээлэлд шууд хандахгүй, харин тэр мэдээллийг өгдөг функц үйлдлийг ашиглана. Харин обьектийн ямар нэг мэдээллийг өөрчлөхийн тулд мөн л обьектийнхоо харгалзах функцийг ашиглана.

Энгийн анги

Дэлхий дээр олон тооны обьект байдаг боловч ижил обьектууд олон байдаг. Тухайлбал нэг ангийн хоёр обьект бүхий дараах энгийн программыг үзье.

```
#include <iostream.h>

class smallobj
        {
        private:
        int somedata;
        public:
        void setdata(int d)
                { somedata=d; }
        void showdata()
                { cout << "\nӨгөгдөл нь " << somedata; }
        };
void main()
{
smallobj s1, s2;
s1.setdata(1066);
s2.setdata(1776);

s1.showdata();
s2.showdata();
}
```

Энд SmallObj программ нь 1 гишүүн (member) өгөгдөл хоёр гишүүн функцтэй байна. Функцүүд нь гадаад ертөнцөөс өгөгдөл рүү хандахад зориулагджээ. Ангийг тодорхойлохдоо class гэсэн түлхүүр үг, араас нь ангийн нэр (smallobj), ангийн бие гэсэн байдлаар тодорхойлно.

Private түлхүүр үг нь гишүүн өгөгдөл болон функцийг обьектийн гаднаас хандахыг хориглож гадаад ертөнцөөс нууцалдаг. Өөрөөр хэлбэл обьектийн хувийн өгөгдөл, функцийг тодорхойлдог.

Обьектын гишүүн функцыг дуудахдаа заавал обьектын нэрийг бичих ёстой. Тухайлбал дээрх жишээнд:

```
s1.setdata(1066);
s2.setdata(1776);
```

гэж дуудсан байна.

Байгуулагч функц (Constructors)

Дээрх жишээнд программын өгөгдөлд setdata функцийн тусламжтай утга оноож байна. Гэтэл заримдаа утгыг оноохоосоо өмнө утга авах дүрмийг ашиглах тохиолдол гарч болдог. Ийм байдлаас зайлсхийхийн тулд обьектийг анх үүсэхэд нь инициализаци хийх буюу анхны утгыг оноож өгөх боломж байдаг. Ингэж инициализаци хийдэг тусгай дүрмийг байгуулагч функц гэдэг. Энэ функц нь обьектийг үүсгэхэд автоматаар дуудагддаг юм.

Бид жишээ болгон тоолуурын программыг үзье.

```
class Counter
        {
        private:
        unsigned int count;

        public:
        Counter()   {count = 0;}
        void inc_count()
        int get_count()
        };

. . .

Counter c1;
cout << c1.get_count;
c1.inc_count();
cout << c1.get_count;
```

Байгуулагч функц нь ангийн нэртэйгээ ижил байх ёстой ба үр дүн буцаадаггүй байх ёстой. Учир нь компилятор байгуулагч функцийг нэрээр нь таньдаг. Байгуулагч нь программаас автомат дуудагддаг учраас өмнө дурдагдсан шинжүүдийг заавал хангадаг байх ёстой.

Устгагч функц (Destructors)

Харин устгагч функц нь обьектийг устгахад автоматаар дуудагддаг дүрэм юм. Устгагч функц нь мөн ангийн нэртэйгээ

ижил байх ёстой ба зөвхөн өмнөх '~' тэмдгээрээ л байгуулагчаас ялгагдана. Жишээлбэл:

```
class Counter
        {
        private:
        unsigned int count;

        public:
        Counter()
                {count = 0;}
        ~Counter() { }
        void inc_count()
        int get_count()
                {return count;}
        };
```

Устгагч функц нь аргумент авдаггүй, утга буцаадаггүй байх ёстой. Энэ функц нь динамик обьектийг санах ойгоос чөлөөлөх үндсэн үүргийг гүйцэтгэдэг.

Обьектийг функцийн аргументад хэрэглэх нь

Бид жишээ болгож уртын хэмжээг нэмэх программ зохиоё.

```
#include <iostream.h>
class Distance
        {
        private:
        int feet;
        float inches;
        public:
        Distance() { }
        Distance(int ft, float in)
                { feet = ft; inches = in; }
        void getdist()
                {
                cout << "\nФут : ";
                cin >> feet;
```

```
                cout << "\nИнч : ";
                cin >> inches;
                }
        void showdist()
                {cout << feet << "\'-"
                     << inches << '\"';}
        void add_dist(Distance, Distance);
        };

void Distance::add_dist(Distance d2, Distance d3)
        {
        inches = d2.inches + d3.inches;
        feet = 0;
        if (inches >= 12.0)
                {
                inches -= 12.0;
                feet++;
                }
        feet +=d2.feet + d3.feet;
        }
void main()
{
Distance dist1, dist3;
Distance dist2(11, 6.25);

dist1.getdist();
dist3.add_dist(dist1, dist2);

cout << "\ndist1 = "; dist1.showdist();
cout << "\ndist2 = "; dist2.showdist();
cout << "\ndist3 = "; dist3.showdist();
}
```

Дээрх программд хоёр конструктор байгаа ба эдгээр нь аргументаараа ялгарч байна. Энэ тохиолдолд хөрвүүлэгч өгөгдсөн параметрээс хамааруч автоматоор таньж харгалзах конструкторыг ажиллуулдаг. Түүнээс гадна гишүүн функцийг ангийн гадна талд тодорхойлж болж байна. Ингэж тодорхойлохдоо функцийн толгойг ангийн нэр, функцийн нэрийг (::-р тусгаарлан) хамт бичсэнийг анхаарах хэрэгтэй.

```
void Distance::add_dist(Distance d2, Distance d3)
```

Дээрх жишээнд обьектийг функцийн аргументэд өгөх тухай үзлээ. Энэ нь обьект гэдэг нь хувьсагч гэдгийг харуулж байгаа хэрэг юм. Иймд обьектийн хувьд утга олгох үйлдлийг хэрэглэж болно. Жишээлбэл:

```
dist3 = dist2
```

Мөн түүнчлэн обьект нь функцийн үр дүн байж ч болно. Тухайлбал дээрх жишээн дэх функцыг дараах байдалтай болгож өөрчилж болно.

```
Distance add_dist(Distance)
```

Ангийн статик өгөгдөл

Нэг ангийн олон обьект үүсгэхэд тэдгээр объект тус бүр өөрийн гишүүдийг санах ойд үүсгэж байдаг. Өөрөөр хэлбэл тухайн анги хэдэн обьекттой байна түүний тоогоор мэдээллүүд нь санах байрладаг. Харин ангийн ямар нэг гишүүн өгөгдлийг static гэж үүсгэвэл тэр нь санах ойд зөвхөн ганц байрлаж бүх обьектуудад ерөнхий хэрэглэгдэж байдаг. Гэхдээ статик өгөгдөлийг зөвхөн обьект дотроос л ашиглах боломжтой. Иймд жишээлбэл статик өгөгдлийг ашиглан тухайн ангид хэдэн обьект байгааг тоолж болох юм. Үүнийг доорх жишээнд харуулъя.

```
Class foo
        {
        private:
        static int count;
        public:
        foo() {count++}
        int getcount() {return count;}
        };
```

Ангийн статик өгөгдөл нь өргөн хэрэглэгддэггүй боловч маш чухал хэрэглэх тохиолдол олонтоо гардаг.

Оператор тодорхойлох

Оператор тодорхойлно гэдэг нь ОХП-ын нэгэн чухал хэсэг юм. Оператор тодорхойлох нь уншихад төвөгтэй их бичлэгтэй программыг ойлгоход хялбар болгох бололцоогоор хангаж өгдөг.

Жишээлбэл:

```
dist3.add_dist(dist1, dist2);
```

гэсэн уртыг нэмэх анги бид дээр тодорхойлж байсан. Харин үүнийг

```
dist3 = dist1 + dist2;
```

гэж бичвэл унших ойлгох, бичихэд маш хялбар байна. Гэвч энд байгаа обьектууд нь хэрэглэгчийн тодорхойлсон төрлийнх тул ингэж бичвэл хөрвүүлэлтийн алдаа гарах болно.

Тэгвэл хэрэглэгчийн төрлүүдэд C++ хэлний +, *, <=, += гэх мэтийн энгийн операторуудыг тодорхойлон хэрэглэх боломж байдаг. Өөрөөр хэлбэл оператор тодорхойлох замаар C++ хэлийг өөрийн дизайнаар өргөтгөх боломжтой юм.

Унар оператор тодорхойлох

Бид өмнө тоолуурын анги тодорхойлсон жишээ үзсэн ба энд inc_count гэсэн гишүүн функц тодорхойлж байсан билээ.

```
Counter c1;
c1.inc_count();
```

Уг нь дээрх бичлэгийг ++c1 гэж бичвэл илүү ойлгомжтой бөгөөд тохиромжтой билээ. Үүний тулд Counter ангийг дараах байдлаар тодорхойлох хэрэгтэй болно.

```
class Counter
        {
        private:
        unsigned int count;
        public:
        Counter() {count = 0;}
```

```
~Counter() { }
void inc_count()
int get_count() {return count;}
void operator ++() {count++;}
};
```

Дээрх жишээнд operator түлхүүр үгийн тусламжтайгаар тухайн ангийн хувьд ++ операторыг тодорхойлж байна. Ийнхүү тодорхойлсны дараа

```
c1++; ++c1;
```

гэж бичих бололцоотой болно. Эцэст нь хэлэхэд тодорхойлогдсон оператор бол ангийн гишүүн функц бөгөөд түүнийг дуудахдаа энгийн оператортой адил дуудан ашиглах бололцоотой байдаг.

Гэхдээ энэ операторыг тодорхойлсны дараа дараах бичлэгийг хэрэглэвэл алдаа өгөх болно.

```
c1 = c2++;
```

Учир нь манай тохиолдолд оператор маань void үр дүнтэй тодорхойлогдсон билээ. Тэгвэл операторын үр дүнг тодорхой зааж өгөх боломж бас байдаг. Жишээлбэл:

```
Counter operator ++()
        {
        count++;
        counter temp;
        temp.count = count;
        return temp;
        };
```

Энэ жишээнд temp гэсэн түр ашиглагдах хувьсагч ашиглажээ. Тэгвэл үүнээс дараах байдлаар зайлж болох юм. Үүний тулд эхлээд дараах хоёр байгуулагч функцийг тодорхойлох хэрэгтэй.

```
Counter() { count = 0;}
Counter(int c) {count = c;}
```

Үүний дараа дараах байдлаар операторыг тодорхйолж болно.

```
Counter operator ++ ()
        {
        count++;
        return Counter(count);
        }
```

Энэ жишээнд

```
Counter(int c) {count = c;}
```

гэсэн байгуулагч функцийг тодорхойлоогүй тохиолдолд алдаа гарна гэдгийг анхаарах хэрэгтэй.

Бинар оператор тодорхойлох

Бинар оператор тодорхойлох нь унар оператортой төстэй байна. Бид үүнийг арифметик оператор, жиших, утга олгох гэсэн тохиолдлуудад үзье.

Арифметик оператор

Бид өмнө уртын хэмжээг нэмэх жишээ программ үзсэн билээ. Энэ программд

```
dist3.add_dist(dist1, dist2);
```

гэсэн үйлдлийн тусламжтайгаар уртын хэмжээнүүдийг нэмж байсан ба үүнийг + операторыг тодорхойлох замаар дараах байдлаар энэ үйлдлийг хийх боломжтой болно.

```
dist3 = dist1 + dist2;
```

Үүний тулд add_dist функцийн оронд + операторыг дараах байдлаар тодорхойлж болно.

```
class Distance
        {
        . . .
        Distance operator + (Distance);
```

```
        . . .
        };

Distance Distance::operator + (Distance d2)
        {
        int f=feet+d2.feet;
        float i=inches+d2.inches;
        if (i>=12.0)
                {
                i-=12.0;
                f++;
                }
        return Distance(f, i);
        }
```

Ингэж тодорхойлсны дараа дээр ярьсан + үйлдлийг хийж болно.

```
dist3=dist1+dist2;
```

Энд dist1 обьектийн оператор гишүүн функц дуудагдаж dist2 аргумент нь болдог. Нэмэх үйлдлийг 2 операндтай гүйцэтгэж болохоос гадна дараах байдлаар хэрэглэж ч болно.

```
dist4=dist1+dist2+dist3;
```

Жишэх оператор

Бид өмнө тодорхойлсон уртын хэмжээг жиших программ зохиоё. Үүний тулд ангийн бие дотор доорх маягийн нэмэлт оруулахад хангалттай.

```
. . .
boolean operator < (Distance);
. . .
boolean Distance::operator < (Distance d2)
{ . . . }
```

Өгөгдлийн төрөл хөрвүүлэх

Бид дараах утга олгох үйлдэл хөрвүүлэх үед алдаа өгөхийг мэднэ.

```
float f;
int t;
t=f;
```

Харин дээрх утга олгох үйлдлийг төрлийг нь хөрвүүлэх байдлаар дараах утга олголтыг хийвэл алдаа өгөхгүй.

```
t = int(f);
```

Үүнтэй ижил байдлаар өөр ангиудын хувьд болоод анги үндсэн төрлүүдийн хооронд утга олгох үйлдэл хийх үед төрөл хөрвүүлэлт яаж хийгдэхийг заах боломж байдаг.

Анги, үндсэн төрөлийг хооронд нь хөрвүүлэх

Бидний жишээ болгон өмнө хэрэглэсэн уртын хэмжээ анги нь Америк уртын хэмжээ билээ. Тэгвэл үүнийг метр системийн уртын нэгж болгон бодит утга руу хөрвүүлэх жишээ үзье.

Энэ хөрвүүлэлтийг хийхийн тулд дараах операторийг ангийн бие дотор хөрвүүлэх ёстой.

```
. . .
operator float()
          {
          return (feet*12+inches)*2.54
          }
 . . .
float mtrs=float(dist2);
```

Дээрхийн эсрэг бодит утгыг энэ анги руу хөрвүүлэх жишээ үзье. Үүний тулд бодит аргумент бүхий байгуулагч функцийг тодорхойлох ёстой байдаг.

```
Distance(float meters)
          {
          float fltfeet=meters*100/2.54/12;
          feet=int(fltfeet);
          inches=12*(fltfeet–feet);
```

```
        }
. . .
Distance dist1 = 2.35;
. . .
dist1 = 1.0;
```

Ялгаатай ангиудын обьектыг хөрвүүлэх

Бид үндсэн төрөл болон хэрэглэгчийн тодорхойлсон анги хоёрын хооронд хөрвүүлэх тухай үзлээ. Одоо хэрэглэгчийн тодорхойлсон ангиудыг хооронд нь хөрвүүлэх тухай үзье. Үүний тулд 1 аргумент байгуулагч функц, хөрвүүлэгч функц хоёрыг хэрэглэх ёстой.

Бид дараах утга олголтыг хийхийг зорьж байна гэж саная.

```
objecta=objectb;
```

Энд тохиолдолд objecta-г destination, objectb-г source обьект гэж нэрлэнэ. Тэгвэл энэ тохиолдолд objecta-ийн ангид хөрвүүлэх функц, objectb-ийн ангид нэг аргументтай байгуулагч функц тодорхойлогдох ёстой.

Суурь анги болон үүссэн анги

Удамших шинж бол ОХП-н маш чухал давуу тал бөгөөд программын өргөтгөх сайхан боломжийг олгодог. ОХП-д тодорхойлогдсон байгаа ямар нэг суурь ангиас (base class) өөр ангийг өргөтгөн үүсгэх (derived class) боломжтой. Ингэж өргөтгөх үед суурь обьектийн бүх гишүүд үүссэн обьектод удамшихаас гадна программ зохиогч өөрийн нэмэлт гишүүдийг тодорхойлж болно. Өөрөөр хэлбэл өмнө нь тодорхойлон зүгшрүүлсэн ангийг дахин өөрчлөх шаардлагагүй харин түүнээс үүссэн ангийг үүсгэж өөрийн өөрчлөлт өргөтгөлийн хийж болно гэсэн үг юм. Бид жишээ болгон өмнө үзсэн тоолуурын ангийг багасгаж тоолох (decrement --) боломжтой анги болгон тодорхойльё. Мөн түүнчлэн хэд хэдэн туслах ойлголтуудыг тусгасан дараах программыг авч үзье.

```
#include <iostream.h>
```

```
class Counter
        {
        protected:
        unsigned int count;

        public:
        Counter() { count = 0;}
        Counter(int c) {count = c;}
        int get_count() {return count;}
        Counter operator ++ ()
                {
                count++;
                return Counter(count);
                }
        };

class CountDn : public Counter
        {
        public:
        Counter operator -- ()
                {
                count--;
                return Counter(count);
                }
        };

void main()
{
CountDn c1;
cout << "\nc1=" << c1.get_count();
c1--; c1--;
cout << "\nc1=" << c1.get_count();
}
```

CountDn анги нь суурь анги болон тодорхойлогдсон гишүүдийг өөртөө агуулна. Өөрөөр хэлбэл дараах байдалтайгаар хэрэглэх боломжтой.

```
. . .
c1++;
. . .
c1--;
. . .
```

Дээрх жишээнд protected гэсэн түлхүүр үгийг хэрэглэжээ. Хамгаалагдсан (protected) гишүүн private гишүүн хоорондоо ерөнхийдөө адилхан юм. Хамгаалагдсан өгөгдөл нь гадаад ертөнцөөс нууцлагдсан байх боловч дараах ангиуддаа удамшдаг байна. Харин private гишүүн үүссэн ангидаа удамшидаггүй байна. Өөрөөр хэлбэл гишүүнийг гадаад ертөнцөөс нууцлах private, protected гэсэн 2 боломж байх боловч энэ нь обьектийн удамшилд ялгаатай нөлөө үзүүлдэг. Иймд дээрх жишээнд Counter ангид count гишүүнийг private гишүүн байсныг protected болгон өөрчилсөн байна.

Үүссэн ангийн байгуулагч функц

Дээрх жишээнд үүссэн ангийн хувьд суурь ангийн байгуулагч функцийг хэрэглэвэл алдаа өгөх болно. Харин суурь ангийн байгуулагч функцийг хэрэглэх тохиолдолд дараах байдлаар ангийг тодорхойлох ёстой.

```
class CountDn : public Counter
        {
        public:
        CountDn() : Counter()
                { }
        CountDn (int c) : Counter(c)
                { }
        Counter operator -- ()
                {
                count--;
                return Counter(count);
                }
  };
```

Дээрх хоёр тохиолдолд суурь ангийн байгуулагчийг дуудаж байна. Харин өөрийн нэмэлт инициализаци хийх кодыг өөрөө нэмж бичиж болох боловч манай тохиололд код байхгүй тул байгуулагчийн бие хоосон байна.

Дахин тодорхойлогдсон гишүүн функц

Үүссэн ангид суурь анги дахь функцийн нэртэй ижил нэртэй функцийг хэрэглэх боломжтой. Өөрөөр хэлбэл суурь анги болон үүссэн ангид нэг ижил гишүүн функцийг хэрэглэх боломжтой. Бидний өмнөх жишээн дэх уртын анги бол зөвхөн эерэг утгатай уртыг хадгалдаг анги юм. Тэгвэл энэ ангиас үүссэн сөрөг уртыг авдаг анги үүсгэе. Энд өмнөх анги дээр зөвхөн тэмдэг (sign) гэсэн гишүүн өгөгдөлийг л нэмье.

```
enum posneg{pos, neg};

class DistSign : public Distance
        {
        private:
        posneg sign;

        public:
        DistSign():Distance()
                { sign = pos; }
        DistSign(int ft, float in, posneg sg=pos)
                :Distance(ft, in)
                { sign = sg; }
        void getdist()
                {
                Distance::getdist();
                char ch;
                cout << "Тэмдгийг оруул(+ эсвэл -):";
                cin >> ch;
                sign=(ch=='+') ? pos:neg;
                }
        void showdist()
                {
                cout << ((sign == pos) ? "(+)" : "(-)");
                Distance::showdist();
                }
        };
```

Энэ жишээнд getdist, showdist гишүүн функцүүд нь эх ангийн мөн ижил нэртэй функцийг ашиглаж дуудсан байгаа дуудахдаа

```
Distance::getdist();
```

Distance::showdist();

гэж бичсэн байгааг анхаарах хэрэгтэй.

Манай жишээнд нэг ангиас нөгөө ангийг өргөтгөн үүсгэлээ. Түүнээс гадна нэг ангиас хэд хэдэн анги зэрэг үүсч болно. Заримдаа суурь эх анги нь дангаараа ашиглагдахгүй боловч түүнээс үүссэн ангиудыг ашиглах тохиолдол ч байдаг ба энэ тохиолдолд суурь ангийг abstract анги гэж нэрлэдэг.

Public ба Private удамшил

C++ хэлэнд суурь ангиас өөр ангийг хоёр янзаар үүсгэх боломжтой.

Манай жишээнд:

Distance : public DistSign

гэж үүсгэсэн билээ. Энд бичигдэж байгаа public түлхүүр үг нь эх ангийн public гишүүнүүдийг гадаад ертөнцөөс ашиглах боломжтой болохыг зааж байна. Харин private түлхүүр үгийг хэрэглэвэл public гишүүд рүү хандах боломжгүй болох өөрөөр хэлбэл эх энгийн аль ч гишүүн өгөгдөл рүү гадаад ертөнцөөс хандах боломжгүй болно. Эдгээр комбинацийг харуулсан нэгэн хялбар жишээг үзье.

```
class A
        {
        private:
        int privdataA;

        protected:
        int protdataA;

        public:
        int pubdataA;
        };

class B : public A
        {
        public:
        void funct()
```

```
                {
                int a;
                a=privdataA; //алдаа
                a=protdataA; //ok
                a=pubdataA;  //ok
                }
        };

class C : private A
        {
        public:
        void funct()
                {
                int a;
                a = privdataA; //алдаа
                a = protdataA; //ok
                a = pubdataA;  //ok
                }
        };

void main
{
int a;
B objB;

a=objB.privdataA;  //алдаа
a=objB.protdataA;  //алдаа
a=objB.pubdataA;  //ok

C objC;

a=objC.privdataA;  //алдаа
a=objC.protdataA;  //алдаа
a=objC.pubdataA;  //алдаа
}
```

Энэ жишээнд гаднаас болон дотоод гишүүн функцээс эх ангийн гишүүд рүү хандах бүх боломжийг харуулсан ба алдаа гарах эсэхийг тайлбараар бичсэн болохоор бүх зүйл ойлгомжтой юм.

Олон зэрэг удамшил

Анги нь нэгээс олон суурь ангиас удамшин үүсэх боломжтой байдаг ба үүнийг олон зэрэг удамшил (Multiple Inheritance) гэж нэрлэдэг. Энэ тохиолдолд бичлэг нь дараах байдалтай байна.

```
class A
  {};
class B
  {};
class C : public A, public B
  {};
```

Олон зэрэг удамшлын байгуулагч функц

Олон ангиас зэрэг удамшсан ангийн байгуулагч функц ямар байхвэ?

Бид жишээ болгон дараах программыг үзэх гэж байна. Бид барилгын байгууллагат банз бэлтгэдэг цехийн мэдээллийг программчилж байна гэж саная. Энэ программд нийлүүлж буй банзны тоо ширхэг, үнэ, боловсруулалтын зэрэглэл, хөндлөн огтлолын талбай (2x4), уртын хэмжээ гэх мэт мэдээллүүд тусгагдах ёстой байг.

Тэгвэл бид эндээс дараах дүгнэлтүүдийг хийх ёстой.

Банз бүрийн хувьд боловсруулалтын зэрэглэл, хөндлөн огтлолын талбай нь хувьсах өгөгдөл байна. Иймд банзны төрөл (Type) гэсэн эдгээр өгөгдлийг агуулах анги хэрэгтэй.

Урт гэсэн өмнөх жишээнд үзсэн Distance анги хэрэгтэй.

Банз (Lumber) гэсэн дээрх хоёр ангиас зэрэг удамшсан шинэ анги хэрэгтэй.

Мөн энд бидний өмнө үзээгүй тэмдэгт мөр гэсэн ойлголт гарч ирэх бөгөөд тэр нь ердөө л олон тэмдэгтүүдийг агуулсан нэг төрөл гэж ойлгоход л болно. Энэ төрлийн тухай хожим дэлгэрэнгүй тайлбарлана.

Ингээд дараах программыг үзье.

```
#include <iostream.h>
#include <string.h>
```

```
const int LEN = 40;

class Type
        {
        private:
        char dimensions[LEN];
        char grade[LEN];

        public:
        Type()
                {
                strcpy(dimensions, "бэлэн биш");
        strcpy(grade, "бэлэн биш");
                }
        Type(char di[], char gr[])
                {
                strcpy(dimensions, di);
                strcpy(grade, gr);
                }
        void gettype ()
                {
                cout << “Банзын х/огтлол(2х4 г.м):“;
                cin >> dimensions;
                cout << “Банзын боловсруулалт“
                    << “(өнгөлөн, зах г.м): “;
                cin >> grade;
                }
   void showtype()
                {
                cout << "\nБанзын х/огтлол:"
                    << dimensions;
                cout << "\nБанзын боловсруулалт:"
                        << grade;
                }
        };

class Distance
        {
        private:
        int feet;
        float inches;
```

```
        public:
        Distance()
                { feet=0; inches=0.0;}
        Distance(int ft, float in)
                { feet=ft; inches=in;}
        void getdist()
                {
                cout << "\nФут : ";
                cin >> feet;
                cout << "\nИнч : ";
                cin >> inches;
                }
        void showdist()
                {
                cout << feet << "\'-" << inches << '\"';
                }
        };

class Lumber : public Type, public Distance
        {
        private:
        int quantity;
        float price;

        public:
        Lumber() : Type(), Distance()
                { quantity=0; price=0.0; }
        Lumber(char di[], char gr[], int ft,
                  float in, int qu, float prc)
                        : Type(di, gr), Distance(ft, in)
                { quantity=qu; price=prc; }
        void getLumber()
                {
                Type::gettype();
                Distance::getdist();
                cout << " Тоо ширхэг: ";
                cin >> quantity;
                cout << " Нэг ширхэгийн үнэ: ";
                cin >> price;
                }
    void showLumber()
                {
                Type::showtype();
```

```
                cout << "\n Урт: ";
                Distance::showdist();
                cout << "\n" << quantity
                    << " ширхэг банзны үнэ нийт "
                    << price * quantity;
                }
        };
void main()
{
Lumber siding;

cout << "\nБанзны тухай өгөгдөл: ";
siding.getLumber();

Lumber studs("2x4", "өнгөлгөөтэй", 8,
                        0.0, 200, 4.45);

cout << "\nSiding "; siding.showLumber();
cout << "\nStuds  "; studs.showLumber();
}
```

Энэ жишээн дахь хамгийн гол шинэ зүйл бол Lumber байгуулагч функц юм. Энэ байгуулагч функц суурь хоёр ангийнхаа байгуулагч функцийг дуудаж байна.

Тодорхойгүй хоёрдмол утга (Ambiguity)

Олон зэрэг удамшлын хувьд дараах тохиолдол гарч болдог. Суурь хоёр анги нь ижил нэртэй функцүүдтэй боловч үүссэн анги нь энэ нэр бүхий гишүүн функц байхгүй байж болно.Тэгвэл үүссэн ангиас энэ нэртэй функцийг хэрэглэвэл ямар ангийн функц хэрэглэх нь тодорхойгүй хоёр утгатай (ambiguity) байдаг ба энэ тохиолдолд хөрвүүлэлт хийгддэггүй. Жишээ:

```
#include <iostream.h>
class A
        {
        public:
        void show()
                { cout << "\nClass A";}
```

```
        };
class B
        {
        public:
        void show()
                {cout << "\nClass B";}
        };
class C : public A, public B
        {
        };
void main()
{
C objC;
objC.show();                //алдаа
objC.A::show(); //ok
objC.B::show(); //ok
}
```

Virtual функц

Virtual гэдэг нь оршин байгаа боловч бодитой байхгүй гэсэн утгатай юм. Виртуаль функц ямар тохиолдолд хэрэглэгдэж болох талаар ярья. Графикийн янз бүрийн тойрог, тэгш өнцөгт, квадрат гэх мэт дүрсүүдийн цуглуулгыг үүсгэн тэдгээрийг массивт хадгалжээ гэж саная. Тэгвэл тэдгээрийг бүгдийг дэлгэцэнд зурах шаардлагатай болжээ гэж саная. Энэ тохиолдолд дүрсийн анги бүрт Draw() гэсэн түүнийг зурах функцийг тодорхойлох ёстой. Мэдээж энэ функц анги бүрийн хувьд өөр кодууд байх нь ойлгомжтой. Ингээд дараах программыг бичиж болох юм.

```
shape* ptrarr[100];
for (int j=0; j<N; j++)
        ptrarr[j]->Draw();
```

Энэ программд Draw() гишүүн функц нь дүрс бүрийн хувьд өөрөөр биелэх функц юм. Үүнийг полиморфизм гэж нэрлэдэг. Өөрөөр хэлбэл дүрс бүрийн хувьд ижил утгыг агуулах боловч өөр өөр үйлдэл хийх юм.

Одоо виртуаль функцийг хэрхэн хэрэглэх тухай тодорхой жишээн дээр ярья.

```
#include <iostream.h>
class Base
        {
        public:
        void show()
                { cout << "\nBase"; }
        };

class Derv1 : public Base
        {
        public:
        void show()
                { cout << "\nDerv1"; }
        };

class Derv2 : public Base
        {
        public:
        void show()
                { cout << "\nDerv2";}
        };

void main()
{
Derv1 dv1;
Derv2 dv2;
Base* ptr;

ptr= &dv1;
ptr->show();

ptr= &dv2;
ptr->show();
}
```

Манай энэ жишээнд дахь программыг ажиллуулахад show() функц хоёр тохиолдолд хоёуланд нь суурь ангийн show функц дуудагдана. Өөрөөр хэлбэл манай программ дараах үр дүнг буцаах болно.

```
Base
Base
```

Гэтэл өмнө нь өөр өөр кодоор биелэх ижил нэртэй функц хэрэгтэй байдаг талаар ярьж байсан. Тэгвэл энэхүү программд бяцхан өөрчлөлт хийхэд манай асуудал шийдэгдэх болно.

```
#include <iostream.h>
class Base
        {
        public:
        virtual void show()
                { cout << "\nBase"; }
        };

class Derv1 : public Base
        {
        public:
        void show()
                { cout << "\nDerv1";}
        };

class Derv2 : public Base
        {
        public:
        void show()
                { cout << "\nDerv2";}
        };

void main()
{
Derv1 dv1;
Derv2 dv2;
Base* ptr;

ptr= &dv1;
ptr->show();

ptr= &dv2;
ptr->show();
}
```

Тэгэхээр дээрх жишээнээс харахад программ биелэхийн өмнө хөрвүүлэлт хийх үед ptr заагч нь хэдийд Derv1, хэдийд Derv2 ангийн обьект руу заах нь тодорхой биш юм. Иймд хөрвүүлэлтийн үед аль show дүрмийг дуудахыг тодорхойлох боломжгүй байдаг. Харин программ биелэх үед энэ холболт хийгддэг тул үүнийг орой холболт гэж нэрлэдэг.

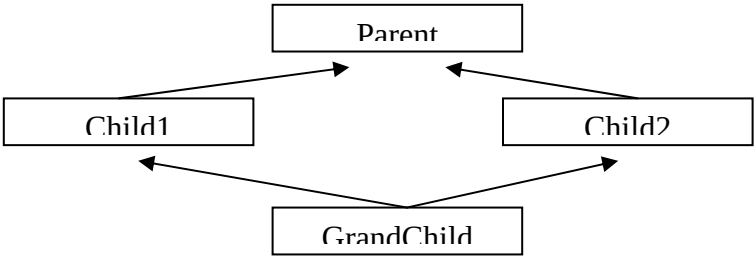
Манай жишээнд Base анги дахь show функц заавал байх шаардлагагүй байх тохиолдол байдаг. Энэ функц зөвхөн тухайн функцийг виртуаль гэж тодорхойлох үед хэрэглэгдэж харин үүссэн ангид энэ функцийг өөр өөрөөр тодорхойлох замаар ашиглах шаардлагатай байж болно. Тэгвэл энэ тохиолдолд суурь ангид виртуаль функцийг дараах байдлаар тодорхойлох ёстой.

```
virtual void show()=0;
```

Энд 0 утга олгож байгаа мэт бичиж байгаа нь хөрвүүлэгчид энэ функц бие байхгүй буюу программын кодын хэсэгт байхгүй гэдгийг зааж өгдөг ба энэ нь программын хэмжээг хэмнэдэг байна.

Тэгэхлээр манай программд тодорхойлогдсон суурь анги Base ангийн обьект нь хэрэггүй болох юм. Харин түүнээс үүссэн ангиудын обьект л хэрэглэгдэх болно. Ийм чанартай суурь ангийг хийсвэр (abstract) анги гэж нэрлэдэг.

Virtual суурь анги



Ийм загвараар ангиудыг үүсгэх тохиолдол гарч болдог. Дээрх загварыг тодорхой жишээн дээр тайлбарлья.

```
class Parent
        {
        protected:
        int Basedata;
        };

class Child1 : public Parent
        {
        };

class Child2 : public Parent
        {
        };

class GrandChild : public Child1, public Child2
        {
        public:
        int getdata()
                {return Basedata;}
        };
```

Тэгвэл энэ жишээнд getdata функц дотор Basedata гишүүн рүү хандах үед тодорхойгүй хоёрдмол утга алдаа гардаг. Учир нь Child1, Child2 ангиуд нь Parent ангиас удамшихдаа тус бүртээ Parent ангиас хуулбарлах маягаар удамшидаг. Ингэж

хуулбарлан удамшихыг subobject гэж нэрлэдэг. Тэгэхээр Child1, Child2 ангиуд нь Basedata өгөгдөлийн хуулбаруудыг тус бүртээ агуулж байгаа тул GrandChild ангид энэ өгөгдөл рүү хандахад ambiguity алдаа гардаг. Тэгвэл энэ асуудлыг шийдэхэд виртуаль суурь ангийг ашигладаг. Үүнийг дараах байдлаар бичнэ.

```
. . .
class Child1 : virtual public Parent
        {
        };
class Child2 : virtual public Parent
        {
        };
class GrandChild : public Child1, public Child2
        {
        public:
        int getdata()
                {return Basedata;}
        };
```

Энэ тохиолдолд бүх зүйл алдаагүй болно. Учир нь үүссэн ангиуд ганц хуулбарыг (share) олон зэрэг хэрэглэх байдлаар удамшидаг. Иймд ганц Basedata өгөгдөл рүү хандах болно.

Найз функцүүд

Обьектийн битүүмжлэл нь обьектийн гадна талаас private болон protected өгөгдөл рүү хандахыг хаадаг. Иймд зарим тохиолдолд боловсруулалт хийхэд хүндрэл учирдаг. Жишээлбэл ялгаатай ангиудын обьектын private мэдээлэлийг боловсруулах болжээ гэж саная. Боловсруулалт хийх функц маань обьектуудыг аргумент болгон аваад түүний private гишүүн рүү хандах ёстой гэж саная. Тэгвэл хэрвээ хоёр анги нь нэг суурь ангиас удамшсан байвал суурь ангид боловсруулалт хийх функцийг тодорхойлох боломжтой. Харин хоёр анги холбоотой биш байвал яах вэ?

Энэ тохиолдолд найз (friend) функцийг хэрэглэнэ. Найз функц хэрэглэсэн дараах жишээг үзье.

```
#include <iostream.h>
```

```
class beta;
class alpha
        {
        private:
        int data;

        public:
        alpha() { data = 3; }
        friend int frifunc(alpha, beta);
        };

class beta
        {
        private:
        int data;

        public:
        beta() {data = 7; }
        friend int frifunc(alpha, beta);
        };

int frifunc(alpha a, beta b)
        { return (a.data + b.data); }

void main()
{
alpha aa;
beta bb;
cout << frifunc(aa, bb);
}
```

Энд frifunc функц нь хоёр ангид хоёуланд нь тодорхойлогдсон тул анги бүрийн private өгөгдөл рүү хандах боломжтой юм. Өөрөөр хэлбэл холбоогүй хоёр ангийн гишүүд рүү нэг функцээр хандах тохиолдолд найз функцийг хэрэглэнэ. Манай жишээнд beta ангийг тодорхойлохоос өмнө түүнийг frifunc функцийн аргументад хэрэглэж байна. Иймд beta нэртэй анги байгаа гэдгийг программын эхэнд заавал хэлж өгөх ёстой байдаг. Программын эхэнд

```
class beta;
```

гэж тодорхойлсон байгааг анхаарах хэрэгтэй. Тэгэхлээр найз функцийг зайлшгүй хэрэглэх шаардлага амьдралд их гардаг. Жишээлбэл бидний өмнө Distance ангид + оператор тодорхойлсоныг бид мэдэж байгаа билээ. Тэгвэл энэ операторыг хэрэглэх тохиолдолуудыг судлая.

```
Distance d1, d2, d3;
d1=d2+10.0;
```

Дээрх байдлаар + операторыг хэрэглэхэд d2 обьектын оператор гишүүн функц дуудагдаж 10.0 нь түүнийг аргумент болдогийг бид мэднэ. Иймд 10.0 тоо нь бидний тодорхойлсон байгуулагч функцийн тусламжтай хөрвүүлэлт хийгдэх учир алдаа гарахгүй. Харин дараах тохиолдолд яахыг сонирхоё.

```
d3 = 10.0 + d2;
```

Энэ тохиолдолд 10.0 тоо нь обьект биш тул алдаа гарна. Тэгэхлээр үүнээс гарахын тулд операторыг найз функц байдалтай тодорхойлдог.

```
class Distance
        {
        . . .
        friend Distance operator+(Distance,Distance);
        . . .
        };

Distance operator+(Distance d1,Distance d2)
        {
        int f=d1.feet+d2.feet;
        float i=d1.inches+d2.inches;
        if (i>=12.0)
        { i-=12.0; f++; };
        return Distance(f, i);
        }
```

Найз ангиуд

Найз функцийг хэрэглэж байгаа тохиолдолд зөвхөн найз функцүүд хоёр ангийн гишүүд рүү хандах боломжтой байна.

