СЭДЭВ 1. ОРШИЛ

1.1 Си хэлний ач холбогдол, ангилал

Си хэл нь анх үйлдлийн системд зориулагдан бүтээгдсэн хэл юм. UNIX үйлдлийн системийг бүтээхдээ программчилалын хүчтэй орчинг бүрдүүлэхийг зорьсон хүмүүс энэ хэлийг боловсруулжээ. UNIX үйлдлийн системийг дэлхий нийтээр өргөн хэрэглэх болсноор программчилалын Си хэлний хэрэглээ ч нэмэгдэх болжээ. UNIX үйлдлийн системийн 90% нь энэ хэл дээр бичигдсэн , тэр ч бүү хэл Си хэлний өөрийнх компилятор нь хүртэл энэ хэл дээр бичигджээ.

Мөн UNIX үйлдлийн системийн бараг бүх үйлчилгээний программ хангамжууд, UNIX үйлдлийн систем дээр ажиллах ихэнх хэрэглээний программ хангамжууд Си хэл дээр бичигдсэн , цаашид ч энэ хэл дээр бичигдэх болно. Иймээс Си хэл нь одоо ч хүртэл программчилалын хүчтэй зэвсэг хэмээн тооцогдсоор байна.

Си хэл бол сурахад хялбар, маш бага кодчилол хийх боломжтой хэл юм.

1.2 Си программын бүтэц , онцлог шинжүүд

Си программ нь функцуудын олонлогоос бүрдэх бөгөөд эдгээр функцууд нь нэг буюу хэд хэдэн эх файлд хадгалагдаж болно. Эдгээр эх файл бүр нь тус тусдаа хөрвүүлэгддэг. (машины хэлд)

Си программд зөвхөн нэг л функц “main” нэртэй байж болох бөгөөд бусад функцууд нь дурын байдлаар нэрлэгдэж болно. Си программ бичихэд ядаж нэг функц заавал бичих бөгөөд энэ нь “main” функц юм. Си хэлний функц нь бас процедурын үүрэг гүйцэтгэнэ. Ө.х Си хэл нь бусад хэл шиг функцийг функц , процедур гэж ялгадаггүй. Си хэл дээр бичигдсэн программ “main” функцээс эхэлж ажиллах бөгөөд хэрвээ программ хэвийн ажилласан бол энэ функцээр үйл ажиллагаагаа дуусгах болно. Программ ажилллах явцад “main” функц нь бусад функцийг дуудаж болох бөгөөд тэр функц нь цааш өөр функц дуудах гэх мэтчилэн үйл ажиллагаа нь үргэлжилнэ. Тухайн функц үйл ажиллагаагаа дуусгамагц түүнийг дуудсан функцэд удирдлагаа шилжүүлнэ.

Программын жишээ :

```
main()
{
   int k;
   printf(“ Now I,ll print pi number ”);
   PrintfPI();
}
```

Үндсэн функц зарлах хэсэг

СЭДЭВ 2. СИ ПРОГРАММЧИЛАЛ ГЭЖ ЮУ ВЭ ?

Хэрвээ та урьд өмнө нь хэзээ ч программ бичиж байгаагүй бол энэ сэдэв танд программ анхлан бичихэд чинь туслана.

2.1 Программ гэж юу вэ ?

Бидний сайн мэдэх компьютер бол тийм ч их ухаантай машин биш юм. Тэр бол зөвхөн зааврыг үг дуугүй дагадаг сайн туслагч төдий юм. Тэрээр таны өгсөн даалгаврыг биелүүлэхээр хэдэн ч өдрөөр , уйтгарлахгүйгээр , амрахгүйгээр ажиллаж чадна. Гэвч тэр юу хийхээ өөрөө бие даан шийдвэрлэж, бодож чадахгүй. Иймээс тэдэнд юу хийхийг нь Программ зохиогч хэлж, зааж өгдөг.

Түүгээр ямар нэг ажил хийлгэхийн тулд түүнд өгч байгаа командуудын олонлогийг программ гэж нэрлэнэ. Одоогоор бидний ашиглах боломжтой мянга мянган программ зохиогдоод байгаа бөгөөд эдгээр нь маш олон төрөл , чиглэлээр зохиогдсон болно.

2.2 Си программ бичихэд юу хэрэгтэй вэ ?

Си хэл дээр программ бичиж түүнийгээ ажиллуулахын тулд юуны түрүүнд Си компилятор хэрэгтэй. Си компилятор нь таны бичсэн программыг компиляци хийх буюу хөрвүүлэх үйл ажиллагааг хийдэг. Хөрвүүлэх гэдэг нь хүнд ойлгомжтой буюу Си хэл дээр бичигдсэн программыг машинд ойлгомжтой буюу командуудын дараалалд хөрвүүлэх үйл ажиллагааг хэлнэ.

2.3 Программчилах процесс

Ихэнх хүмүүс программ бичихдээ дараах дарааллыг баримталдаг.