Гэтэл заримдаа ялгаатай ангийн бүх гишүүн функцүүд өөр ангийн private гишүүд рүү хандах шаардлага гарч болно. Энэ тохиолдолд найз ангийг хэрэглэдэг. Ингээд найз ангийг хэрэглэсэн хялбархан жишээ үзье.

```
#include <iostream.h>
class alpha
        {
        private:
        int data1;

        public:
        alpha() { data1 = 99; }
        friend class beta;
        };

class beta
        {
        public:
        void func1(alpha a)
                {cout << "\ndata1 = " << a.data1;}
        void func2(alpha a)
                {cout << "\ndata1 = " << a.data1;}
        void func3(alpha a)
                {cout << "\ndata1 = " << a.data1;}
        };

void main()
{
alpha a;
beta b;

b.func1(a);
b.func2(a);
b.func3(a);
}
```

Энэ жишээнд alpha ангийн найзыг beta гэж заасан тул beta анги alpha ангийн бүх гишүүд рүү дурааараа хандах болно.

Статик функцүүд

Бид өмнө статик өгөгдлийн тухай ярьж байсан. Тэгвэл статик өгөгдөл рүү хандахад заавал тодорхой нэг обьектоос хандах шаардлагагүй байдаг. Учир нь статик өгөгдөл нь бүх обьектуудад ерөнхий байдаг аль ч обьектоос хандахад адилхан байна. Тэгвэл ийм өгөгдөл рүү хандахад статик функцийг ашиглах нь тохиромжтой байдаг. Статик функцийг дуудахдаа обьектын нэрийг биш түүний ангийн нэрийг ашигладаг. Жишээ:

```
class gamma
        {
        private:
        static int total;

        public:
        static showtotal();
                { cout << total; }
        };

. . .

gamma g;
gamma::showtotal();

. . .
```

Утга олгох

Аливаа обьектуудын хооронд утга олгох үйлдлүүдийг хэрэглэж болдог билээ. Тэгвэл утга олгох үйлдэл хийх үед обьектын гишүүн өгөгдөл тус бүрийг хуулах байдлаар утга олголт явагддаг. Утга олгох үйлдлийг дараах байдлаар бичихийг бид мэднэ.

```
a2=a1;
```

Дээрхээс гадна обьектийг дараах байдлаар үүсгэх боломжтой.

alpha a2(a1);

Өөрөөр хэлбэл эхний тохиолдол нь утга олгох үйлдэл, харин хоёр дахь нь утга олгох үйлдэл биш байдаг ба үүнийг хуулж үүсгэх (copy constructor) гэж нэрлэдэг. Хоёр дугаар тохиолдолыг мөн дараах байдлаар бичиж болох ба энэ тохиолдолд мөн утга олгох үйлдэл биш байдаг.

alpha a2=a1;

Программ бичих явцад зарим тохиолдолд эдгээр үйлдлүүдийг өөрчлөх шаардлага гардаг. Жишээлбэл дараах ангийн хувьд үзье.

```
#include <iostream.h>
#include <stddef.h>
class String
        {
        private:
        char *str;

        public:
        String()
                { str = NULL;}
        String(char *s)
                {
                str=new char[strlen(s)+1];
                strcpy(str, s);
                }
        void changestr()
                {
                delete str;
                str = "Шинэ тэмдэгт мөр";
                }

        void showstr()
                { cout << "\ns2=" << str; }
        };

void main(char *, int)
{
String s1 = "s1 обьект дахь тэмдэгт мөр";
String s2;
```

```
s2=s1;
s2.showstr();

s1.changestr();
s2.showstr();
}
```

Энэ жишээнд 2 дахь s2.showstr() командаар

s1 обьект дахь тэмдэгт мөр

гэсэн мэдээлэл дэлгэцэнд гарах болно. Учир нь s2 = s1 утга олгох үйлдлийг хэрэглэх үед зөвхөн хаягийг хуулсан тул s1, s2 обьектууд үнэн хэрэгтээ нэг ижил тэмдэгт мөрийг заах болно. Үүний дараа changestr функцээр тэмдэгт мөрийг устгах үед s1, s2 ангийн гишүүний зааж байсан мөр устах болно. Тэгэхлээр энэ тохиолдолд энгийн утга олгох үйлдэл нь учир дутагдалтай болж байна. Иймд утга олгох үйлдэл, хуулан үүсгэх арга зэргийг өөрчлөх шаардлага зайлшгүй гардаг.

Утга олгох үйлдэлийг хэрхэн өөрчлөхийг үзье.

```
void operator = (String& s)
        {
        if (str == NULL)
                delete str;
        str = new char[strlen(s.str)+1];
        strcpy(str, s.str);
        }
```

Ингэж утга олгох үйлдэлийг тодорхойлсоны дараа

s1 = s2

утга олголтыг хийхэд санах s1 обьектын зааж байгаатай тэнцүү хэмжээний мужийг санах ойд үүсгэн хаягийг нь s2 обьектын str гишүүнд олгоод тэмдэгт мөрийг дараа нь хуулж байна. Иймд обьект тус бүр нь өөрийн тэмдэгт мөрийг заах болно гэдэг нь ойлгомжтой.

Оператор гишүүн функцүүд удамшидаг бол харин утга олгох оператор удамшидаггүй болохыг анхаарах хэрэгтэй.

Дээрх утга олголтыг тодорхойлсны дараа хэрэв хуулан үүсгэх аргыг хэрэглэвэл дахиад өмнөх асуудалтай тулгарна.

Учир нь хуулан үүсгэх аргын үед утга олгох оператор ажилладаггүй болохыг анхаарах хэрэгтэй. Энэ үед хуулах байгуулагч ажилладаг байна. Хуулах байгуулагчийг дараах байдлаар тодорхойлох ёстой болно.

```
String(String& s)
        {
        if (str == NULL)
                delete str;
        str = new char[strlen(s.str)+1];
        strcpy(str, s.str);
        }
```

Энд байгуулагчийг

```
String(String s)
```

гэж тодорхойлж болохгүйг анхаарах хэрэгтэй.

this заагч

Аливаа гишүүн функц өөрийн обьектын хаягийг хэрэглэх шаардлага гарвал this заагчийг ашилах хэрэгтэй. Өөрөөр хэлбэл this заагч нь обьект руугаа өөр рүүгээ заасан заагч юм.

Тэгэхээр обьект энэ заагчийг дараах байдлаар хэрэглэж болох юм.

```
bool compare(char *str)
        { return strcmp(this->str, str); }
```

Мөн түүнчлэн утга буцаах үед хэрэглэж болно.

```
String& operator = (String& s)
        {
        if (str==NULL)
                delete str;
        str=new char[strlen(s.str)+1];
        strcpy(str, s.str);
        return *this;
        }
```

Бүлгийн дүгнэлт

Бидний өмнө үзэж байсан программчлалын аргыг уламжлалт буюу процедур хандалтат программчлал гэж нэрлэдэг. Учир нь ийм аргаар программчлахад программын мэдээлэл буюу түүнийг боловсруулах функц, процедурууд нь тус тусдаа байдаг юм. Ийм тохиолдолд зөв мэдээллийг буруу процедураар боловсруулах аюултай байдаг аж.

Харин энэ бүлэгт бид нэгэн шинэ аргыг үзэж байгаа юм. Обьект хандалтат программчлал гэх энэ арга нь мэдээлэл болон түүнийг боловсруулах процедуруудыг нэгэн цул обьект болгон авдгаараа өмнөх аргаас ялгаатай. Ийм үед процедур нь зөвхөн өөрт хамаатай мэдээллийг л боловсруулах болно.

Обьект төрлийн хувьсагчийг обьектийн анги гэж нэрлэнэ.

Нэг обьект нь өөр обьектээс удамшиж болно. Ө.х өөр нэгэн ангийн бүх шинж чанаруудыг өөртөө агуулж болно. Удамшсан ангийг child class, удамшуулсан ангийг parent class гэж нэрлэдэг.

Уг ангийн обьектийг санах ойд үүсгэх үед автоматаар дуудагдан ажилладаг функцийг байгуулагч функц гэж нэрлэнэ. Харин санах ойгоос устгах үед ажилладаг функцийг устгагч функц гэж нэрлэнэ. Байгуулагч функц нь ангийн нэртэй ижил нэртэй байх ёстой бөгөөд харин устгагч функц нь байгуулагчаасаа нэрийн өмнөх '~' тэмдгээрээ л ялгагдана.

Обьектийн ангийг бусад төрлийн хувьсагчдын нэгэн адил функцийн аргументэд өгч, функцийн утгад буцааж болно. Мөн түүнийг бүтцийн талбар болгож ч болно. Обьект дотор обьект агуулагдаж чадна.

Нэг ангийн олон обьект үүсвэл обьектийн талбарууд тэр тоогоор санах ойд дахин дахин үүсдэг. Харин нэг ангийн бүх обьектийн дунд ерөнхий хэрэглэгддэг хувьсагчийг статик хувьсагч гэдэг. Статик хувьсагчийг static гэсэн нөөц үг ашиглан тодорхойлно.

Обьект маань нэгэнт хэрэглэгчийн тодорхойлсон төрөл учраас түүн дээр арифметик үйлдлүүд болон стандарт операторуудыг шууд ашиглаж болохгүй. Харин ингэж ашиглахыг хүсвэл обьектдоо уг операторыг тодорхойлж өгөх хэрэгтэй.

Мөн төрөл хөрвүүлэх операторыг тодорхойлж өгч болно. Ингэснээр обьект төрлийг стандарт төрлийн хувьсагчид утга оноох боломжтой болох юм.

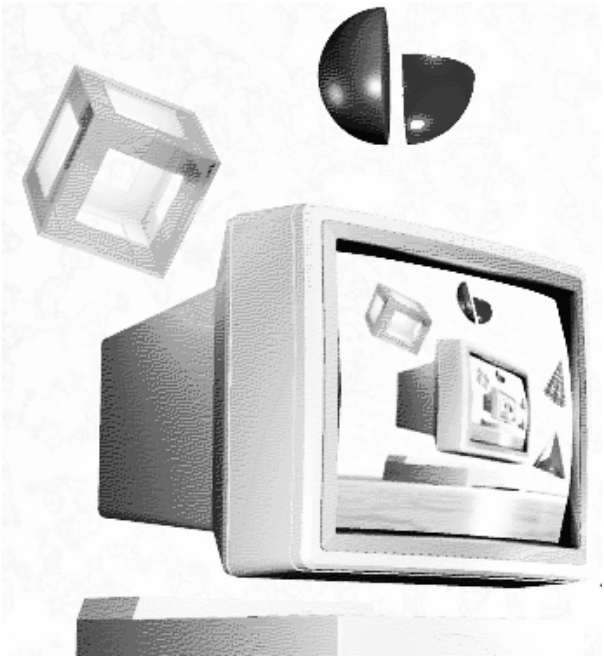
Өөрөөсөө олон ангиудыг удамшуулдаг боловч өөрөө шууд хэрэглэгддэггүй ангийг абстракт суурь анги гэж нэрлэнэ.

Ангийн гишүүд 3 янзаар тодорхойлогдож болно. Хэрэв private маягаар тодорхойлогдвол тэрхүү гишүүн рүү ангийн гаднаас, ө.х. бусад анги болон үндсэн программаас шууд хандах боломжгүй юм. Харин public зарчмаар тодорхойлогдвол тэр гишүүн рүү хаанаас ч хандаж болно. Харин protected зарчмаар тодорхойлогдсон бол зөвхөн түүнээс удамшсан анги л тэр гишүүн рүү хандах боломжтой.

Удамшиж байгаа анги бүрд дахин тодорхойлогдож байх шаардлагатай функцийг виртуаль болгон тодорхойлох ёстой. Виртуаль функц төдийгүй виртуаль анги байж болно.

Ямар ч обьектэд өөрийгөө заасан this гэсэн заагч автоматаар тодорхойлогдсон байдаг. Уг обьектийн ямар нэгэн гишүүн обьектийнхоо хаягийг ашиглах шаардлага гарвал энэ заагчийг ашиглаж болно.

6-р бүлэг



-Массив
-Обьектуудын массив
-Массивыг зөв зохион байгуулах нь
-Тэмдэгт мөр

6-р бүлэг

Бодит амьдрал дээр бид үргэлж бүлэг юмтай харьцдаг. Программ бичиж байх явцад ч ижил төрлийн олон өгөгдөлтэй ажиллах шаардлага маш их хэмжээгээр гардаг. Ийм зорилгоор ашиглагддаг механизмыг C++-д массив гэж нэрлэнэ. Массив нь мянга мянган ижил төрлийн элементтэй ажиллах чадвартай. Массивын элементүүд нь бүхэл, бутархай тодийгүй бүтэц төрөлтэй ч байж болно. Массив, бүтэц хоёр нь олон өгөгдлийг нэгтгэдгээрээ төстэй боловч бүтэц нь олон төрлийн өгөгдлүүдийг, харин массив нь нэг төрлийн өгөгдлүүдийг хадгалдаг онцлогтой юм. Мөн өөр нэгэн онцлог нь бүтцийн элементүүдэд нэрээр нь ханддаг бол харин массивын элементүүдэд ямар ч нэр байхгүй харин индекс гэсэн ойлголтыг ашигладаг.

Массив нь Pascal, Basic мэтийн ихэнх программчлалын хэлэнд өргөн ашиглагддаг. Энэ бүлэгт бид массивын энгийн хэрэглээнээс эхлээд түүнийг ангийн гишүүн болгох, обьектүүдийг хадгалахад хэрхэн ашиглах тухай дэлгэрэнгүй үзэх болно. Эдгээрийг үзсэнээр таны обьект хандалтат программчлалын тухай ойлголтоо нэлээн хэмжээгээр гүнзгийрүүлэх боломжийг олох болно. Энэ бүлгийн төгсгөлд бид тэмдэгтийн массив болох string –ийн тухай үзнэ.

Массивын тухай үндсэн ойлголт

Доорх жишээ массивын тухай үндсэн ойлголт өгнө. Энэ программд хүмүүсийн насыг массив ашиглан санах ойд хадгалж байна.

```
//replay.cpp
#include <iostream.h>

void main()
{
int age[4];

cout << endl;
for (int j=0; j<4; j++)
        {
        cout << “Насаа оруулж өгнө үү : ”;
        cin >> age[j];
        }
for (j=0; j<4; j++)
        cout << “\nТаны оруулсан нь: “ << age[j];
}
```

Эхний давталт хэрэглэгчээс тоо асууж, өгсөн тоог “нас” гэсэн массивт хадгалж байна. Харин сүүлийн давталт нь тэдгээрийг массиваас уншиж, дэлгэцэнд хэвлэнэ.

```
Насаа оруулж өгнө үү : 44
Насаа оруулж өгнө үү : 16
Насаа оруулж өгнө үү : 23
Насаа оруулж өгнө үү : 68

Таны оруулсан нь: 44
Таны оруулсан нь: 16
Таны оруулсан нь: 23
Таны оруулсан нь: 68
```

Массив тодорхойлох

Бүх л төрлүүдийг нэгэн адил массив төрлийн хувьсагчийг ч ашиглахаасаа өмнө урьдчилан тодорхойлсон байх ёстой. Мөн бусад төрлийн адил энэ тодорхойлолт нь хувьсагчын нэр болон төрлөөс тогтоно. Харин бусад төрлөөс ялгарах ганц зүйл нь массивын хэмжээ буюу элементийн тоо гэсэн шинэ бичлэг нэмэгдэнэ. Энэ хэмжээг шууд төрлийн нэрийн хойно залгуулан бичих бөгөөд дөрвөлжин хаалтаар хашсан байна. Өмнөх жишээнд тодорхойлсон массив нь бүхэл тоон төрлийн 4 элемент агуулах ёстой юм. Массивын хэмжээ нь ямар нэгэн тогтмол тоо буюу тогтмол тоон дээр хийгдсэн үйлдэл байж болно.

Массивын элементүүд

Бүтцийн гишүүдийг талбар гэж нэрлэдгийн адил массивын гишүүдийг элемент гэж нэрлэдэг. Массивын элементүүд бүгд ижил төрөлтэй ч тус бүрдээ өөр өөр утга агуулж байх болно. Массивын элемент бүрд төрлийнх нь хэмжээгээр санах ой хуваарилаж, массивын нэр нь тэрхүү үргэлжилсэн урт мужийн эхлэлийг зааж байх болно. Бидний үзсэн өмнөх жишээнд уг массивд зориулж (4 элемент * 2 байт) 8 байтын хэмжээтэй санах ойг авна гэсэн үг. Массивын эхний элемент 0 гэсэн индекстэй байна. Харин хамгийн сүүлийн элемент нь (n-1) гэсэн индекстэй (n биш!). Тухайлбал өмнөх жишээнд байгаа массив нь 0-3 индекстэй 4 элементээс тогтох юм.

Массивын элементэд хандах нь

Дээрх жишээнд бид массивын элемент бүрд 2 удаа хандаж байна. Эхний удаад уг массивын элемент бүрт гарнаас утга оноож өгч байгаа.

```
cin >> age[j];
```

Харин дараагийн удаад тэдгээрийг дэлгэцэнд хэвлэн гаргаж байна.

```
cout << “\nТаны оруулсан нь “ << age[j];
```

Ингэж хандахдаа массивын нэрийг залгуулан дөрвөлжин хаалтанд хашигдсан j гэсэн хувьсагчийг ашиглажээ. Массивын аль элементэд хандахыг j гэсэн хувьсагч ашиглан зохицуулж байна.

Массивын элементүүдийг дундажлах нь

Массивтай ажиллах өөр нэгэн жишээ үзье. Энэ жишээ нь хэрэглэгчийн оруулсан 6 тоог массивт хадгалж, тэдгээрийн дунджийг тооцон гаргаж өгөх болно.

```
//sales.cpp
#include <iostream.h>

const int SIZE=6;

void main()
{
float sales[SIZE];
cout << “\n6 өдрийн орлогыг дараалуулан оруул\n”;
for (int j=0; j<SIZE; j++)
        cin >> sales[j];

float total=0;
for (j=0; j<SIZE; j++)
```

```
        total+=sales[j];
float average=total/SIZE;
cout << “Дундаж орлого=” << average;
}
```

Программ доорх байдлаар ажиллана.

```
6 өдрийн орлогыг дараалуулан оруул
352.64
867.70
781.32
867.35
746.21
189.45
Дундаж орлого=634.11
```

Энд ашиглагдаж байгаа шинэлэг санаа нь массивын хэмжээг тогтмол болгон тодорхойлж, түүнийгээ программын туршид ашигласан байна.

```
const int SIZE=6;
```

Массивын хэмжээг ингэж тогтмолоор тодорхойлохын гол ашиг тус нь программчилж байх үед массивын хэмжээг өөрчлөх ажиллагааг хялбар болгодогт байгаа юм. Зөвхөн энэ тогтмолын утгыг өөрчлөхөд программ дахь массивтай холбоотой бүх үйлдэл дэх элементийн тоо автоматаар өөрчлөгдөх юм. Программд ингэж тогтмолоор ашиглагдах нэрнүүдийг голчлон том үсгээр тодорхойлдог. Энэ нь программыг уншиж байх явцад өөрчлөгдөхгүй тогтмол утгуудыг шууд ялган харах боломж олгоно.

Массивд анхны утга оноох

Бусад төрлүүдийн нэгэн адил массивыг тодорхойлох үед анхны утгыг нь оноож өгч болно. Доорх жишээнд энэ тухай үзүүлнэ.

```
//days.cpp
#include <iostream.h>

void main()
{
int month, day, total_days;
int days_per_month[12]={31, 28, 31, 30, 31, 30,
                                        31, 31, 30, 31, 30, 31};

cout << “\nСарыг оруулна уу(1 to 12): ”;
cin >> month;
cout << “Өдрийг оруулна уу(1 to 31): ”;
cin >> day;
total_days=day;
for (int j=0; j<month-1; j++)
          total_days+=days_per_month[j];
cout << “Жилийн эхнээс өнгөрсөн өдрийн тоо : ”
          << total_days;
}
```

Уг программ жилийн эхнээс өгсөн өдөр хүртэлх хоногийн тоог гаргах юм (Гэхдээ өндөр жилийн хувьд нэг хоногийн зөрүү гарч болно.). Программын ажиллах жишээ нь:

```
Сарын оруулна уу(1 to 12): 3
Өдрийг оруулна уу(1 to 31): 11
Жилийн эхнээс өнгөрсөн өдрийн тоо : 70
```

Хэрэглэгч гараас сар, өдрийг оруулж өгсний дараагаар программ өмнөх бүх саруудын нийт өдрийг тоолж, нийлбэрийг total_days хувьсагчид хадгалж байна. Харин сар бүр дэх өдрийг days_per_month массиваас унших болно. Жишээ нь: Хэрэглэгч 5 сар

гэж өгсөн бол өмнөх 4 сарын (31, 28, 31, 30) гэсэн утгууд нэмэгдэх юм.

Массивт өгч байгаа анхны утгууд нь бүхлээрээ нэг хаалтанд бичигдсэн байх бөгөөд өөр хоорондоо таслалаар зааглагдана. Массивтайгаа тэнцүүгийн тэмдгээр холбогдсон байх ёстой.

Массивыг тодорхойлох үед элементийн тоог зааж өгөөгүй байвал компилятор автоматаар анхны утгуудыг тоолж, элементийн тоог тогтоож өгдөг.

```
int days_per_month[]={31, 28, 31, 30, 31, 30,
                                    31, 31, 30, 31, 30, 31};
```

Хэрэв массивын оноосон утгын тоо массивын элементийн тооноос цөөн байвал үлдсэн элементүүд 0 утга авна. Харин оноосон утга элементээс олон байгаа тохиолдолд алдаа гарна.

Олон хэмжээст массив

Өмнөх жишээнүүдэд бид дандаа шугаман массив буюу нэг хэмжээст массивуудыг авч үзсэн билээ. Өөрөөр хэлбэл, массивын нэг элементийг тодорхойлоход бидэнд нэг л индекс шаардагдаж байсан. Харин доорх жишээнд хоёр хэмжээст массив тодорхойлон ашигласан байна.

```
//salemon.cpp
#include <iostream.h>
#include <iomanip.h>

const int districts=4;
const int months=3;

void main()
{
```

```
int d, m;
float sales[districts][months];

cout << endl;
for (d=0; d<districts; d++)
        for (m=0; m<months; m++)
        {
        cout << d+1 << "-р дүүргийн ";
        cout << m+1 << "-р сарын орлого "<< ": ";
        cin >> sales[d][m];
        }
cout << "\n\n";
cout << "                        Сар\n"
cout << "                1       2       3"
for (d=0; d<districts; d++)
        {
        cout << "\nДүүрэг " << d+1;
        for (m=0; m<months; m++)
                cout << setioflags(ios::fixed)
                        << setiosflags(ios::showpoint)
                        << setprecision(2)
                        << setw(10)
                        << sales[d][m];
        }
}
```

Энэ программ sales гэсэн 2 хэмжээст массивт утгууд уншиж түүнийг хүснэгт маягт оруулан дэлгэцэнд хэвлэж байна.

```
1-р дүүргийн, 1-р сарын орлого 3964.23
1-р дүүргийн, 2-р сарын орлого 4135.87
1-р дүүргийн, 3-р сарын орлого 4397.98
2-р дүүргийн, 1-р сарын орлого 867.75
2-р дүүргийн, 2-р сарын орлого 923.59
2-р дүүргийн, 3-р сарын орлого 1037.01
3-р дүүргийн, 1-р сарын орлого 12.77
3-р дүүргийн, 2-р сарын орлого 378.32
3-р дүүргийн, 3-р сарын орлого 798.22
4-р дүүргийн, 1-р сарын орлого 2983.53
4-р дүүргийн, 2-р сарын орлого 3983.73
4-р дүүргийн, 3-р сарын орлого 9494.98
```

	Сар		
	1	2	3
Дүүрэг 1	3964.23	4135.87	4397.98
Дүүрэг 2	867.75	923.59	1037.01
Дүүрэг 3	12.77	378.32	798.22
Дүүрэг 4	2983.53	3983.73	9494.98

Хоёр хэмжээст массивын тодорхойлолтод тус бүрийг дөрвөлжин хаалтаар хашсан хоёр индекс ашиглагдана.

```
float sales[districts][months];
```

Бичлэгээс үзвэл олон хэмжээст массивыг массивын массив гэж ойлгож ч болно. Өөрөөр хэлбэл, sales гэдэг нь districts тооны элементтэй массив бөгөөд элемент бүр нь months тооны элементтэй массив байна гэсэн үг. Хэрэв 3 хэмжээст массив бол массивын массивын массив болох юм.

Хоёр хэмжээст массивын элементэд хандахад мөн л 2 индексийг нь ашиглаж ажиллана.

```
sales[d][m]
```

Индекс бүр дөрвөлжин хаалтанд бичигдэх ёстойг анхаарах хэрэгтэй. Программчлалын бусад хэл шиг sales[d, m] гэсэн бичлэг ашиглаж болохгүй.

Тоог форматлах нь

Өмнө үзсэн salemon жишээнд дэлгэцэнд мөнгөний утгууд харуулж байгаа юм. Ийм утгуудыг дэлгэцэнд харуулахад доорх маягийн шаардлагууд тавигдана. Үүнд: хэвлэгдэж буй бодит тоо бүр таслалын хойно 2 оронтойгоор, мөн тэдгээр утгууд нь

бүгд таслалаараа тэгшрэн хэвлэгдсэн байх ёстой юм байна.

C++-ийн стандарт урсгалуудад ийм тохируулгуудыг хийх боломжуудтай байдаг. Бид урьд өмнө нь setw() гэсэн функцийг нь үзэж байсан. Харин одоо өөр хэдийг нэмж үзье.

```
cout << setioflags(ios::fixed)
        << setiosflags(ios::showpoint)
        << setprecision(2)
        << setw(10)
        << sales[d][m];
```

ios гэсэн ангийн long төрлийн утгын бит бүр нэг төлөвийг тодорхойлж байдаг. Анги гэж юу болох, түүнтэй хэрхэн ажиллаж байгааг энэ удаад тайлбарлалгүй орхие.

Дээрх жишээнд setiosflags гэсэн функцийг ашиглан тоог форматлаж байна. Ө.х. эхний хоёр мөр нь тоог дэлгэцэнд харуулах форматыг тогтоож өгч байгаа юм (Харин энэхүү форматыг хүчингүй болгоё гэвэл resetiosflags гэсэн функцийг ашигладаг юм). Эхний мөрөнд fixed гэсэн флагийг тогтоож өгснөөр тоог бэхлэгдсэн таслалтай харуулахыг (3.45e-3 биш, 0.0345), харин showpoint флагаар тооны таслалыг үргэлж харуулахыг (123 биш, 123.00) тус тус зааж өгч байна. Харин 3 дахь мөрөнд setprecision функцээр таслалын ард гарах оронгийн тоог зааж өгнө. Энэ бүхнийг cout урсгал руу оруулж өгсний дараагаар дэлгэцэнд хэвлэгдэх тоо дээр дурдсан форматуудаар хэвлэгддэг.

Олон хэмжээст массивд анхны утга оноох

Ганц хэмжээст массивын нэгэн адил олон хэмжээст массивт ч анхны утга оноож болно. Харин ингэхийн тулд хэд хэдэн давхар хаалт ба таслалууд ашиглагдах тул тун хянуур хандах нь зүйтэй. Одоо saleinit гэсэн жишээ авч үзье.

```
//saleinit.cpp
#include <iostream.h>
#include <iomanip.h>

const int districts=4;
const int months=3;

void main()
{
int d, m;

float sales[districts][months]=
                              {{1432.07, 234.50, 654.01},
                               {322.00, 13838.32,17589.88},
                               {9328.34, 934.00, 4492.30},
                               {12838.29, 2332.63, 32.93}};

cout << "\n\n";
cout << "                          Cap\n"
cout << "                  1       2       3"
for (d=0; d<districts; d++)
        {
        cout << "\nДүүрэг " << d+1;
        for (m=0; m<months; m++)
                cout << setioflags(ios::fixed)
                        << setiosflags(ios::showpoint)
                        << setprecision(2)
                        << setw(10)
                        << sales[d][m];
        }
}
```

2 хэмжээст массив гэдэг бол массивын массив гэж дээр хэлсэн билээ. Анхны утга оноох үйлдлийн бичлэг ч энэ зарчим дээр үндэслэгдсэн байна.

Өөрөөр хэлбэл, дэд массив бүр нэг хаалтан дотор, харин массив бүр хоорондоо таслалаар зааглагдсан байна.

Массивыг функцийн аргументэд өгөх нь

Бусад бүх төрлийн нэгэн адил массивыг функцийн аргументэд дамжуулан өгч болно. Дахин нэг жишээ үзэцгээе.

```
//salefunc.cpp
#include <iostream.h>
#include <iomanip.h>

const int districts=4;
const int months=3;

void display(float[districts][months]);

void main()
{
float sales[districts][months]=
                                {{1432.07, 234.50, 654.01},
                                 {322.00, 13838.32, 17589.88},
                                 {9328.34, 934.00, 4492.30},
                                 {12838.29, 2332.63, 32.93}};
display(sales);
}

void display(float funsales[districts][months])
{
int d, m;

cout << “\n\n”;
cout << “                          Cap\n”
cout << “                  1       2       3”
for (d=0; d<districts; d++)
          {
          cout << “\nДүүрэг ” << d+1;
```

```
        for (m=0; m<months; m++)
                cout << setioflags(ios::fixed)
                        << setiosflags(ios::showpoint)
                        << setprecision(2)
                        << setw(10)
                        << sales[d][m];
        }
}
```

Функцийг тодорхойлохдоо массив төрлийн аргументийг элементийн төрөл болон массивын хэмжээгээр тодорхойлж өгдөг.

```
void display(float[districts][months])
```

Массивыг шинээр тодорхойлохдоо түүний хэмжээг заавал зааж өгөх шаардлагагүй байдаг тухай дээр өгүүлсэн билээ. Тийм учраас дээрх бичлэгийг доорх байдлаар өөрчилж бас болно.

```
void display(float[][months])
```

Харин 2 дахь индексийг хоосон өгч болохгүй. Учир нь, аргументэд ирэх массивын хэмжээ тодорхойгүй байж болох боловч, харин уг массивын нэг элементийн хэмжээ тодорхой өгөгдсөн байх ёстой юм.

Функцийг дуудах болоход энгийн төрөлтэй хувьсагчидтай адилаар шууд массив төрлийн хувьсагчийг аргументэд бичиж өгөх юм.

```
display(sales);
```

Харин энгийн хувьсагчаас ялгаатай нь энгийн хувьсагчийг функцийн аргументэд дамжуулахад түүний утга нь функц дэх хувьсагчид хувилагдан очдог бол ийм нийлмэл бүтцийг дамжуулахад түүний утга нь

хувилагддаггүй, харин хаяг нь дамждагт байгаа юм. Бид функц доторх массивын утгыг өөрчлөхөд үндсэн программ дахь массивын утга өөрчлөгдөх болдог. Ө.х. funsales гэдэг нь sales массивын өөр нэгэн шинэ нэр л болж байгаа юм.

Бүтцийн массив

Массивын нэг элемент нь энгийн төрөл төдийгүй бүтэц буюу struct ч байж болно.

```
//partaray.cpp
#include <iostream.h>

const int size=4;

struct part
        {
        int ModelNumber;
        int PartNumber;
        float cost;
        }

void main()
{
int n;
part apart[size];

for (n=0; n<size; n++)
        {
        cout << endl;

        cout << “Загварын дугаар: ”;
        cin >> apart[n].ModelNumber;
        cout << “Дэд загварын дугаар: ”;
        cin >> apart[n].PartNumber;
        cout << “Үнэ: ”;
        cin >> apart[n].cost;
        }
```

```
for (n=0; n<size; n++)
        {
        cout << “\nЗагвар ” << apart[n].ModelNumber;
        cout << “Дэд загвар ” << apart[n].PartNumber;
        cout << “Үнэ ” << apart[n].cost;
        }
}
```

Энэ программд байгаа part гэсэн бүтэц нь 3 талбараас тогтож байна. Харин apart гэсэн массив нь part төрлийн элементүүдтэй байгаа юм. Энэ массив нь 4 элементтэй бөгөөд уг программ элемент бүрийн талбаруудын утгыг хэрэглэгчээс асууж, түүнийгээ дэлгэцэнд хэвлэж гаргадаг юм. Программ дэлгэцэнд доорх маягаар ажиллана.

```
Загварын дугаар: 44
Дэд загварын дугаар: 4954
Үнэ: 133.45

Загварын дугаар: 44
Дэд загварын дугаар: 8431
Үнэ: 97.59

Загварын дугаар: 77
Дэд загварын дугаар: 9343
Үнэ: 109.99

Загварын дугаар: 77
Дэд загварын дугаар: 4297
Үнэ: 3456.55

Загвар 44 Дэд загвар 4954 Үнэ 133.45
Загвар 44 Дэд загвар 8431 Үнэ 97.59
Загвар 77 Дэд загвар 9343 Үнэ 109.99
Загвар 77 Дэд загвар 4297 Үнэ 3456.55
```

Бүтэц төрлийн массивын талбарт хандахын тулд массивын элементэд хандах бичиглэл, бүтцийн талбарт хандах бичиглэлийг хоршуулан ашиглана.

apart[n].ModelNumber

Массивыг ангийн гишүүн болгон ашиглах

Анги дотор массивыг гишүүн болгон тодорхойлж болно. Жишээ нь: үүний нийтлэг хэрэглээ нь стек (stack) гэж нэрлэгдэх өгөгдлийн тусгай бүтэц юм.

Зоогийн газарт цэвэрхэн таваг хадгалах зориулалттай тусгай тавиур байдгийг стек гэж нэрлэдэг. Стекийн дээрээс шинэ таваг нэмэхэд бусад таваг бага зэрэг доош сууж өгдөг бөгөөд харин хамгийн дээрээс нь нэг таваг авахад бусад таваг дээшилж, дараагийн таваг хамгийн дээд талд гарч ирнэ. Стект хамгийн сүүлд орсон таваг хамгийн эхэнд стекээс авагддаг.

MS-DOS үйлдлийн системийг ашигладаг компьютерүүдэд өгөгдлийг хадгалах зориулалт бүхий стек гэсэн санах ой байдаг. Өмнө өгүүлж байснаар функцийн аргумент болон буцах хаягууд ч энд хадгалагдаж байдаг юм. Stackaray гэсэн дараагийн жишээг авч үзье.

```
//Stackaray.cpp
#include <iostream.h>

const int max=100;

class Stack
        {
        private:
        int st[max];
        int top;
```

```
        public:
        Stack() { top=0; }
        void push(int var) { st[++top]=var; }
        int pop()          { return st[top--]; }
        };

void main()
{
Stack s1;

s1.push(11);
s1.push(22);
cout << "1: " << s1.pop() << endl;
cout << "2: " << s1.pop() << endl;
s1.push(33);
s1.push(44);
s1.push(55);
s1.push(66);
cout << "3: " << s1.pop() << endl;
cout << "4: " << s1.pop() << endl;
cout << "5: " << s1.pop() << endl;
cout << "6: " << s1.pop() << endl;
}
```

Stack гэсэн анги нь st гэсэн массив ба top гэсэн бүхэл тоон утгаас тогтож байна. Энд st нь стект байгаа өгөгдлүүдийг, top нь стекийн орой буюу хамгийн сүүлд орсон элементийн дугаарыг зааж өгөх болно. Стект шинээр элемент нэмэхэд шинэ утга массивт бичигдэж, top –ийн утга нэгээр нэмэгдэж шинэ элементийн заах юм. Харин стекээс утга авахад top –ийн утга нэгээр хорогдоно (Нэгэнт хандах боломжгүй болж байгаа учраас хуучин утгыг заавал арилгах шаардлагагүй). Стект шинээр утга нэмэхдээ push гэсэн функцийг, стекээс утга авахдаа pop гэсэн функцийг тус тус ашиглана. Программын ажиллагаа доорх байдлаар байна.

```
1: 22
2: 11
```

```
3: 66
4: 55
5: 44
6: 33
```

Үр дүнгээс харахад стект сүүлд орсон элемент эхэндээ гардаг нь тодорхой байна.

Харин одоо операторын өмнөх ба хойдох тэмдэглэгээний тухай товч үзэцгээе. Дээрх жишээний push функцэд байгаа комманд нь утга оноох үйлдлээс өмнө top-ийн утга өөрчлөгдөн гэж илэрхийлж байгаа юм.

```
st[++top]=var;
```

Харин pop функцэд байгаа доорх үйлдэл нь массивын элементийн утгыг уншсаны дараагаар top-ийн утга өөрчлөгдөхийг хэлж байгаа болно.

```
return st[top--];
```

Энэхүү Stack анги нь өгөгдөл хадгалах бүхэл механизмийг дүрсэлж байгаа учраас энэ нь OOP(Object Oriented Programming – Объект хандалтат программчлал)-ийн нэгэн жишээ юм. Өгөгдлийг хадгалах дараалал (queue), олонлог (set), холбоост жагсаалт (linked list) гэх мэтчилэн олон механизмийн тухай дараагийн бүлгүүдэд тайлбарлах болно.

Обьектүүдийн массив

Өмнөх бүлэгт бид обьект дотор массив агуулагдах тухай үзлээ. Мөн үүний яг эсрэгээр массивын элемент нь обьект байж болох тухай одоо үзэх гэж байна.

Distance төрлийн массив

Үүнээс өмнө бүлгүүдэд бид Distance гэсэн анги тодорхойлон ашиглаж байсан билээ. Одоо энэ ангийн обьектээс тогтсон массивыг ашигласан нэгэн жишээ авч үзье.

```
//englaray.cpp
#include <iostream.h>

const int max=100;

class Distance
        {
        private:
        int feet;
        float inches;

        public:
        void getdist()
                {
                cout << “\nФут: ”; cin >> feet;
                cout << “Инч: ”; cin >> inches;
                }
        void showdist()
                { cout << feet << “\’-”
                        << inches << “\””; }
        };

void main()
{
Distance dist[max];
int n=0;
char ans;
cout << endl;

do
        {
```

```
            cout << “Дугаар ” << n+1;
            dist[n++].getdist();
            cout << “Дахиад оруулах уу(y/n)?: ”;
            cin >> ans;
            }
while (ans!=’n’);

for (int j=0; j<n; j++)
            {
            cout << “\nДугаар ” << j+1 << “:”
            dist[j].showdist();
            }
}
```

Энэ программыг ашиглаж байгаа хэрэглэгч 100 хүртэл тооны бичлэгийг үүсгэн хадгалж болно. Нэг бичлэг оруулсны дараагаар дахин өөр бичлэг оруулах эсэхийг лавлана. Хэрэв оруулахгүй бол давталт төгсч, хэрэглэгчийн оруулсан бүх бичлэгүүдийг дэлгэцэнд хэвлэж гаргах юм. Энэ программын ажиллаж буй нэгэн жишээ доор өгөгдсөн байна.

```
Дугаар 1
Фут: 5
Инч: 4
Дахиад оруулах уу(y/n)?: y
Дугаар 2
Фут: 6
Инч: 2.5
Дахиад оруулах уу(y/n)?: y
Дугаар 3
Фут: 5
Инч: 10.75
Дахиад оруулах уу(y/n)?: n

Дугаар 1: 5’-4”
Дугаар 2: 6’-2.5”
Дугаар 3: 5’-10.75”
```

Мэдээж хэрэг, нэгэнт ингээд оруулсан бичлэгүүд дээр ямар ч дурын үйлдлийг хийж болно.

Массивын хязгаарын тухайд

Энэ программ do гэсэн давталтыг ашиглан хэрэглэгчээс бичлэгүүдийг авч байна. Харин массивын элементийн тоог max буюу 100 гэж тогтоосон байгаа. Хэрэв хэрэглэгч 100-аас олон бичлэг оруулвал яах вэ? Энэ тохиолдолд дээрх программ үл ойлгогдох тодорхойгүй гэмтэлд хүрэх нь ойлгомжтой. Учир нь хэрэглэгчийн оруулсан бичлэг хэдэн ширхэг болж байгааг хаана ч тооцохгүй байгаа юм. Хэрэв программ массивын төгсгөлөөс хойш өгөгдөл бичвэл түүнийг компилятор ч, үйлдлийн систем ч аль аль нь мэдэхгүй. Тэрхүү өгөгдөл нь дараагийн хувьсагчийн утгыг өөрчлөх, эсвэл программын биелэх кодыг өөрчлөх зэргээр буруу нөлөө үзүүлэх юм.

Ийм байдал нь программчлагчийг массивтай ажиллаж байх үед түүний хязгаарыг үргэлж шалгаж байхыг шаардаж байгаа юм. Хэрэв массив дүүрвэл хэрэглэгчид мэдээлэл өгөх доорх мөрүүдийг өмнөх программын do давталт дотор хийж өгөх хэрэгтэй юм.

```
If (n>=max)
        {
        cout << “\nМассив дүүрэн байна!!!”;
        break;
        }
```

Энэ мөрүүд нь мэдээллийг хэрэглэгчид харуулаад давталтыг таслах болно.

Массив доторх обьекттэй ажиллах нь

Энэ программ дахь анги нь яг өмнөх программуудад тодорхойлогдож байсантайгаа нэгэн адил тодорхойлогдоно. Харин үндсэн программд энэ ангийн нэг хувьсагч биш 100 элементтэй массив тодорхойлогдож байна. Ангийн гишүүн функцэд хандахын тулд доорх маягийн бичлэг шаардагдах болно.

```
dist[j].showdist();
```

Юуны өмнө массивын нэр ба хашилт бүхий массивын индекс байна. Дараа нь обьектийн талбарт хандах цэг үйлдэл, эцэст нь гишүүн хувьсагч болон функцийн нэр байна. Харин getdist функцийг дуудахад дараах бичлэгийг ашиглана.

```
dist[n++].getdist();
```

Энд массивын индекс дээр шууд нэмэгдүүлэх үйлдлийг (increment operator) гүйцэтгэж байгаа юм. Энэ нь ердийн ашиглагддаг “n++” гэсэн үйлдлээс огт ялгаагүй бөгөөд тодруулан хэлбэл массивын элементэд хандахын өмнө n-ийн утга нэгээр нэмэгдэхийг зааж өгч байна.

Хөзрийн массив

Обьектоос тогтсон массивын өөр нэгэн жишээг авч үзье. Энэ программ нь өмнө бүлэгт үзсэн CARDOBJ гэсэн жишээ программын үргэлжлэл болно. Тэр жишээнд тодорхойлж байсан card обьектоос

тогтсон 52 элемент бүхий массив үүсгэн ашиглах болно.

```
//cardaray.cpp
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

enum Suit { clubs, diamonds, hearts, spades };

const int jack=11;
const int queen=12;
const int king=13;
const int ace=14;

enum Boolean { false, true };

class card
        {
        private:
        int number;
        Suit suit;

        public:
        card()
                { };
        void init(int n, Suit s)
                { Suit=s; number=n; }
        void display();
        Boolean isEqual(card);
        };

void card::display()
        {
        if (number>=2 && number<=10)
                cout << number;
        else
                switch (number)
                        {
                        case jack: cout << “J”; break;
                        case queen: cout << “Q”; break;
                        case king: cout << “K”; break;
```

```
                        case ace: cout << “A”; break;
                        }
        switch (suit)
                {
                case clubs: cout << char(5); break;
                case diamonds: cout << char(4); break;
                case hearts: cout << char(3); break;
                case spades: cout << char(6); break;
                }
        }

void main()
{
card deck[52];
int num;
Suit su;
cout << endl;
for (int j=0; j<52; j++)
        {
        int num=(j%13)+2;
        Suit su=Suit(j/13);
        deck[j].init(num, su);
        }
cout << “\nЭрэмбэлэгдсэн хөзрүүд:\n”;
for (j=0; j<52; j++)
        {
        deck[j].display();
        cout << “ “;
        if (!((j+1) % 13))
                cout << endl;
        }
randomize();
for (j=0; j<52; j++)
        {
        int k=random(52);
        card temp=deck[j];
        deck[j]=deck[k];
        deck[k]=temp;
        }
cout << “\nХолилдсон хөзрүүд:\n”;
for (j=0; j<52; j++)
        {
        deck[j].display();
```

```
        cout << “, “;
        if (!((j+1) % 13))
                cout << endl;
        }
}
```

Энэ программ эхлээд массивын элементүүдэд бүх хөзрийн утгыг дэс дараалан олгоод дэлгэцэнд хэвлэж байна. Дараа нь хөзрүүдийг холиод, дахин хэвлэж байгаа юм. Дэлгэц дээр бунд, дөрвөлжин, цэцэг, гэлийг дүрслэхдээ IBM-ийн стандарт ASCII кодыг ашиглана гэж тооцлоо. Үүнтэй холбогдуулан доорх хэдэн ойлголтыг үзэх нь зүйтэй.

IBM-ийн стандарт тэмдэгтүүд

Компьютерийн текст горимд дүрслэгддэг бүх тэмдэгтүүдийг агуулсан массивыг ASCII кодын хүснэгт гэж нэрлэдэг. Энэхүү ASCII кодын тэмдэгт бүр өөрийн индекс буюу ASCII дугаартай байна. 32-оос өмнөх дугаартай тэмдэгтүүдийг тусгай тэмдэгтүүд гэнэ. Өмнөх программын display() гэсэн процедур дотор дэлгэц дээр цэцэг, дөрвөлжин, бунд гэлийг дүрслэхийн тулд харгалзан 5, 4, 3, 6 дугаартай ASCII тэмдэгтүүдийг ашиглаж байгаа юм. Программ дэлгэцэнд доорх үр дүнг харуулах болно.

```
Эрэмбэлэгдсэн хөзрүүд:
2□ 3□ 4□ 5□ 6□ 7□ 8□ 9□ 10□ J□ Q□ K□ A□
2□ 3□ 4□ 5□ 6□ 7□ 8□ 9□ 10□ J□ Q□ K□ A□
2□ 3□ 4□ 5□ 6□ 7□ 8□ 9□ 10□ J□ Q□ K□ A□
2□ 3□ 4□ 5□ 6□ 7□ 8□ 9□ 10□ J□ Q□ K□ A□

Холилдсон хөзрүүд:
3□ 9□ 6□ K□ 8□ 4□ 7□ 4□ 3□ 3□ A□ 2□ 9□
6□ 7□ 9□ 8□ Q□ Q□ 10□ J□ 6□ 4□ J□ K□ 5□
```

3□ J□ 5□ K□ Q□ 10□ 8□ 2□ 6□ A□ 4□ J□ 8□
10□ 2□ Q□ 10□ 5□ A□ K□ 7□ 5□ A□ 2□ 9□ 7□

Санамсаргүй тоо

Программ зохиогчийн хувьд тоог санамсаргүйгээр сонгож авах шаардлага олонтаа гардаг. Энэ жишээ программд хөзрийг холихын тулд бид санамсаргүй тоог ашиглаж байгаа юм. C++-д санамсаргүй тоо гаргаж авахын тулд хоёр алхам хэрэгтэй болдог. Эхний алхамд санамсаргүй тоо үүсгэгчийг ачаалж, ажилуулах шаардлагатай. Үүнийг хийхийн тулд randomize() гэсэн функцийг дуудаж байна. Энэ функцийг ачаалахын тулд stdlib.h, time.h гэсэн 2 толгой файлыг ашиглах болно.

Санамсаргүй тоог гаргаж авах ажиллагааг random() гэсэн функцийг ашиглан гүйцэтгэнэ. Уг функц ганц аргументтэй бөгөөд энэ аргумент нь гарч ирэх санамсаргүй тооны дээд хязгаарыг зааж өгөх юм. 0-оос 51-ийн хооронд санамсаргүй утга гаргаж авахын тулд random (52) гэсэн бичлэгийг ашиглаж байна. Бид бүх хөзрөөрөө давталт хийж, хөзөр бүрийг санамсаргүй гаргасан өөр хөзөртэй солих зарчмаар нийт хөзрийг хольж байгаа болно.

Тэмдэгт мөр

Массивууд дотроос хамгийн түгээмэл ашиглагддаг тэмдэгт мөр хэмээх төрлийн тухай энэ бүлэгт үзнэ. Тэмдэгт мөр гэдэг нь тэмдэгтүүдийн массив бөгөөд ийм төрлийн хувьсагч болон тогтмолууд байж болно.

Тэмдэгт мөр төрлийн хувьсагч

Тэмдэгт мөр төрлийн хувьсагчийг ашигласан дараах хялбар жишээг үзье.

```
//stringin.cpp
#include <iostream.h>
const int max=80;

void main()
{
char str[max];

cout << “\nМөр оруулж өгнө үү: ”;
cin >> str;
cout << “Таны оруулсан мөр: ” << str;
}
```

Тэмдэгт мөр төрлийн хувьсагчийг тодорхойлох нь дурын массивыг тодорхойлохоос бичлэгийн хувьд огт ялгаагүй юм. Харин заавал индекс ашиглалгүйгээр бүхэл массивтай ажиллаж болдгоороо бусад төрлийн массиваас ялгаатай байдаг юм. Жишээ нь, энэ программд cin урсгалаас тэмдэгтүүдийг уншиж авахын тулд зөвхөн тэмдэгт мөр массивын нэрийг л ашигласан байна. Хэрэглэгчийн гараас дарсан товч бүр уг массивын нэг элемент болж бичигдэх болно.

Нэг тэмдэгт бүр санах ойд нэг байт эзэлнэ. C++-ийн массивт бичсэн байгаа тэмдэгт мөр бүр заавал 0-р тэмдэгтээр төгссөн байх ёстойг мэдэж авах нь зүйтэй. Энэхүү төгсгөлийг null тэмдэгт гэх бөгөөд cout урсгал тэмдэгт мөрийг дэлгэцэнд хэвлэхдээ энэхүү null тэмдгээр мөрийн төгсгөлийг мэдэрдэг байна.

Хэрэв хэрэглэгч тэмдэгт мөрийн уртаас илүү гарсан өгөгдөл оруулж өгвөл бусад массивын нэгэн

адил тэрхүү илүү гарсан өгөгдөл нь дараагийн хувьсагчийн утгыг өөрчлөн бичигдэнэ. Үүнээс болж программд болон цаашилбал системд аюул учруулж магагдгүй юм. Гэхдээ C++-д гараас оруулж буй өгөгдлийн уртыг хязгаарлах боломж байдаг байна.

```
//safetyin.cpp
#include <iostream.h>
#include <iomanip.h>
const int max=20;

void main()
{
char str[max];

cout << "\nМөр оруулж өгнө үү: ";
cin >> setw(max) >> str;
cout << "Таны оруулсан мөр: " << str;
}
```

Стандарт оролтын урсгалын setw гэсэн функцийг ашиглан гараас оруулах өгөгдлийн уртыг хязгаарладаг байна. Өөр нэг анхаарах зүйл бол дээрх программ str гэсэн тэмдэгт мөрөнд 19-өөс олон тэмдэгт авахгүй. Үүний учир нь, өмнө дурдсан мөрийн төгсгөл дэх null тэмдэгт 1 байт эзлэх ёстой юм.

Тэмдэгт мөр төрлийн тогтмол

Бусад бүх төрлийн нэгэн адил тэмдэгт мөрийг тодорхойлох үедээ анхны утга оноон өгч болно.

```
//strinit.cpp
#include <iostream.h>

void main()
{
```

```
char str[]=”Thou art too dear for my possessing”;
cout << str;
}
```

Тэмдэгт мөрд өгч байгаа тогтмол утга нь энгийн өгүүлбэрийн зарчмаар, ердөө л хашилтан дотор бичсэн байна. Уг нь массивт анхны утга оноохдоо элемент бүрийн утгыг таслалаар зааглан, хаалтан дотор бичиж өгдөг билээ. (Гэхдээ тийм бичиглэлийг ч ашиглаж болно). Энэ нь C++-ийн зүгээс программ зохиогчид үзүүлж байгаа хөнгөлөлт гэж ойлгож болох юм.

Хоосон зай агуулсан тэмдэгт мөр

Өмнө үзсэн жишээ программд хэрэв та гараас хэд хэдэн үгээс тогтсон өгүүлбэр оруулж өгсөн бол буцаад дэлгэцэн дээр уг өгүүлбэр тань биш, харин зөвхөн түүний эхний үг л хэвлэгдэнэ. Ө.х.

```
Мөр оруулж өгнө үү: Law is bottomless pit.
Таны оруулсан мөр: Law
```

C++-ийн стандарт оролтын урсгал cin нь хоосон зай буюу space-ийг өгөгдөл зааглах тэмдэг гэж ойлгодогт гол учир байгаа юм. Жишээ нь: Программ дотор

```
cin >> var1 >> var2 >> var3;
```

гэсэн комманд байжээ. Хэрэглэгч гараас өгөгдлийг өгөхдөө

```
15 25 35
```

гэж бичихэд л хангалттай. Ингэсэн тохиолдолд var1 нь 15, var2 нь 25, var3 нь 35 гэсэн утгуудыг харгалзан авна. Үүнтэй яг ижлээр гараас оруулсан өгөгдлийн зөвхөн эхний үгийг тэмдэгт мөрд авах бөгөөд үлдсэн өгөгдлүүд хаана ч орохгүй хаягдах болж байгаа юм. Харин space агуулсан тэмдэгт мөрийг авах шаардлага гарвал бид өөр функц ашиглах шаардлагатай болно.

```
//blanksin.cpp
#include <iostream.h>
const int max=80;

void main()
{
char str[max];

cout << “\nМөр оруулж өгнө үү: ”;
cin.get(str, max);
cout << “Таны оруулсан мөр: ” << str;
}
```

Олон мөр бүхий өгөгдөл

Хоосон мөр агуулсан тэмдэгт мөрийг ингэж авдаг бол олон мөрөөс тогтсон өгөгдлийг яаж авах вэ? Үүнийг хийхийн тулд өмнөх cin урсгалын get гэсэн функцэд 3 дахь аргументыг өгөх шаардлагатай болно. Энэхүү 3 дахь аргументээр ямар тэмдэгт орж иртэл тэмдэгт мөрийг уншихыг зааж өгөх юм. Ө.х. энэхүү 3 дахь аргументийн стандарт утга нь Enter товч байдаг гэсэн үг. Одоо жишээгээ үзэцгээе.

```
//linesin.cpp
#include <iostream.h>
const int max=2000;
char str[max];
```

```
void main()
{
cout << “\nМөр оруулж өгнө үү:\n”;
cin.get(str, max, ‘$’);
cout << “Таны оруулсан мөр:\n” << str;
}
```

Одоо та хэд л бол хэдэн мөр өгөгдлийг оруулах боломжтой боллоо. Хэрэглэгч ‘$’ товч дартал эсвэл өгөгдлийн урт 2000-аас давтал гараас тэмдэгт мөрийг авсаар байх болно.

```
Мөр оруулж өгнө үү:
Аргалын утаа боргилсон
Малчны гэрт төрсөн би
Атар хээр нутгаа
Өлгий минь гэж санадаг.$

You entered:
Аргалын утаа боргилсон
Малчны гэрт төрсөн би
Атар хээр нутгаа
Өлгий минь гэж санадаг.
```

Мөрийг хувилах нь

Бид мөртэй ажиллахдаа массив гэсэн утгаар нь авч үзвэл тэмдэгт тус бүртэй ажиллах болно. Дараагийн программ тэмдэгт мөрийг давталтын аргаар хувилж байна.

```
//strcopy1.cpp
#include <iostream.h>
#include <string.h>
const int max=80;

void main()
```

```
{
char str1[]=”Энэ бол миний төрсөн нутаг”
          “Монголын сайхан орон”;

char str2[max];

for (int j=0; j<strlen(str1), j++)
           str2[j]=str1[j];
str2[j]=’\0’;
cout << endl << str2;
}
```

Энэ программд 2 тэмдэгт мөр тодорхойлогдож байгаа бөгөөд нэгнийх нь анхны утгыг оноож өгсөн байна. Дараа нь давталт ашиглан str1-ийн тэмдэгт бүрийг str2-д хувилж байгаа юм. Мөн энэ программд бидний өмнө ашиглаж байгаагүй нэг толгой файлыг ашиглаж байна. Энэ бол string.h юм. Тус файлд тэмдэгт мөртэй ажилладаг нэлээн хэдэн функц агуулагдсан байдаг юм. Эндээс бид strlen() гэсэн функцийг ашиглалаа. Энэ нь аргументэд өгсөн тэмдэгт мөрийг уртыг өгөх бгөөд тэрхүү уртыг ашиглан бид str1-ийн бүх тэмдэгтэй ажиллаж боломжтой болно. Харин энэ функц нь null тэмдэгтийг тооцдоггүй учраас шинээр хувилагдсан str2 тэмдэгт мөрийн төгсгөлд бид null тэмдэгтийг бичиж өгөх ёстой. Хэрэв энэ тэмдэгтийг нэмэлгүйгээр уг тэмдэгт мөрийг дэлгэцэнд хэвлэвэл жинхэнэ утгын араас хамаагүй тэмдэгтүүд дэлгэцэнд хэвлэгдэн гарахад хүрэх юм.

Гэхдээ тэмдэгт мөрийг хувилахад дээрх давталтын аргыг хэрэглэх нь тун ч болхи хэрэг бөгөөд дээр дурдсан толгой файлд байдаг стандарт функцүүдийг ашиглах нь зүйтэй юм.

```
//strcopy2.cpp
#include <iostream.h>
#include <string.h>
const int max=80;
```

```
void main()
{
char str1[]=”Энэ бол миний төрсөн нутаг”
          “Монголын сайхан орон”;

char str2[max];

strcpy(str2, str1);
cout << endl << str2;
}
```

Анхааруулахад, шинэ тэмдэгт мөр 1-р аргументэд, хувилагдах хуучин мөр нь 2-р аргументдээ өгөгдөнө.

Тэмдэгт мөрийн массив

Массивын массив байж болдгийн нэгэн адилаар тэмдэгт мөрийн массив бас байж болно.

```
//straray.cpp
#include <iostream.h>

const int days=7;
const int max=10;

void main()
{
char star[days][max]={“Ням”, “Даваа”,
                      “Мягмар”, “Лхагва”,
                      “Пүрэв”, “Баасан”,
                      “Бямба” };

for (int j=0; j<days; j++)
          cout << star[j] << endl;
}
```

Программ ажиллаад дэлгэцэнд доорх үр дүнг хэвлэж гаргана.

Ням
Даваа
Мягмар
Лхагва
Пүрэв
Баасан
Бямба

Тэмдэгт мөр нь өөрөө массив учраас тэмдэгт мөрийн массив гэдэг бол мөн л массивын массив болох юм. Эхний индекс нь массивт хэдэн тэмдэгт мөр байхыг, харин хоёр дахь индекс нь тэмдэгт мөр тус бүр ямар урттай байхыг тус тус зааж өгөх болно. Харин ийм бүтцийн нэг дутагдал нь массив доторх тэмдэгт мөрүүд заавал ижил урттай байх шаардлага юм. (Үүнийг заагч ашиглаж л засч болно)

Тэмдэгт мөрийг ангийн гишүүн болгон ашиглах нь

```
//strpart.cpp
#include <iostream.h>
#include <string.h>

class part
        {
        private:
        char partname[30];
        int partnumber;
        float cost;

        public:
        void setpart(char pname[], int pn, float c)
                {
```

```
                strcpy(partname, pname);
                partnumber=pn;
                cost=c;
                }
        void showpart()
                {
                cout << "\nНэр=" << partname;
                cout << ", дугаар=" << partnumber;
                cout << ", үнэ=₮" << cost;
                }
        };

void main()
{
part part1, part2;
part1.setpart("Деталь 1", 4473, 217.55);
part2.setpart("Деталь 2", 9924, 419.25);
cout << "\n1-р бие: "; part1.showpart();
cout << "\n2-р бие: "; part2.showpart();
}
```

Программд part ангийн 2 обьект тодорхойлж, тэдгээрийн setpart функцүүдийг нь ашиглан гишүүдийнх нь анхны утгуудыг тодорхойлж өгч байна. Дараа нь тэдгээрийг showpart функцүүдийг ашиглан дэлгэцэнд харуулна.

```
1-р бие:
Нэр=Деталь 1, дугаар=4473, үнэ=₮217.55

2-р бие:
Нэр=Деталь 2, дугаар=9924, үнэ=₮419.25
```

Аргументэд ирсэн тэмдэгт мөрийг гишүүн тэмдэгт мөр лүү хувилахын тулд өмнө үзсэн strcpy() функцийг ашиглаж байна. Уг толгой файлд тэмдэгт мөрүүдийг хувилахаас гадна, хоёр тэмдэгт мөрүүдийг нийлүүлэх, жиших гэх мэтчилэн үйлдлүүдийг гүйцэтгэх боломжтой функцүүд агуулагдаж байдаг.

Хэрэглэгчийн тодорхойлсон тэмдэгт мөр төрлүүд

С++-д байгаа стандарт тэмдэгт мөрд бас дутагдалтай тал байдаг. Жишээ нь,

```
str2=str1
```

гэсэн үйлдлийг шууд хийх боломжгүй. Харин өөрийн тэмдэгт мөр төрлийг зохиовол энэ мэт асуудлуудаа шийдвэрлэх боломжтой болох юм. Доорх программд string гэсэн анги тодорхойлжээ.

```
//strobj.cpp
#include <iostream.h>
#include <string.h>

const int sz=80;

class String
        {
        private:
        char str[sz];

        public:
        String()
                { str[0]="\0"; }
        String(char s[])
                { strcpy(str, s); }
        void display()
                { cout << str; }
        void concat(String s2)
                {
                if (strlen(str)+strlen(s2.str) < sz)
                        strcat(str, s2.str);
                else
                        cout << "\nМөр дэндүү урт байна";
                }
        };
```

```
void main()
{
String s1(“Шинэ жилийн мэнд хүргэе!”);
String s2=“Баяр хүргэе!”;
String s3;

cout << “\ns1=”; s1.display();
cout << “\ns2=”; s2.display();
cout << “\ns3=”; s3.display();

s3=s1;
cout << “\ns3=”; s3.display;

s3.concat(s2);
cout << “\ns3=”; s3.display;
}
```

Энгийн тэмдэгт мөр нь ердөө л тэмдэгтийн массив байдаг бол бидний зохиосон энэхүү анги нь зөвхөн тэмдэгтийн массив төдийгүй хэд хэдэн функцийг өөртөө агуулсан байна. Мөн түүнчлэн, анги болгосны үр дүнд дээр өгүүлсэн шууд утга оноох үйлдлийг ч хийх боломж гарч ирж байгаа юм.

Дээрх ангид байгаа тэмдэгт мөрүүд нь sz буюу 80 тэмдэгтийн урттай байна. Уг анги нь 2 байгуулагч функцтэй байна. Эхний байгуулагч нь хоосон тэмдэгт мөр үүсгэдэг байхад дараагийнх нь тэмдэгт мөрд аргументэд өгсөн утгыг оноож өгдөг юм байна.

```
String s3;
String s1(“Шинэ жилийн мэнд хүргэе!”);
```

Мөн бүх нэг аргументтэй байгуулагчдыг нэгэн адилаар 2 дахь байгуулагч функцийг доорх маягаар дуудаж болно.

```
String s1=“Шинэ жилийн мэнд хүргэе!”;
```

Мөн түүнчлэн тэмдэгт мөрийг дэлгэцэнд хэвлэдэг display гэсэн функц, хоёр тэмдэгт мөрүүдийг нэмдэг concat гэсэн функцүүдийг агуулсан байна.

```
String s3(“Шинэ жилийн мэнд хүргэе!”);
String s2=“Баяр хүргэе!”;

s3=s1;
s3.concat(s2);
cout << “\ns3=”; s3.display;
```

гэсэн үйлдлүүдийн үр дүнд дэлгэц дээр “Шинэ жилийн мэнд хүргэе! Баяр хүргэе!” гэсэн текст хэвлэгдэх юм. Дээрх программын нийт үр дүн нь доорх байдалтай байна.

```
s2=Баяр хүргэе!
s3=
s3=Шинэ жилийн мэнд хүргэе!
s3=Шинэ жилийн мэнд хүргэе! Баяр хүргэе!
```

Массивыг зөв зохион байгуулах нь

Бидний бичиж буй программд бүх аймгуудын нэрийг массивт хадгалах шаардлага гарлаа гэж үзье. Үүнийг хадгалах нэг арга бол тэмдэгт төрлийн 2 хэмжээст массив ашиглах явдал юм. Массивын эхний индекс нь аймгийн дугаарыг, дараагийн индекс нь аймгийн нэрний үсгийн дугаарыг зааж өгөх юм. 18 аймаг байх учраас массивын эхний хэмжээс 18 байж болно. Харин хоёр дахь хэмжээс нь аймгуудын нэрнүүдийн хамгийн урттай нь тэнцүү байж таарна. Ө.х 10 (Баянхонгор, Өвөрхангай, Баян-Өлгий) бөгөөд аймаг бүрийн нэр 10 байт санах ой ашиглана гэсэн үг.

Ингэж зохион байгуулах нь программчлагчийн хувьд хялбар байж болох боловч санах ойг тун ч хэмнэлтгүй ашиглаж байгаа хэрэг юм. Жишээ нь, Увс гэхэд 10 байтын ердөө 3 нь ашиглагдаж, цаана нь 7 байт ашиглалтгүй хаягдаж байна. Иймэрхүү маягаар том хэмжээний массив үүсгэвэл ямар их санах ойг сул хаях нь ойлгомжтой юм.

Харин үүний оронд массивын 2 дахь хэмжээсийг тогтмол бусаар, харгалзах нэрнүүдэд тааруулан сонгож авах нь зүйтэй юм. Тухайлбал Увс гэхэд л 4 байт, Завхан гэхэд л 7 байтыг л ашиглана(тэмдэгт мөр 0 тэмдгээр төгсөх ёстой учраас нэг нэг байт илүү авагдана). Энэ тухай дараагийн жишээгээр харуулъя.

```
char Aimags1[ 3 ][ 11 ]=
   { “Архангай”, “Баянхонгор”, “Баян-Өлгий” };

char *Aimags2[ 3 ]=
   { “Архангай”, “Баянхонгор”, “Баян-Өлгий” };
```

Энд ашиглагдаж байгаа бичлэг нь заагч хувьсагч тодорхойлж байгаа хэрэг юм. Заагчийн тухай дэлгэрэнгүй ойлголтыг 7-р бүлэгт тайлбарлана. Бид “Архангай” гэсэн үгний “х” үсгэнд хандахын тулд дараах бичлэгүүдийг ашиглана.

```
Aimags1[0][2]
Aimags2[0][2]
```

Хэдийгээр эдгээр нь бичлэгийн хувьд адилхан байгаа боловч санах ойд үүсэх болон тэдэнд хандах байдал нь тун их ялгаатай.

Эхний мөрийг авч үзье.

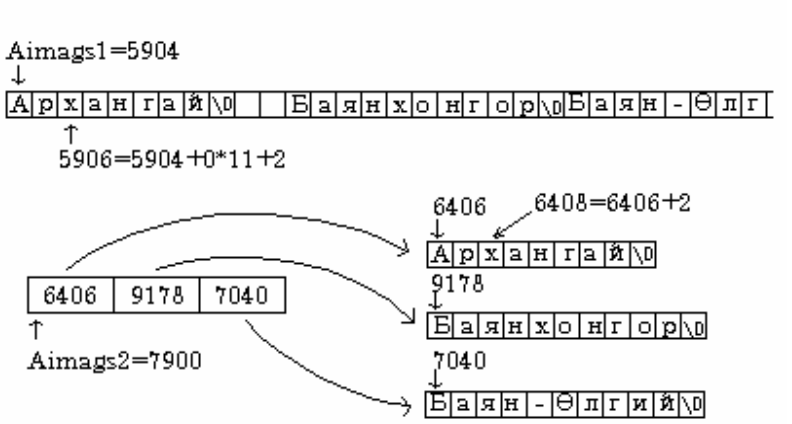
```
Aimags1[0][2]
```

Түүний 0 гэсэн индекс нь Aimags1 массивын 1-р элементэд хандахыг зааж байна. Түүнд хандахын тулд массивын санах ойд байрлаж буй хаягаас 0*11 байтаар шилжинэ. Учир нь Aimags1 массивын нэг элемент бүр нь 11 байтыг эзэлж байгаа шүү дээ. Дараагийн 2 гэсэн индекс нь уг шилжилтэсээ дахиад 2 байтаар шилжиж уг тэмдэгт мөрийн 3 дахь элементэд хандахыг хэлж өгч байна.

Тэгвэл 2 дахь мөрийг товч авч үзье.

Aimags2[0][2]

Түүний 0 гэсэн индекс нь Aimags2 гэсэн заагч массивын 1-р элементэд хандахыг зааж байгаа бөгөөд ингэхдээ массивын санах ой дахь эхлэлээс 0*4 байтаар шилжинэ. Санах ойн хаягийг хадгалахад 4 байт шаардагддаг учраас ингэж үржиж байгаа юм. Үүний дараа 2 гэсэн индексийн дагуу олсон хаягаасаа дахин 2 байт шилжиж шаардлагатай үсгэн дээр хүрч байгаа юм.



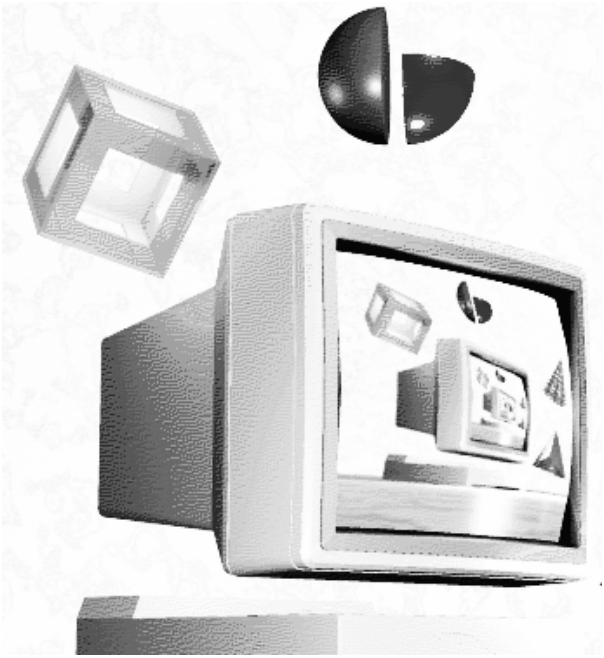
Бүлгийн дүгнэлт

Массив гэдэг нь ижил төрлийн бүлэг өгөгдлийг хэлнэ. Массивт байгаа өгөгдлүүдийг элементүүд гэнэ. Массивын элементийн төрөл нь дурын энгийн төрлүүд төдийгүй, бүтэц болон анги байж болно. Элемент бүрт харгалзах нэг тоо байх бөгөөд түүнийг индекс гэж нэрлэнэ. Массивыг анхлан тодорхойлох үед түүний элементүүдэд шууд анхны утга оноож болно. Массив нь хэд хэдэн хэмжээстэй байж болно. Жишээ нь, 2 хэмжээст массив гэдэг бол массивын массив юм.

Массивыг функцийн аргументэд өгвөл бусад төрлийн адил түүний утга нь хувилагдан очдоггүй бөгөөд харин массивын хаяг нь дамждаг. Массив обьектийн гишүүн байж болно.

Тэмдэгтүүдийн массивыг тэмдэгт мөр гэж нэрлэнэ. Тэмдэгт мөр бүхэн 0-р тэмдэгт буюу null тэмдэгтээр төгсөж байх ёстой. Тэмдэгт мөртэй ажиллах функцүүд string.h гэсэн толгой файлд агуулагдаж байдаг. Тэмдэгт мөрийн массив байж болох бөгөөд энэ нь массивын массив буюу 2 хэмжээст массивтай яг ижил ойлголт юм. Тэмдэгт мөрийг анги болон бүтцийн гишүүн болгон ашиглаж болно.

7-р бүлэг



-Текст горим
-График горим
-Өнгө
-Геометр дүрсүүд
-Дуу гаргах ба дүрс хөдөлгөх
-График горимын текст

7-р бүлэг

Turbo C++ болон Borland C++-ийн аль аль нь графиктай холбоотой функцүүдийг агуулсан толгой файлуудтай байдаг. Эдгээр функцүүд таны өмнө өөр нэгэн шинэ ертөнцийг нээх болно. Ө.х. та эдгээр функцүүдийг ашигласнаар дэлгэц дээр цөөн тооны хэдэн тэмдэгтийг биш, харин өөрийн хүссэн дурын зүйлийг зурах боломж гарч ирж байгаа юм. График болон обьект хандалтат программчлал нь хоорондоо холбогдон ажиллахад маш тохиромжтой бөгөөд дэлгэц дээр байгаа нэг дүрсийг нэг обьект болгон зохион байгуулахад тун эвтэйхэн байдаг.

Turbo C++-ийн график функцүүд нь текст болон график горимын гэсэн 2 ангилалтай байдаг.

Текст горимын функцүүд нь текстийг дэлгэцийн хүссэн хэсэгт янз бүрийн форматаар бичихэд зориулагдана. Ийм функцүүд нь дурын дэлгэц болон адаптертай зохицон ажиллах тул та текст горимын функцүүдийг монохром (ганц өнгөтэй дэлгэц) системд ч ашиглаж болно. Мөн IBM-ийн стандарт тэмдгүүдийг ашиглан текстийн хүрээ мэтийн хялбар дүрсүүдийг ч текст горимд хэвлэж болно.

Харин график горимын функцүүд нь CGA, EGA, VGA, SVGA гэсэн адаптерүүдийг шаарддаг. (Дэлгэцийн адаптерын тухай хойно гарах тайлбарыг уншаарай) График горимын функцүүд нь цэг, шугамаас эхлээд тойрог, тэгш өнцөгт зурах, битүү хүрээн доторх хэсгийг будах зэрэг график үйлдлүүдийг гүйцэтгэх чадвартай.

Мөн энэ бүлэгт график дүрсүүдийг хэрхэн обьект маягаар зохион байгуулах тухай үзэх юм. Хэдийгээр обьектийн тухай шинэ ойлголт нэмэхгүй ч өмнө үзсэн ойлголтыг нэлээн гүнзгийрүүлж өгөх зорилготой юм.

Бид графикийн тухай бүх функцийг энэ бүлэгт үзэж амжихгүй тул хамгийн түгээмэл ашиглагдах, чухал хэдхэн функцийг түүвэрлэн үзэж байна.

Текст горимын функцүүд

Компьютер анхлан асаахад автоматаар текст горимд шилждэг. Текст хэмээх энэхүү горимд дэлгэц хэсэг хэсэг матрицуудад хуваагдах бөгөөд нийт 80 тэмдэгт багтаах 25 мөрөөс тогтоно. (Бас 40x25, 80x43-ийн горимууд гэж байна) Бидний өмнө үзэж байсан cout гэсэн стандарт урсгал нь хамгийн энгийн текст горимын функц болно. Текст горимын нэмэгдэл функцүүд нь бүтэн дэлгэц төдийгүй цонхонд ч ажиллах чадвартай байдаг. Цонх гэдэг нь дэлгэцний нэг хэсэг тэгш өнцөгт мужийг хэлнэ.

Энд сануулахад, эдгээр үзэж байгаа функцүүд нь зөвхөн MS-DOS үйлдлийн системд л ажиллана. Өөр бусад UNIX мэтийн систем, Turbo болон Borland C++-ээс бусад компилятор дээр ажиллахгүй.

Window() функц

Энэ функцээр дэлгэцэнд цонх үүсгэх бөгөөд функцийг ашиглахын тулд conio.h гэсэн толгой файлыг зааж өгсөн байх ёстой. Уг функц 4 аргумент авах бөгөөд тэдгээр нь харгалзан цонхны зүүн, дээд, баруун, доод координатуудыг илэрхийлж байна. Энэ функцийг ашиглан цонх үүсгэсэн тохиолдолд дэлгэцэнд текст хэвлэх функцүүд нь уг цонхон дотор ажиллах болно. (Дэлгэцний стандарт цонх нь бүтэн дэлгэц байдаг.) Цонхонд багтахгүй текст цонхны

дараагийн мөрөнд үргэлжлэн гарна. Хэрэв цонхны хамгийн доод мөр дууссан бол илүү гарсан өгөгдөл хэвлэгдэхийн тулд цонхон дахь өгөгдөл дээш гүйх (scroll) болно.