- Программаа яг юу хийхийг тодорхойлно. (Үйл ажиллагааг нь төсөөлнө.)
- Алгоритмаа тодорхойлоод программаа бичнэ.
- Си программыг дурын текст боловсруулагч программ дээр бичиж болно. Гэхдээ ихэнх Си компиляторууд нь өөртөө текст боловсруулагчтай байдаг. Си хэл дээр бичигдсэн программ нь .C өртгөтгөлтэй файлд хадгалагдана.
- Программаа хөрвүүлэх
- Программын алдааг хянаж, засварлах
- Си компилятор нь танд алдааны тухай мэдээллүүд өгдөг. Хэрвээ алдаагаа засаж дууссан бол программ ажиллахад бэлэн боллоо гэсэн үг.
- Программаа ажиллуулах, үр дүнг нь үзэх

Одоогоор хэрэглэгдэж байгаа ихэнх Си компиляторууд нь маш өргөн боломжтой бөгөөд энэ бүх үйл ажиллагааг нэгэн программаас хийх боломжийг бүрдүүлсэн байдаг. Жишээ нь Turbo C компилятор нь манайд өргөн хэрэглэгддэг бөгөөд энэ нь программ бичих , үр дүнг нь харах зэрэг үйл ажиллагааг нэг дэлгэцэнд меню ашиглан хийх боломжийг бүрдүүлсэн сайн компилятор юм.

СЭДЭВ 3. ДЭЛГЭЦЭНД МЭДЭЭЛЭЛ ХЭВЛЭХ

3.1 printf функц

Таны программ ажиллаж дуусаад тодорхой үр дүнг гаргах бөгөөд түүнийг бид үзэх буюу дэлгэцэнд үр дүн нь хэвлэгдэх зайлшгүй шаардлагатай байдаг. Си хэлэнд үүнийг хэрэгжүүлдэг **printf** гэсэн функц байдаг. Энэ функц нь дэлгэцэнд тоо, тэмдэгт , үг гаргахад ашиглагдана. printf функц нь маш өргөн боломжтой.

3.2 printf функцын формат

Энэ функцын үндсэн формат нь :

printf(Удирдлагын тэмдэгт мөр [,өгөгдөл]);

Тайлбар: Си хэлэнд функцуудын форматыг бичиж үзүүлэхдээ зарим тусгай тэмдэглэгээг хэрэглэдэг ба энд [ ] гэсэн тэмдэглэгээг ашигласан байна. Ер нь тэмдэглэгээг зөвхөн танд ойлгуулах үүднээс л ашигладаг ба энэ нь тайлбарлаж буй функцын бичлэгт ордоггүй юм. Иймээс энэ функцыг ашиглахдаа [ ] хаалтыг бичихгүй байх болно. **Энэ хаалтанд бичигдсэн өгөгдөл түүнийг бичихгүй байж болно гэсэн утгыг агуулдаг.**

Удирдлагын тэмдэгт мөр нь тухайн өгөгдөл ямар хэлбэрээр дэлгэцэнд гарахыг тодорхойлдог.

printf(“ Жишээ %d”,1); /* Энэ жишээнд Жишээ 1 гэсэн үгийг хэвлэж байна*/

Си хэлэнд тэмдэгт мөрийг заавал хашилтанд (“”) бичдэг. Тэгэхээр удирдлагын тэмдэгт мөр нь заавал хашилтанд бичигдэнэ.

Си хэлэнд бичигдсэн команд бүр “;”-аар төгсөх ёстой. Өөрөөр хэлбэл энэ нь команд дуусч байна, үүнийг биелүүлэх ёстой гэдгийг Си энэ тэмдэгтээр мэдэх болно.

3.3 Тэмдэгт мөр хэвлэх

Бид дэлгэцэнд тэмдэгт мөр мэдээлэл хэвлэхдээ энэ функцыг ашиглана.

printf(“ Сайн байна уу ! ”);

Энэ комманд биелэгдэхэд дэлгэцэнд **Сайн байна уу !** гэсэн үг хэвлэгдэнэ.

printf(“ Сайн байцгаана уу !”);
printf(“ Хичээлээ эхлэцгээе !”); гэсэн коммандууд юу хэвлэхийг үзэцгээе.

Сайн байцгаана уу ! Хичээлээ эхлэцгээе !

Си нь тэмдэгт мөр хэвлээд дараа нь курсорыг дараагийн мөрт автоматаар шилжүүлдэггүй. Тэгвэл ингэж курсорын байрлалыг удирдахдаа *Удирдлагын тэмдэгт мөр* дотор тусгай тэмдэгтүүдийг ашигладаг.

3.4 Курсорын байрлал удирдах тэмдэгтүүд

Си хэлэнд маш олон ийм тэмдэгтүүд байдаг ба тэднээс зарим хэсгийг нь л их өргөн ашигладаг.

Хүснэгт 3.1 Курсорын байрлал удирдах тэмдэгтүүд

Код	Тайлбар
\n	Шинэ мөр
\a	Дуут дохио гаргах
\t	Tab тэмдэгт (Хэд хэдэн хоосон зай)
\\	\ тэмдэгтийг өөрийг нь хэвлэх
\”	“ тэмдэгт хэвлэх
\r	Мөрийн эхэнд

“\a” тусгай тэмдэгтийг хэвлэхэд дуут дохио гардаг. Өмнөх жишээнд бид хоёр тэмдэгт мөр хэвлэхэд тэд 2 мөрт биш , 1 мөрт гарч байсан . Тэгвэл одоо \n тусгай тэмдэгт ашиглан энэ үйлдлийг гүйцэтгэе. Эхлээд эхний тэмдэгт мөрөө хэвлээд дараа нь курсорыг дараагийн мөрт шилжүүлэх шаардлагатай.

printf(“ Сайн байцгаана уу !\n”);
printf(“ Хичээлээ эхлэцгээе !”);

Си тэмдэгт мөрийг хэвлэж байгаад \n тэмдэгт дайралдангуут курсорын байрлалыг дараагийн мөрт шилжүүлдэг. Ингэсний дараа хэвлэгдэх тэмдэгтүүд шинэ байрлалаас цааш хэвлэгдэх болно.