Анхааруулахад, дэлгэц дээр үүссэн цонхонд автоматаар хүрээ болон өнгө үүсэхгүй учраас бид текст бичтэлээ цонх хаана үүссэнийг мэдэх боломжгүй. Текст горим нь компьютерийн стандарт горим тул текст горимын функцүүдийг ашиглахын тулд ямар нэгэн бэлтгэл функц ажилуулах шаардлагагүй.

```
//window.cpp
#include <iostream.h>
#include <conio.h>

const int len=500;

class box
        {
        private:
        int left, top, right, bottom;
        char str[len];

        public:
        box(int l, int t, int r, int b, char s[])
                {
                left=l; top=t; right=r; bottom=b;
                strcpy(str, s);
                window(left, top, right, bottom);
                cputs(str);
                }
        void erase()
                {
                window(left, top, right, bottom);
                clrscr();
                }
        };

void main()
```

```
{
clrscr();

char s1[]="The disk drive is jammed\n\r\
        or on fire, or the CPU\n\r\
        is flooded. Correct the\n\r\
        situation and try again. \n\r\
        Press Enter to continue: ";

char s2[]="This is a string that will wrap at the\
        right edge of the window, probably\
        breaking words inappropriately in the\
        process.";

box a(25, 6, 55, 18, s1);
getch();
a.erase();

box b(1, 1, 25, 10, s2);
getch();

}
```

Дээрх box гэсэн ангийн байгуулагч функц нь 5 аргумент авч байна. Эхний 4 нь цонхны 4 талын координат, харин сүүлийн аргумент нь уг цонхонд хэвлэгдэх текстийг өгөх юм.

cputs() функц

Гаралтын стандарт функц cout-ийг ашиглан дэлгэцэн дээр өгөгдөл хэвлэхэд тэр нь идэвхтэй цонхноос хамаарахгүй гардаг. Тэр ч бүү хэл, дэлгэцтэй ажиллах өөр ихэнх функцүүд ч цонхноос хамаардаггүй. Тийм учраас цонх ашиглаж байгаа тохиолдолд цөөн хэдэн функцийг л ашиглах боломжтой. Эдгээрийн дотроос хамгийн түгээмэл ашиглагдах нь энэхүү cputs() функц юм.

Харин энэ функцийн дутагдал нь зөвхөн ганц ‘\n’ гэсэн тэмдгээр шинэ мөрөнд шилжүүлэх үйлдэл хийдэггүй. Үнэн хэрэгтээ, шинэ мөрөнд шилжих үйлдэл нь 13-р тэмдэгт (мөр шилжүүлэх), 10-р тэмдэгт (мөрийн эхлэлд аваачих) гэсэн 2 тэмдгээр хийгддэг юм. Эхний тэмдэг нь текст курсорыг эгц доош нь дараагийн мөрөнд аваачиж, харин 2 дахь тэмдэгт нь уг мөрийнхээ эхлэлд авчирдаг байна. Гаралтын стандарт функц нь үүнийг товчлоод зөвхөн ‘\n’ буюу 13-р тэмдгээр шинэ мөрөнд шилжүүлдэг. Харин энэхүү cputs() функц нь зөвхөн ‘\n’ гэж бичиж өгвөл дээр дурдсанаар зөвхөн доод талын мөр лүү шилжүүлж орхино. Уг мөрийнх нь эхлэлд авчирахын тулд ‘\r’ буюу 10-р тэмдгийг нэмж өгөх шаардлагатай болж байгаа юм. Ийм учраас мөр шилжүүлэхийн тулд “\n\r” гэсэн коммандыг ашиглаж байна.

clrscr() функц

clrscr() функц нь цонхонд байгаа текстийг арилгаж цэвэрлэдэг болно. Энэ функц дээрх жишээнд 2 удаа ашиглагджээ. Эхний удаад (ямар нэгэн цонх тодорхойлогдоогүй учраас) бүтэн дэлгэцийг цэвэрлэж байна. Мөн box ангийн erase() функцэд энэ функцийг дуудаж ашигласан байгаа.

Программын үндсэн бие дотор 2 тэмдэгт мөрийг тодорхойлж, түүнийгээ 2 цонхонд бичиж байна. Эхний цонх дэлгэцийн төвд байрлах бөгөөд хэрэглэгч товч дарсны дараа нөгөө цонхыг үүсгэх юм.

box ангийг өргөтгөх нь

Дээрх жишээн дэх box ангийг дахин тодорхойлцгооё. Гэхдээ энэ цонхонд зөвхөн текст бичээд зогсохгүй, уг цонхоо тойруулан хүрээ татаж болох юм. Ингэснээр бидний тодорхойлсон box анги өмнөхөөсөө илүү цонхтой төстэй болж байна. Үүнийг бид мөн л IBM-ийн стандарт тэмдгүүдийг ашиглан хийх болно. Үний тулд бид дээрх ангидаа хэдэн шинэ гишүүн функц нэмж өгье.

```
//boxes.cpp
#include <iostream.h>
#include <conio.h>

const int LEFT=25;
const int TOP=6;
const int RIGHT=55;
const int BOTTOM=18;
const int LEN=500;

class box
        {
        private:
        int left, top, right, bottom;
        char str[len];
        int color;

        public:
        box()
                {
                left=LEFT; top=TOP;
                right=RIGHT; bottom=BOTTOM;
                color=WHITE;
                }
        box(int l, int t, int r, int b, int c=WHITE)
                {
                left=l; top=t; right=r;
                bottom=b; color=c;
                }
        void text(char s[])
                { strcpy(str, s); }
        void draw();
```

```
        void erase()
                {
                window(left, top, right, bottom);
                clrscr();
                }
        };

void box::draw()
        {
        erase;
        window(left, top, right, bottom+1);
        int width=right-left+1;
        int height=bottom-top+1;
        textcolor(color);

        for (int j=1; j<=width; j++)
                {
                gotoxy(j, 1); putch(char(205));
                gotoxy(j, height); putch(char(205));
                }

        for (j=1; j<=width; j++)
                {
                gotoxy(1, j); putch(char(186));
                gotoxy(width, j); putch(char(186));
                }

        gotoxy(1, 1); putch(char(201));
        gotoxy(width, 1); putch(char(187));
        gotoxy(1, height); putch(char(200));
        gotoxy(width, height); putch(char(188));

        window(left+2, top+1, right-2, bottom-3);
        cputs(str);

        window(left, top, right, bottom+1);
        gotoxy(3, height-1);
        cputs("Дурын товч дарна уу:");
        textcolor(WHITE);
        }

void main()
{
```

```
clrscr();
box a;
box b(1, 1, 31, 8, YELLOW);

a.text("\"The time, \" said the \n\r\
      Duchess, \"is surely ripe;\n\r\
      make haste, lest seconds\n\r\
      spoil.\"");

b.text("Should you continue \n\r\
      along the preset path you\n\r\
      risk investigation\n\r\
      for felonious chicanery.");

a.draw();
getch();

a.erase();
b.draw();
getch();

b.erase();
b.text("Баярлалаа");
b.draw();
getch();
}
```

Жишээ программ дахь box анги нь 2 байгуулагч функцтэй байна. Эхнийх нь ямар ч аргументгүй бөгөөд дэлгэцийн голд стандарт цонх үүсгэдэг байна. Харин дараагийнх нь хэрэглэгчийн тодорхойлсон хэмжээ, өнгийг ашиглан цонх үүсгэх юм. Цонхонд гарах текстийг text() гэсэн гишүүн функцийг ашиглана. Ингэснээр хэрэглэгч цонхны хэмжээ, координатыг өөрчлөхгүйгээр, текстийг нь шууд өөрчлөх боломжтой болж байна. Цонхыг draw() гэсэн дүрмээр зурж, erase() гэсэн дүрмээр цэвэрлэх юм. Одоо draw() дүрэмд ашиглагдсан функцүүдийг тайлбарлая.

gotoxy() функц нь текст курсорыг өгсөн координатад байрлуулдаг юм. Курсорын байрлаж

байгаа газраас эхлэн текст хэвлэгддэг учраас энэ функцийг ашиглан текстийн гарах координатыг зохицуулж болно. Параметрт курсор байрлах мөр ба баганын дугаарыг өгөх бөгөөд энэ дугаар нь дэлгэцтэй биш, цонхтой харьцангуй гэдгийг анхаарах нь зүйтэй.

putch() функц курсорын байрлаж байгаа газар нэг тэмдэгт хэвлэж гаргана. Параметрт хэвлэгдэх тэмдэгтийг өгнө. Бид энэ функцийг ашиглан хүснэгтийн хүрээний тэмдэгтүүдийг арилгаж байгаа юм.

5 аргумент бүхий байгуулагчийн сүүлийн буюу 5 дахь аргумент нь цонхны хүрээг зурах өнгийг зааж өгч байна. Тэрхүү өнгийг өөрчлөхийн тулд draw() дүрэм нь textcolor() гэсэн функцийг ашиглана. Энэ функцийн параметрт өгч болох өнгүүдийг доорх хүснэгтээс харна уу.

Өнгөний дугаар	Өнгөний нэр
0	BLACK(хар)
1	BLUE(хөх)
2	GREEN(ногоон)
3	CYAN(цэхэр)
4	RED(улаан)
5	MAGENTA(ягаан)
6	BROWN(бор)
7	LIGHTGRAY(тод саарал)
8	DARKGRAY(бараан саарал)
9	LIGHTBLUE(тод хөх)
10	LIGHTGREEN(тод ногоон)
11	LIGHTCYAN(цэхэр)
12	LIGHTRED(тод улаан)
13	LIGHTMAGENTA(тод ягаан)
14	YELLOW(шар)
15	WHITE(цагаан)
128	BLINK(анивчих)

Түүнчлэн тэмдэгтийн өөрийн өнгөнөөс гадна түүний фоны өнгийг мөн өөрчилж болно. Үүний тулд textbackground() функцийг ашиглана. Харин энэ функц нь параметртээ дээрх хүснэгтийн зөвхөн эхний 8 өнгийг л авч чадна.

draw() дүрэм дотор window() функц хэд хэдэн удаа дуудагдсан байна. Энэ бүрийг нэг нэгээр тайлбарлавал, хамгийн эхний удаад жинхэнэ цонхыг үүсгэх комманд байна. Доод зах нь жинхэнэ bottom хувьсагчаас нэгээр илүү байгаагийн учир нь цонхны хамгийн баруун доод буланд тэмдэгт зурахад цонх дээш нэг мөрөөр гүйдгээр тайлбарлагдаж байгаа юм. Ингэж гүйлгэхгүйн тулд цонхны доод хязгаарыг нэгээр илүү авч байна. Ингээд хүрээг бэлдсэний дараагаар window() функц дахин дуудагдана. Энэ удаад жинхэнэ текст хэвлэгдэх хэсэг буюу цонхыг үүсгэж байна. Жинхэнэ цонхны гадна хүрээг дарж бичихгүйн тулд текст бичих цонхоо багаар авч байна. Хамгийн доод мөрөнд “Press any key to continue:” гэсэн текст хэвлэхийн тулд 3 дахь удаад window() функцийг дуудаж байна.

График горимын функцүүд

Текст горимд та ердөө л цөөн тооны хэдэн тэмдэгт дүрсэлдэг бол харин график горимд та энгийн ганц цэгээс эхлээд шугам болон төрөл бүрийн дүрсүүдийг дүрслэх боломжтой. Текст горимд ердөө л 2000 тэмдэгт (80x25) дүрсэлдэг бол график горимд та 307200 цэгийг (640x480) удирдан зохион байгуулах болно. Харин ямар дүрсийг ямар түвшинд яаж зурахыг зөвхөн таны уран сэтгэмж, авьяас чадвар шийдэх юм.

График горимтой ажиллахын тулд юуны өмнө graphics.lib файлыг Options менюний Libraries гэсэн дэд менюний тусламжтайгаар linker-т зааж өгөх ёстой юм. Хэрэв ингэж заагаагүй бол таны программ дахь график функцүүдийг “тодорхойлогдоогүй функц” гэж үзэх болно.

График горимын функцүүд нь текст горимынхоос эрс ялгаатай. Өөрийн шаардлагатай график горимтой ажиллахын тулд багагүй хэмжээний бэлтгэл ажил хийх шаардлагатай. Эдгээр үйлдлүүдийг ezball хэмээх дараагийн жишээ программаас харж болно. Энэ анги дэлгэцэнд зурагдах тойргийн төвийг set() функцээр сонгон өгч, draw() функцээр зурдаг байна. Бүх ball обьект RAD гэсэн глобал хувьсагчийг тойргийн радиус болгон авах болно.

```
//ezball.cpp
#include <graphics.h>
#include <conio.h>

const int RAD=75;

class ball
        {
        private:
        int xCo, yCo;

        public:
        ball()
                { xCo=0; yCo=0; }
        void set(int x, int y)
                { xCo=x; yCo=y; }
        void Draw()
                { circle(xCo, yCo, RAD); }
        };

void main()
{
int driver, mode;
```

```
driver=DETECT;
initgraph(&driver, &mode, “C:\\BORLANDC\\BGI”);

ball b1;
ball b2;
ball b3;

b1.set(320, 150);
b2.set(255, 150);
b3.set(385, 150);

b1.draw();
b2.draw();
b3.draw();

getch();
closegraph();
}
```

initgraph() функц

Энэ функц нь график горимын бусад бүх функцээс өмнө ажилласан байх ёстой. Энэхүү initgraph() функц нь дэлгэцийг график горимд оруулдаг юм.

Дэлгэцийн орох график горимыг сонгох 2 үндсэн арга байна. Ямар график горимд орохоо та өөрөө сонгон авч болно, мөн системээр сонгуулж болно. Өмнө үзсэн жишээнд системээр автомат сонгуулсан байна. Хэрэв та дэлгэцийн горимуудын тухай нарийн сайн ойлголтгүй бол сонголтыг системээр хийлгэх нь зөв юм. Харин өөрөө сонгохоор шийдсэн бол доорх жижиг бүлгүүдийг анхааралтай уншаарай.

Дэлгэцийн драйвер

Горим сонгох initgraph() функцийн эхний аргумент нь дэлгэцийн график драйвер юм. График драйвер гэдэг нь программ хангамжийг дэлгэцтэй холбож өгдөг видеоадаптерын нэр юм. Энэ аргументийн сонголтуудыг дараагийн хүснэгтэд үзүүлэв.

Дэлгэцийн адаптер	Аргументэд өгөх утга
CGA адаптер	CGA
EGA адаптер	EGA
VGA адаптер	VGA

Энэ аргументэд өгч байгаа тогтмол утгуудыг graphics.h толгой файлд тодорхойлж өгсөн байдаг. Хэрэв та EGA адаптертай бол зөвхөн EGA горимыг л сонгож чаддаг байхад хэрэв VGA адаптертай бол EGA, CGA, VGA алиныг ч сонгож авах боломжтой байдгийг мэдэж авах нь зүйтэй.

Дэлгэцийн горим

Дараагийн аргумент нь орох график горимын дугаарыг заана. Нэг дэлгэцийн драйвер хэд хэдэн горимд орж чадна. Уг горимуудыг дараагийн хүснэгтээр үзүүлье.

Драйвер	Горим	Горимын үзүүлэлт
CGA	CGAC0	320x200, 4 өнгө, палитр 0
	CGAC1	320x200, 4 өнгө, палитр 1
	CGAC2	320x200, 4 өнгө, палитр 2
	CGAC3	320x200, 4 өнгө, палитр 3
	CGAHI	640x200, 2 өнгө
EGA	EGALO	640x200, 16 өнгө
	EGAHI	640x350, 16 өнгө
VGA	VGALO	640x200, 16 өнгө

	VGAMED	640x350, 16 өнгө
	VGAHI	640x480, 16 өнгө

Палитр гэдэг бол өнгөний олонлогийг хэлж байгаа юм. CGA адаптер тус бүр 4 өнгөөс тогтох 4 палитртай байдаг. Харин EGA ба VGA адаптерууд палитр ашигладаггүй. Доор нэгэн товчхон жишээ харуулж байна.

```
int driver=EGA;
int mode=EGAHI;
initgraph(&driver, &mode, “c:\\borlandc\\bgi”);
```

Системээр сонгуулах нь

Хэрэв та энэ ажлыг системд даалгавал систем драйвер ба горимуудыг шалгаж үзээд боломжтой хамгийн сайн горимд оруулдаг. Ингэж таниулахын тулд горимын утганд DETECT гэж бичиж өгөх хэрэгтэй. Үүнийг өмнөх жишээ программаас та бүхэн ойлгож авсан билээ.

Хаяг авах оператор

Дээрх жишээнд initgraph() функцийг дуудахдаа түүний эхний 2 аргументүүдийг өгөхдөө шууд өмнөх хүснэгтүүдийн тогтмол утгыг өгөөгүй байна. Түүний оронд хувьсагч тодорхойлж, уг хувьсагчиддаа дээрх тогтмол утгуудаас оноож, харин тэдгээрийгээ аргумент болгон өгөхдөө хувьсагчийн нэрийн өмнө ‘&’ гэсэн оператор ашигласан байна. Үүний учрыг тайлбарлая.

Энэ операторыг ашиглан бид тэрхүү тогтмол утгуудыг өөрийг нь биш, харин тэдгээр утгуудыг

хадгалж байгаа санах ойн хаягийг өгч байгаа хэрэг юм. Энэ нь үйлдлийн хувьд аргументийг хуурамч нэрээр дамжуулахтай яг ижил үйлдэл болно. Эдгээрийн тухай дараагийн 6-р бүлэгт дэлгэрэнгүй үзэх тул одоо энэ тухайд санаа тавих шаардлагагүй.

График драйверийн зам

initgraph() функцэд өгч байгаа 3 дахь аргумент бол график драйверийн файлын байрлаж байгаа зам юм. График драйверийг автоматаар сонгосон ч гэсэн энэхүү аргументийг заавал бичиж өгөх хэрэгтэй. C++-ийг суулгах үед эдгээр драйверийн файлуудыг '{C++-ийг суулгасан каталог} \BGI' гэсэн замаар бичиж өгдөг. (C++-ийг суулгасан каталоги нь ихэвчлэн C:\BORLANDC байдаг. Энэ тохиолдолд уг зам нь C:\\BORLANDC \\BGI болно.) Уг каталогид байгаа график драйверийн файлууд *.BGI гэсэн өргөтгөлтэй байна. CGA драйверийн файл нь CGA.BGI гэсэн нэртэй, EGA ба VGA драйверийн файл нь EGAVGA.BGI гэсэн нэртэй байна гэсэн үг.

Энэ замд файлын нэрийг оруулах шаардлагагүй бөгөөд том жижиг үсгийн ялгаа байхгүй. Энэхүү зам нь тэмдэгт мөр учраас (' ') хашилтанд бичигдэх ёстой. (\ ) тэмдэг нь C++-ийн хувьд тусгай тэмдэг учраас ('\n' г.м.) энэ тэмдгийг 2 дахин бичиж байж, нэг тэмдэг гэж ойлгох болно. Ийм программ нь зөвхөн C++-ийг суулгасан, уг заасан зам дээр тэрхүү драйверийн файл байгаа тохиолдолд ажиллана. Харин үүнээс хамааралгүй ажилладаг болгохын тулд арайхан өөр алгоритмыг хэрэглэхэд хүрнэ.

circle() функц

Жишээ программдаа бид ball ангийн 3 обьект үүсгэж, set() функцийг нь ашиглан тэдний координатыг тогтоож өглөө. Харин үүний дараа уг обьектүүдийг draw() гэсэн дүрэм ашиглан зурах юм. Харин draw() дүрэм нь C++-ийн стандарт circle() гэсэн функцийг ашиглан дэлгэцэнд тойрог зурж байна.

Уг circle() функц 3 аргумент авах бөгөөд эхний 2 нь тойргийн төвийн координат, 3 дахь нь тойргийн радиус юм. Тойргийн радиусыг RAD гэсэн глобал хувьсагчаас, харин төвийн координатыг xCo, yCo гэсэн локал хувьсагчуудаас авна.

```
circle(xCo, yCo, RAD);
```

closegraph() функц

Программ графиктай ажиллаж дуусаад гарахдаа өмнө нь байсан дэлгэцийн горимд шилжих нь зүйтэй юм. Хэрэв ингэхгүй бол дараагийн программ огт ажиллахгүй байх ч юм уу, эсвэл үйлдлийн систем график горимд текст бичих гэж оролдох зэрэг алдаанууд гарахад хүрнэ.

Хуучин горимыг тогтоох, энэ горимын санах ойд ашиглаж байсан бүгдийг чөлөөлөхийн тулд бид closegraph() функцийг ашиглах ёстой.

Өнгө

Хэрэв та өнгөт дэлгэц ашиглаж байгаа бол график горимын төрөл бүрийн дүрсүүдийг өөр өөр

өнгөөр зурах боломж бий. Мөн битүү хүрээн доторхийг өнгөөр дүүргэн будаж ч болно.

```
//colorbal.cpp
#include <graphics.h>
#include <conio.h>

const int RAD=75;

class ball
        {
        private:
        int xCo, yCo;
        int linecolor;
        int fillcolor;

        public:
        ball()
                {
                xCo=0; yCo=0;
                linecolor=WHITE; fillcolor=WHITE;
                }
        void set(int x, int y, int lc, int fc)
                {
                xCo=x; yCo=y;
                linecolor=lc; fill color=fc;
                }
        void Draw()
                {
                setcolor(linecolor);
                setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
                circle(xCo, yCo, RAD);
                setfillstyle(SOLID_FILL, fillcolor);
                floodfill(xCo, yCo, linecolor);
                }
        };

void main()
{
int driver, mode;
driver=DETECT;
initgraph(&driver, &mode, “C:\\BORLANDC \\BGI”);
```

```
ball b1;
ball b2;
ball b3;

b1.set(100, 150, YELLOW, RED);
b2.set(200, 150, YELLOW, GREEN);
b3.set(300, 150, YELLOW, BLUE);

b1.draw();
b2.draw();
b3.draw();

getch();
closegraph();
}
```

Энэ жишээнд өмнө үзсэн ball ангийн set() функц нь 4 аргументтэй болж өргөжжээ. Эхний 2 аргумент нь өмнөхтэй адилаар тойргийн төвийн координат, харин сүүлийн 2 аргумент нь тойргийн хүрээний болон будагдах өнгүүдийг тус тус зааж өгч байна. Хамгийн ихээр өөрчлөгдсөн дүрэм нь draw() дүрэм юм.

```
setcolor(linecolor);
setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
circle(xCo, yCo, RAD);
setfillstyle(SOLID_FILL, fillcolor);
floodfill(xCo, yCo, linecolor);
```

Энд байгаа функцүүдийн тайлбарыг үзэцгээе.

setcolor() функц

Энэ функц нь дэлгэцэнд дүрс зурах өнгийг сонгож өгдөг юм. Эдгээр функцүүдэд хэрэглэгдэж байгаа line (шугам) гэсэн үг нь зөвхөн шулуун шугамыг

биш, бүхий л дүрсийн хүрээ шугамыг хэлж байгаа юм. Дүрсийн будагдах өнгөнд энэ функц нөлөө үзүүлэхгүй.

EGA ба VGA адаптеруудад нийтдээ 16 өнгийг ашиглах боломжтой байдаг. Эдгээр өнгө нь өмнө текст горимд ашиглагдаж байсан өнгөнүүдтэй нэгэн ижил утгатай байна. Бидний жишээнд байгаа ball ангийн обьектүүд бүгд хүрээнийхээ өнгийг шар гэж тогтоосон байна.

setlinestyle() функц

Бид энэ функцийг ашиглан дүрсийн хүрээг зурах шугамны хэмжээ, хэлбэрийг тогтоож өгч болно. Түүний эхний аргумент нь шугамны хэлбэрийг зааж өгнө.

Дугаар	Шугамны хэлбэр
0	SOLID_LINE(тасралтгүй)
1	DOTTED_LINE(цэглэсэн)
2	CENTER_LINE(цэг зураас маягийн)
3	DASHED_LINE(зурааслсан)
4	USERBIT_LINE(хэрэглэгчийн тодорхойлсон)

Харамсалтай нь энэ хэлбэрүүд зөвхөн шулуун шугаманд нөлөөлөх бөгөөд бидний тойрогт бол энэ аргументийг яаж ч өгсөн ялгаагүй юм. Дараагийн аргумент нь хэрэв эхний аргумент 4 утгатай сонгодсон бол ашиглагдах юм. Бусад тохиолдолд үүнийг 0 гэж өгөх хэрэгтэй. Харин 3 дахь аргумент доорх хоёр утгаас авдаг юм.

Дугаар	Шугамны өргөн
0	NORM_WIDTH(1 цэгийн өргөнтэй)
3	THICK_WIDTH(3 цэгийн өргөнтэй)

setfillstyle() функц

Одоо битүү хүрээн доторхийг яаж будахыг үзэцгээе. Эхлээд бид будах ажиллагаанд шаардлагатай өнгө болон хэлбэрийг сонгож өгөх хэрэгтэй. Энэ функцийн эхний аргумент доорх утгуудаас авна.

Дугаар	Шугамны өргөн
0	EMPTY_FILL(будахгүй)
1	SOLID_FILL(битүү)
2	LINE_FILL(---)
3	LTSLASH_FILL(////(нарийн))
4	SLASH_FILL(////(өргөн))
5	BKSLASH_FILL(\\\\(нарийн))
6	LTBKSLASH_FILL(\\\\(өргөн))
7	HATCH_FILL(+++)
8	XHATCH_FILL(XXX)
9	INTERLEAVE_FILL(3 цэгийн өргөнтэй)
10	WIDE_DOT_FILL(. . . . .)
11	CLOSE_DOT_FILL(. . . . . .)
12	USER_FILL(хэрэглэгчийн)

Харин 2 дахь аргумент нь будах өнгийг зааж өгөх юм. Өгч буй өнгө нь мөн л текст горимынхтой ижил байна. Дээрх программд 3 тойргийг улаан, ногоон, цэнхэр гэсэн 3 өөр өнгөөр буджээ.

floodfill() функц

floodfill() функц нь жинхэнэ будах үйлдлийг гүйцэтгэж байна. Будах ажиллагааг өмнө зурсан тойргийн төвөөс setfillstyle() функцээр тогтоосон өнгө, хэлбэрийг ашиглан зурж эхлэнэ. Харин будах үйл

ажиллагааг 4 дэх параметрт өгсөн өнгөнд тултал гүйцэтгэдэг байна. Энэхүү 4 дэх аргументийг будах өнгө гэж андуурч болохгүй. Хэрэв таны дүрс битүү биш бол дэлгэц дүүртэл ч будагдаж болзошгүй шүү.

Шугам болон тэгш өнцөгт

Дараагийн жишээ нь өмнөх жишээтэй тун төстэй боловч тойргийн оронд тэгш өнцөгт зурдаг анги нэмэгдсэн байна.

```
//restline.cpp
#include <graphics.h>
#include <conio.h>

const int w=75;

class rect
        {
        private:
        int xCo, yCo;
        int linecolor;
        int fillcolor;

        public:
        rect()
                {
                xCo=0; yCo=0;
                linecolor=WHITE; fillcolor=WHITE;
                }
        void set(int x, int y, int lc, int fc)
                {
                xCo=x; yCo=y;
                linecolor=lc; fill color=fc;
                }
        void Draw()
                {
                setcolor(linecolor);
```

```
                setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
                rectangle(xCo-w, yCo-w, xCo+w, yCo+w);
                setfillstyle(SOLID_FILL, fillcolor);
                floodfill(xCo, yCo, linecolor);
                line(xCo-w, yCo+w, xCo+w, yCo-w);
                }
        };

void main()
{
int driver, mode;
driver=DETECT;
initgraph(&driver, &mode, "C:\\BORLANDC \\BGI");

rect r1;
rect r2;
rect r3;

r1.set(80, 150, YELLOW, RED);
r2.set(250, 150, YELLOW, GREEN);
r3.set(420, 150, YELLOW, BLUE);

r1.draw();
r2.draw();
r3.draw();

getch();
closegraph();
}
```

Энэ программд шинээр ашиглагдаж байгаа 2 функцийг доор тайлбарлав.

rectangle() функц

Энэ функц нь тэгш өнцөгтийн хүрээ зурдаг болно. Тэрхүү тэгш өнцөгтийн 4 захын координатыг уг функцийн аргументүүдээр өгнө.

```
rectangle(left, top, right, bottom);
```

line() функц

Уг функц нь өгсөн 2 цэгийг шулуунаар холбодог юм. Эхний 2 аргумент нь шулуун татах эхний цэгийн хэвтээ ба босоо координатууд, сүүлийн 2 нь төгсгөлийн цэгийн хэвтээ ба босоо координатуудыг тус тус тодорхойлно.

line(x1, y1, x2, y2);

Зурагдах шулууны хэмжээ, хэлбэр, өнгө нь өмнө үзсэн функцүүдээр тодорхойлогдоно.

Олон өнцөгт болон удамшлын тухай

Бидний үзэх дараагийн жишээ нь өмнөх жишээнүүд дэх ball ба rect гэсэн ангиудаас удамшуулан гурвалжин гэсэн ангийг үүсгэн ашиглана. Мөн энэ жишээ нь олон өнцөгт зурах fillpoly() коммандыг та бүхэнд танилцуулах юм.

```
//multshap.cpp
#include <iostream.h>
#include <conio.h>

const int w=75;

class shape
        {
        protected:
        int xCo, yCo;
        int linecolor;
        int fillcolor;

        public:
        shape()
                {
```

```
                xCo=0; yCo=0;
                linecolor=WHITE; fillcolor=WHITE;
                }
        void set(int x, int y, int lc, int fc)
                {
                xCo=x; yCo=y;
                linecolor=lc; fill color=fc;
                }
        void Draw()
                {
                setcolor(linecolor);
                setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
                setfillstyle(SOLID_FILL, fillcolor);
                }
        };

class ball : public shape
        {
        public:
        ball() : shape()
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void Draw()
                {
                shape::draw();
                circle(xCo, yCo, w);
                floodfill(xCo, yCo, linecolor);
                }
        };

class rect : public shape
        {
        public:
        rect() : shape()
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void Draw()
                {
                shape::draw();
                rectangle(xCo-w, yCo-w, xCo+w, yCo+w);
                floodfill(xCo, yCo, linecolor);
```

```
                moveto(xCo-w, yCo+w);
                lineto(xCo+w, yCo-w);
                }
        };

class tria : public shape
        {
        public:
        tria() : shape()
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void Draw()
                {
                int triarray[]={ xCo, yCo-w,
                                        xCo+w, yCo+w,
                                        xCo-w, yCo+w };
                shape::draw();
                fillpoly(3, triarray);
                }
        };

void main()
{
int driver, mode;
driver=DETECT;
initgraph(&driver, &mode, "C:\\BORLANDC\\BGI");

ball b1;
rect r2;
tria t3;

b1.set(80, 150, YELLOW, RED);
r2.set(250, 150, YELLOW, GREEN);
t3.set(420, 150, YELLOW, BLUE);

b1.draw();
r2.draw();
t3.draw();

getch();
closegraph();
}
```

shape ангийн тухай

Үүнээс өмнө үзсэн 2 жишээнд байгаа ball ба rect гэсэн ангиудыг та анзаарч харсан бол тэдгээрт ижилхэн юм маш олон байгааг харж болно. Тэдгээрийн аль аль нь дүрсийн төв ба хүрээний болон будагдах өнгийг хадгалах гишүүн талбаруудтай байна. Мөн адилхан set() ба draw() гэсэн гишүүн функцүүдтэй байна.(Гэхдээ мэдээж draw() дүрмүүд нь өөр өөр үйлдлүүд хийнэ.)

C++-ийн уламжилсан анги нь эх ангийнхаа бүх шинж чанарыг агуулсан байдгийг бид мэдэх билээ. Энэ чанарыг ашиглан бид shape гэсэн анги үүсгэж, ball ба rect ангиудыг уг ангиас уламжлуулан авсан юм. Энэхүү shape анги нь нөгөө 2 ангийнхаа бүх нийтлэг шинж чанарыг агуулж байгаа юм.

Олон өнцөгт

Сүүлийн жишээнд бид tria гэсэн шинэ ангийг үүсгэлээ. Гурвалжин дүрсийг хадгалж байдаг энэхүү анги нь мөн л shape ангиас уламжлан авч байгаа юм. Гурвалжин гэдэг бол шулуун шугамуудаар холбогдсон хамгийн энгийн битүү олон өнцөгт дүрс юм. Энэ функц нь бидний өмнө үзсэнүүдээс арай нарийн буюу ө.х. төвөгтэй бүтэцтэй функц болох юм. Түүний ашиглахын өмнө бид программдаа нэгэн массив тодорхойлж өгөх ёстой. Жишээ нь, 4 өнцөгт зурах бол доорх загварын массив тодорхойлогдох шаардлагатай.

```
int polyarray[]={x1, y1, x2, y2, x3, y3, x4, y4}
```

Энэ массивыг ашиглан квадрат, параллелограмм гэх мэт 4 талаас тогтсон бүх дүрсийг зурах боломжтой.

Олон өнцөгт зурах функц 2 аргумент авна. Эхний аргумент нь уг өгөгдлийг ашиглан хэдэн талтай дүрс зурахыг заах юм. Харин дараагийн аргумент нь цэгүүдийг координатуудыг хадгалж байгаа массивын хаягийг авах ёстой. (Ямар ч хаалт, хашилт хэрэггүй. Зөвхөн массивын нэр)

```
fillpoly(4, polyarray);
```

Функц эхлээд хамгийн эхний цэгээс 2 дахь цэг рүү шугам татна. Дараа нь 2 дахь цэгээс 3 дахь руу, 3 дахиас 4 дэх цэг рүү тус тус шулуунууд татна. Харин хамгийн эцэст нь 4 дүгээр цэгээс буцаагаад 1 дүгээр цэг рүү шулуун татаж, уг дүрсийг битүү болгодог юм. Та дүрсийг битүү болгохын тулд эхний цэгийг эцэст нь дахин өгөх шаардлагагүй. Ийнхүү хүрээг зурсны дараагаар сонгосон хэлбэр болон өнгийг ашиглан олон өнцөгтийн доторх хэсгийг будаж дүүргэнэ. Харин drawpoly() функцийг ашиглан будагдаагүй олон өнцөгт зурж болно.

Дуу гаргах бөгөөд хөдөлгөөн оруулах

Одоо тун сонирхолтой нэгэн жишээ үзэх гэж байна. Бидний өмнөх жишээнд үүсгэсэн 3 ангийг ашиглан тэдгээр дүрсүүд дэлгэц дээр 777 тоглоомын маягаар эргэлдэн гарч байх программ юм. Мөн C++-ийн стандарт фунцкийг ашиглан авиа гаргах болно.

Программ эхлэхэд дэлгэц дээр квадрат хэлбэрийн 3 цонх гарч ирнэ. Цонх бүрт доорх 4 дүрсийн аль нэг нь ээлж дараалан гарсаар байх юм. Үүнд: улаан тойрог, цэнхэр тойрог, квадрат, гурвалжин. Дэлгэц дээр нэг дүрс солигдох бүрийд нэг дохио гарна. Яваандаа дүрсүүд солигдох хурд багассаар багассаар, эцэст нь дүрсүүд бүрмөсөн зогсож, хоёр үргэлжилсэн дохио дуугарах болно. Хэрэв гурван улаан тойрог таарч зогсвол та хожих юм.

```
//slot.cpp
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <dos.h>

const int w=15;
const int mar=10;

class shape
        {
        protected:
        int xCo, yCo;
        int linecolor;
        int fillcolor;

        public:
        shape()
                {
                xCo=0; yCo=0;
                linecolor=WHITE; fillcolor=WHITE;
                }
        void set(int x, int y, int lc, int fc)
                {
                xCo=x; yCo=y;
                linecolor=lc; fill color=fc;
                }
        void draw()
```

```
                {
                setcolor(linecolor);
                setlinestyle(SOLID_LINE, 0, THICK_WIDTH);
                setfillstyle(SOLID_FILL, fillcolor);
                }
        };

class ball : public shape
        {
        public:
        ball() : shape()
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void draw()
                {
                shape::draw();
                circle(xCo, yCo, w);
                floodfill(xCo, yCo, linecolor);
                }
        };

class rect : public shape
        {
        public:
        rect() : shape()
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void draw()
                {
                shape::draw();
                rectangle(xCo-w, yCo-w, xCo+w, yCo+w);
                floodfill(xCo, yCo, linecolor);
                moveto(xCo-w, yCo+w);
                lineto(xCo+w, yCo-w);
                }
        };

class tria : public shape
        {
        public:
        tria() : shape()
```

```
                { }
        void set(int x, int y, int lc, int fc)
                { shape::set(x, y, lc, fc); }
        void draw()
                {
                int triarray[]={ xCo, yCo-w,
                                 xCo+w, yCo+w,
                                 xCo-w, yCo+w };
                shape::draw();
                fillpoly(3, triarray);
                }
        };

class noshape : public shape
        {
        public:
        void erase()
        {
        int border[]={ xCo-w-mar, yCo-w-mar,
                       xCo+w+mar, yCo-w-mar,
                       xCo+w+mar, yCo+w+mar,
                       xCo-w-mar, yCo+w+mar };
        setfillstyle(SOLID_FILL, DARKGRAY);
        fillpoly(4, border);
                }
        };

class cherry : public ball, public noshape
        {
        public:
        cherry() : ball()
                { }
        void set(int x, int y)
                {
                ball::set(x, y, WHITE, RED);
                noshape::set(x, y, WHITE, RED);
                }
        void draw()
                { erase(); ball::draw(); }
        };

class grape : public ball, public noshape
        {
```

```
        public:
        grape() : ball()
                { }
        void set(int x, int y)
                {
                ball::set(x, y, WHITE, BLUE);
                noshape::set(x, y, WHITE, BLUE);
                }
        void draw()
                { erase(); ball::draw(); }
        };

class square : public rect, public noshape
        {
        public:
        square() : rect()
                { }
        void set(int x, int y)
                {
                rect::set(x, y, WHITE, GREEN);
                noshape::set(x, y, WHITE, GREEN);
                }
        void draw()
                { erase(); rect::draw(); }
        };

class pyramid : public tria, public noshape
        {
        public:
        pyramid() : tria()
                { }
        void set(int x, int y)
                {
                tria::set(x, y, WHITE, GREEN);
                noshape::set(x, y, WHITE, GREEN);
                }
        void draw()
                { erase(); tria::draw(); }
        };

class wheel : public shape
        {
        private:
```

```
        cherry ch;
        grape gr;
        square sq;
        pyramid py;

        public:
        wheel()
                { xCo=0; yCo=0; }
        void set(int x, int y)
                {
                xCo=x; yCo=y;
                ch.set(xCo, yCo);
                gr.set(xCo, yCo);
                sq.set(xCo, yCo);
                py.set(xCo, yCo);
                }
        void draw();
        };

void wheel::draw()
{
setcolor(WHITE);
rectangle(xCo-w-mar, yCo-w-mar,
        xCo+w+mar, yCo+w+mar);
switch(random(4))
        {
        case 0 : ch.draw(); break;
        case 1 : gr.draw(); break;
        case 2 : sq.draw(); break;
        case 3 : py.draw(); break;
        }
}

void main()
{
const int number=60;
int driver, mode;

driver=DETECT;
initgraph(&driver, &mode, "c:\\borlandc\\bgi");
randomize();

wheel w1;
```

```
wheel w2;
wheel w3;

w1.set(100, 100);
w2.set(160, 100);
w3.set(220, 100);

for (int j=0; j<number; j++)
        {
        w1.draw();
        w2.draw();
        w3.draw();
        sound(100); delay(20); nosound();
        delay(j * j / 20);
        }
sound(400); delay(400);
sound(500); delay(800); nosound();
getche();
closegraph();
}
```

Энэ программ нь бидний өмнө үзэж байсан жишээнүүдийг бодвол нэлээн урт, төвөгтэй бүтэцтэй учраас бид программын тухай нэлээн тайлбарлах шаардлагатай болно.

Программ дахь удамшлын тухай

Үзэж буй энэхүү жишээ нь бидний өмнөх жишээнүүдэд тодорхойлогдсон ангиудыг ашиглаж байгаа билээ. Тэр ангиудаас уламжуулан мөн шинэ ангиуд үүсгэсэн байна. Тодруулбал, улаан өнгөтэй ball ангиас cherry гэсэн шинэ анги, хөх өнгөтэй ball ангиас grape гэсэн анги, ногоон өнгө бүхий rect ангиас square гэсэн анги, мөн өнгөтэй tria ангиас pyramid гэсэн ангийг тус тус үүсгэжээ. Уламжилсан ангиуд нь

өмнөх ангийнхаа set() функцийг шууд дуудан ажиллах бөгөөд гэхдээ өнгийг нь тогтмолоор өгч байна.

```
void cherry::set(int x, int y)
        {
        ball::set(x, y, WHITE, RED);
        noshape::set(x, y, WHITE, RED);
        }
```

Бид аль нэгэн цонхонд шинэ дүрс зурахын өмнө уг цонхыг цэвэрлэх шаардлагатай. Ингэж арчихгүй бол тойрог зурагдаж байсан газар гурвалжин зурвал цонхонд үл ойлгогдох дүрс үүсэхэд хүрнэ. Энэ шаардлагын үүднээс noshape гэсэн шинэ анги тодорхойлогджээ. Энэ анги erase() гэсэн ганцхан гишүүн функцтэй байна.

Шинэ тодорхойлогдсон ангиуд ball мэтийн ганц ангиас биш харин тус бүр хоёр ангиас удамшиж байна. Эдгээр ангиуд noshape ангиас удамшсанаараа цонхонд дүрс зурахын өмнө уг цонхоо цэвэрлэх чадалтай болж байгаа юм. Энэхүү noshape анги нь мөн л өөртөө координат хадгалах зориулалттай учраас мөн shape ангиас удамшиж байна.

Хамгийн сүүлийн анги wheel нь мөн л shape-ээс удамшиж байна. Гэхдээ энэ нь өмнөх ангиуд шигээ дүрс зурах зориулалттай биш, харин дүрснүүд солигдох нэг цонхыг зурах болно.

Программын ажиллагааны тухай

Программын үндсэн биед wheel ангийн 3 обьектийг үүсгэжээ. Тэдгээрийг set() функцийг нь ашиглан дэлгэц дээр байрлуулж бөгөөд давталт дотор цонхыг дахин дахин зурж байна. Уг wheel анги дотор cherry, grape, square, pyramid ангийн нэг нэг

обьектүүдийг мөн үүсгэсэн байна. set() гишүүн функц цонхнуудыг байрлуулаад зогсохгүй, тэдгээр дээр зурагдах ёстой дүрсүүдийн ч координатуудыг тогтоож өгөх юм. Харин draw() функц цонхны хүрээг зураад, дээрх дөрвөн янзын дүрснээс аль нэгийг нь санамсаргүй тоон дээр үндэслэн зурах болно. Ингээд draw() дүрэм давталт дотроос дуудагдах бүрд уг цонхонд байгаа хуучин дүрсийг арилгаж, шинэ дүрс зурна гэсэн үг. Гурван цонхон дээрх дүрс зурагдсаны дараа программ нэг дохио дуугаргаж, хэсэг хугацаанд зогсох болно. Тэрхүү зогсох хугацаа давталт бүрд нэмэгдсээр, давталт дуусах үед дүрс бүрэн зогсох юм.

Дуу гаргах функцүүд

C++-д дуу гаргахын тулд доорх алхмуудыг гүйцэтгэнэ. Үүнд:

1. Дууг гаргах
2. Хэсэг хугацааны турш хүлээх
3. Дууг алга болгох

Үүнийг бид жишээ программдаа доорх байдлаар гүйцэтгэсэн байна.

```
sound(100); delay(20); nosound();
```

Эдгээр функцүүдийг ашиглахын тулд dos.h гэсэн толгой файлыг тодорхойлсон байх шаардлагатай.

sound() функц нь аргументэд өгсөн утгаас хамааруулан янз бүрийн авиа гаргадаг юм. Аргумент нь гарах авианы хэлбэлзлийг Герцээр илэрхийлсэн тоон утга байх ёстой бөгөөд 15-аас 3000 хүртэл утга

авч чадна. Функцийг дуудах үед түүний үргэлжлэх хугацааг зааж өгөхгүй.

Харин уг авиа хэр удаан үргэлжлэн дуугарахыг delay() гэсэн функцээрээ тохируулах болно. Ерөнхий зориулалтаараа delay() функц нь программын ажиллагааг хэсэг хугацаанд зогсоодог юм. Уг функц нь мөн ганц аргумент авах бөгөөд тэр нь ажиллагаа зогсох хугацааг миллисекундээр (1/1000 секунд) илэрхийлж байгаа болно. Ө.х. авиаг хагас секунд дуугаргахын тулд delay(500) гэж өгнө гэсэн үг юм.

Харин хамгийн сүүлийн nosound() функц нь гарч байгаа авиаг зогсоох болно.

Энэхүү жишээ программаараа бид обьект хандалтат программчлалын үндсийг үзлээ гэж бодож болно. Энгийн ангиас удамшуулан нэлээн төвөгтэй ангиудыг үүсгэж болохыг мэдэж авлаа. Хэрэв бид программаа процедур хандалтат программчлалаар зохион байгуулсан бол өөр өөр обьектүүдийн хоорондох харилцан хамаарал үүсгэх асуудал тун төвөгтэй болох юм.

График горим дахь текст

График горимд гарч байгаа текст нь текст горимд гардгаа бодвол илүү боломж ихтэй байдаг. Тухайлбал, текстийг заавал зөвхөн нэг маягаар биш, хэд хэдэн янзын фонт ашиглан, хүссэн хэжээгээрээ зурж болно. Мөн уг текстээ хүссэн чиглэлдээ бичиж болно.

График горимд текст гаргах анги

Бидний одоо үзэх жишээнд gstring гэсэн анги ашиглагдаж байна. Энэ анги нь дэлгэцэнд гарах текст болон түүний янз бүрийн график шинж чанаруудыг хадгалж байх юм. Үүнд: эхлэлийн координат, текстийн фонт, түүний чиглэл, текстийн хэмжээ болон өнгө, мөрүүд аль талаараа тэгшрэх, мөн текстийн өргөн өндрийн харьцаа гэсэн мэдээллүүд орно.

Уг ангийн аргумент авдаггүй байгуулагч функц нь энэхүү шинж тэмдгүүдийн стандарт утгуудыг өгдөг байна. Тэдгээр утгуудыг бүгдийг гишүүн функц ашиглан өөрчилдгөөр зохион байгуулсан байна.

```
//grast.cpp
#include <graphics.h>
#include <conio.h>
#include <string.h>

class gstring
        {
        protected:
        char str[80];
        int xCo, yCo;
        int font;
        int direction;
        int size;
        int color;
        int horzjustify;
        int vertjustify;
        int multx, divx;
        int multy, divy;

        public:
        gstring()
                {
                str[0]='\0';
                xCo=0; yCo=0;
                font=DEFAULT_FONT;
                direction=HORIZ_DIR;
                size=4;
                color=WHITE;
```

```
                horzjustify=LEFT_TEXT;
                vertjustify=TOP_TEXT;
                multx=1; divx=1;
                multy=1; divy=1;
                }
        void drawtext()
                {
                moveto(xCo, yCo);
                settextstyle(font, direction, size);
                setcolor(color);
                settextjustify(horzjustify, vertjustify);
                setusercharsize(multx,divx,multy,divy);
                outtext(str);
                }
        void settext(char s[])
                { strcopy(str, s); }
        void setposition(int x, int y)
                { xCo=x; yCo=y; }
        void SetFont(int f)
                { font=f; }
        void SetDirection(int d)
                { Direction=d; }
        void SetSize(int s)
                { size=s; }
        void SetColor(int c)
                { color=c; }
        void SetHJust(int hj)
                { horzjustify=hj; }
        void SetVJust(int vj)
                { vertjustify=vj; }
        void SetHorzSize(int m, int d)
                { size=0; multx=m; divx=d; }
        void SetVertSize(int m, int d)
                { size=0; multy=m; divy=d; }
        };

void main()
{
int driver, mode;
driver=DETECT;
initgraph(&driver, &mode, “c:\\borlandc\\bgi”);

gstring s1, s2, s3, s4, s5, s6;
```

```
s1.SetText("Анхны тохиргоо");

s2.SetText("Гот маягийн фонт");
s2.SetFont(GOTHIC_FONT);
s2.SetPosition(0, 75);

s3.SetText("Босоо санскрит фонт");
s3.SetFont(SANS_SERIF_FONT);
s3.SetPosition(600, 0);
s3.SetDirection(VERT_DIR);

s4.SetText("Голлосон триплекс фонт");
s4.SetFont(TRIPLEX_FONT);
s4.SetPosition(300, 150);
s4.SetHJust(CENTER_TEXT);

s5.SetText("Голлосон санскрит фонт");
s5.SetFont(SANS_SERIF_FONT);
s5.SetSize(6);
s5.SetPosition(300, 225);
s5.SetDirection(CENTER_TEXT);

s6.SetText("Босоо нарийн триплекс фонт");
s6.SetFont(TRIPLEX_FONT);
s6.SetPosition(0, 300);
s6.SetHorzSize(2, 3);
s6.SetVertSize(4, 1);

s1.drawText();
s2.drawText();
s3.drawText();
s4.drawText();
s5.drawText();
s6.drawText();

getch();
closegraph();
}
```

Программын үндсэн биед gsrting ангийн 6 обьект үүсгэсэн байна. Эхнийх нь бүх стандарт

тохируулгаараа ашиглагдаж байна. Харин 2 дахь нь стандарт фонтны оронд sans_serif фонтыг ашигласан байна. 3 дахь нь дэлгэц дээр хэвтээгээр байрлах бөгөөд 4 ба 5 дахь нь координатынхаа хувьд голлон байрласан байна. Харин 6 дахийн хувьд текстийн өргөн өндрийн харьцаа өөрчлөгдсөн байгаа юм.

Текстийг яг дэлгэц дээр гаргах үйлдлийг drawtext() гэсэн дүрмээр хийж байгаа билээ. Харин одоо энд байгаа стандарт функцүүдийг дэлгэрэнгүйгээр авч үзье.

moveto() функц

C++ нь текстийг дэлгэцэнд гаргахдаа идэвхтэй байрлал (Current Position) дээр хэвлэдэг юм. Энэхүү функц нь тэр идэвхтэй байрлалын утгыг өөрчилж өгдөг юм. Горим шинээр тогтоогдоход идэвхтэй байрлал (0, 0) болдог.

settextstyle() функц

Энэхүү функцийг ашиглан текстийн фонт, чиглэл болон хэмжээг өөрчилж болно. C++ нь доорх 5 янзын стандарт фонттой байдаг.

Утга	Фонт
DEFAULT_FONT	Энгийн фонт
TRIPLEX_FONT	Times хэлбэрийн фонт
SMALL_FONT	Жижиг фонт
SANS_SERIF_FONT	Helvetica хэлбэрийн фонт
GOTHIC_FONT	Хуучин англи хэлбэрийн фонт

Харин эдгээр фонтууд нь ашиглагдахын тулд initgraph()-ийн 3-р аргументэд өгсөн каталогид байх ёстойг анхаарах хэрэгтэй. Ийм фонтны файл нь CHR гэсэн өргөтгөлтэй байдаг. Жишээ нь: TRIP.CHR гэвэл Triplex фонт, SANS.CHR гэвэл Sans Serif Font байх жишээтэй.

Функцийн 2 дахь аргумент нь HORIZ_DIR (хэвтээ) ба VERT_DIR (босоо) гэсэн утгуудын аль нэгийг авах ёстой. Босоо чиглэлтэй текст нь дороосоо дээш, харин хэвтээ чиглэлтэй бол зүүнээс баруун тийш бичигдэх болно.

Хамгийн сүүлийн аргумент нь текстийн хэмжээг зааж өгнө. Хамгийн бага нь 1 хэмжээтэй байх бөгөөд 2 гэж сонговол текст 1 хэмжээтэйгээсээ 2 дахин том гарна гэсэн үг.

settextjustify() функц

Уг функц текстийг идэвхтэй байрлалтай харьцангуйгаар яаж байрлахыг зааж өгдөг юм. Эхний аргумент нь уг текст хэвтээ байрлалдаа хэрхэн байрлахыг, харин дараагийн аргумент нь босоо байрлалдаа хэрхэн зохицохыг тохируулах юм.

Утга	Тайлбар
LEFT_TEXT	Идэвхтэй байрлалаас баруун тийш бичнэ(и.байрлал зүүн талдаа байна)
CENTER_TEXT	Идэвхтэй байрлалаас хоёр тийш бичнэ(и.байрлал төвдөө байна)
RIGHT_TEXT	Идэвхтэй байрлалаас зүүн тийш бичнэ(и.байрлал баруун талдаа байна)

Утга	Тайлбар
TOP_TEXT	Идэвхтэй байрлал дээд

	талдаа байна
CENTER_TEXT	Идэвхтэй байрлал голд нь байна
BOTTOM _TEXT	Идэвхтэй байрлал доод талдаа байна

setusercharsize() функц

Энэ функцийг ашиглан тэмдэгтийн өргөн өндрийг өөрчлөх замаар текстийн пропорци, харьцааг өөрчилж болох юм. Зөвхөн текстийн хэмжээг 0 гэж сонгосон тохиолдолд л энэ функц нөлөө үзүүлж чадна. Учир нь уг функц фонтны хэмжээг автоматаар 4 гэж сонгодогт байгаа юм. Функц 4 аргумент авна. Эхний хоёр аргумент нь текстийн өргөний харьцаа бөгөөд хүртвэр ба хуваарь нь болно. Жишээ нь: хэрэв эхний 2 аргументийг 5 ба 2 гэж өгвөл текстийн өргөний харьцаа 5/2 буюу (хэмжээ нь 4 гэдгийг санавал) текст 10 хэмжээтэй текст шиг өргөнтэй гарна гэсэн үг юм. Үүнтэй адилаар сүүлийн 2 аргумент нь текстийн өндрийн харьцааг илэрхийлнэ.

outtext() функц

График горимын текст нь ийм функцийн тусламжтайгаар дэлгэцэнд хэвлэгддэг юм. Уг функц аргументдээ хэвлэгдэх текстээ авдаг байна. Бусад хэвлэх функцүүд буюу гаралтын стандарт урсгал график горимд ажилладаггүйг анхаарч авах хэрэгтэй.

Бүлгийн дүгнэлт

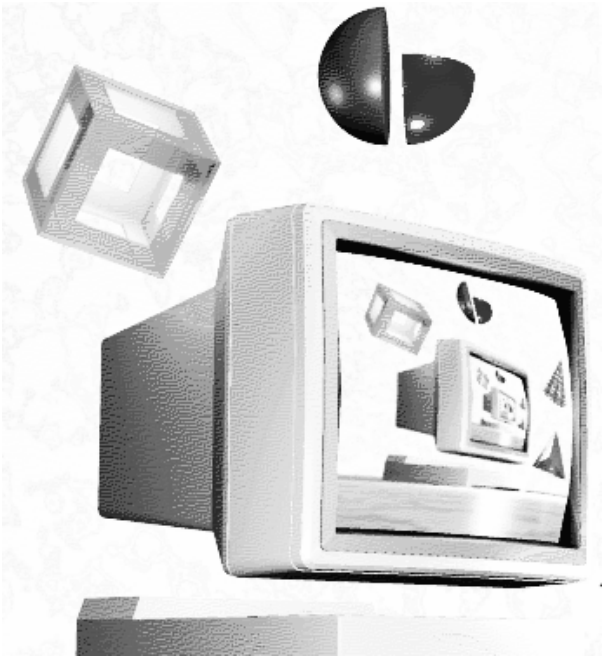
С++-ийн дэлгэцтэй ажиллах функцүүдийг текст горимын болон график горимын гэж 2 ангилна. Эхний буюу текст горимын функцүүд нь цонх гэж нэрлэгдэх, дэлгэцний нэг хэсэгт текстийг гаргахтай холбоотой юм. Дэлгэц дээр цонх үүсгэхийн тулд window() гэсэн стандарт функцийг ашиглана.

Дараагийн бүлэг буюу график горимын функцүүд нь видео адаптераас хамааран ажилладаг. Ийм функцүүдийг ашиглах программ нь график initgraph() функцийг ашиглан дэлгэцийн горимыг өөрчлөх бөгөөд мөн closegraph() функцээр хуучин горимд нь оруулж байх ёстой.

С++-ийн стандарт функцүүдийг ашиглан график горимд энгийн шулуунаас эхлээд тойрог, дөрвөлжин, олон өнцөгт зэрэг дүрсүүдийг сонгосон өнгө, хэмжээ, хэлбэрийг ашиглан зурж болно. Мөн график горимд төрөл бүрийн текстийг хэвлэж чадах бөгөөд түүний хэмжээ, өнгө, байрлал, өргөн өндрийн харьцаа өөрчлөгдөх боломжтой байдаг.

Графикийн элементүүдийг обьект маягаар зохион байгуулах нь туйлын зохистой бөгөөд энэ нь обьект хандалтат программчлалыг ашиглах үндэс суурь болно

8-р бүлэг



-Хаяг ба заагч
-Заагч ба массив
-Заагч ба функц
-Заагч ба тэмдэгт мөр
-Обьектийн заагч
-Өгөгдлийн бүтэц
-Заагчийн заагч

8-р бүлэг

Заагч гэдэг бол ер нь программчлалын хэл, тэр дундаа C++-ийн салшгүй чухал хэсгүүдийн нэг юм. Энэ бүлэгт заагчийн тухай ойлголтоос эхлээд түүнийг ашиглах хэд хэдэн жишээтэй танилцах юм.

Заагчийг дараах ерөнхий тохиолдуудад ашигладаг юм.

- Олон элементтэй массивт хандах
- Параметр нь өөрчлөгддөг функцийг зохиож ашиглах
- Массив болон тэмдэгт мөрийг функцийн аргументэд өгөх
- Системээс санах ой “хулгайлах”
- “Холбоост мод”, “Дараалал” зэрэг бүтцүүдийг үүсгэх

Заагч төрөл нь C++-д бусад программчлалын хэлнүүдээс илүү өргөн ашиглагдах бөгөөд түүнийхээ хэмжээгээр ашиглахад ч илүү дөхөм байдаг. Энэ бүлгийг үзсэний дараа “заагчийг огт ашиглахгүй байж болох л юм байна” гэсэн бодол төрж болно. Учир нь C++-д заагчаар хийж байгаа үйлдлүүдийг энгийн аргаар хийх боломжтой байна. Жишээ нь, массивийг заагч ашиглалгүйгээр зохион байгуулж болно.

Гэхдээ заагчийг ашигласнаар C++-ийг илүү хүчирхэг шинжийг мэдэж авна гэдгийг л ойлгох хэрэгтэй. Үүний тод жишээ нь өгөдлийг холбоост мод ба дараалал хэлбэртэй зохион байгуулах явдал юм. Та өгөгдлөө ингэж зохион байгуулснаар их хэмжээний өгөгдөлтэй ажиллах, хайлт хийх, эрэмбэлэх зэрэг ажиллагааг асар богино хугацаанд хийж чадна.

Хэрэв та урьд нь C-ийг ашиглаж байсан бол энэ бүлгийн эхний хагасыг судлалгүй орхиж болно. Харин

сүүлийн хагас буюу new ба delete функцуудийн тухай, гишүүн функцийг ашиглах, мөн холбоост дараллын тухай хэсгүүдийг анхааралтай судлах хэрэгтэй.

Хаяг болон заагчийн тухай

Заагчийг дараах тун энгийн ойлголтоос эхлэж судлая.

Компьютерийн санах ойд байгаа байт бүр өөрийн хаягтай байдаг. Хаяг гэдэг нь яг л байшингийн хаягтай ижил ойлголт юм. Ө.х. хаяг нь 0-ээс эхлээд 1, 2, 3 гэх мэтчилэн үргэлжилнэ. Хэрвээ компьютерийн санах ой тань 640KB бол хаяг нь 655359-өөр, харин 1MB бол 1048575-аар тус тус дуусна гэсэн үг.

Ямар ч программ ажиллахын тулд дискнээс санах ойд ачаалагдах бөгөөд ингэснээрээ санах ойн тодорхой хэсэг хаягийг эзэлж авдаг. Ө.х программын бүх коммандууд болон өгөдлүүд санах ойн аль нэг хаяг дээр байрлах болно гэсэн үг.

Хаяг авах ‘&’ оператор

Энэхүү операторыг ашиглан та хувьсагчийн санах ойд байрлаж байгаа хаягийг авах боломжтой болно. Энэ операторыг varaddr гэсэн доорх жишээнд ашиглаж үзүүлжээ.

```
//varaddr.cpp
#include <iostream.h>

void main()
{
int var1=11;
int var2=22;
```

```
int var3=33;

cout << endl << &var1
    << endl << &var2
    << endl << &var3;
}
```

Энэ программд 3 хувьсагч тодорхойлж, анхны утгуудыг оноожээ. Харин сүүлийн мөрөнд эдгээр 3 хувьсагчийн санах ойд байрлаж байгаа хаягуудыг хэвлэж байна.

Хувьсагчийн жинхэнэ хаяг нь маш олон зүйлээс хамаарна. Уг компьютерт байгаа үйлдлийн системийн хэмжээ, тухайн үед санах ойд ажиллаж байгаа программуудын тоо болон хэмжээ г.м. Тийм учраас энэ программ ажиллах бүрдээ ижил хаяг хэвлэхгүй байж болно. Программын үр дүнгийн нэг жишээ нь:

```
0x8f4ffff4
0x8f4ffff2
0x8f4ffff0
```

Хувьсагчийн хаяг гэдэг бол түүний утга биш гэдгийг мартаж болохгүй. Дээрх 3 хувьсагчийн утга нь харгалзан 11, 22, 33 юм. (Харин хаяг нь эдгээр тоонуудад огт хамаагүй!)

Энд хэвлэгдсэн утгууд нь өмнөө ‘0x’ гэсэн байгаа нь уг тоо 16-тын системээр харагдаж байгааг илэрхийлж байгаа юм. Ер нь санах ойн хаяг ихэвчлэн 16-тын системээр илэрхийлэгддэг. Хэрвээ та 16-тын системийн тухай мэдэхгүй бол нэг их санаа зовсны хэрэггүй. Энд анхаарах ганц зүйл нь 3 хаяг гурвуулаа ижил эхэлсэн бөгөөд хоорондоо зөвхөн 2 байтаар л ялгаатай байна. Үүний учир нь int төрөл санах ойд 2 байт эзэлдгээр тайлбарлагдана. (65535 хүртэл тоог илэрхийлэхийн тулд 16 бит буюу 2 байт) Бид char төрлийн хувьсагч тодорхойлсон бол 1 байтаар, double

төрлийн хувьсагч тодорхойлсон 8 байтаар хаяг нь шилжинэ гэсэн үг юм.

Мөн хаягууд буурах дарааллаар байрласан байгааг анзаарах хэрэгтэй. Программын эхэнд тодорхойлсон хувьсагчууд stack-т (төгсгөлөөсөө эхлэл рүүгээ явдаг) санах ойд хадгалагддагт гол учир байгаа юм. Харин программын явцад тодорхойлогдсон хувьсагчид нь heap гэж нэрлэгдэх эхлэлээсээ төгсгөл рүүгээ явдаг санах ойд байрладаг.

Хаяг авах & операторыг функцийн параметрт бичигддэг & (4 дүгээр бүлэгт үзсэн) оператортой андуурч ойлгож болохгүй.

Заагч хувьсагч

Өмнөх жишээнд хийсэн хувьсагчийн хаягийг хэвлэх үйлдэл нь бодит амьдрал дээр бараг шаардлагагүй юм. Харин программд зайлшгүй шаардагдах нэг зүйл бол *хувьсагчийн хаягийг хадгалж байдаг хувьсагч* юм. Бид бүхэл, бутархай, тэмдэгт гээд олон төрөл үзсэн билээ. Тэгвэл одоо хаяг хадгалдаг заагч гэсэн төрлийг нэмж үзье.

Заагч хувьсагчийн төрөл нь зааж байгаа хувьсагчийнхаа төрөлтэй адил байна гэж ойлгож болохгүй. Ө.х. int төрлийн заагч нь int-тэй ижил хэмжээний санах ой эзлэхгүй. Энэ тухай дараагийн жишээн дээр тайлбарлая.

```
//ptrvar.cpp
#include <iostream.h>

void main()
{
        int var1=11;
        int var2=22;
```

```
        cout << endl << &var1 << endl << &var2;

        int *ptr;

        ptr=&var1;
        cout << endl << ptr;

        ptr=&var2;
        cout << endl << ptr;

}
```

Энэ программ var1, var2 гэсэн хувьсагчууд тодорхойлж, харгалзан 11, 22 гэсэн утгууд оноожээ. Дараа нь тэдгээрийн хаягийг хэвлэсэн байна. Үүний дараа ptr гэсэн заагч хувьсагч тодорхойлсон байна. Энд байгаа (*) нь заагч гэсэн утга санааг илэрхийлнэ. Ө.х ‘ptr хувьсагчийн int төрлийн хувьсагчийн хаягийг хадгална’, эсвэл ‘ptr хувьсагч нь int төрлийн хувьсагч руу заана ’ гэсэн үг. Харин ямар ч төрлийн хувьсагчийг зааж чадах заагчийг доорх байдлаар тодорхойлдог.

```
pointer ptr;
```

Энэ тохиолдолд заагчийн утга дээр үйлдэл хийхдээ тун анхааралтай байх шаардлагатай болно.

Энд нэг зүйл сонирхуулахад бид заагч хувьсагч тодорхойлохдоо (*) тэмдгийг хувьсагчийнхаа нэрэнд ойрхон бичсэн байна. Уг нь хаана ч байрлаж болох боловч энэ нь төрөлдөө биш хувьсагчдаа хамаатай гэсэн утга санааг илэрхийлэхийн тулд ийм бичлэгийг ашигладаг байна. Өөрөөр хэлбэл, доорх бичлэгүүд хөрвүүлэгч компиляторын хувьд огт ялгаагүй гэсэн үг.

```
Float *fptr;
Float * fptr;
```

```
Float* fptr;
```

Заагчийн утгын тухайд

Бидний үзсэн жишээнд var1 гэсэн хувьсагчид 11 гэсэн утга оноож байсан. Үүнтэй яг адилаар ptr хувьсагчийн утга нь 0x8f4ffff4 гэч мэтчилэн санах ойн хаяг байх ёстой юм.

Заагчийг анхлан тодорхойлоход тэр нь ямар ч утгагүй байдаг. Өөрөөр хэлбэл ямар ч санах ой руу заагаагүй байх болно. Магадгүй энэ нь ямар нэгэн утгатай байж болох бөгөөд гэхдээ тэр нь жинхэнэ ёсоороо олгогдоогүй, худлаа хаяг байх юм. Тийм учраас бид заагчдаа юуны өмнө утга олгох хэрэгтэй болно. Өмнөх ptrvar гэсэн жишээнд

```
Ptr=&var1;
```

гэж утга оноож байгаа юм. Үүний дараа хэвлэгдэж байгаа утга нь var1 хувьсагчийн санах ойд байрлаж байгаа хаяг билээ.

Эцэст нь хэлэхэд заагч хувьсагч юу зааж байгаагаа огт мэдэхгүй. Зөв утга оноогоогүй тохиолдолд тэрбээр огт хүсээгүй газар руу, тухайлбал программын кодууд руу (өгөгдөл рүү) ч юм уу, үйлдлийн системийн хэсэг рүү зааж байж болох юм. Энэ үед тэр хаяг руу юм бичвээс системийг гацах хүртэл байдалд оруулж болохыг үргэлж санаж бай. Ийм учраас заагч хувьсагчид заавал утга оноож байх хэрэгтэй.

Зааж буй хувьсагчид хандах нь

Бид ямар нэгэн хувьсагчийн нэрийг мэдэхгүй ч хаягийг нь мэддэг боллоо. Тэгвэл тэр хувьсагчийн утгыг “оролдож” болох уу?. Тэгвэл хувьсагчийн нэрийг

мэдэхгүй ч хаягийг нь ашиглан утгатай нь ажиллаж болно гэж хариулна. Доорх ptracc гэсэн жишээг үзье.

```
//ptracc.cpp
#include <iostream.h>

void main()
{
int var1=11 ;
int var2=22 ;

int *ptr ;

ptr=&var1 ;
cout << endl << *ptr;

ptr=&var2 ;
cout << endl << *ptr;
}
```

Энэ жишээ нь үүний өмнөх ptrvar жишээтэй тун төстэй харагдавч энд тун жаахан ялгаа бий. Өмнөх жишээнд хаягуудыг нь хэвлэж байсан бол энэ удаад уг хаяган дээрх утгуудыг хэвлэж байна. Үр дүн нь:

```
11
22
```

Үүнийг хэвлэж байгаа мөрөнд ptr хувьсагчийн өмнө од тавьсан байгаа нь заагчийн утгыг бус, харин утга болж буй санах ойн хаяган дээрх утгыг авна гэж хэлж байгаа юм. Ийм учраас ptr нь var1 -ийг зааж байгаа үед *ptr нь 11 гэсэн утга, ptr нь var2 –ийг зааж байгаа үед *ptr нь 22 гэсэн утгатай байгаа юм.

Ийм од ашигласнаар та зөвхөн утгыг хэвлэх биш, тэр хувьсагч дээр хийж болох бүх үйлдлийг гүйцэтгэх боломжтой болно. Доорх жишээг анхааралтай үзэхэд үүнийг мэдэж болно.

```
//ptrto.cpp
#include <iostream.h>

void main()
{
int var1, var2;
int *ptr;

ptr=&var1;
*ptr=37;
var2=*ptr;

cout << endl << var1 << “,” << var2 ;
}
```

Үр дүн нь:

```
37,37
```

гэж байна. Өөрөөр хэлбэл, энэ тохиолдолд var1=37, *ptr=37 гэсэн үйлдлүүд нь яг ижил гэсэн үг юм.

Үүнийг нэг амьдралын жишээн дээр тайлбарлая. Та найздаа захиа бичээд түүнийгээ найзынхаа шуудангийн хайрцагт шууд хийчихэж болно. Эсвэл дугтуйд хийгээд шуудангийн албанд аваачиж өгч болно, тэгвэл шуудан зөөгч уг захиаг түүний хайрцагт хийнэ. Энэ хоёр процесс ижил үр дүнтэй боловч, өөр аргаар хийгдэж байгаа билээ. Эхний тохиолдолд шууд, 2 дахь тохиолдолд дамжуулан хандаж байгаа хэрэг юм. Бидний үзсэн жишээнд бид хувьсагчаар шууд хандахын оронд, заагч ашиглаж байгаа нь заагчийн үүргийг онц их ойлгуулахгүй байж болох юм. Гэхдээ хойно үзэх бүлгүүдэд заагчийн жинхэнэ үүргүүд буюу хандаж болохгүй газар руу хандах тухай үзэх болно.

VOID Заагч

Заагчийн үүргийн тухай цааш үзэхийн өмнө заагчийн нэгэн сонин төрлийн тухай үзье. Бидний үзсэнээр заагчид хаяг олгохдоо зөвхөн таарах төрлийн хувьсагчийн л хаягийг өгөх ёстой. Тухайлбал, int төрлийн заагчид float төрлийн хувьсагчийн хаягийг өгч болохгүй. Гэтэл нэгэн цагт ямар ч төрлийн хувьсагчийг зааж чадах заагч хэрэг болох нь зайлшгүй. Энэ үед доорх байдлаар тодорхойлсон void заагчийг ашигладаг юм.

```
void *ptr;
```

Доорх жишээнд ийм тодорхойгүй заагчийг төрөлтэй заагчийн оронд хэрхэн хэрэглэхийг үзүүлж байна.

```
//ptrvoid.cpp
#include <iostream.h>

void main()
{
int intvar;
float flovar;

int *ptrint;
float *ptrflo;
void *ptrvoid;

ptrint=&intvar;
//ptrint=&flovar; //Алдаатай мөр

//ptrflo=&intvar; //Алдаатай мөр
ptrint=&flovar;

ptrvoid=&intvar;
ptrvoid=&flovar;
}
```

Ptrvoid нь тодорхойгүй заагч учраас дурын төрлийн хувьсагчийн хаягийг авч чадаж байна.

Заагч ба массив

Заагч болон массив нь хоорондоо тун чухал хамааралтай төрлүүд юм. Бид 8 дугаар бүлэгт массивын элементэд хэрхэн хандахыг үзсэн билээ. Бяцхан давтлага болгож arrnote жишээг үзье.

```
//arrnote.cpp
#include <iostream.h>

void main()
{
int intarray[5]={31, 54, 77, 52, 93};

for (int j=0; j<5; j++)
        cout << endl << intarray[j];
}
```

Энэ жишээ нь массивын элементүүдийг дараалан хэвлэж байна. Жишээ нь, j нь 3 утгатай байхад intarray [3] буюу массивын 4 дэх элементэд хандана. Үр дүн нь:

```
31
54
77
52
93
```

Тэгвэл заагч хувьсагч нь массивын элементэд дээрх аргаар илүү ойлгомжтой ханддагийг дараах жишээ харуулна.

```
//ptrnote.cpp
#include <iostream.h>

void main()
{
int intarray[5]={31, 54, 77, 52, 93};

for (int j=0; j<5; j++)
        cout << endl << *(intarray+j) ;
}
```

Программын үр дүн өмнөхөөс огт ялгаагүй. Массивын нэр гэдэг бол бусад хувьсагчийн нэгэн адил уг массивын хаягийг өгөх бөгөөд харин j шилжилтийг нэмснээр массивын j дүгээр элементийг заах болно. Гэтэл энд нэг зүйл анхаарах хэрэгтэй. Intarray бол массивын эхлэлийн хаяг, харин массив нь int төрлийнх. Тэгэхээр нэг элемент нь 2 байт эзлэнэ, гэтэл бид эхлэлээс нь j*2 биш, ердөө j байт шилжүүлж байгаа билээ. Ингэхээр j=4 байх үед бид 4-р элемент биш, 2-р элементэд хандах болох нь ээ. Гэхдээ үүнд санаа бүү зов. C++-ийн компилятор нь энэ “алдааг” өөрөө залруулдаг ажээ. Ө.х. intarray гэдэг бол int массив учраас түүний заагчийн шилжилт нь 2 байтаар явах ёстой гэж ойлгодог байна. Ингээд заагчийг тодорхойлохдоо төрлийг нь зааж өгөхийн бас нэг онцлогийг мэдэж авлаа. Int төрлийн заагчийн шилжилт нь 2 байтаар, double төрлийн заагчийн шилжилт нь 8 байтаар байна гэсэн үг юм.

Заагч тогтмолууд ба заагч хувьсагчууд

J-г intarray дээр нэмэхийн оронд шууд нэмэгдүүлэх (++) оператор ашиглаж болох уу?

Энэ асуултанд болохгүй гэж хариулна. Учир нь тогтмолын утгыг өөрчилж болохгүй. Intarray гэдэг бол системээс уг массивт оноож өгсөн санах ойн хаяг бөгөөд программ дуустал intarray –ийн утга өөрчлөгдөхгүй, тэр чигээрээ байх ёстой. Intarray тогтмол учраас intarray++ гэсэн үйлдэл нь 7++ гэснээс огт ялгаагүй хэрэг болно (Харин windows ажиллаж байх үед систем энэ хувьсагчийн хаягийг автоматаар сольж болно. Гэхдээ энэ явц бидэнд огт мэдэгдэхгүй.). Харин хаягийг өөрчилж болохгүй ч түүн рүү заасан заагчийг өөрчилж болно гэдгийг дараачийн жишээнээс үзэж болно.

```
//ptrinc.cpp
#include <iostream.h>

void main()
{
int intarray[5]={31, 54, 77, 52, 93};
int *ptrint;
ptrint=intarray;

for (int j=0; j<5; j++)
          cout << endl << *(ptrint++) ;
}
```

Бид ptrint хувьсагч тодорхойлж, уг хувьсагчид массивын хаягийг өгснөөр массивын бүх элементэд *(ptrint++) гэсэн үйлдлээр дараалан хүрэх боломж олж байгаа юм. Ptrint бол тогтмол биш учраас өөрчлөгдөж болж байна.

Заагч ба функц

6 дугаар бүлэгт функцэд аргумент дамжуулах гурван арга байдаг тухай үзсэн билээ. Хэрэв функц нь аргументдээ ирж

байгаа хувьсагчийн үндсэн программ дахь утгыг өөрчлөх шаардлагатай бол түүнийг энгийн утга аргументээр дамжуулж болохгүй. Учир нь ийм аргументийн утгын хуулбар нь л функцэд ирдэг. Харин энэ тохиолдолд заагч аргументийг ашиглах хэрэгтэй.