Тусгай тэмдэгтүүдийг ашигласан жишээ :

```
printf(“Тэмдэгт\tмөр\tхэвлэв.\n”);      /* Тэмдэгт мөр хэвлэв. */
printf(“Дохио дуугарав. \a\n”);          /* Дохио дуугарав.     */
printf(“Тэр \”Үгүй\” гэж хэлсэн.\n”);    /* Тэр “Үгүй” гэж хэлсэн. */
printf(“\\ тэмдэгтийг хэвлэв.\n”);       /* \\ тэмдэгтийг хэвлэв.  */
```

3.5 Хөрвүүлэлтийн тусгай тэмдэгтүүд

Та тэмдэгт болон тоо хэвлэхдээ тэдгээрийг яаж хэвлэгдэхийг тодорхойлох хэрэгтэй болдог. Ингэж тодорхойлохдоо хөрвүүлэлтийн тусгай тэмдэгтүүдийг ашигладаг.

Хүснэгт 3.2 Хөрвүүлэлтийн тусгай тэмдэгтүүд

Тэмдэгт	Тайлбар
%d	**int төрлийн утгыг төлөөлнө.**
%i	**int төрлийн утгыг төлөөлнө.**
%c	**Зөвхөн нэг тэмдэгт**
%s	**Тэмдэгт мөр**
%o	unsigned int (8-т тооллын систем)
%u	unsigned int

%x	unsigned int (16-т тооллын систем)
%X	unsigned int (16-т)
%f	Бодит тоон утга [-]dddd.ddd
%e	Бодит тоон утга [-]d.ddd e [+/-]ddd
%%	% тэмдэгт
%p	XXXX:YYYY санах ойн хаяг
%hd, %hi, %ho, %hx	short int
%ld, %li, %lo, %lx	long int
%le, %lE, %lf, %lg	double
%Le, %LE %Lf, %Lg	long double

Жич : Хамгийн хялбар бөгөөд юуны түрүүнд бидэнд хэрэг болох тусгай тэмдэгтүүдийг тод хараар бичэв.

Тоо, тэмдэгт хэвлэхдээ *Удирдлагын тэмдэгт мөр* дотор хөрвүүлэлтийн тусгай тэмдэгтүүдийг бичих ба дараа нь *Удирдлагын тэмдэгт мөрийн* ард хэвлэх утгуудаа таслалаар зааглан харгалзуулан жагсааж өгдөг.

Жишээ :

printf(“%s %d %f %c\n”,”Hi”,14,-8.76,’x’);

Энэ жишээ нь дараах үр дүнг хэвлэнэ.

Hi 14 -8.760000 x

“Hi” тэмдэгт мөр нь хашилтанд бичигдэнэ. Харин Си хэлэнд нэг тэмдэгтийг ‘’ хашилтанд бичдэг. –8.76 гэсэн бодит тоог хэвлэхэд Си түүнийг -8.760000 болгож хэвлэсэн байна. Си хэлэнд бодит тоог хэвлэхдээ % ба f тэмдэгтийн хооронд . тэмдэгтийг хэрэглэн хэвлэгдэх хэлбэрийг тодорхойлно.

Жишээ :

printf(“%f %.3f %.2f %.1f”, 4.5678, 4.5678, 4.5678, 4.5678);

Үр дүн : **4.567800 4.578 4.57 4.6**

3.6 fprintf() функц

Энэ функц нь өгөгдлийг урсгал (принтер,диск,файл) хэмээн нэрлэгдэх виртуал төхөөрөмжид гаргадаг.

Бичих хэлбэр : **int fprintf(FILE *stream, const char *format[, argument, . . .]);**

Энэ функц нь UNIX үйлдлийн системд өргөн хэрэглэгддэг.

Жишээ :

```
#include<stdio.h>

int main(void)
{
    int n=3;
    float x=45.875;

    /* Дэлгэцэнд хэвлэх */
    printf(“ Сайн байна уу ? \n”);

    /* Принтерт хэвлэх */
    fprintf(stdprn,“ Сайн байна уу ? \n”);

    /* Дэлгэцэнд хэвлэх */
    printf(“ x=%f ба n=%d \n”,x,n);
```

```
    /* “Принтерт  хэвлэх  */
    fprintf(stdprn,“ x=%f  ба  n=%d \n”,x,n);
}
```

СЭДЭВ 4. ХУВЬСАГЧ БА ӨГӨГДЛИЙН ТӨРЛҮҮД

Бүх программчилалын хэлүүд хувьсагч тодорхойлж, өгөгдөлтэй ажиллах тодорхой механизмуудтай байдаг. Си хэлэнд энгийн өгөгдлийн төрлүүд болох бүхэл, бодит тоо, тэмдэгтэн төрлүүдтэй ажиллах олон арга замууд байдаг.

Бусад программчилалын хэлүүдэд байдаггүй тийм аргууд нь Си хэлийг улам ч хүчирхэг болгодог. Энэ бүлэгт үзэх санаанууды Си хэлэнд эхлэн суралцагч хэн бүхэн сайн ойлгох хэрэгтэй. Өгөгдлийн төрлүүд болон тэдгээртэй ажиллах үйлдлүүдийн олонлог нь Си хэлийг улам уян хатан болгодог ч , зарим талдаа буюу Си хэлийг төгс эзэмшиж ойлгоход хүндрэлтэй байдаг талтай.