Хувьсагчийг функцэд дамжуулах

Бидний өмнө үзсэнээр функцийн аргументийг хоёрдогч нэрээр дамжуулж болно. Доорх жишээнээс үүнийг харцгаая.

```
//passref.cpp
#include <iostream.h>

void main()
{
void centimize(double &);

double var=10.0
cout << endl << “var=“ << var << “ inches”;

centimize(var);
cout << endl << “var=“ << var << “ centimeters”;
}

void centimize(double& v)
{
v *=2.54;
}
```

Энэ жишээнд var гэсэн хувьсагчийн утгыг инчээс сантиметр лүү хөрвүүлж байна. Centimize функцийн параметрт шууд л хувьсагчийг дамжуулна (гэхдээ өөр нэрээр!). Өөрөөр хэлбэл, v ба var хувьсагчууд нь нэг зүйлийн хоёр нэр болж байгаа юм. Дээрх программын үр дүн:

```
var=10 inches
var=25.4 centimeters
```

Харин одоо заагчаа ашиглаж үзье.

```
//passptr.cpp
#include <iostream.h>

void main()
{
void centimize(double *);

double var=10.0
cout << endl << “var=“ << var << “ инч”;

centimize(&var);
cout << endl << “var=“ << var << “ см”;
}

void centimize(double* v)
{
*v *=2.54;
}
```

Үр дүн нь өмнөхтэйгээ яг адилхан байх болно. Харин энэ тохиолдолд ямар нэгэн утга бус, ямар нэгэн хуурамч нэр ч бус, харин хувьсагчийн хаяг дамжих юм. Функц дотроос энэ хаяганд хандах нь функцийн гадна байгаа хувьсагчид хандана гэсэн үг. Харин энд байгаа олон (*)-ны хоорондбудилж болохгүй. (*v * = 2.54 гэсэн нь *v = *v * 2.54 –тэй адил үйлдэл) Функц дотор *v-тэй ажиллана гэдэг нь функцийн гаднах var –тай ажиллахтай яг ижилхэн болно. Хоёрдогч нэрээр дамжуулах, заагчаар дамжуулах нь компиляторын хувьд яг ижил үйлдэл боловч харагдах байдлаараа тэс өөр хоёр механизм юм.

Массивыг функцэд дамжуулах

5 дугаар бүлгээс эхлэн функцийн аргументэд массив дамжуулдаг нэгэн жишээг бид үзэж байгаа билээ. Өдийг хүртэл заагчийг үзээгүй тул шууд массиваар нь дамжуулан хэрэглэсээр байгаа юм. Харин одоо заагчийг ашиглан массивыг хэрхэн дамжуулахыг үзье.

```
//passarr.cpp
#include <iostream.h>

const int max=5;
void main()
{
void centimize(double *);

double varray[max]={10.0, 43.1, 95.9, 59.7, 87.3};

centimize(varray);

for (int j=0; j<max; j++)
          cout << endl << “varray[” << j << “]=“
        << varray[j] << “ centimeters”;
}

void centimize(double * ptrd)
{
for (int j=0; j<max; j++)
          *ptrd++*=2.54;
}
```

Функцийн толгой бичлэг нь энгийн хувьсагч дамжуулахтай ижилхэн байна. Нэгэнт массивын нэр гэдэг нь массивын хаяг учраас функцийг дуудахдаа centimize(varray) гэж бичихэд л хангалттай. Харин функц дотор энэ хаяг нь ptrd заагчид очно. Ингээд *ptrd++ *=2.54 гэсэн мөрөнд бүх үйлдэл хийгдэж байна.

*ptrd++ байгаа 2 үйлдлийн аль нь түрүүлж биелэх вэ? Өөрөөр хэлбэл заагчийн утга өөрчлөгдөх үү, эсвэл зааж буй хаяган дээрх утга нь өөрчлөгдөх үү? Энэ хоёр оператор ижил эрэмбэтэй үйлдлүүд юм. Хэрэв ямар нэгэн бүлэг үйлдэл хоёроос олон оператор агуулсан байвал компилятор тэдгээрийг баруун талаас нь эхлэн биелүүлдэг. Тийм учраас (++) оператор эхлээд харин (*) оператор сүүлд нь биелэх болно.

Массивын элементүүдийг эрэмбэлэх

Энэ бүлэгт заагч ашиглан массив эрэмбэлэх тухай 2 жишээ үзнэ.

Эхний программ нь 4 дугаар бүлэгт үзсэн reforder программаас зөвхөн хоёрдогч нэрний оронд заагч ашиглаж байгаагаараа л ялгагдана. Функц нь аргументдээ ирсэн 2 тоог эрэмбэлэх бөгөөд 2 дахь нь их байвал 1 дэхийнх нь өмнө нь оруулна гэсэн үг.

```
//ptrorder.cpp
#include <iostream.h>
void main()
{
void order(int*, int*);

int n1=99, n2=11;
int n3:=22; n4:=88;

order(&n1, &n2);
order(&n3, &n4);

cout << endl << " n1=" << n1;
cout << endl << " n2=" << n2;
cout << endl << " n3=" << n3;
cout << endl << " n4=" << n4;
}
```

```
void order(int* numb1, int* numb2)
{
if (*numb1 > *numb2)
        {
        int temp=*numb1;
        *numb1=*numb2;
        *numb2=temp;
        }
}
```

Order функц нь өмнө нь үзэж байсан reforder программын нэг адил боловч ялгаа нь функцийн аргументэд жиших тоонуудын хаяг нь ирж байгаа юм. Программын үр дүн нь доорх маягаар байна.

```
n1=11;
n2=99;
n3=22;
n4=88;
```

Одоо order функцийг ашигладаг ptrsort гэсэн дараагийн жишээг үзье.

```
//ptrsort.cpp
#include <iostream.h>

void main()
{
void bsort(int*, int);
const int N=10;

int arr[N]={37,84,62,91,11,65,57,28,19,49};

bsort(arr, N);

for (int j=0; j<N; j++)
        cout << arr[j] << " ";
}

void bsort(int *ptr, int n)
        {
```

```
        void order(int*, int*)
        int j, k ;

        for (j=0; j < N-1; j++)
                for (k=j+1; k < n; k++)
                        order(ptr+j, ptr+k);
        }
void order(int* numb1, int* numb2)
        {
        if (*numb1 > *numb2)
                {
                int temp=*numb1;
                *numb1=*numb2;
                *numb2=temp;
                }
        }
```

Энэ программын үр дүн нь доорх байдлаар байна.

```
11 19 28 37 49 57 62 65 84 91
```

Заагч ба тэмдэгт мөр

Массивын тухай үзэж байх үед тэмдэгт мөр гэдэг нь char төрлийн массив болохыг тайлбарласан билээ. Тийм учраас массивын элементүүдийн нэгэн адилаар тэмдэгт мөрийн тэмдгүүдэд заагчийг ашиглан хандаж болно.

Тэмдэгт мөрийн заагч

Доорх жишээнд массивын ба заагчийн хэлбэрээр хандах 2 төрлийн тэмдэгт мөрийг авч үзнэ.

```
//twostr.cpp
#include <iostream.h>

void main()
{
char str1[]="Массив болгон тодорхойлов";
char *str2="Заагч ашиглан тодорхойлов";

cout << endl << str1;
cout << endl << str2;

//str1++;          //Тогтмол учраас өөрчлөгдөхгүй
str2++;

cout << endl << str2;
}
```

Ихэнх утгаараа энэ хоёр тэмдэгт мөр нь ижилхэн байх юм. Жишээ нь: Тэдгээрийг нэгэн аргаар хэвлэж болно, функцийн аргументэд өгч болно гэх мэт. Гэхдээ ганцхан том ялгаа бий. Str1 бол энгийн нэг хаяг, str2 бол хувьсагч юм. Тийм учраас str1 нь өөрчлөгдөж болохгүй байхад str2 чөлөөтэй өөрчлөгдөх боломжтой.

Анхааруулга: Хэдийгээр str2 чөлөөтэй өөрчлөгдөж болох хэдий ч энэ нь тэмдэгт мөрийнхөө эхлэлийг зааж байх ёстойг мартаж болохгүй. Программын доорх үр дүнг сайтар анхаарах хэрэгтэй.

```
Массив болгон тодорхойлов
Заагч ашиглан тодорхойлов
аагч ашиглан тодорхойлов
```

Тэмдэгт мөрийг функцэд дамжуулах

Одоо үзэх жишээнд аргументдээ ирсэн тэмдэгт мөрийн үсэг бүрийг дэлгэцэнд хэвлэдэг функц ашиглагдана.

```
//ptrstr.cpp
#include <iostream.h>

void main()
{
void dispstr(char *);
char str[]=“ Эрхийг сурахаар бэрхийг сур ”;
dispstr(str);
}
void dispstr(char * ps)
        {
        cout << endl;
        while (*ps)
                cout << *ps++;
        }
```

Функцийн параметрт тэмдэгт мөр өгөгдөж байна. Уг нь str нь тогтмол боловч функцийн аргументэд ирснээр ps гэсэн заагчид хаяг нь очиж байгаа юм. Ийм учраас заагчийг ашиглан бүх тэмдэгтийг “ дамжин явж “ болно. Давталт нь “\0” тэмдэг гарч иртэл хийгдэх бөгөөд 0 тэмдэг гарч ирэхэд давталтын нөхцөл худал болно.

Заагчийг ашиглан тэмдэгт мөрийг хувилах

Өмнөх жишээнүүдэд заагчийг ашиглан массиваас унших тухай үзлээ. Мөн заагчийг ашиглан массивт утга бичиж болно. Тэмдэгт мөрийг хувилах дараагийн жишээг авч үзье.

```
//copystr.cpp
#include <iostream.h>
void main()
{
void copystr(char *, char *);

char* str1=“ Өөрийгөө ялах гэдэг бол\
          хамгийн том ялалт юм.”;
char str2[80];

copystr(str2, str1);
cout << endl << str2;
}
void copystr(char *dest, char * src);
          {
          while (*src)
                    *dest++=*src++;
          *dest=‘ \0 ’;
          }
```

Программын хамгийн гол хэсэг нь нэг хаяган дээрх тэмдэгтийг уншиж өөр хаяг руу бичих

```
*dest++=*src++;
```

гэсэн комманд юм. 2 заагч хоёулаа нэмэгдэх бөгөөд хуулж дууссаны дараа шинэ мөрийн тэгсгөлд 0 дугаар тэмдэгтийг бичиж байна.

Бидний үзэж байгаа тэмдэгт мөртэй ажиллах ихэнх коммандууд нь C++-ийн стандарт санд байрлах бөгөөд тэр нь STRING.H гэсэн файл юм. Энэ санд байх ихэнх коммандууд тэмдэгт мөрийг аргументдээ авахдаа заагч хэлбэрийг ашигласан байдаг. Бидний бичсэн copystr функцийг уг стандарт сан дахь strcpy функцтэй харьцуулан үзье. Санд доорх байдлаар тодорхойлогдсон байна.

```
char* strcpy(char* dest, const char* src)
```

Функц 2 аргумент авч байна. Хоёр дахь аргументэд байгаа const гэсэн үг нь src гэсэн энэ аргументийн зааж буй тэмдэгнүүд өөрчлөгдөж болохгүй гэсэн үг юм. Энэ функц нь dest мөрийн хаягийг функцийн үр дүнд буцаадаг. Үүнээс бусад нь бидний copystr функцээс ялгаагүй болно.

Тэмдэгт мөрийн заагчийн массив

Бүхэл болон бодит тоон массив байдгийн адилаар бас заагчийн массив байж болно. Ихэвчлэн тэмдэгт мөрийн заагчаар массив үүсгэдэг.

Өмнө 5 дугаар бүлэгт бид тэмдэгт мөрийн массив ашиглах жишээ үзсэн билээ. Ингэхэд гарах гол дутагдал нь массивт байгаа тэмдэгт мөрүүд заавал ижил урттай байх шаардлагатай болдог. Энэ нь санах ойг хэмнэлт багатай ашиглахад хүргэнэ.

Харин энэ асуудлыг заагчийн массив хэрхэн шийдэхийг одоо үзэцгээе.

```
//ptrtostr.cpp
#include <iostream.h>
const int DAYS=7;

void main()
{
char* arrptrs[DAYS]={“Ням”, “Даваа”, “Мягмар”,
                    “Лхагва”, “Пүрэв”,
                    “Баасан”, “Бямба” } ;

for (int j=0; j < DAYS; j++)
          cout << arrptrs[j] << endl;
}
```

Программын үр дүн нь:

Ням
Даваа
Мягмар
Лхагва
Пүрэв
Баасан
Бямба

Массив дахь мөрүүдийг иймэрхүү байдлаар тодорхойлсон үед уг мөрүүдийг санах ойд цувуулан үүсгэх болно. Ийм учраас ямар нэгэн мөрийг авахын тулд массивын эхлэлээс шилжүүлэх аргаар биш, харин индексийн аргаар шилжих хэрэгтэй.

Санах ойг зохион байгуулах нь. New ба Delete оператор

Бид санах ойн муж авахын тулд дандаа массивыг ашигласаар ирсэн. Жишээ нь:

Int arr[100]

Энэ нь санах ойгоос 100 ширхэг бүхэл тоонд зориулсан муж буюу тодруулбал 200 байтыг авч, arr гэсэн хувьсагчид хаягийг нь өгч байгаа юм. Ингэж тодорхойлоход бид программаа бичиж байхдаа уг массив ямар хэмжээтэй байхыг мэдэж байх шаардлагатай болж байна. Программ нэгэнт эхэлсэн хойно массивын хэмжээг өөрчлөх ямар ч боломжгүй юм. Жишээ нь:

```
cin >> size;
int arr[size];
```

гэсэн бичлэгийг компилятор зөвшөөрөхгүй. Гэтэл программ дотор хэрэглэгдэх мужийн урт нь программ биелэж байх явцад өөрчлөгдөх шаардлага тун их гардаг.

NEW оператор

Энэ оператор нь өгсөн хэмжээний мужийг санах ойгоос авч, мужийн хаягийг буцааж өгдөг. Newintro хэмээх доорх жишээг үзэцгээе.

```
//newintro.cpp
#include <iostream.h>
#include <string.h>

void main()
{
char *str="Эдээр биеэ чимэхээр эрдмээр биеэ чим.";
int len=strlen(str) ;

char *ptr;
ptr=new char[len+1];

strcpy(ptr, str);
cout << endl << "ptr=" << ptr ;

delete ptr;
}
```

ptr = new char [len+1] гэсэн үйлдэл нь strlen фукцээр авсан урт дээр 0 дугаар тэмдэгтэд зориулсан 1 байтыг нэмсэнтэй тэнцэх хэмжээний санах ойг авч, хаягийг нь ptr хувьсагчид өгч байгаа юм. Харин авах санах ойн хэмжээг дөрвөлжин хаалтанд хийхээ мартаж огт болохгүй. Хэрэв та дугуй хаалт ашиглавал алдаа гарахгүй боловч, буруу үр дүнд хүрэх болно. Бид нэгэнт ижил хэмжээтэй санах ойг үүсгэсэн учраас

strcpy фукцийг ашиглан тэмдэгт мөрүүдийг хувилж болно.

DELETE оператор

Хэрвээ таны программ маш олон удаа new операторыг хэрэглэн санах ой авбал эцэстээ системийн санах ой дүүрч, компьютер цаашид ажиллах боломжгүй болж, систем зогсоход хүрнэ. Иймд санах ойг зүй зохистой ашиглахын тулд программд авсан санах ойн муж бүрийг чөлөөлж байх ёстой. Энэ зорилгоор delete операторыг ашигладаг.

```
Delete ptr;
```

Энэ комманд нь ptr хувьсагчийн зааж байгаа санах ойн мужийг чөлөөлж, системд эргүүлэн өгнө. Хэрвээ та энэ үйлдлийг орхигдуулбал, программ дуусах үед заагч хувьсагч устах боловч түүний зааж буй санах ой чөлөөлөгдөхгүй. Компьютер унтартал хэн ч хэзээ ч олж авахгүй санах ойн хаягдмал муж болон үлдэх болно.

Харин энэ операторыг ажиллуулснаар уг санах ойг зааж буй заагч хувьсагч устахгүй. Харин хоосон заагч болон үлдэнэ. Санах ойг нь чөлөөлсөн заагчтай ажиллавал утга олгоогүй заагчтай ажилласны нэгэн адил алдаанд хүргэж болзошгүй.

New оператор ашигласан string ангийн тухай

New операторыг ихэвчлэн байгуулагч функц дотор бичдэг. Одоо үзэх жишээнд бидний 7 дүгээр бүлэгт үүсгэж байсан string ангийг өөрчлөх болно.

Бидний өмнө тодорхойлсон ангид тэмдэгт мөрийн урт нь тогтмол учраас санах ойн ашиглалт муутай байсан билээ. Одоо доорх жишээг үзэцгээе.

```
//newstr.cpp
#include <iostream.h>
#include <string.h>

class string
        {
        private:
        char* str;

        public:
        string(char* s)
                {
                int length=strlen(s) ;
                str=new char[length+1] ;
                strcpy(str, s);
                }
        ~string()
                {
                delete str;
                }
        void display()
                {
                cout << str ;
                }
        };

void main()
{
string s1="Юу ч хийдэггүй хүн юунд ч ороолддоггүй";

cout << endl << "s1=";
s1.Display;
}
```

Энэ анги доторх өгөгдөл нь ердөө str гэсэн ганцхан заагч юм. Уг заагч нь new операторoор үүсгэгдсэн санах ойн мужийг зааж байх бөгөөд ангийг

санах ойгоос устгах үед үүсгэсэн муж автоматаар устгагдах болно.

Байгуулагч функцэд (конструктор) нь шинээр санах ой авч, орж ирсэн тэмдэгт мөрийг уг санах ойдоо хувилан авна. Харин өмнө нь бид устгагч функц (деструктор) бараг үзээгүй билээ. Гэхдээ ангийг устгах үед автоматаар ажилладаг функцийг устгагч функц гэдэг тухай бид өмнө нь үзэж байсан. Энэ жишээний устгагч функц нь санах ойд авсан мужийг чөлөөлөх үүрэгтэй юм. Ийм учраас программ дуусах үед санах ойг бүрэн чөлөөлж байгаа тухайд санаа зовох шаардлагагүй юм.

Обьектийн заагч

Заагч нь энгийн төрлүүдийн нэгэн адил обьектийг ч бас зааж болно. Өмнө нь бид Distance гэсэн ангийн dist гэдэг обьектийн тухай үзэж байсан.

Зарим тохиолдолд программ дотор тун ч олон обьект үүсгэх шаардлага гардаг. Харин бид нэгэнт new операторыг үзсэн учраас энэ нь тийм ч төвөгтэй асуудал биш боллоо. Доорх жишээг үзэцгээе.

```
//englptr.cpp
#include <iostream.h>

class Distance
        {
        private:
        int feet;
        float inches;

        public:
        void getdist()
                {
                cout << “\nФут: ”;
```

```
                cin >> feet;
                cout << “\nИнч: ”;
                cin >> inches;
                }
        void showdist()
                {
                cout << feet << “\’-” << inches << ‘\”’;
                }
        };

void main()
{
Distance dist;
Dist.getdist() ;
Dist.showdist() ;

Distance* distptr;
Distptr=new Distance;
Distptr->getdist() ;
Distptr->showdist() ;
}
```

Обьектийн гишүүнд хандах нь

Өмнөх жишээнд обьектийн энгийн болон заагч байдлаар 2 янз тодорхойлжээ. Эхнийх нь энгийн обьект учраас үүсгэх бөгөөд талбаруудад хандах бидний өмнө үзсэний дагуу байна. Харин 2 дахь тохиолдолд new операторыг ашиглан обьектийн заагч үүсгэжээ. Уламжлалт аргаар бол обьектийн гишүүнд бид доорх байдлаар хандах болно.

```
(*distptr).getdist();
```

Гэхдээ энэ нь бичлэгийн хувьд эвгүй учраас С++-д заагч обьектийн гишүүнд доорх байдлаар ханддаг юм.

```
distptr->getdist();
```

Программын үр дүн нь:

```
Фут: 10
Инч: 6.25
10'-6.25"

Фут: 6
Инч: 4.75
6' – 4.75"
```

Обьектийн заагчийн массив

Программд обьектийн заагчийн массивыг ашиглах нь бүлэг обьекттой ажиллахад хамгийн тохиромжтой зохион байгуулалт юм. Дараагийн жишээнд person гэсэн ангийн заагчуудын массивыг үүсгэнэ.

```
//ptrobjs.cpp
#include <iostream.h>

class person
        {
        protected:
        char name[40];

        public:
        void setName()
                {
                cout << "Нэрээ оруулна уу : ";
                cin >> name;
                }
        void printName()
                {
                cout << "\n Нэр: " << name;
                }
        };
```

```
void main()
{
person* persPtr[100];
int n=0;
char choice;

do
        {
        persPtr[n]=new person;
        persPtr[n]-> setName();
        n++;
        cout << “Дахиад оруулах уу (y/n)? ”;
        cin >> choice;
        }
while (choice==’y’);

for (int j=0; j<n; j++)
        {
        cout << “\nДугаар ” << j+1;
        persPtr[j]-> printName();
        }
}
```

Программ доорх байдлаар ажиллана.

```
Нэрээ оруулна уу : Richie
Дахиад оруулах уу (y/n)? y
Нэрээ оруулна уу : John
Дахиад оруулах уу (y/n)? y
Нэрээ оруулна уу : Sarah
Дахиад оруулах уу (y/n)? N
Дугаар 1
Нэр: Richie
Дугаар 2
Нэр: John
Дугаар 3
Нэр: Sarah
```

Энэ жишээнээс харж байхад обьектийн гишүүн функцэд обьектийн гишүүний нэгэн адил бичиглэлээр ханддаг ажээ.

```
persPtr[j]->getName();
```

Холбоост жагсаалт

Одоо холбоост жагсаалтын жишээг үзэцгээе. Холбоост жагсаалт бол массивын нэг адил өгөгдлийг хадгалах нэг төрлийн арга юм. Холбоост жагсаалт бол элементийнх нь тоо тогтмол биш байдгаараа массиваас ялгаатай.

Заагчийн “гинж”

Холбоост жагсаалтын бүх элемент new операторoор үүсгэгдэх бөгөөд шинэ элемент бүр өмнөх элементтэйгээ мөн л заагчийг ашиглан холбогддог юм. Элементүүд массив шиг заавал санах ойн дараалсан мужид байрлах албагүй, нэгэнт холбоосоор холбогддог учраас санах ойн хаана ч байрлаж болно.

Бидний жишээнд нэг элемент бүр нь link бүтэцтэй, харин бүхэл холбоост жагсаалт нь linked list ангийнх байх юм. Link бүтэц нь нэг бүхэл тоо ба дараагийн элементийг заах заагчаас тогтох болно. Харин холбоост жагсаалтын заагч нь холбоост жагсаалтынхаа хамгийн эхний элементийг зааж байна.

```
//linklist.cpp
#include <iostream.h>

struct link
        {
```

```
        int data;
        link *next;
        };

class linklist
        {
        private:
        link *first;

        public:
        linklist()
                { first=NULL; }
        void additem(int d);
        void display();
        };

void linklist::additem(int d)
        {
        link *newlink=new link;
        newlink-> data=d;
        newlink-> next=first;
        first=newlink;
        }

void linklist::display()
        {
        link *current=first;
        while (current!=NULL)
                {
                cout << endl << current-> data;
                current=current-> next;
                }
        }

void main()
{
linklist li;

li.additem(25);
li.additem(36);
li.additem(49);
li.additem(64);
```

```
li.display();
}
```

Холбоост жагсаалтыг анх үүсгэхэд түүний жагсаалтын эхлэлийг заах first заагчид NULL утга өгч байна. Энэ нь уг заагч юу ч заахгүйг илэрхийлж байгаа юм.

Additem гэсэн дүрмээр жагсаалтад элемент нэмж байна. Элемент нь жагсаалтын эхэнд нэмэгдэх (сүүлд нь нэмж болох боловч энэ нь ялигүй төвөгтэй болно) бөгөөд энэ явцыг нэг бүрчлэн тайлбарлая.

```
link *newlink=new link;
```

Энэ мөр нь шинээр link бүтэцтэй элемент үүсгэж, хаягийг нь newlink гэсэн хувьсагчид авч байна. Үүний дараа шинэ элементийн утгуудыг оноож байна. Сүүлийн 2 мөр нь элементийг жагсаалтад оруулж байна. Оруулахдаа хуучин жагсаалтын эхлэл заагчийн утгыг шинэ элементийн дараагийн заагчид өгч, харин шинэ элементийг жагсаалтын эхлэл болгон first заагчид өгч байна.

```
newlink->next=first;
first=newlink;
```

Одоо жагсаалтыг хэвлэж байгаа явцыг бага зэрэг тайлбарлая. Жагсаалтыг нэгэнт үүсгэсэн үед түүнийг дараалуулан хэвлэх нь нэг их төвөгтэй биш алхам юм. Гол зарчим нь гинжний эхлэлээс дараа дараагийн элемент рүү явах бөгөөд next гишүүний утга NULL болтол давтах юм. Өөрөөр хэлбэл,

```
cout << endl << current-> data
```

гэсэн коммандаар өгөгдлийг хэвлэж,

current=NULL

нөхцөл үнэн болтол

current=current->next

коммандаар дараагийн элемент рүү очих болно. Программ доорх үр дүн үзүүлнэ.

64
49
36
25

Жагсаалтын давуу талуудыг тодорхой тайлбарласан. Харин дутагдал нь ямар нэгэн элемент дээр очихын тулд заавал гинжний эхлэлээс хөдөг явдал юм. Энэ нь цаг хугацааны алдагдалтай болно гэсэн үг.

Өөрийгөө агуулдаг обьект

Энэ тухай тун товчхон үзье. Өмнөх жишээн дэх link бүтцийн гишүүн нь link бүтцийн заагчаар тодорхойлогдсон байгаа билээ. Өөрөөр хэлбэл ийм бүтцийн гишүүн нь өөртэйгээ ижил обьект рүү зааж байна. Харин өөрийгөө заадаг ийм гишүүн нь заагч биш энгийн төрөл байхыг компилятор зөвшөөрөхгүй. Ө.х:

```
Class sample
        {
        sample *ptr ;               //Зөвшөөрнө
        sample obj ;      //Зөвшөөрөхгүй.
        }
```

Linklist – ийг өргөтгөх нь

Бидний энэ жишээг дахин өргөжүүлэх боломжтой юм. Тухайлбал, link бүтцэд ганц бүхэл тоон гишүүнээс гадна, олон гишүүн, тэр дунд массив, бичлэг, обьект гэх мэтчилэн нийлмэл төрлүүд, тэр ч бүү хэл заагч ч байж болно.

Мөн linklist ангид гишүүн функцүүд нэмж болно. Тухайлбал өгсөн байрлал дээр элемент нэмэх, өгсөн байрлалаас элемент устгах дүрмүүд байж болно. Мөн энд дутагдалтай нэг дүрэм бол устгагч функц юм. Энэ функц нь гинжийг чөлөөлдөг байх ёстой юм.

Заагчийн заагч

Одоо үзэх жишээгээр энэ бүлгээ төгсгөе. Энэ нь үндсэндээ ptrobjs программтай адил боловч ptrsort программын order, bsort функцуудийг ашиглаж person ангийн обьектуудыг цагаан толгойн дарааллаар эрэмбэлэх болно.

```
//persort.cpp
#include <iostream.h>
#include <string.h>

class person
        {
        protected:
        char name[40];

        public:
        void setName()
                {
                cout << “Нэрээ оруул : ”;
```

```
                cin >> name;
                }
        void printName()
                {
                cout << "\n Нэр: " << name;
                }
        char *getname()
                {
                return name;
                }
        };

void main()
{
void bsort(person **, int);
person* persPtr[100];
int n=0;
char choice;

do
        {
        persPtr[n]=new person;
        persPtr[n]-> setName();
        n++;
        cout << "Дахиад оруулах уу(y/n)? ";
        cin >> choice;
        }
while (choice=='y');

cout << "\nЭрэмбэлээгүй жагсаалт: ";
for (int j=0; j<n; j++)
        persPtr[j]-> printName();

bsort(persPtr, n);

cout << "\nЭрэмбэлсэн жагсаалт: ";
for (int j=0; j<n; j++)
        persPtr[j]-> printName();
}

void bsort(person **pp, int n)
        {
        void order(person*, person*)
```

```
        int j, k ;

        for (j=0; j < N-1; j++)
                for (k=j+1; k < n; k++)
                        order(pp+j, pp+k);
        }
void order(person* pp1, person* pp2)
        {
        if (strcmp((*pp1)->getName(),
          (*pp2)->getName()) > 0)
                {
                person* tempptr=*pp1;
                *pp1=*pp2;
                *pp2=tempptr;
        }
}
```

Энэ программ нь хэрэглэгчийн оруулсан 100 хүртэл нэрийг эрэмбэлнэ. Программын ажиллагаа доорх байдлаар харагдана.

```
Нэрээ оруул: Washington
Дахиад оруулах уу(y/n)? y
Нэрээ оруул: Adams
Дахиад оруулах уу(y/n)? y
Нэрээ оруул: Jefferson
Дахиад оруулах уу(y/n)? y
Нэрээ оруул: Madison
Дахиад оруулах уу(y/n)? n

Эрэмбэлээгүй жагсаалт:
Washington
Adams
Jefferson
Madison

Эрэмбэлсэн жагсаалт:
Adams
Jefferson
Madison
Washington
```

Энд хэрэв та анзаарсан бол бид жинхэнэ өгөгдлүүдийг эрэмбэлсэнгүй. Харин тэдгээрийг зааж буй хаягуудыг эрэмбэллээ. Энэ нь цаг хугацааны хувьд асар их хэмнэлттэй бөгөөд заагчийн ашгийг жинхэнэ утгаар нь гаргаж байгаа юм.

Бид bsort функцийн эхний аргументийг 2 давхар одтой тодорхойлсон байгаа билээ. Энэ нь ямар нэгэн шинэ зүйл биш бөгөөд ерөөсөө л заагчийн заагчийг илэрхийлж байгаа юм. Дамжуулах утга нь person обьектийн заагч байгаа бөгөөд харин функцийн аргументэд очихдоо тэр утгын заагч очих ёстой учраас л тэр. Ө.х бид жинхэнэ person обьектуудыг биш, тэдний заагчийг хооронд нь солих учраас тэдгээр заагчийн заагч функцэд дамжих ёстой.

Тэмдэгт мөрүүдийг жиших

Order функцэд ашиглагдсан strcmp функц нь аргументдээ өгсөн 2 мөрийг харгалзах үсгүүдээр жишиж үзээд доорх утгуудын нэгийг буцаадаг байна.

<0	Эхний мөр 2 дахь мөрийн өмнө бол
0	Хоёр мөр тэнцүү бол
>0	Эхний мөр 2 дахь мөрийн дараа бол

Заагчийн тухай санамжууд

Заагч нь программын болон системийн хамгийн сайн зохион байгуулагч төдийгүй, хамгийн аюултай дайсан юм шүү. Хэрэв программчлагч заагчийг зөв зохистой зохицуулж чадахгүй бол, хэрэв заагчийг

хяналтаасаа гаргах юм бол, new ба delete функцүүдийг зөв ашиглаж чадахгүй бол тэдгээр нь таны хүссэн газар луу бус, санах ойн өөр газар луу заана. Өөрөөр хэлбэл, системийн болон программын биелэх кодуудын хэсэг рүү зааж байна гэж саная. Энэ үед таны программ тэр хаяг руу ямар нэгэн зүйл бичиж орхивол программ болон системийн код буруу биелэгдэж, ингэснээрээ компьютер гацах хүртэл аюултай байдалд оруулах юм. Ийм байдлаас зайлсхийхийн тулд watch цонхыг ашиглах хэрэгтэй юм. Энэ цонхонд дурын хувьсагчийн утга төдийгүй, санах ойн дурын хаяг дээрх өгөгдлийг харж болдог. Үүнийг ашиглан программын буруу ажиллаж байгаа хэсгийг илрүүлж болох юм.

Бүлгийн дүгнэлт

Энэ бүлэгт бид заагчийн тухай өргөн дэлгэр мэдлэгийг олж авлаа.

Компьютерийн санах ойн нэг байт бүр өөрийн хаягтай байдаг бөгөөд тэр хаягийг заадаг хувьсагчийг заагч хувьсагч гэж нэрлэдэг. Программд тодорхойлогдсон хувьсагч бүр тухайн хувьсагчид оноож өгсөн санах ойн мужийн хаягийг хадгалж байдаг. Энэ утгаар нь энгийн хувьсагчдыг тогтмол заагч гэж нэрлэж болох юм. Тийм хувьсагчдын хаягийг (&) функц ашиглан авч болдог.

Харин заагч хувьсагч бол санах ойн хаана ч зааж болох чөлөөтэй хувьсагч юм. Ө.х. заагчид анхнаас нь оноож өгсөн санах ойн хаяг байдаггүй буюу заагч тодорхойгүй байдаг. Заагчийг (*) ашиглан тодорхойлно. Заагчийг тодорхойлохдоо заавал төрлийг нь зааж өгөх шаардлагатай бөгөөд тэр

төрлийг нь ашиглан компилятор заагч дээрх арифметик үйлдлийг хийдэг.

Массивын элементэд заагч ашиглан хандаж болно. Бусад хувьсагчийн адил массивын хаяг нь тогтмол боловч энэ хаягийг заагчид өгснөөр массивын дурын элемент дээр шууд очих боломжтой болох юм.

Функцийн аргументэд заагчийг өгснөөр функц жинхэнэ хувьсагчтай шууд ажиллах боломжтой болдог. Аргументийг заагчаар дамжуулах нь хоёрдогч нэрээр дамжуулахтай компиляторын хувьд ижилхэн үйлдэл боловч хэрэглэгчийн хувьд арай ялгаатай юм.

Тэмдэгт мөр нь массиваар тодорхойлогдохоос гадна мөн л заагчаар тодорхойлогдож болно.

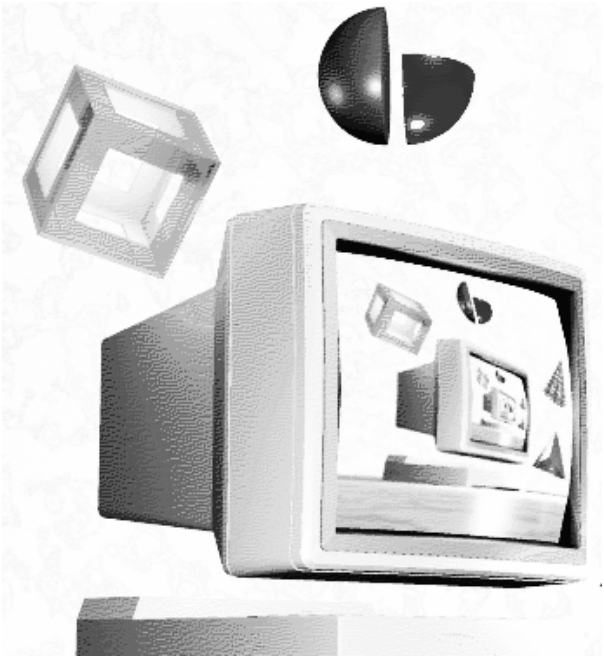
New оператор өгсөн хэмжээний санах ойг нөөцлөн авч, тэр мужийн хаягийг өгдөг. Энэ оператор программ ажиллаж байх үед хувьсагч үүсгэхэд ашиглагдана. Ингэж авсан санах ойн мужийг delete операторoop чөлөөлдөг.

Заагч ямар нэг обьектийг зааж байхад тухайн обьектийн гишүүн өгөгдөл болон функц рүү (->) операторыг ашиглан ханддаг.

Анги болон бүтэц нь өөртөө өөртэйгээ ижил төрлийн заагч агуулсан байж болно. Энэ нь холбоост жагсаалт мэтийн өгөгдлийн тусгай бүтцүүдэд чухал үүрэгтэй.

Заагчийн заагч байж болно. Энэ тохиолдолд бичлэг нь int** ptr гэсэн хэлбэртэй болно.

9-р бүлэг



-Урсгалын тухай ойлголт

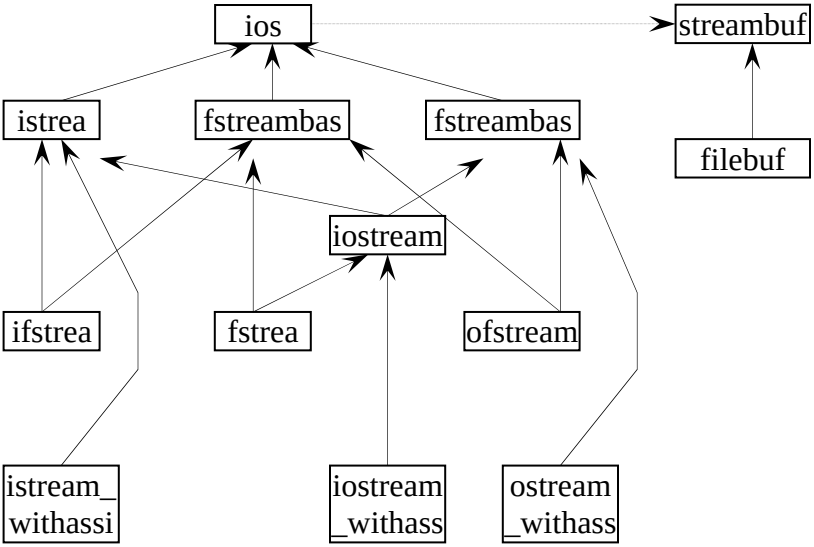
9-р бүлэг

Урсгалууд (Streams)

Урсгал гэдэг нь ямар нэг мэдээллийн урсгалуудыг нэрлэсэн ерөнхий нэр юм. Ялгаатай урсгалууд нь ялгаатай төрлийн мэдээллүүд рүү ханддаг. Урсгал ангиуд нь харгалзах мэдээллийн төрлүүдтэйгээ ажиллах гишүүн функцүүдтэй байдаг.

Урсгалын ангиудын удамшил

Урсгалын ангиудын удамшилыг харуулсан ургийн модыг юуны түрүүнд үзье.



istream анги нь урсгалаас задлах оператор гэж нэрлэгдэх ‘>>’ операторыг тодорхойлсон байдаг ба мөн ostream анги нь урсгал руу оруулах оператор гэж нэрлэгдэх ‘<<’ операторуудыг тодорхойлсон байдаг. Эдгээр хоёр анги нь хоёулаа ios ангиас үүссэн ангиуд юм.