4.1 Идентификаторууд

Хувьсагчид болон функцийн нэрийг идентификатор гэнэ. Си хэлний идентификатор нь үсэг, тоо, доогуур зураас тэмдэгтээс бүрдсэн 32 тэмдэгт хүртэлх урттай байж болно. Идентификаторийг том болон жижиг үсгээр бичиж болох ба Си хэлэнд том жижиг үсгийг ялгаатайд тооцдог. Өөрөөр хэлбэл Count, count , COUNT гэсэн гурван нэр нь гурвуулаа ялгаатай нэрүүд болно. Практикаас үзэхэд өөрийн бичих программаа илүү ойлгомжтой болгохын тулд хувьсагчдын нэрийг жижиг үсгээр, тогтмолуудыг том үсгээр бичиж ашигладаг.

Зөв идентификатор	Буруу идентификатор
far_out	Tom’s
TIME	whicha$macallit
age	%dollar
time	do-it

Зөвлөгөө :

- ❖ Хувьсагчийн нэрийг үсгээр эхлэх
- ❖ Хувьсагчийг түлхүүр үгтэй ижлээр нэрлэж болохгүй. (do, while , if , int, float , case . . .)
- ❖ Хувьсагчдыг гүйцэтгэж байгаа үүрэгтэй нь зохисон нэрээр нэрлэх
- ❖ Дотоод хувьсагч ялангуяа индекс зэргийг богино , гадаад хувьсагчийг арай уртаар нэрлэхэд тохиромжтой.

4.2 Түлхүүр үгүүд

Си хэлэнд бүх түлхүүр үгийг жижиг үсгээр бичдэг. Хувьсагчийн нэр нь түлхүүр үгтэй яг ижил байж болохгүй. Гэхдээ бичлэгийн хувьд түлхүүр үгээс ялгаатай байж болно.

Жишээ нь :

Түлхүүр үг	Хувьсагч
for	FOR
if	IF

Хүснэгт 4.1

int	extern	else	char

register	for	float	typedef
do	double	static	while
struct	goto	switch	union
return		case	long
sizeof	default	short	break
auto	unsigned	continue	if
asm	cdecl	huge	interrupt
pascal	void	const	volatile
enum	signed	_cs	near
_es	_ss	far	_ds

4.3 Энгийн өгөгдлийн төрлүүд

Си хэлэнд 4 үндсэн энгийн төрөл байдаг.

int — Энэ төрлийн хувьсагч бүхэл тоон утга агуулна. Гэхдээ авах утгууд нь зарим төрлийн систем дээр өөр өөр байдаг боловч ихэнхдээ 1 үг буюу 2 byte урттай байдаг.
Int утга : -32768 . . . 32767 (тэмдэгтэй)
unsigned int : 0 . . . 65535 (тэмдэггүй)

char — Энэ төрлийн хувьсагч нь тэмдэгт утга агуулна. Char утга нь 1 byte –ийн хэмжээтэй байдаг.
Char утга : -128 . . . 127 (тэмдэгтэй)
unsigned char : 0 . . . 255 (тэмдэггүй)

float — Энэ төрлийн хувьсагч нь бодит тоон утга агуулна. float утга нь 4 byte-ийн хэмжээтэй байдаг ба 10E-38 … 10E+38 завсар дахь бодит тоон утга агуулах чадалтай.

double — Энэ нь float төрлийн адил бодит тоон утга агуулах ба 8 byte хэмжээтэй.

4.4 Хувиргагчид

Си хэлэнд дараах хувиргагчдыг хэрэглэн өгөгдлийн төрлийг хувиргаж болдог.

short int — Энэ нь int утгаас их биш утга авна гэсэн үг. Энэ төрлийн хэмжээ нь системээс хамааралтай байдаг. Зарим системд энэ утга нь int утгын хагас хэмжээтэй байхад заримд нь адил хэмжээтэй байдаг.

long int — int-ээс 2 дахин том гэсэн үг. Мөн системээс хэмжээ нь хамаардаг ба ихэнхдээ 2 үгийн урттай байдаг.

long float — float утгаас 2 дахин том буюу double төрөл гэсэн үг.

unsigned int — Энэ нь int төрлийн утгыг тэмдэггүйгээр хадгална гэсэн үг. Өөрөөр хэлбэл дан эерэг утга авах буюу int утгаас 1 битээр том гэсэн үг.

Хүснэгт 4.2 Өгөгдлийн төрлүүдийн хэмжээ

Төрөл	byte-аар	битээр
char	1	8
int	2	16
short	2	16
long	4	32
unsigned	2	16
float	4	32

double	8	64

short, long, unsigned хувиргагчид нь int утганд хэрэглэгддэг. Иймээс Си хэлэнд дараах бичлэгүүд адил утгатай байдаг.

short int = short
long int = long
unsigned int = unsigned

4.5 Хувьсагч ба санах ойн хаяг

Си хэлэнд хувьсагчдыг тэдгээрийн санах ойн хаяг нь төлөөлдөг.

int cost;

Тэгвэл энэ cost хувьсагчийн утга нь санах ойн дараалсан 2 үүрэнд хадгалагдах ба энэ үүрнийх нь хаягийг **&** үйлдлээр авч болно. Өөрөөр хэлбэл **&cost** гэвэл энэ хувьсагчийн санах ой дахь хаяг нь болно. & хаяг авах үйлдэл нь Си хэлэнд маш өргөн хэрэглэгддэг.

4.6 sizeof үйлдэл

Си хэлэнд sizeof үйлдлээр тухайн хувьсагчийн утгыг санах ойд хадгалахад хичнээн byte шаардагдаж байгааг тодорхойлдог.

int k;

үед **sizeof(k)** нь 2 гэсэн үр дүнг буцаах болно. Учир нь int төрөл нь 2 byte хэмжээтэй байдаг шүү дээ. Мөн sizeof үйлдлийг Си хэл дээрх илэрхийлэлд ашиглаж болдог.

x=s/sizeof(k);

sizeof нь функц биш , үйлдэл бөгөөд иймээс ч түүнийг дуудахдаа заавал хаалт хэрэглэх шаардлагагүй.

a=sizeof(cost);
a=sizeof cost; /* Энэ 2 үйлдэл ижил чанартай. */

Харин тодорхой нэг төрлийн хэмжээг мэдэхийн тулд sizeof –ийг ашиглаж байгаа тохиолдолд заавал хаалт хэрэглэнэ.
n=sizeof(int);

4.7 Хувьсагчид

Хувьсагч гэдэг бол тодорхой төрлийн утгыг хадгалж чадах обьект юм. Хувьсагчийг зарлахдаа өгөгдлийн төрөл болон идентификаторыг ашигладаг.

Бичигдэх нь : **<төрөл> <утга>**

Жишээ нь :

char ch;
short i,j;

Си хэлэнд хувьсагчийг зарлахгүй байж болдог. Хэрэв хувьсагч нь зарлагдаагүй бол түүнийг int төрөлтэй гэж тооцно. Ө.х Си хэлэнд int хувьсагчийг зарлахгүй байж болно. Гэхдээ бүх хувьсагчаа зарлаж байх нь илүү найдвартай.

Хувьсагчид анх утгатай болох хүртлээ тодорхойгүй төлөвт байна. Хувьсагчид анхны утга олгох 2 арга байдаг.
1. Хувьсагчийг анх зарлахдаа анхны утгыг нь олгож болно.

```
int last = 32767;
char capc='C';
float pi = 3.141593;
double expo=1.0e6;
```

Энэ арга нь тодорхой хэдэн төрөлд л үйлчилдэг.

2. Утга олгох үйлдэл ашиглан анхны утга олгож болно.

```
main()
{
    int last;
    char capc;
    float pi;
    double expo;
    last = 32767;
    capc='C';
    pi = 3.141593;
    expo=1.0e6;
}
```

4.8 Тогтмолууд

Программын амьдралын туршид тогтмол утгатай байх обьектыг тогтмол гэж нэрлэнэ. Си хэлэнд тогтмолыг зарлах үндсэн 2 арга байдаг.

1. #define түлхүүр үгээр тогтмолыг зарлах
2. const түлхүүр үгээр зарлах

#define тэмдэглэгээг ашиглан зарлахдаа эхлээд тогтмолын нэр , дараа нь оруулах утгыг тодорхойлдог. Харин #define – ийг ашиглахдаа ; -аар төгсгөдөггүйг анхаар.

Бичигдэх хэлбэр нь : **#define <Тогтмолын нэр> <утга>**

#define TRUE 1

#define коммандыг main() функцийн биенээс гадна ашиглах ёстой.

```
#include <stdio.h>
#define TRUE 1
#define FALSE 0
main()
{
  . . .
}
```

‘const’ түлхүүр үгийг ашиглан тогтмол зарлах :

Бичигдэх хэлбэр : **const <төрөл> <Нэр>=<утга>;**

Жишээ :

```
const float pi=3.1415;
const int Max=100;
```

Тогтмолыг ингэж зарласны дараа үндсэн программд дараах үйлдлүүдийг хийхийг хориглоно.

```
main()
{
   pi=3.1;
   Max=50;
   pi=pi+1;
}
```

4.9 Хадгалах ангилал

Си хэлэнд олон янзын хадгалах ангилал байдаг. Хадгалах ангилалаас нь хамаарч обьектын амьдралын хугацаа , ажиллах хүрээ тодорхойлогддог. Ихэнх хэлүүдэд зөвхөн ажиллах хүрээ нь тодорхойлогдсон байдаг. Өөрөөр хэлбэл хувьсагч, функц нь тодорхой модульд л ашиглагдах бөгөөд бусад модульд бол тодорхойгүй байна. Харин Си хэлд амьдралын хугацаа гэдэг бол чухал ойлголт байдаг. Хэрэв хувьсагч тодорхой нэг функцэд тодорхойлогдсон бол тухайн функцын ажиллах хугацаанд л амьдарна. Тэгэхээр түүний амьдрах хугацаа нь функцын ажиллах хугацаа байна. Гэхдээ хадгалах ангилалаасаа хамаарч амьдрах хугацаа нь ийм байх уу үгүй юу гэдэг нь тодорхойлогдоно. Хадгалах ангилал нь long, unsigned гэх мэт хувиргагчидтай адилаар тодорхойлогддог. Өргөн хэрэглэгддэг хадгалах ангилалууд нь **auto, static, register, extern** юм. Утга нь тухайн программын амьдралын хугацаанд хадгалагддаг, харин ажиллах хүрээ нь нэг функцын хүрээгээр тодорхойлогддог хувьсагчийг статик хувьсагч гэнэ. Статик хувьсагчийг **static** түлхүүр үгээр зарлана. Гадаад хувьсагч болон гадаад функцыг ашиглахдаа түүнийг **extern** түлхүүр үгээр зарлан ашигладаг. Static болон extern хувьсагчдын тухай бид сүүлд функц сэдвийг үзэхдээ дахин танилцуулах болно. **auto** –оор тодорхойлогдсон хувьсагчид түүнийг агуулж байгаа функцууд дуудагдах