Бид урсгалын тухай өмнө үзээгүй байсан болохоор cout, cin-ийг функц гэж тайлбарлаж байсан билээ. Өмнө үзсэн cout обьект нь дэлгэц рүү хандсан стандарт гаргах төхөөрөмжийн обьект бөгөөд энэ нь ostream_withassign ангийн обьект юм. Харин ostream_withassign анги нь ostream ангиас удамшсан анги юм.

Үүнтэй төстэй istream ангиас удамшисан istream_withassign ангийн cin обьект стандарт оруулах төхөөрөмжийн урсгал юм.

Стандарт оруулах, гаргах төхөөрөмжийн обьектууд нь iostream.h файлд тодорхойлогдсон байдаг. Харин дискийн файлтай ажиллах урсгалууд нь fstream.h файлд тодорхойлогдсон байдаг.

Урсгалын ангиуд

Ургийн модны зургийг харахад ios анги нь бүх урсгалын ангиудын язгуур анги юм. Энэ анги бүх төрлийн өгөгдөлийг оруулах гаргах үйлдлүүдийг агуулсан анги юм. ios анги нь streambuf анги руу заасан заагчийг агуулдаг ба энэ анги нь орох гарах мэдээллийн буфер юм.

istream болон ostream ангиуд нь ios ангиас удамшах ба ерөнхийдөө оруулах гаргах үйлдлүүдэд зориулагдсан ангиуд юм. istream анги нь урсгалаас задлах оператор >> -ийг, харин ostream анги нь урсгалд оруулах << операторыг агуулж байдаг.

iostream ангиуд олон зэрэг удамшилаар istream, ostream ангиудаас зэрэг удамшин үүссэн анги юм.

istream_withassign, ostream_withassign, iostream_withassign ангиуд нь харгалзан istream, ostream, iostream ангиудаас үүссэн ба урсгалыг ямар нэг сувагуудтай холбоход зориулагдсан ангиуд юм.

Файлтай холбогдох дараах хэдэн ангиуд байдаг. Файлыг унших байдлаар холбогдох ifstream, файлд гаргах байдлаар холбогдох ofstream болон гаргах ба оруулах байдлаар холбогдох fstream ангиуд байдаг.

ifstream, ofstream, fstream ангиуд нь fstream.h файлд агуулагддаг ба бусад нь iostream.h файлд агуулагддаг. Гэхдээ fstream.h файл нь iostream.h файлыг давхар агуулдаг учир fstream.h файлыг хэрэглэж байгаа тохиолдолд iostream.h толгой файлыг давхар тодорхойлох шаардлагагүй юм.

Ингээд урсгалыг хэрхэн хэрэглэж болох тохиолдолуудыг дэлгэрүүлэн үзье.

Тэмдэгт мөрийг оруулах, гаргах

Бид эхлээд дискэнд тэмдэгт мөр бичих хялбар программ үзье.

```
#include <fstream.h>

void main()
        {
        ofstream outfile("test.txt");

        outfile << "Анхны тэмдэгт_ мөр\n";
        outfile << "Цааш үргэлжилж байна.\n";
 }
```

Бид дээрх жишээнд ofstream ангийн outfile обьектийг үүсгэх үедээ түүнд файлын нэрийг өгч байна. Ингэсэн үед дискний идэвхтэй каталогид өгсөн нэртэй файл үүсч байгаа юм. Үүний дараа урсгал руу оруулах операторын тусламжтай мэдээллийг файл руу бичиж байна. Ингээд программ дуусах үед outfile обьект устгах функцийг дуудах бөгөөд энэ үед файлыг хаах үйлдэл хийгдэх ба энэ үед урсгалын буферт орсон өгөгдөл дискэнд (буюу нээлттэй файлд маань) бичигддэг юм.

Одоо дискийн файлаас тэмдэгт мөрийг хэрхэн унших жишээ үзье.

```
#include <fstream.h>

void main()
        {
        const int MAX=80;
        char buffer[MAX];
        ifstream infile("test.txt");
        while (infile)
                {
                infile.getline(buffer, MAX);
                cout << buffer;
                }
        }
```

Энэ тохиолдолд урсгал руу оруулах операторыг ашиглахгүй харин getline() функцийг хэрэглэж байна. Учир нь тэмдэгт мөрийг уншиж байгаа тохиолдолд энэ функцийг ашиглах нь зохимжтой байдаг юм. Энэ функц нь \n тэмдэгт хүртэл уншаад үр дүнг буферт шилжүүлнэ. Харин 3-р аргумент буфферийн хэмжээ байх ёстой.

Уг урсгалаар нээгдсэн файлын заагч нь төгсгөл дээрээ хүрсэн эсэхийг бид урсгалын нэрийг ашиглан шалгаж байна.

Тэмдэгт оруулах, гаргах

Урсгалаас тэмдэгтийг унших, гаргахад get(), put() функцүүдийг ашиглана.

Жишээлбэл:

```
. . .
outfile.put(str[j]);
. . .
infile.get(ch);
. . .
```

Обьектыг оруулах, гаргах

Урсгалд обьектийг оруулах, гаргах боломжтой. Ингээд обьектийг дискэнд бичих дараах жишээг үзье.

```
#include <fstream.h>

class person
        {
        protected:
        char name[40];
        int age;

        public:
        void getdata(void)
                {
                cout << "Нэрээ оруул:"; cin >> name;
                cout << "Насаа оруул: "; cin >> age;
                }
        };
```

```
void main()
{
person pers;

pers.getdata();
ofstream outfile("person.dat");
outfile.write((char *)&pers, sizeof(pers));
}
```

Тэгэхээр обьектийг дискэнд write() функцийн тусламжтай бичих боломжтой. Энэ функц нь хоёр аргументтай функц юм. Эхний аргумент нь бичигдэх обьектийн хаяг байна. Харин дараагийн аргумент нь обьектийн хэмжээ байх ёстой энэ хэмжээг sizeof операторын тусламжтай авдаг.

Одоо обьектийг дискнээс хэрхэн уншихыг үзье. Ингэж уншихад read() функц ашиглагдана.

```
infile read((char *)&pers, sizeof(pers));
```

fstream анги

fstream анги нь оруулах болон гаргахад зориулагдсан урсгал юм. Үүнийг дараах байдлаар ашиглаж болно.

```
. . .

fstream file;

file.open(“person.dat”,ios::app|ios::out|ios::in);
file.write((char *)&pers, sizeof(pers));

. . .

file.write((char *)&pers, sizeof(pers));
file.seekg(0);
file.read((char *)&pers, sizeof(pers));

. . .

if (!file.eof())
```

```
        file.read((char *)&pers, sizeof(pers));

. . .

file.close();

. . .
```

Файлтай холбоотой урсгалыг нээхдээ open() функцийг ашиглаж болно. Энэ функц нь файлын нэр болон горимын битүүдийг авдаг. Эдгээр битүүдийн тогтмолыг ios ангид статик тогтмолоор зарласан байдаг. Манай жишээнд app гэсэн бит хэрэглэсэн байгаа нь тухайн файлын төгсгөлд мэдээлэл нэмж бичих зорилгоор нээж байгаа хэрэг юм. Ингээд горимын битүүдийг товч тайлбарлая.

Горимын бит	Тайлбар
in	файлыг уншихаар нээнэ.
out	файлд бичихээр нээнэ.
app	файлыг төгсгөлд нэмж бичихээр нээнэ.
ate	файлыг унших болон бичихээс өмнө идэвхтэй байрлалаас хойшхи хэсгийг таслана.
nocreate	файлыг нээхэд байхгүй бол алдаа өгөхийг заана.
noreplace	файлыг нээхэд тухайн файл байвал алдаа өгөхийг заана.
binary	файлыг текст биш хоёртын файл байдлаар нээхийг заана.

Ингээд файлтай ажиллах үедээ файлын төгсгөлийг eof() функцийн тусламжтай мэдэх бололцоотой байдаг.

Файлын заагчууд

Файлтай ажиллах бүх урсгалууд нь файлд бичих байрлалын заагч (put pointer) болон файлаас унших байрлалын заагч (get pointer) гэсэн хоёр заагчуудтай байдаг. Эдгээр нь

файлын хэд дэх байт руу бичих ба хэд дэх байтаас уншихад бэлэн байгааг зааж байдаг. Унших заагчийг байрлуулахын тулд seekg() функцийг, бичих заагчийг байрлуулахдаа seekp() функцийг тус тус хэрэглэдэг байна.

Манай жишээнд seekg(0) гэж хэрэглэсэн унших байрлалыг файлын эхэнд байрлуулж байна. Мөн түүнчлэн энэ функц нь хоёр аргументтай хэрэглэгдэж болно. Энэ тохиолдолд эхний аргумент шилжилтийн хаяг харин дараагийн аргумент нь аль байрлалаас шилжилтийн хаягийг тооцохыг заадаг. Хоёрдугаар аргумент нь файлын эхлэл-beg, файлын төгсгөл-end, идэвхтэй байрлал-cur гэсэн гурван утга авч болно. Жишээлбэл:

```
seekp(-10,ios::end);
```

гэх мэтээр файлын заагчийг шилжүүлж болно. Мөн seekg() функц нь дээрхтэй ижил хэрэглэгдэнэ.

Энэ хоёр функцийн эсрэг үүрэгтэй tellp(), tellg() функцүүд байдаг ба эдгээр нь файлын заагчийн байрласан утгыг буцаадаг байна. Иймд файлын хэмжээг дараах байдлаар мэдэх боломжтой юм.

```
filelength = infile.seekg(0, ios::end);
```

Өмнөх жишээнүүдэд программ хаагдах болон обьект устах үед файл хаагдаж байна. Хэрэв файлыг шууд хаах шаардлага гарвал close() функцийг хэрэглэж болно.

Алдааг боловруулах

Бид файлтай ажиллах үедээ алдаа хэдийд гарах, түүнийг яаж боловсруулах тухай яриагүй билээ. Тэгвэл алдааг боловсруулахад урсгалын төлөвийн битүүд хэрэглэгддэг. Төлөвийн битүүд нь төлөвийн байтад агуулагддаг.

Урсгалын төлөвийн байтыг авч үзье.

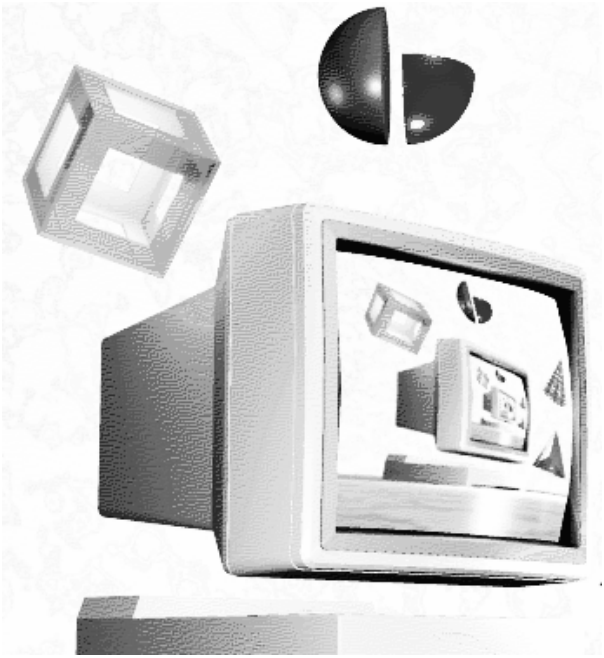
				hardfail	bad	fail	eof

Дээрх битүүдийг товч тайлбарлья.

eofbit	файлын төгсгөлийн бит

failbit	Унших бичих үйлдэл алдаатай явагдсан
badbit	Үйлдэл биелээгүй (Файл нээж байгаа бол файл нь байхгүй байж болно)
hardfail	Физик алдаа

10-р бүлэг



-Дүрийн төрөл буюу template анги

10-р бүлэг

Дүрийн төрөл-Templates

Дүрийн төрөл буюу параметрт төрөл гэж нэрлэгддэг энэ төрөл нь хоорондоо уялдаа холбоотой ангиуд болон функцүүдийг үүсгэх боломжийг олгодог.

Функцийн дүр

Бид max(x, y) гэсэн x, y аргументыг ихийг нь буцаах функц бичих тохиолдолд аргументуудын төрөлийг заавал тодорхой заах ёстой. Гэтэл дурын төрөлийн аргументыг өгөх шаардлага гардаг. Эсвэл төрөл бүрийн хувьд энэ функцийг давтан тодорхойлох шаардлага гарна. Харин үүнийг дараах макрог ашиглан шийдэх боломж байдаг.

```
#define max(x, y) ((x > y) ? x : y)
```

Гэвч макрог хэрэглэх нь C++ хэлний хувьд хуучирсан хэрэгцээнээс гарсан арга юм. Тэгээд ч энэ аргыг хэрэглэхэд нийцэхгүй хоёр төрлийн хувьд жиших үйлдлийг хийхгүй ба хөрвүүлэлт хийх үед алдаа гардаг. Тухайлбал int болон struct төрөл хоёрын хооронд жиших үйлдэл хийгдэхгүй.

Иймд макрог хэрэглэхээс гадна дүрийн буюу параметрт төрөл template-ийг ашиглах боломж байдаг. Дүрийг ашиглаж дахин давтан тодорхойлж болох функцүүдийн дүр зургийг гаргаж болно. Фунцкийн аргумент нь ямар төрөл байхыг дүрд параметр болгон дамжуулж өгдөг. Ингээд дүрийг манай тохиолдолд хэрхэн тодорхойлохыг үзье.

```
template <class T> T max(T x, T y)
        {
        return (x > y) ? x : y;
        };
```

Энд дүрд дамжиж байгаа параметр нь class T юм. Ингээд max функцийг хэрэглэх хөрвүүлэгч ямар төрөл хэрэглэж байгааг харгалзан зөв хөрвүүлэлтийг хийдэг. Жишээлбэл:

```
int i;
```

```
Myclass a, b;

int j=max(i,0);
/* аргумент нь энэ тохиолдолд бүхэл тоо байна*/
Myclass m=max(a, b);
/* Харин энэ тохиолдолд хэрэглэгчийн тодорхойлсон төрөл
байна*/
```

Тэгэхээр max функц нь дотроо > операторыг хэрэглэж байгаа тул энэ операторыг хэрэглэж болох бүх төрлийг жиших бололцоотой болно. Дээрх төрлийн дүрийг функцийн дүр гэж нэрлэдэг. Харин энэ дүрийн тодорхой төрөл дээрх функцийн дүр функц гэж нэрлэдэг.

Дүр функцийг дахин тодорхойлох

Дүр функц нь үндсэн төрлүүдийн хувьд өмнөх жишээнд үзсэнтэй ижлээр шууд хэрэглэж болох боловч аргумент нь хаяг байхад болон шаардлагатай үйлдлийг өөр функцээр хийх тохиолдол зэрэгт дүр функцийг тухайн тохиолдолд тохируулан дахин давтан тодорхойлж болно. Ингэж давтан тодорхойлсноор дүр функцийг хэрэглэх хүрээг өргөтгөж байгаа хэрэг юм. Жишээлбэл хоёр тэмдэгт мөрийг жишихэд хаягуудыг нь жишихээс тэмдэгт мөрүүдийг хооронд нь жишихгүй юм. Иймд энэ тохиолдолд бид дүр функцийг тэмдэгт мөрийн хувьд дараах байдлаар дахин тодорхойлох хэрэгтэй.

```
#include <string.h>
. . .
char *max(char *x, char *y)
        {
        return(strcmp(x, y)>0) ? x:y;
        }
. . .
```

Ангийн дүр

Ангийн дүр нь функцийн дүртэй төстэйгээр ижил дүр төрхтэй өөр ангиудыг тодорхойлоход зориулагддаг. Тухайлбал

бид нэг хэмжээст массив буюу вектор гэсэн анги тодорхойльё. Тэгвэл векторын элементүүд бүхэл тоо, бодит тоо гэх мэт ямар ч төрөл байж болох билээ. Тэгэхээр үүнийг мөн л дүрийн тусламжтай гүйцэтгэнэ. Тэгэхээр ангиудыг үүсгэхэд зориулагдсан дүрийг ангийн дүр гэдэг. Манай тохиолдолд вектор ангийн дүр нь ямар ч төрөлийн элементүүдийг векторыг үүсгэх боломжтой байх ёстой. Үүнийг заахдаа ангийн дүрд шаардлагатай өгөгдлийн төрлийг параметр болгон дамжуулдаг. Түүнээс гадна энэ ангийн дүрд элемент нэмэх, хасах, дугаараар нь хандах гэх мэт үндсэн үйлдлийг бүгдийг тодорхойлж болно. Ингээд вектор ангийн дүрийг дараах байдлаар тодорхойлж болох юм.

```
template <class T> class Vector
        {
        T *data;
        int size;

        public:
        Vector(int);
        ~Vector()
                { delete[ ] data; }
        T& operator[] (int i)
                { return data[i]; }
        };

template <class T> Vector<T>::Vector(int n)
        {
        data = new T[n];
        size = n;
        };
```

Дээрх жишээнээс харахад ангийн дүрийг тодорхойлохдоо template түлхүүр үгийг бичээд араас дүрийн параметрийг бичжээ. Харин араас нь залгуулж анги болохыг заасан class түлхүүр үгийг бичээд нэрийг нь тодорхойлжээ. Манай ангийн дүр нь параметрээр дамжин ирэх ямар нэгэн дурын төрлийн n элементүүдтэй үргэлжилсэн мужийг санах ойд үүсгэх дүр юм. Мөн түүнчлэн энэ дурын элемент рүү индексээр хандах операторыг тодорхойлжээ.

Ингээд энэ ангийн дүрийг ашиглан тодорхой нэг анги үүсгэвэл тэр ангийг дүр анги гэж нэрлэнэ. Дүр ангийг дараах байдлаар үүсгэж болно.

```
typedef Vector<float> FloatArr;
```

Зарим тохиолдолд эхлээд дүр ангийг тодорхойлолгүй шууд обьектийг үүсгэх боломжтой. Жишээлбэл:

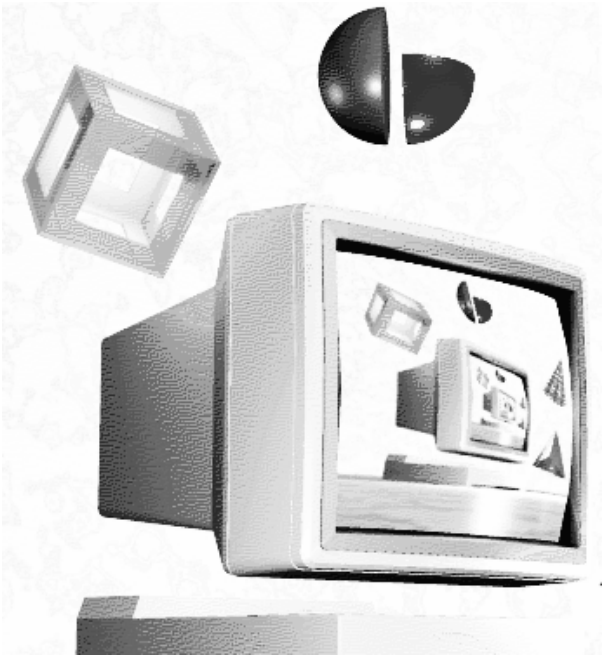
```
Vector<float> x(5);
```

Одоо ийнхүү обьектийг тодорхойлсоны дараа түүнийг программд хэрэглэх боломжтой болно.

```
x[0]= 1.0;
cout << "\n" << x[0];
for (int i = 1; i < 5; ++i)
        {x[i] = x[i-1] + 0.23;
cout << "\n" << x[i];}
. . .
```

Манай жишээнд ангийн дүр нь ганц параметр авдагаар тодорхойлогджээ. Түүнээс гадна зарим ангийн дүр нь олон параметр авах боломж байдаг.

11-р бүлэг



-Borland ангиудын сан

11-р бүлэг

Borland ангиудын сан - Borland Class Library

C++ хэл нь ангиудын бэлэн сантай байдаг. Эдгээр ангиуд нь include\classlib фолдер дотор агуулагдаж байдаг. Энд жишээлбэл бидний өмнөх жишээнүүдэд үзэж байсантай ижил string ангийг тодорхойлсон байдаг. Тэгвэл энэ ангийг ашигласан нэгэн хялбархан жишээ үзье.

```
#include <iostream.h>
#include <cstring.h>

void main()
{
string str1("А.Отгонбаяр");
string str2="Программ хангамжийн тэнхим";

cout << "\nstr1=" << str1;
cout << "\nstr2=" << str2;

str1=str2;

cout << "\nstr1=" << str1;

string str3(str2);

if (str2==str3)
        cout << "\nstr2 = str3 байна.";
}
```

Эдгээр ангиудын сан нь ерөнхийдөө агуулагч, агуулагч биш гэсэн хоёр төрлийн ангиудад хуваагддаг. Манай дээрх жишээнд хэрэглэсэн string анги нь агуулагч биш юм. Учир нь дотроо өөр обьектийг агуулдаггүй зөвхөн тэмдэгт мөр гэсэн мэдээллийг агуулж байдаг.

Ингээд агуулагч ангиудыг товчхон үзье.

Агуулагч TStack

```
#include <iostream.h>
#include <classlib\\stacks.h>
#include <cstring.h>

typedef TStack<string> StrStack;

void main()
{
StrStack stk;

string s1("Амарбат");
string s2("Отгонбаяр");
string s3("Идэрбаясгалан");

stk.Push(s1);
stk.Push(s2);
stk.Push(s3);

while (!stk.IsEmpty())
        {
        string& temp = (string&)stk.Pop();
        cout << temp;
        cout << "\n";
        }
}
```

Stack анги нь стекэд элемент нэмэх, стекээс элемент хасах Push(), Pop() гэсэн хоёр чухал гишүүн функцтэй. Түүнээс гадна стеикийн хоосон эсэхийг үздэг IsEmpty() функц байдаг. Стекийг үүсгэн ашиглахад TStack ангийн дүр ашиглагддаг.

Агуулагч анги

Агуулагч ангиуд нь өөр ямар нэг обьектуудыг цуглуулан хадгалах боломжтой ангиуд юм. Агуулагч ангиуд урьдчилан тодорхойлогдсон 13 янзын өгөгдлийн бүтцээр обьектуудыг хадгалах боломжтой. Өгөгдөлийн бүтэц нь стек, дараалал,

жагсаалт гэх мэт бүтцүүдтэй байж болно. Бид стек ангийг товчхон үзлээ.

Тэгэхээр ямарваа мэдээллийг агуулагч ангид хадгалахын тулд хамгийн чухал нь ямар өгөгдлийн бүтцээр мэдээлэл хадгалах вэ гэдгээс хамааран агуулагч ангийг зөв сонгож авах ёстой. Агуулагч анги бүр мэдээллийг хадгалах боловсруулах нь ялгаатай байдаг. Харин агуулагч ангиуд нь хоёр төрлийн хэсэгт хуваагддаг. Энд нэгдүгээрт өгөгдөлийн хийсвэр төрөлийн (Abstract Data Types - ADT), хоёрдугаарт өгөгдөлийн үндсэн бүтэцүүдийн (Fundamental Data Structures - FDS) гэсэн хоёр төрөл байдаг.

ADT ба FDS ангиуд

ADT-д нь ихэнхи өгөгдлийн бүтцүүдтэй ажиллах үндсэн үйлдэлүүд тодорхойлогдсон байдаг. Жишээлбэл стекийн хувьд стекэд обьект нэмэх push() үйлдэл, стекээс обьект авч хасах pop() үйлдэл гэх мэт үндсэн үйлдэлүүд тодорхойлогдсон байдаг. Стекийн хувьд стекэд ямар төрлийн мэдээлэл агуулахыг параметрээр зааж өгөх бололцоотой ангийн дүрүүд юм. Иймд эдгээр ангийн дүрийн тусламжтай ямар ч төрлийн мэдээлэл агуулах стек буюу тодорхой дүр ангийг үүсгэн хэрэглэх бололцоотой. Стекээс гадна дараалал, цуглуулга, цуваа гэх мэт өгөгдөлийн бүтцүүдтэй ажиллах үндсэн үйлдлүүдийг тодорхойлсон байдаг. Бүх ADT ангиуд нь FDS ангиудтай холбоотой байдаг ба өгөгдлийн бүтэцийн үйлдэлүүдийг биелүүлэхэд хэрэглэгддэг.

Ингээд ангиудын урьдчилсан товч тайлбарыг өгье.

ADT ангиуд

Анги	Тайлбар
Association	Бүлэг. Түлхүүр болон утгын бүлэг
Array	Массив. Бүх обьектууд нь дугаар бүхий хэсгүүдийн цуглуулга юм. Обьектууд нь эрэмбэлэгдээгүй байдаг ба обьектийн тоог нэмэх болон хасах нь төвөгтэй байдаг.
Bag	Цүнх. HashTable-тай адилхан.

Set	Олонлог. HashTable-тай адилхан боловч нэг обьект зөвхөн нэг л байдаг.
Dictionary	Толь. Association буюу бүлгүүдийг агуулсан олонлог.
Stack	Стек. Обьектууд руу зөвхөн оройгоос хандах боломжтой. LIFO зарчмаар обьектууд руу нь ханддаг.
Queue	Дараалал. Обьект нэг төгсгөлөөс ороод нөгөө төгсгөлөөс гардаг. FIFO зарчмаар обьектууд руу хандана.
Deque	Стек, дараалалын нэгдэл. Обьектууд нь хоёр төгсгөлөөрөө орж, гардаг.

FDS ангиуд

Анги	Тайлбар
HashTable	Эмхэлээгүй хүснэгт. Ямарваа обьект руу шууд хандахад амархан байдаг боловч обьектуудад дараалуулсан боловсруулалт хийхэд удаан байдаг. Обьектуудыг түлхүүрийн тусламжтай нэмэх хасах боломжтой. Обьектууд руу индексээр хандах боломжгүй. Ижил обьект нэгээс олон байж болно.
List	Холбоост жагсаалт. Обьектийг зөвхөн эхэнд нь нэмэх, хасах боломжтой. Ямарваа обьектруу шууд хандахад удаан боловч обьектуудад дараалуулсан боловсруулалт хийхэд хурдан. Обьект болгон нь дараагийн элемент рүүгээ заасан заагчтай байдаг.
DoubleList	Хос холбоост жагсаалт. Обьектийг эхэнд болон төгсгөлд нь хоёр талаас нь нэмэх хасах боломжтой жагсаалт. Обьект болгон өмнөх болон дараагийн элементийн заагчийг агуулж байдаг.
Vector	Вектор. Индексээр нь хандаж болох элементүүдийн жагсаалтыг санах ойн үргэлжилсэн мужид хадгалдаг. Энэ нь С хэлни энгийн массивтай адил төрөл юм.

Энд агуулагч ангиудыг товчхон тайлбарлалаа. Агуулагч анги бүр нь дэд төрлүүдийг агуулдаг. Иймд дэд ангиудыг ямар

дүрмээр нэрлэдгийг тайлбарлаад дараа нь тэдгээрийг судлах нь хялбар юм.

ADT ангиудын нэршил

Нэрлэх дүрмийг дараах нэрэн дээр товч тайлбарлая.

TMISArrayAsVectorIterator

Нэрний эхний T үсэг нь энэ тухайн дүр буюу template болохыг илтгэж байна. Иймд бүх төрлийн ангиудын дүр нь T үсгээр эхлэн нэрлэгдэнэ. Иймд ангийн дүрд бид параметр дамжуулан өгөх замаар дурын төрлийн мэдээллийг агуулах агуулагч дүр ангиудыг тодорхойлох бололцоотой болно.

M үсэг нь мэдээллийг санах ойд зохион байгуулах тухай тусгайлан зааж өгөхийг илтгэдэг. Хэрэв M үсэг байхгүй бол агуулагч өөрөө санах ойн зохион байгуулалтыг гүйцэтгэх болно. Харин M үсэг байгаа тохиолдолд санах ойг зохион байгуулах обьектийг агуулагч ангид параметр болгон дамжуулж өгдөг.

I (indirect) үсэг нь агуулагч анги обьектуудыг биш обьектуудын заагчийг хадгалахыг заадаг.

S үсэг нь эрэмбэлэгдсэн байдлаар хадгалахыг заадаг.

Array үг нь ADT ангийн нэр юм. Манай тохиолдолд массивыг хэрэглэжээ.

As үг нь ADT нэрийг холбогдох FDS нэрээс тусгаарлана.

Vector үг нь FDS ангийн нэр бөгөөд манай тохиолдолд массив санах ойд хэрхэн зохион байгуулагдахыг зааж байна. Vector нь ихэнхи ADT ангиудтай холбогддог. Харин зарим ангиуд ADT ангиуд нь Vector ангитай биш өөр FDS ангитай холбогддог. Манай жишээнд байгаа Array анги нь зөвхөн Vector ангитай холбогддог.

Iterator үг нь ямарваа үйлдийг бүх обьектуудад давтан хэрэглэх боловсруулалт хийх боломжийг олгодог. Үүнийг дараа дэлгэрүүлэн үзнэ.

FDS ангиудын нэршил

Дараах жишээн дээр тайлбарлая.

TMISListImp

TMIS кодуудыг бид ADT–ийн хувьд тайлбарлалаа.

List үг нь өгөгдөлийн бүтцийг тодорхойлдог. Манай тохиолдолд жагсаалт болохыг тодорхойлж байна.

IMP үг нь тухайн бүтцийг хэрэгжүүлэх гүйцэтгэхэд хэрэглэнэ гэсэн утгыг илтгэнэ. Манай жишээнд жагсаалт төрөлийн өгөгдөлийг хадгалах тодорхой обьектийг үүсгэх шаардлага гарвал заавал IMP түлхүүр үгийг хэрэглэнэ.

Ангийн нэрийн функциональ кодууд

Бид энэ кодуудын MIS гэсэн гурван үсгийг тайлбарласан билээ. Эдгээрээс гадна дараах хоёр код байдаг.

C үсэг нь ангуулагч анги дахь обьектуудыг тоолж байхыг заана.

D агуулагч ангид обьектын заагч биш обьектууд өөрсдөө хадгалагдахыг заадаг. Зөвхөн Association ангид хэрэглэгдэнэ.

Эдгээрээс гадна T, M, I, S гэсэн кодууд байдаг бөгөөд эдгээрийн өмнө тайлбарласан билээ.

Ангиудын товчилсон нэрс

Ангиудыг дээрх дүрмүүдээр нэрлэдэг боловч ихэнх нэрсийг хэрэглэхэд хялбаршуулан богиносгосон байдаг ба эдгээр товчилсон нэрсийг үзье.

Товчилсон нэр	Ангийн нэр
TArray	TArrayAsVector
TArrayIterator	TArrayAsVectorIterator
TBag	TBagAsVector
TBagIterator	TBagAsVectorIterator
TBinaryTree	TBinaryTreeImp
TBinaryTreeIterator	TBinaryTreeIteratorImp
TDictionary	TDictionaryAsHashTable
TDictionaryIterator	TDictionaryAsHashTableIterator

TDeque	TDequeAsVector
TDequeIterator	TDequeAsVectorIterator
TDoubleList	TDoubleListImp
TDoubleListListIterator	TDoubleListIteratorImp
TList	TListImp
TListIterator	TListIteratorImp
TQueue	TQueueAsVector
TQueueIterator	TQueueAsVectorIterator
TSet	TSetAsVector
TSetIterator	TSetAsVectorIterator
TStack	TStackAsVector
TStackIterator	TStackAsVectorIterator

Агуулагч ангиудыг хэрэглэх

Бид агуулагч ангиудын тухай товчхон үзлээ. Одоо тэдгээрийг программд хэрхэн хэрэглэхийг үзье. Аливаа агуулагч ангийг хэрэглэхийн тулд өмнө үзсэн ангийн дүрүүдийг хэрэглэх хэрэгтэй. Эдгээр дүрүүд нь агуулагч ангид хадгалах төрлийг параметр болгон авдаг. Зарим төрлүүд нь санах ойг хэрхэн зохион байгуулахыг заах обьектийг параметр болгон авах тохиолдол бий. Ингэж тодорхойлохдоо эхлээд дүр ангийг тодорхойлоод дараа нь обьектийг үүсгэн ашиглах байдлаар ажиллаж болно. Энэ тохиолдолд дараах байдлаар ангийг тодорхойлно.

```
typedef TArray<TPoint> TPoints;
```

Энд TPoints нэртэй TPoint төрөлийн обьектуудыг хадгалах агуулагч анги массивыг тодорхойлж байна.

Зарим тохиолдолд эхлээд ангийг тодорхойлолгүй шууд обьектийг үүсгэх боломжтой. Жишээлбэл:

```
TArray<TPoint> PointsArray(10);
```

Энэ тохиолдолд TPoint төрөлийн 10 обьектыг хадгалах PointsArray обьектийг үүсгэж байна.

Дээрх хоёр тохиолдолд агуулагч обьектод обьектуудыг шууд хадгална. Түүнээс гадна агуулагч обьектод обьектуудыг биш харин тэдгээрийн хаягийг хадгалах боломжтой. Өөрөөр хэлбэл заагчуудын массивыг үүсгэх боломжтой болох жишээтэй юм.

Үүнийг үүсгэхдээ I (indirect) үсгийг ашиглан дараах байдлаар тодорхойлох болно.

```
TIArray<TPoint> PPoints
```

Энэ тохиолдолд PPoints анги нь обьектуудын хаягийг агуулах агуулагч ангийн төрөл болох юм.

Агуулагч ангиудыг заагчуудын эсвэл обьектуудын агуулагч болгон сонгон авах нь тухайн үед ямар төрлийн өгөгдөлтэй ажиллаж байгаагаас хамаардаг. Обьектуудын болгон сонгон авсан тохиолдолд ямарваа обьектийг агуулагч обьектод нэмэн оруулахад тухайн обьектийг хуулан агуулагч ангид хийдэг. Иймд обьектийг агуулагч анги руу хийсний дараа анхны обьектийг санах ойгоос устгаагүй тохиолдолд энэ обьект санах ойд хоёр хувь болдог жишээтэй юм. Энэ тохиолдолд агуулагч анги обьектийн өөр доторхи хуулбар луу хандахаас анхны обьект руу ханддаггүй байна. Гэтэл зарим тохиолдолд нэг обьектийн хэд хэдэн агуулагч ангид ашиглах тохиолдол гардаг. Энэ тохиолдолд заагчуудын агуулагч ангийг ашиглах нь зүйтэй байдаг. Гэхдээ энд бүх агуулагч ангиуд энэ обьектийг ашиглаж дуустал санах ойгоос устгаж болохгүй нь ойлгомжтой.

Бүх агуулагч ангиуд нь санах ойг зохион байгуулах анхны анги TStandardAllocator ангийг хэрэглэдэг. Харин зарим агуулагч ангиуд өөр байдлаар санах ойг зохион байгуулах боломжтой байдаг. Үүний тулд санах ойг зохион байгуулах ангийн тусламжтай өөрийн санах ойг зохион байгуулах ангийн ангийг үүсгээд энэ ангийн нэрийг ямар нэг агуулагч ангиуд дамжуулан өгдөг. Ингэснээр тухайн агуулагч анги бидний өгсөн санах ойг зохион багйуулах функцийн тусламжтайгаар санах зохион байгуулалтыг гүйцэтгэдэг. Үүнийг дэлгэрүүлэн судлах шаардлагатай бол alloctr.h толгой файлаас харж болох юм. Энд

санах ойг зохион байгуулагч анги нь санах ойд шинэ элемент үүсгэх new оператор тодорхойлогдсон байх ёстой ба энэ оператор нь void* төрөлийн аргумент авна. Энэ параметр нь агуулагчид нэмэх обьектийн заагч байна. Түүнчлэн агуулагчаас элементийг устгах delete оператор тодорхойлогдсон байх ёстой. Тэгэхээр хэрэв бид өөрсдөө санах ойг зохион байгуулах MyMemManager төрөлийг тодорхойлсон бол дараах байдлаар дарааллыг ашиглаж болох юм.

```
TMQueueAsVector <MyClass, MyMemManager>
MyQueue(100);
```

Энэ тохиолдолд MyClass төрлийн 100 элементийг хадгалах дарааллыг үүсгэж байгаа бөгөөд санах ойг зохион байгуулах өөрийн төрөлийг дамжуулан өгч байна. Энд дарааллын ангийн дүрийг хэрэглэхдээ M функцианаль кодыг хэрэглэсэн учир өөрийн санах ойн зохион байгуулагчийг параметрт дамжуулан өгөх ёстой.

Агуулагч ангиуд нь элементүүдийг эрэмбэлэгдсэн байдлаар агуулж болох ба энэ тохиолдолд ангийн дүрийг хэрэглэхдээ I функциональ кодыг хэрэглэх ёстой. Гэхдээ агуулагчид хадгалагдах обьектууд нь жиших оператор тодорхойлсон байх ёстой.

Үйлдэл давтагч анги

Программ зохиох явцад агуулагч обьектод байгаа бүх обьектод үйлдлийг давтах шаардлага гардаг. Бүх агуулагч ангийн дүрүүд нь өөрийн үйлдэл давтагч ангийн дүртэй байдаг. Өөрөөр хэлбэл агуулагч анги болгон үйлдэл давтагч өөр ангитай уялдаа холбоотой байдаг ба үйлдэл давтах ажлыг агуулагч биш үйлдэл давтагч анги нь гүйцэтгэж байдаг найз анги юм.Ингээд давтагч ангиуд нь давталтыг гүйцэтгэхийн тулд дараах гишүүдтэй байдаг.

Current() идэвхтэй обьектийн заагч.

Restart() давталтыг шинээр эхний элементээс эхлүүлэх функц.

++ Идэвхтэй заагчийг дараах элемент рүү шилжүүлнэ.

Цувааны ангийн хувьд -- үйлдэл бас тодорхойлогдсон байдаг.

Гэхдээ давтагч ангийг хэрэглэлгүй зарим тохиолдолд агуулагчийн ForEach() функцийг ашиглах боломж байдаг. Энэ фунцкийг бид хожим үзнэ. Агуулагч ангийн давтагч обьектийг дараах байдлаар хэрэглэнэ.