үед оршин тогтнож эхэлдэг. Удирдлага функцээс шилжихэд auto хувьсагч санах ойгоос алга болно. Тухайн машины бодит боломж ямар байгаагаас хамааруулан register төрлийн хувьсагчдыг ашигладаг. Олон дахин ашиглагдах хувьсагчийг буюу их давтамжтай хэрэглэгдэх тоолууруудыг register хувьсагчаар зарлаж болох бөгөөд энэ хувьсагч нь процессорын регистрийг эзэмших учир бусад хувьсагчийг бодвол хурдан ажилладаг.

auto int k; **static int i=1;**
register int t; **extern int n;**

4.10 Төрлийн хувиргалт

Си нь ялгаатай төрлүүдийн хооронд хувиргалт хийдэг. Гэхдээ программ зохиогч түүний үр дүнг тооцоолж , анхаарч байх ёстой. Жишээ нь : char утганд int утга олгоход халилт гарах магадлалтай. Илэрхийлэл дэх обьектууд нь ялгаатай төрөл байхад Си дараах дүрмийг баримталдаг.

Илэрхийлэл	Тайлбар
float үйлдэл **int**	үед int утгыг float болгон хөрвүүлж бодолт хийх ба илэрхийллийн үр дүн float утгатай байна.
char үйлдэл **int**	үед char утгыг int болгон хөрвүүлж бодолт хийх ба илэрхийллийн үр дүн int утгатай байна.
float үйлдэл **double**	үед float утгыг double болгон хөрвүүлж бодолт хийх ба илэрхийллийн үр дүн int утгатай байна.
int үйлдэл **short**	үед short утгыг int болгон хөрвүүлж бодолт хийх ба илэрхийллийн үр дүн int утгатай байна.
int үйлдэл **long**	int утгыг long руу хөрвүүлнэ. Үр дүн нь int утгатай
unsigned үйлдэл **int**	unsigned утгыг int-д хөрвүүлнэ. Үр дүн нь int утгатай

СЭДЭВ 5. ХЭРЭГЛЭГЧЭЭС ӨГӨГДӨЛ УНШИХ

5.1 scanf() функц

Энэ функц нь гараас өгөгдөл уншина. Программ зохиох үед хэрэглэгчээс төрөл бүрийн өгөгдөл унших зайлшгүй шаардлага гардаг. Тиймээс юуны түрүүнд энэ функцыг ашиглаж сурах явдал чухал юм. Бичлэгийн хувьд энэ функц нь printf –тэй нэлээд адилхан байдаг , иймээс ч сурахад илүү дөхөм байх болно.

Хэрэглэгчээс өгөгдөл уншихдаа ихэнх тохиолдолд **printf**, **scanf** хоёр функцыг хослуулан ашигладаг. Эхлээд printf –ээр хэрэглэгчээс унших утгын тухай буюу асуултаа хэвлээд , дараа нь scanf – аар утгаа уншдаг.

Бичигдэх хэлбэр : scanf(Удирдлагын Тэмдэгт Мөр [,хувьсагч]);
int scanf(“Удирдлагын Тэмдэгт Мөр”,Хувьсагч1, . . . ХувьсагчN);

Таны программ биелж байгаад scanf функц дуудагдангуут, программын үйл ажиллагаа зогсож, хэрэглэгчээс өгөгдөл оруулж дуусахыг хүлээдэг. Хэрэглэгч өгөгдөл оруулаад дуусахдаа ENTER товч дарах ёстой бөгөөд тэгэхэд уншигдсан утгууд нь харгалзах хувьсагчиддаа очин , удирдлага scanf коммандын дараагийн коммандад шилждэг. scanf нь printf-тэй адил хөрвүүлэлтийн тусгай тэмдэгтүүдийг ашигладаг.

Харин курсор удирдах тусгай тэмдэгтүүдийг (\n\a\t . . .) удирдлагын тэмлэгт мөр дотор хэзээ ч бүү ашигла. Ашиглавал функцын үйл ажиллагаа буруу явагдах

болно. Мөн scanf-ыг үргэлж printf-тэй цуг хэрэглэ. Хэрэв printf-ээр асуултаа тавихгүй бол хэрэглэгч өгөгдөл оруулах ёстой гэдгийг ойлгохгүй байх болно.

Жишээ :

```
printf("Таны массив хичнээн элементтэй байх вэ ?");     /*  Асуултаа тавих   */
scanf("%d",&n);
```

Гэхдээ зарим шинж чанарыг нь авч үзвэл scanf нь тийм ч хялбархан функц биш юм. scanf –ыг ашиглахад гардаг нэг хэцүү асуудал нь **"scanf –ын Удирдлагын тэмдэгт мөрийг ямар байдлаар бичсэн яг түүнтэй адилаар хэрэглэгч гараас өгөгдөл оруулах ёстой"** юм.
Жишээ : Гараас int утгыг age хувьсагчид уншья.

```
scanf("=%d",&n);
```

Энэ тохиолдолд хэрэглэгч гараас тоогоо оруулахдаа өмнө нь заавал = тэмдэг бичиж байх хэрэгтэй. Учир нь scanf ингэж хүсч байна. Тэгэхээр scanf-ын Удирдлагын тэмдэгт мөрийг бичиж байхдаа онц шаардлагагүй л бол хэрэгцээгүй, илүү тэмдэгтийг ашиглахгүй байх хэрэгтэй.

scanf-ыг ашиглахад гардаг өөр нэг асуудал нь **&** тэмдэгтийг ашиглах явдал юм.

scanf –д хувьсагчдын өмнө & тэмдэгтийг ашигладаг. Шаарддаг учрыг сүүлд бид "Функц" гэсэн сэдвийг үзэж байхад мэдэх болно. Харин нэг тохиолдолд буюу "%s" тэмдэгтийг ашиглаж байгаа , өөрөөр хэлбэл тэмдэгт массивтай ажиллаж байх үед & тэмдэгтийг ашигладаггүй.

Дүрэм : Хэрэглэгчээс int, float, char, double,long гэх мэт энгийн төрлийн утгыг уншихыг хүсвэл & тэмдэгтийг хувьсагчийн нэрийн өмнө заавал ашигла. Харин хэрэглэгчээс тэмдэгт мөр буюу тэмдэгт массив уншихыг хүсвэл & тэмдэгтийг ашиглахгүй.

Дараах программд хэрэглэгчээс овог, нэр , нас, хүндийг нь асууж байна. Тэмдэгт мөр уншихдаа & ашиглаагүй , харин нөгөө 2 хувьсагчид & ашиглаж байгааг анзаарна уу. Программд хэрэглэгчийн овог нэрийг нэг scanf –аар цугт нь уншиж чадаагүй байна. Учир нь scanf нь нэгэн зэрэг 2 үг буюу тэмдэгт мөр салгаж уншиж чадахгүй. юм.

```
#include <stdio.h>
main()
{ int age;
  float weight;
  char first[15], last[15];
    printf("Та нэрээ  оруулна уу ?");   /*   Асуулт   */
    scanf("%s",first);        /*   & тэмдэгт ашиглаагүй .  */
    printf("Та овгоо  оруулна уу ?");   /*   Асуулт   */
    scanf("%s",last);         /*   & тэмдэгт ашиглаагүй .  */

    printf("Та хэдэн настай вэ ?");         /*   Асуулт   */
    scanf("%d",&age);          /*   & тэмдэгт ашигласан .  */
    printf("Та хэдэн кг жинтэй вэ ?");   /*   Асуулт   */
    scanf("%f",&weight);     /*   & тэмдэгт ашигласан .  */

    printf(" Таны оруулсан мэдээллүүд \n");
    printf(" Овог, Нэр :  %s  %s\n",last,first);
```

```
    printf("Жин : %.0f\n",weight);    /*  0  байгаа учир бутархай оронг хэвлэхгүй.*/
    printf("Нас : %d", age);
}
```

Программ ажиллаад дараах үр дүнг үзүүлнэ.

```
    Та нэрээ  оруулна уу ?  Бат
    Та овгоо  оруулна уу ? Дорж
    Та хэдэн настай вэ ? 20
    Та хэдэн кг жинтэй вэ ? 60

    Таны оруулсан мэдээллүүд
    Овог, Нэр :  Дорж  Бат
    Жин : 60
     Нас : 20
```

Жишээ :

```
main()
{ int i;   char c;    float f;    char Name[20];
  scanf("%d",&i);    scanf("%c",&c);
  scanf("%f",&f);     scanf("%s",Name);
}
```

Жишээ :

```
main()
{  int i,j,k;
   scanf("%d  %d  %d",&i,&j,&k);
}
```

Та scanf-ыг ашиглан хэрэглэгчээс тодорхой форматтай өгөгдлийг уншиж болно. Жишээлбэл он сар уншихад он,сар, өдөр гэсэн утгууд нь тодорхой тусгаарлагчтай (03/05/99) байдаг. Тэгвэл одоо хэрэглэгчээс mm/dd/yy гэсэн форматаар он сар уншия.

```
scanf("%d/%d/%d",&month,&day,&year);
```

Энэ тохиолдолд хэрэглэгч яг дээрх форматаар л оруулах ёстой.

Дүрэм :

- ❖ Хэрэглэгчээс өгөгдөл уншихдаа scanf функцыг ашиглана.
- ❖ scanf-ыг ашиглахдаа удирдлагын тэмдэгт мөр бичдэг ба энэ нь хэрэглэгч өгөгдлөө яаж оруулахыг тодорхойлсон тэмдэгт мөр байна.
- ❖ scanf-ыг ашиглахаасаа өмнө printf-ээр заавал асуултаа хэвлэ.
- ❖ scanf-д массиваа өөр хувьсагч ашиглахдаа & тэмдэг ашигла.
- ❖ scanf-д массивын нэрийн өмнө & тэмдэг бүү ашигла.

5.2 getchar() функц

getchar() нь гараас зөвхөн ганц тэмдэгт уншина, харин putchar нь дэлгэцэнд нэг тэмдэгт хэвлэнэ. Та зөвхөн нэг тэмдэгт гараас унших, эсвэл дэлгэцэнд нэг тэмдэгт хэвлэхийг хүсвэл эдгээр функцыг ашигла.