1. Агуулагч анги болон давтагч ангийг тодорхойлно. Ингэж төрөл тодорхойлохдоо ангийн дүрүүдийг ашиглан typedef түлхүүр үгт параметрт төрөлийг тодорхойлно. Жишээлбэл:

```
typedef TArray<TPoint> TPoints;
typedef TArrayIterator<TPoint>
TPointsIterator;
```

2. Ийнхүү агуулагч болон үйлдэл давтагч дүр ангиудыг тодорхйолсны дараа энэ ангиудын обьектуудыг тодорхойлох ёстой. Эхлээд давтагч обьектоос өмнө агуулагч обьектийг тодорхойлох ёстой. Дараа нь түүнийг давтагч обьектод дамжуулах ёстой. Эхлээд агуулагч обьектийг дараах байдлаар үүсгэж болно.

```
TPoints Line(10);
```

Эсвэл

```
. . .
TPoints* Line;
. . .
Line=new TPoints(10, 0, 10);
. . .
```

гэх мэтээр обьектийг үүсгэж болно.

Ийнхүү агуулагч обьектийг үүсгэсний дараа үйлдэл давтагч обьектийг үүсгэх хэрэгтэй.

```
TPointsIterator i(*Line);
```

Одоо давталтыг гүйцэтгэх ажил үлдсэн бөгөөд үүнийг Current болон ++ гишүүдийг ашиглан гүйцэтгэж болно.

```
TPoint point(10, 20);
Line->Add(point);
. . .
while (i)
```

```
{
i++;
TPoint p = i.Current();

if (!first)
        LineTo(p);
else
        {
        MoveTo(p);
        first = false;
        }
}
```

Манай жишээнд агуулагч ангид цэгүүдийн обьектийг хадгалсан гэж үзвэл тэдгээрийг холбон зурах кодыг бичлээ. Энэ жишээнд байгаа

```
TPoint p = i.Current();
```

утга олголтыг хялбаршуулахад зориулан утга олгох оператор давтагч ангийн дүрд тодорхойлогдсон байдаг тул

```
TPoint p = i++;
```

гэж утга олгосон ч болох юм.

Агуулагч ангийн гишүүдэд үйлдэл давтагч дүр ангийн тусламжтайгаар үйлдэл давтаж болохоос гадна үйлдэл давтагч гишүүн функцүүдийг ашиглаж болно. Энэ тохиолдолд агуулагч ангийн өөрийн гишүүн функцүүд хэрэглэгдэх тул үйлдэл давтагч өөр ангийг хэрэглэхгүй юм.

Үйлдэл давтагч функц

Агуулагч ангиуд тодорхой үйлдлийг өөрийн обьектуудад давтахад үйлдэл давтах ангийг ашиглаж болохоос гадна үйлдэл давтагч ForEach(), LastThat(), FirstThat() гишүүн функцүүдийг ашиглаж болдог.

ForEach() функц нь агуулагч ангид байгаа бүх обьектод тодорхой үйлдлийг хийдэг. ForEach() функц нь үйлдэл гүйцэтгэх функцийг бүх обьектуудад дууддаг.

FirstThat(), LastThat() өгсөн нөхцөлийг хангах элементүүдийн хамгийн эхнийхийг болон хамгийн сүүлийн

элементийн олдог. Нөхцөлийг шалгах функцийг бүх обьектуудад дуудан эндээс хамгийн эхнийхийг болон сүүлийнхийг олдог.

Тэгэхээр дээрх функүүдээр дуудагдах обьектуудад үйлдэл гүйцэтгэх функцийг программ зохиогч өөрөө бичих ёстой. Энэ функц нь агуулагдаж байгаа обьектуудын хаяг, мөн void төрлийн заагч гэсэн хоёр аргументийг авах ёстой. Харин FirstThat(), LastThat() функцүүдэд хэрэглэх үйлдэл гүйцэтгэх функцийг бичих тохиолдолд энэ функц нь нөхцөлийг шалган нөхцөлийг хангаж байгаа тохиолдолд тэгээс ялгаатай утгыг, хангаагүй тохиолдолд тэг утгыг функц бичих ёстой.

Санамж: Үйлдэл гүйцэтгэх функц дотроос агуулагчийн хэмжээг өөрчлөх үйлдлийг хийж болохгүй. Жишээлбэл Detach(), Flash() зэрэг функцүүдийг хэрэглэж болохгүй. Үйлдэл давтагч функцүүдийг хэрэглэсэн хялбархан жишээг үзье.

```
class Contained
        {
        . . .
        };

void Show(Contained& c, void *)
        {
        cout << c << endl;
        }

void UseForEach(TArrayAsVector<Contained>& vect)
        {
        vect.ForEach(Show, 0);
        }

. . .
TArrayAsVector<Contained> vect(10);

. . .
UseForEach(vect);

. . .
```

Агуулагч обьектийг устгах

Агуулагч ангиуд нь обьектийг хадгалж байгаа тохиолдолд обьектуудыг хуулан авч хадгалдаг харин заагчуудыг агуулж

байгаа тохиолдолд обьектууд руу заах заагч хадгалдаг тухай өмнө ярьсан. Үүнийг шууд ба шууд биш (Direct & Indirect) агуулагчууд гэж нэрлэдэг ба тэдгээрийн нэр I, D функцоиналь кодуудаар ялгарч байдаг тухай ярьж байсан. Шууд агуулагчууд нь агуулагч обьектийг устгахад өөртөө агуулж байгаа бүх обьектийг автоматаар устгадаг. Харин шууд биш агуулагчууд заагчуудыг устгах боловч заагдаж байгаа обьектуудыг устгадаггүй. Иймд хэрвээ заагдаж байгаа обьектуудыг устгах шаардлагатай бол программ зохиогч өөрөө энэ асуудлыг шийдэх ёстой. Энд тухайлбал дараах тохиолдол гарч болно.

Шууд биш хэд хэдэн агуулагч обьектууд нэг обьект руу зэрэг заан түүнийг зэрэг хэрэглэж болох юм. Энэ тохиолдолд аль нэг агуулагч дотроос обьектийг устгавал бусад агуулагчууд ойлгохгүйд хүрч алдаа гарна. Иймд тухайн обьектыг агуулж байгаа шууд биш агуулагчуудын хамгийн сүүлийн агуулагч устах үед л обьектийг устгах ёстой болно.

Тэгэхээр санах ойг зохион байгуулах эдгээр ажлыг товчоор Object Ownership буюу өмч обьект гэж нэрлэдэг.

Харин одоо шууд агуулагч дотроос обьектийг устгах үндсэн аргуудыг үзье.

Бүх ADT обьектуудын хувьд Destroy() функцээр агуулагчаас обьектийг устгана. Харин Destroy() гишүүн функц байхгүй агуулагчийн тохиолдолд Detach() болон Flush() функцээр обьектийг хасахдаа TShouldDelete::Delete аргументаар санах ойгоос устгахыг заадаг.

Харин FDS агуулагчийн хувьд Detach болон Flush функцийг дуудах ба эдгээр нь del гэсэн тоог аргумент болгон авдаг. Энэ тоог тэгээс ялгаатайгаар тааруулсан тохиолдолд элементийг устгах болно.

Обьектийг агуулагчаас хасаад санах ойгоос устгахгүй үлдээх аргуудыг үзье.

ADT агуулагчуудын хувьд Detach болон Flush функцийг дуудахдаа TShouldDelete::NoDelete утгыг аргументэд дамжуулна. Харин тухайн обьектийн хаягийг ямар нэг заагчид хадгалж үлдээхгүй бол энэ обьект нь эзэнгүй болж орхигдоход хүрэхийг анхаарах хэрэгтэй. Ингээд энэ эзэнгүй болсон обьект руу дараа нь хандах боломжгүй болж санах ойд үлдэнэ.

FDS агуулагчийн хувьд Detach болон Flush функцүүдийг дуудахдаа del тоог тэгтэй тэнцүү болгож өгөх хэрэгтэй.

Агуулагч ангитай ажиллах түгээмэл аргууд

Агуулагчид элемент Add() гишүүнийг хэрэглэнэ. Харин стекэд элемент нэмэхдээ Push(), цуваанд PutLeft() PutRight(), дараалалд Put() функцүүдээр элемент нэмнэ.

Агуулагчид элемент хайхдаа Find() функцийг ашиглана.

Агуулагчаас элемент хасахдаа Detach(), устгахдаа Destroy() функцүүдийг ашиглана. Бүх элементийг устгахдаа Flush() функцийг ашиглана. Эдгээрээс гадна стекээс элемент хасахдаа Pop(), цуваанаас элемент хасахдаа GetLeft(), GetRight(), дараалалаас Get() функцүүдээр элемент хасдаг.

Обьектуудад үйлдэл давтахдаа үйлдэл давтагч ангиудыг эсвэл үйлдэл давтагч ForEach(), FirstThat(), LastThat() зэрэг функцүүдийг хэрэглэнэ.

Одоо түгээмэл хэрэглэгдэх агуулагч ангиудыг хэрэглэсэн хялбар жишээнүүдийг үзье. Бид стектэй ажиллах ангийг үзсэн байгаа билээ.

Агуулагч анги Tarray - Массив

TArray анги нь обьектуудын массив үүсгэхэд зориулагдсан анги юм. Ингээд энэ ангийг хэрэглэсэн хялбар жишээ программ үзье.

```
#include <iostream.h>
#include <classlib\arrays.h>
#include <cstring>

typedef TArray<string> strarr;
typedef TArrayIterator<string> strarriter;

void main()
{
strarr arr(2,0,3);

string* ptrs1=new string("a");
string* ptrs2=new string("b");
string* ptrs3=new string("c");
string* ptrs4=new string("d");
string* ptrs5=new string("e");
```

```
arr.Add(*ptrs1);
arr.Add(*ptrs2);
arr.Add(*ptrs3);
arr.Add(*ptrs4);

cout << "\nIndividual elements:";
strarriter j(arr);
int i=0;
while (j!=0)
          {
          cout  << j.Current()
         << "\n-ийн Элементийн дугаар нь " << i;
          i++; j++;
          }
arr.AddAt(*ptrs5,2);

i = 0;
j.Restart();
while (j!=0)
          {
          cout << j.Current()
        << "\n-ийн элементийн дугаар нь " << I;
    i++; j++;
          }
cout << "\n3 дугаар элемент нь " << arr[3];
}
```

Энэ программд TArray ангийн arr обьектийг 0-ээс эхлээд 2 хүртэл элементтэйгээр үүсгэж байна. Массив дүүрсэн тохиолдолд массивын хэмжээ 3-р нэмэгдэхийг зааж байна. Ингээд дараа нь шинэ обьектуудыг үүсгэн түүнийг arr массив руу нэмж байна. Ингээд массив элементийг тавихдаа Add(), AddAt() функцүүдийг ашиглаж болох юм. Add() функцийн массивын төгсгөлд харин AddAt() функц нь тодорхой байрлалд элементийг тавьдаг. AddAt()функцээр массивын элементийн тоо буюу хэмжээ нэмэгдэхгүй. Үйлдэл давтах ангийн тусламжтай j обьектийг үүсгэн дэлгэцэнд массив дахь обьектуудыг хэвлэхэд хэрэглэж байна. Түүнчлэн j обьектийн Current() гишүүн функцээр обьект руу хандаж байгаа ба ++ операторын тусламжтай дараагийн элемент рүү шилжиж байна. Харин Restart() гишүүнээр идэвхтэй заагчийг дахин эхэнд нь байрлуулж байна.

Программын төгсгөлд массивын элемент рүү индексээр нь хандах боломжтой болохыг харуулж байна.

Агуулагч анги TList – Жагсаалт

Tilt нь TListImp ангийн хялбаршуулсан нэр билээ. Энэ нь анги нь элементийг эхэнд нь нэмж болох холбоост жагсаалт юм. Өөрөөр хэлбэл энэ тохиолдолд стектэй ижил зохион байгуулалттай. Гэхдээ бүх элементүүдэд нь ижил үйлдэл хийх боловсруулалт хийх боломжтой байдаг. Ингээд энэ ангийг ашигласан хялбар жишээг үзье.

```
#include <iostream.h>
#include <classlib\listimp.h>
#include <cstring>

typedef TListImp<string> strlist;

void main()
{

void actionFunc(string&, void*);
strlist lst;

string s1("Отгонбаяр");
string s2("Хашбат");
string s3("Амарбат");
string s4("Лхагва");

lst.Add(s1);
lst.Add(s2);
lst.Add(s3);
lst.Add(s4);

cout << "\nЛистэн дэх элементүүд: "
    << lst.GetItemsInContainer();
lst.ForEach(actionFunc, 0);
cout << "\nЛистэн дэх элементүүд:"
    << lst.GetItemsInContainer();
while (!lst.IsEmpty())
          {
          string& temp=(string&)lst.PeekHead();
          cout << "\n" << temp;
          lst.Detach(temp);
          }
```

```
cout << "\nЛистэн дэх элементүүд: "
    << lst.GetItemsInContainer();
}

void actionFunc(string& obj, void *)
        {
        cout << "\n" << obj;
        }
```

Дээрх жишээнд хэрэглэгдсэн функцүүдийг товч тайлбарлья.

GetItemsInContainer() функц нь аливаа агуулагч ангид хэдэн элемент агуулагдаж байгааг авдаг функц юм.

Обьектуудад үйлдэл давтахдаа ForEach() функцийг ашигласан байна. Энэ функцийг хэрэглэж болохоос гадна TListImpIterator ангийг аиглаж болохыг бид мэднэ.

PeekHead() функцээр жагсаалтын эхний элементийн хаягийг авч байгаа боловч элементийг жагсаалтад үлдээж байна.

Detach() функцээр жагсаалтаас заасан элементийг хасна. Манай тохиолдолд толгойн элементийн хаягийг өгч байгаа тул эхний элементийг жагсаалтаас хасна. Харин обьектийг санах ойгоос устгахгүй.

Агуулагч анги TQueue - Дараалал

Дараалал гэдэг нь ямарваа үйлчилгээний газарт үйлчлүүлэгч хүмүүсийн дараалал очертой адилхан юм. Нэг захаас хүмүүс орон нөгөө захаас гардаг билээ. Өөрөөр хэлбэл FIFO зарчмаар элементүүд нь урсдаг.

Дараалалтай дараах байдлаар ажиллах боломжтой. Жишээлбэл дараалалд элементийг хэрхэн нэмэхийг үзье.

```
#include <classlib\queues.h>
typedef TQueue<string> strQueue;

. . .
string *str1=new string("aa");
string *str2=new string("bb");
strQueue que;
```

```
que.Put(*str1);
que.Put(*str2);
```

Тэгэхээр Put() функцийн тусламжтай дараалалд элемент нэмэх боломжтой. Харин дарааллаас элементийг хэрхэн авах, хасахыг үзье.

```
while (!que.IsEmpty())
   {
   string& temp=(string&)que.Get();
   cout << "\n" << temp;
   }
```

Энд хэрэглэгдэж байгаа Get() функцээр дараалалаас элментийг аваад түүнийг дараалалаас хасдаг. Дээрх Put(), Get() функцүүдээс гадна дараалалын хамгийн зүүн болонбаруун талын обьектын заагчийг авах PeekLeft(), PeekRight() функцүүд байдаг. Эдгээр нь дараалалаас элементийг хасахгүй.

Хэрэглэгчийн ангийг агуулагчид хадгалах

Агуулагчид бид string ангийг хадгалах жишээг үзэж ирлээ. Харин программ зохиогч өөрөө анги тодорхойлон түүнийгээ агуулагч хадгалах боломжтой байдаг. Үүнийг тулд агуулагчид хадгалах төрөлийг эхлээд тодорхойлох ёстой ба үүний агуулагч ангийн дүрийг хэрэглэн агуулагч дүр ангийг тодорхойлох хэрэгтэй. Ингээд үүний дараа бусад ажил нь ижил байх болно. Жишээлбэл дараах ангийг үүсгэе. Энэ ангид хэрэгтэй гэж үзсэн гишүүн функцүүд болон оперторуудыгтодорхойлох ёстой. Манай жишээнд байгуулагчид болон жиших операторуудыг тодорхойлж байна.

```
class Contained
        {
        public:
        Contained(int i = 0) : intValue(i)
                {}
        Contained (const Contained & c)
                {intValue=c.intValue;}
        int operator==(const Contained& c)
                {return intValue==c.intValue;}
```

```
int operator<(const Contained& c)
        {return intValue<c.intValue;}

private:
int intValue;
friend ostream&
        operator<<(ostream&, const Contained&);
};
```

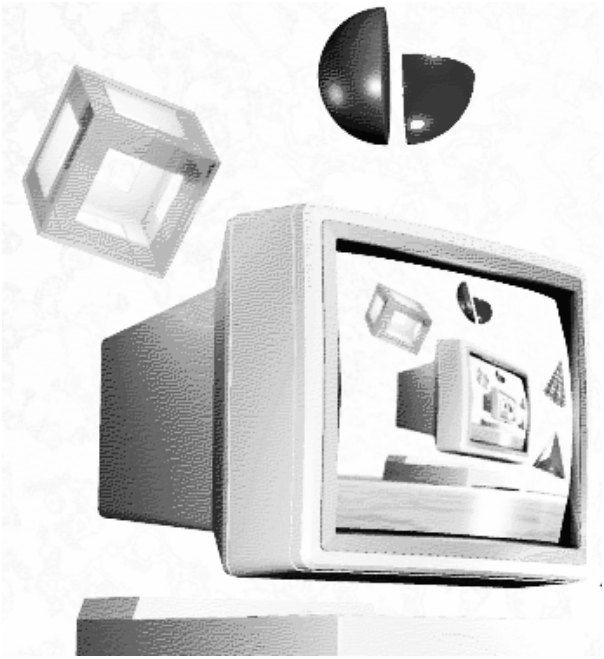
Ингээд хадгалах ангийг тодорхойлсны дараа агуулагч ангийг дараах байдлаар тодорхойлж болно.

```
typedef TArray<contained> MyArray;
```

Чухал гишүүн функцүүд

Анги	Функц	Тайлбар
TStack	Push()	Стекэд обьект нэмнэ.
	Pop()	Стекээс обьектийг авч хасна.
	Top()	Стекийн оройн заагчийг өгнө.
TQueue	Put()	Дараалалд обьект нэмнэ.
	Get()	Дараалалаас обьектийг авч хасна.
TDeque	PutLeft()	Цувааны зүүн захад обьект нэмнэ.
	PutRight()	Цувааны баруун захад обьект нэмнэ.
	GetLeft()	Цувааны зүүн захын обьектийг авч хасна.
	GetRight()	Цувааны баруун захын обьектийг авч хасна.
	PeekLeft()	Цувааны зүүн захын заагчийг өгнө.
	PeekRight()	Цувааны баруун захын заагчийг өгнө.

TListImp	Add()	Жагсаалтын эхэнд обьект нэмнэ.
	Detach()	Жагсаалтын эхнээс обьектийг авч хасна.
	PeekHead()	Жагсаалтын эхний обьектийг авна.
Tarray	Add()	Массивын төгсгөлд обьект нэмнэ.
	AddAt()	Тодорхой байрлал дахь обьектийг солино.
	[ ]	Обьект руу индексээр нь хандана.
ThashTableImp	Add()	Заасан байранд обьект нэмнэ.
	Detach()	Заасан обьектийг хасна.
	Find()	Заасан обьектийг хайж олно.
TbagAsVector	HashTable-г үз	
Tset	Add()	Обьект хэрэв байхгүй бол нэмнэ.
Tdictionary	Add()	Тусгай бүлгийг нэмнэ.



Дасгал ажил ба бодлогууд

Дасгал ажил ба бодлогууд

1. Өгсөн тооны хүрдийг хэвлэж гаргадаг программ зохио. Ингэхдээ нэг мөрөнд 10 багана хэвлэгдэхээр тохируулж өг. Ө.х программын үр дүн дараах байдлаар гарах ёстой.

```
Тоо оруулна уу? 7
7       14      21      28      35      42      49      56      63
        70
77      84      91      98      105     112     119     126     133
        140
147     154     161     168     175     182     189     196     203
        210
```

2. Метрээр өгсөн тоог миллиметр рүү, мөн урвуугаар миллиметрээр өгсөн тоог метр лүү хөрвүүлдэг программ зохио. Хэрэглэгчээс хаанаас хааш нь хөрвүүлэхийг урьдчилан асуух хэрэгтэй. Үр дүнд дэлгэцэнд доорх зүйлс хэвлэгдэнэ гэсэн үг.

```
м-ээс мм-рүү хөрвүүлэх бол 0,
эсрэгээр бол 0-ээс өөр утга өгнө үү : 0
Хэмжээг өгнө үү (м-ээр) : 1.56
1560 мм
```

3. Хэрэглэгчийн гараас оруулсан товч бүрийг getche() функцээр авч түүнийгээ тоо болгон хэвлэх программ бич. Шинэ оруулсан тэмдэгтээ хуучин тоон дээрээ нэмэхдээ арваар үржүүлэх ёстойгоо мартаж болохгүй. Хэрэв тоо биш, үсэг оруулсан байвал түүнийг тоондоо оруулахгүй. Жишээ нь :

```
Өгөгдөл : 123йы456ф
Тоо : 123546
```

4. Хамгийн энгийн тооны машин зохио. Хэрэглэгчээс тоо, үйлдлийн тэмдэг, тоо оруулахыг хүсэх бөгөөд эдгээр өгөгдлүүдээ хоосон зайгаар зааглан оруулах ёстойг анхаарах хэрэгтэй. Үйлдлийг хийж үр дүнг дэлгэцэнд хэвлээд, дахиж өгөгдөл оруулах эсэхийг асуух хэрэгтэй.

Тоо Үйлдэл Тоо : 10 / 3
Хариу : 3.333333
Дахиж оруулах уу? y
Тоо Үйлдэл Тоо : 12 + 100
Хариу : 112
Дахиж оруулах уу? n

5. Хэрэглэгчээс n, m тоонуудыг аваад, n^m–ийг хэвлэж гаргадаг программ зохиож бич.

6. While давталтыг ашиглан хэрэглэгчээс тоонууд авна. Хэрэглэгч сөрөг тоо өгсөн тохиолдолд өмнө оруулсан тоонуудын max, min-ийг хэвлээд дуусдаг программыг зохио.

7. While давталтыг ашиглан хэрэглэгчээс тоонууд авна. Хэрэглэгч сөрөг тоо өгсөн тохиолдолд өмнө оруулсан тоонууд өсч, буурч байгаа тухай хэвлэж дуусдаг программ зохио.

8. While давталтыг ашиглан хэрэглэгчээс тоонууд авна. Хэрэглэгч сөрөг тоо өгсөн тохиолдолд өмнө оруулсан тоонуудын нийлбэрийг хэвлээд дуусдаг программыг зохио.

9. Өмнөх бодлогуудыг do…while давталтаар хийж болох уу? Хэрэв болохоор бол программыг зохио.

10. Хэрэглэгчийн оруулж өгсөн тэмдэгт мөрөнд байгаа бүх жижиг үсгүүдийг том үсэг болгон хувиргах программ бич. Тэмдэгтүүд болон том үсгийг өөрчлөлгүй үлдээнэ.

11. Өмнөхийн адил том үсгүүдийг жижиг болгон хувиргах программ бич.

12. Доор нэгдүгээрт өгөгдсөн мөрүүдийг хоёрдугаарт өгөгдсөн мөрүүдээр сольж болох уу?

```
if (x>max)          If (x>max)
max=x;              Max=x;
if (x<min)          else
min=x;              min=x;
```

13. for давталтыг ашиглан 1-ээс 300 хүртэл тоонуудын нийлбэрийг хэвлэж гаргах программ зохио.

14. Мөн давталтыг ашиглан 1-ээс 15 хүртэл тоонуудын үржвэрийг хэвлэж гарга.

15. Доорх программ ямар үр дүн өгөх вэ?

```
int x=6;

if (x<2)
        cout << “2-оос бага”
else if (x<4)
        cout << “2-оос их боловч 4-оос бага”
        else
                cout << “4-өөс их”
```

16. Тэгвэл энэ программ ямар үр дүн өгөх вэ?

```
int x=6;
```

```
if (x>2)
        cout << “2-оос их”
else if (x>4)
        cout << “2-оос их, 4-өөс их”
        else
                cout << “4-өөс их”
```

17. Хэрэглэгчийн оруулж өгсөн тэмдэгт мөрөнд байгаа таслалын тоог хэвлэдэг программ бич.

18. Оруулж өгсөн тэмдэгт мөрөнд зүүн болон баруун хаалтны тоо тэнцүү байгаа эсэхийг шалгах программ бич.

19. Өмнөх программыг өргөтгөн бүх төрлийн хаалт, хашилтыг ({ }, [ ], (), “ “, ‘ ‘) зэрэг шалгадаг программ зохио.

20. Аль ч талаас нь уншихад ижилхэн байдаг тоо болон мөрийг паиндром гэж нэрлэдэг. Жишээ нь: 121, 11, 45678987654, хэсэх, харах, молом гэх мэт. Дашрамд хэлэхэд монгол хэлний хамгийн урт палиндром үг бол “хадгалагдах” гэсэн үг байдаг юм байна. Өгсөн тоо палиндром эсэхийг шалгах программ зохио.

21. Өгсөн тэмдэгт мөр палиндром эсэхийг шалгаж, хариу өгөх программыг зохиож бич.

22. Хоорондоо 2-оор зөрөөтэй анхны тоонуудыг ихэр анхны тоонууд гэдэг (3 ба 5, 101 ба 103 гэх мэт). 1000-аас бага бүх ихэр анхны тоог хэвлэж гарга.

23. Өгсөн тэмдэгт мөрөнд байгаа дараалсан хоосон зайнуудыг нэг хоосон зайгаар солих программ бич.

24. Он тооллын эхлэлээс өгсөн өдрийг хүртэл нийт өнгөрсөн өдрийн тоог олох программ зохио.

25. Өгсөн оны өгсөн сарын хуанлийг хэвлэж гарга (өмнөх программаа ашиглана).

26. Хэрэглэгчийн өгсөн жилд Баасан гаригт 13-ны өдөр таарч байгаа сарын дугаарыг хэвлэж гарга.

27. Факториалыг тооцох рекурсив биш программ зохио.

28. Доорх дарааллын N-р гишүүнийг рекурсив аргаар тооцож гаргах программ зохио.

S(N)=2+4+6+…+2*N

29. Робот нэг удаа алхахдаа 1,2 эсвэл 3 метр зам туулна. N метр замыг хэдэн янзаар туулж болохыг тооцоолох программыг зохиож бич. Мөн рекурсивийг ашиглана. Жишээ нь:

Метр	Боломжууд
1	1
2	1+1 2
3	1+1+1 1+2 2+1 3
4	1+1+1+1 1+2+1 1+3 2+1+1 2+2

	3+1

30. Хэрэглэгчийн оруулж өгсөн үсгээр уг үсгийг нь томоор зурдаг функцүүдийг бич. Жишээ нь, P үсэг оруулбал доорх байдлаар хэвлэх юм.

```
PPPPPPPPPPPP
PPP      PPP
PPP      PPP
PPP      PPP
PPP      PPP
PPPPPPPPPPPP
PPP
PPP
PPP
PPP
PPP
```

31. Функц болон switch оператор ашиглан доорх загварын программ зохио. Программ хэрэглэгчээс OP, NUM1, NUM2 гэсэн хувьсагчуудыг авна. Хэрэв OP-ийн утга 1 байвал Add гэсэн функц дуудагдана. Уг функц NUM1 болон NUM2-ийн утгуудыг хооронд нь нэмж үр дүнг хэвлэж гаргана. Түүнчлэн OP-ийн утгаас хамааран хасах, үржих, хуваах үйлдлүүдийн функцийг зохио.

32. Функцийн аргументэд функц өгөх аргыг ашиглан дээрх программыг бич.

33. Мөн өмнөх программыг функцийн заагч ашиглан зохио.

34. Доорх томъёогоор бодлого бодох программыг зохиож бич.

c(n,r)=n!/(r!*(n-r)!)

35. Хэрэглэгчийн өгсөн тэмдэгт мөрийг (эсвэл үгийг) Морзын үсгээр дэлгэцэнд хэвлэх программ зохио. Морзын үсгийг доор үзүүлэв.

A ._	J .___	S …
B _…	K _._	T _
C _._.	L ._..	U .._
D _..	M __	V …_
E .	N _.	W .__
F .._.	O ___	X _.._
G __.	P .__.	Y _.__
H ….	Q __._	Z __..
I ..	R ._.	

Энэ программд цэг хэвлэдэг нэг макро, зураас хэвлэдэг бас нэгэн макро байх шаардлагатай. Харин үсэгнүүдийг хэвлэх макрог доорх маягаар зохиовол зохимжтой.

```
#define     Print_A     Print_Dot, Print_Dash
#define     Print_E     Print_Dot
#define     Print_I     Print_E, Print_E
#define     Print_H     Print_I, Print_I
```

36. Доорх нийлбэрийг тооцох программ зохиож бич.

$$S(N)=\sum_{i=1}^{N}\frac{1}{i^2}$$

37. Доорх томъёогоор бодлого бодох рекурсив программыг нь зохиож бич.

$$\begin{cases} N, & \mathrm{mod}(M,N)=0 \\ \end{cases}$$

gcd(M,N)=
gcd(N,R), mod(M,N)=R

38. Санамсаргүй тоо гаргах rnd() гэсэн функцийг зохио. Санамсаргүй тоог системийн цаг ашиглан гаргаж болох юм. Учир нь системийн цаг хэзээ ч давтагдахгүйд байгаа юм. Системийн цаганд янз бүрийн хувиргалт хийж өөрчлөх хэрэгтэй. Жишээ нь, системийн цаг, минут, секундыг бүгдийг хооронд нэмээд язгуур авах г.м. Ийм тохиолдолд давхардсан тоо гарах магадлал тун бага байна.

39. Хоёр тэмдэгт мөрийг хооронд нь нэмж үр дүнг эхний мөрөнд өгдөг concat() функц зохио.

40. Хоёр дахь аргументэд өгсөн тэмдэгт мөрийн утгыг эхний аргументэд өгсөн мөрөнд оноож өгдөг copy() функцийг зохио.

41. 3 хэмжээст массивын элементүүдийг харгалзуулан нэг хэмжээст массивт хуулдаг программ зохиож бич. Жишээ нь:

```
char arr1[10][4][2]
char arr2[80]
```

гэж тодорхойлсон бол

```
arr2[0]=arr1[0][0][0];
arr2[1]=arr1[0][0][1];
arr2[2]=arr1[0][1][0];
arr2[3]=arr1[0][1][1];
arr2[4]=arr1[0][2][0];
…
arr2[79]=arr1[9][3][1];
```

гэх мэтчилэн утга оноогдоно.

42. 100 элементтэй массив үүсгээд түүний зөвхөн эхний 3 элементэд 1, 3, 5 гэсэн утгуудыг онооё. Харин түүнээс хойших (i>3) элементүүдийг доорх томъёогоор гаргаж үр дүнг дэлгэцэнд хэвлэ.

num[i]=n[0]+n[1]+n[2]+n[3]+…+n[i-1]

43. Цагаан толгойн үсгүүдийг 6x10 хэмжээст массивт хадгалаад хэрэглэгчийн өгсөн тэмдэгт мөрийг дэлгэцэнд тэрхүү массивийг ашиглан гаргадаг программ зохиож бич.

```
   A      M        M   БББББББББ
  A A     M M     M M   Б
 A   A    M  M   M  M   БББББББББ
 A AAA A    M   M M   M   Б       Б
A      A   M    M    M   Б       Б
A      A   M         M   БББББББББ
```

44. Хэрэглэгчийн оруулж өгсөн тэмдэгт мөрийг урвуугаар нь хэвлэдэг программ бич.

45. 2-р аргументэд өгөгдсөн тэмдэгт мөр нэгдүгээр аргументэд өгөгдсөн тэмдэгт мөр дотор байж байвал 1 утга, эсрэг тохиолдолд 0 утга буцаах SubCheck() функц зохио.

Функц	Буцах утга
SubCheck(‘Сайн байна уу?’,’Сайн’)	1
SubCheck(‘Hi, I`m John’,’m’)	1
SubCheck(‘Программчлалын хэл’,’C++’)	0

46. Өмнөх программыг өргөтгөж, 2 дахь мөр 1 дэх дотор олдсон тохиолдолд түүний байрлалыг авчирдаг FindSub() функцийг зохиож бич.

Функц	Буцах утга
SubCheck(‘Сайн байна уу?’,’Сайн’)	0
SubCheck(‘Hi, I`m John’,’m’)	6
SubCheck(‘Программчлалын хэл’,’C++’)	-1

47. Та бүхэн гадаадад хэвлэгдсэн номын ард ISBN 0-8065-0959-7 гэсэн бичиг олонтаа үзэж байсан байх. ISBN гэдэг нь International Standard Book Number гэсэн үг юм. Түүний хойно байрлах 10 оронтой тоо нь уг номын тухай мэдээлэл байна. Эхний нэг орон нь уг номыг үйлдвэрлэсэн газрын группын дугаар юм. 0 гэдэг нь англи хэлээр ярьдаг нэгэн улс гэсэн утгатай ажээ. Дараагийн 4 орон нь номыг үйлдвэрлэсэн үйлдвэрийн дугаар юм. 8065 нь Citadel Press гэсэн үйлдвэр юм байна. Харин 0959 гэсэн дараагийн 4 тоо нь уг номын дугаар аж. Харин хамгийн сүүлийн 1 орныг шалгах тоо гэж нэрлэдэг юм. Энэхүү шалгах тоо нь өмнөх 9 орныг нь оронгийнх нь дугаараар үржүүлж, нэмсэн тоо байдаг. Ө.х. эхний тоон дээр 2 дахь тоог 2-оор үржүүлж нэмээд, мөн дээр нь 3 дахь тоог 3-аар үржүүлж нэмнэ г.м. Уг нэмсэн тоог 11-д хуваагаад гарсан үлдэгдэл нь энэ тоо болно. Хэрэв үлдэгдэл 10 гарвал энд тоо биш ‘X’ гэсэн үсэг байрлана.

0+2*8+3*0+4*6+5*5+6*0+7*9+8*5+9*9=249

249 –ийг 11-д хуваагаад 7 гэсэн сүүлийн орон гарч байна.

Өгсөн ISBN дугаарыг зөв эсэхийг шалгах тооных нь тусламжтай шалгаад, зөв бурууг мэдээлэх программ зохио.

48. Тэмдэгт мөрийн тусламжтайгаар олон оронтой тоо хоороңд нь нэмэх программ зохиож бич.

49. Тэмдэгт мөр маягаар өгсөн Ром тоог энгийн 10-тын тоо болгон хөрвүүлэх программ бич.

50. Хоёр ширхэг матрицийг хооронд үржүүлж 3 дахь матрицыг үүсгэх программыг зохио.

51. Ойролцоогоор 500 орчим үгтэй сонины өгүүлэлд байгаа үгнүүдийг, мөн уг үг хэдэн удаа давтагдан орсныг хадгалах массив үүсгэж, массив дахь өгөгдлийг дэлгэцэнд хэвлэж гарга. Массивын нэг элемент нь wd (үг), cnt (тоо) гэсэн 2 талбар бүхий бүтэц байна.

52. 2-оос их бүх эерэг, тэгш тоонууд 2 анхны тооны нийлбэрт тавигдаж болно гэсэн Голдбахын тодорхойлолт байдаг. Хэрэглэгчийн оруулж өгсөн тоог эерэг, тэгш эсэхийг нь шалгаад, түүнийг 2 анхны тооны нийлбэрт тавих программыг зохиож бич. Жишээ нь, 52 нь 47 ба 5-ийн нийлбэр болно.

53. Өгсөн текстийг орчуулах бага хэмжээний орчуулагч программ зохио. Үүний тулд 2 массив зохионо. Эхний массивт англи үгийг, дараагийн массивт монгол үгүүдийг харгалзуулан бичнэ. Жишээ нь, эхний массивын 10-р элемент 'WE' гэсэн үг байгаа бол дараагийн массивын 10-р элемент нь 'БИД' гэж байх жишээтэй. Үгнүүдийг шууд харгалзуулан орчуулахаас, ямар нэгэн дүрмийг шалгах шаардлагагүй. Ө.х. 'I am teacher' гэж өгсөн бол үр дүн нь 'Би бол багш' гэж шууд орчуулагдах юм.

54. Хөзрийг дүрсэлж байгаа 52 элементтэй массивыг үүсгэе. Нэг элемент нь уг хөзрийн өнгө болон тоог илэрхийлэх талбарууд бүхий бүтэц байна. Эхлээд

массивын бүх элементийг өнгө болон тоогоор нь дэлгэцэнд хэвлэж гаргах хэрэгтэй. Харин үүний дараа хөзрөө холино. Ингэхдээ хамгийн эхний хөзрийг санамсаргүй нэгэн хөзөртэй, дараа нь 2 дахь хөзрийг мөн санамсаргүй нэгэн хөзөртэй холино г.м. Ингээд холилдсон хөзрүүдийг мөн дэлгэцэнд хэвлэж гарга.

55. Холбоост жагсаалтын элементүүдийн тоог олдог функцийг зохиож бич.

56. Холбоост жагсаалтын элементүүдийг урвуу эрэмбээр нь байрлуулах программ зохио. Ө.х. өмнөх жагсаалтын хамгийн эхэнд байсан элемент шинэ жагсаалтад хамгийн сүүлийн элемент болно гэсэн үг.

57. Арифметик илэрхийллийн үр дүнг бодож гаргадаг программыг зохио. Үүнд стекийг ашиглана. Стек гэдэг бол орой нь хөдөлдөггүй, харин төгсгөл дээр нь оролт гаралтын бүх үйлдэл хийгдэж байдаг бас нэгэн төрлийн өгөгдлийг хадгалах арга (холбоост жагсаалтын адил) билээ. Стект хамгийн сүүлд орсон өгөгдөл эхэнд нь гардаг онцлогтой. Тэмдэгт мөрийг шалгаж үзээд үйлдэл гарвал стект хийж, харин тоо гарвал түүнийг авч хадгалах хэрэгтэй. Хоёр дахь удаагаа тоо гарсан тохиолдолд өмнө хадгалсан тоо болон одоогийн тоо хоёрын хооронд хамгийн сүүлд стект орсон үйлдлийг хийнэ. Үйлдлийн үр дүнг мөн шинэ тоо гэж үзэх бөгөөд хэрэв эхнийх болж байвал хадгалж, хоёр дахь болж байвал түүн дээр үйлдэл хийх маягаар үргэлжилнэ.

58. Холбоост жагсаалтад байгаа өгсөн дугаартай хоёр элементийг солих программ бич. Хэрэв солих боломжгүй бол (буруу индекс өгөгдсөн гэх мэт) энэ тухай мэдээлнэ.

59. Холбоост жагсаалт дотор өгсөн элементийг хайж олдог программ зохио.

ГАРЧИГ