Жишээ :

```
#include <stdio.h>
main()
{     int ch;
```

```
    printf(“ Тэмдэгтээ оруулна уу ? ”);
    ch=getchar();
    printf(“\n You typed the character ”);
    putchar(ch);
}
```

Энэ команд нь int утга буцаах ба энэ нь дарагдсан товчны код байна.

5.3 getch() функц

getch() нь оруулах өгөгдлийг буферлэдэггүй. Өөрөөр хэлбэл өгөгдлөө цуглуулан, ENTER товчийг хүлээдэггүй . Үүнээс үүдэх нэг дутагдал нь буруу оруулсан өгөгдлөө BACKSPACE товчоор залруулж, арилгаж болдоггүй. Харин getchar()-аар өгөгдлөө унших үед хэрэглэгч BACKSPACE товч дарж болно. Гэтэл getch()-д ийм боломж байхгүй. Дараах жишээнд getch()-ээр 2 тэмдэгт уншиж байна.

```
printf(“Хоёр тэмдэгт оруулна уу ? \n”);
first=getch();
second=getch();
```

getch() нь getchar() шиг ENTER товч хүлээдэггүй учраас түүнээс илүү хурдан ажиллана.

5.4 getche() функц

getch() нь оролтын тэмдэгтүүдийг дэлгэцэнд дүрсэлдэггүй. Иймээс та дэлгэцэнд дүрслэхийг хүсвэл дараа нь putch –ийг дуудах шаардлагатай болдог.

```
printf(“Хоёр тэмдэгт оруулна уу ? \n”);
first=getch();
putch(first);
second=getch();
putch(second);
```

Гэтэл дээрх үйлдлийг гүйцэтгэхийн тулд getche функцыг ашиглахад л хангалттай. getche нь оролтын тэмдэгтүүдийг дэлгэцэнд дүрсэлдэг.

Дүрэм

- Нэг тэмдэгт унших, бичихийг хүсвэл getchar(), putchar()-г ашигла.
- Гараас Enter товчийг хүлээж зогсохыг хүсвэл getchar-ийг ашигла.
- Хэрэглэгч товч дарангуут түүнийг дэлгэцэнд дүрсэлэхгүйгээр уншихыг хүсвэл getch()-ийг ашигла.
- Эдгээр Оролт гаралтын функцүүдтэй ажиллахдаа char биш int хувьсагч ашигла. Учир нь зарим үйлдлийн системүүд char төрлөөс их утгаар товчлуурыудаа кодлосон байдаг.

5.5 СИ ПРОГРАММЫН БҮТЭЦ

Си программын бүтцийг агуулгаар нь дараах 5 бүлэг хэсэгт хувааж үзнэ.

1. Холболтын хэсэг
2. Тогтмолуудыг зарлах хэсэг
3. Хувьсагчийг зарлах хэсэг
4. Функцүүдийг зарлах хэсэг
5. Үндсэн функц

1.Холболтын хэсэг

Программд стандарт функц процедур холбох шаардлага тулгардаг бөгөөд холболтыг **#include** түлхүүр үгийн тусламжтайгаар гүйцэтгэнэ.
Бичигдэх хэлбэрт : **#include <Файлын нэр>** эсвэл
#include "Файлын нэр"

Файлын нэр нь тухайн функцуудыг агуулах .h өргөтгөлтэй файлын нэр байна. Эдгээр файлууд c:\tc\include санд байрлана.

Файлуудын жишээ :

- **stdio.h** - Үндсэн оролт гаралтын функцүүд энэ файлд тодорхойлогдоно
  printf - Дэлгэцэнд өгөгдөл хэвлэх
  scanf - Гараас утга унших
  . . .
- **stdlib.h** - Төрөл бүрийн хөрвүүлэлтийн функц, программын удирдлага шилжүүлэх функцууд байна.
  atoi - Тэмдэгт мөрийг тоо болгох
  itoa - Тоог тэмдэгт мөр болгох
  . . .
- **alloc.h , malloc.h** - Санах ойтой харьцдаг функцууд
- **math.h** - Математик функцууд
  cos() - Косинус олох, sin() - Синус олох
  log() - Логарифм олох, sqrt() - Тооны кв язгуур олох
  pow() - x тоог y зэрэгт дэвшүүлэх
- **ctype.h** - Тэмдэгтийг шалгах, өөрчлөх функцуудыг агуулна.
  isalpha() - Тэмдэгт нь үсэг мөн эсэхийг шалгана.
  isdigit() - Тэмдэгт нь тоо мөн эсэхийг шалгана.
- **time.h** - Он сар өдөр болон цагтай ажиллах функцуудыг агуулна.
  clock() - Программ эхэлснээс хойших хугацааг буцаах
  time() - Системийн идэвхтэй цагийг буцаах
- **conio.h** - Дэлгэцтэй ажиллах функцуудыг агуулна.
  gotoxy() - Курсорын байрлалыг өөрчлөх
  cprintf() - Өнгө зааж өгч хэвлэх
- **string.h** - Тэмдэгт мөртэй ажиллах функцууд
  strcpy() - Нэг тэмдэгт мөрийг нөгөөд олгох функц
  strcat() - Хоёр тэмдэгт мөрийг залгах функц

2.Тогтмолуудыг зарлах хэсэг

Энэ хэсэгт тогтмол хэмжигдэхүүний нэр, авах утгыг зарладаг.

1. const float pi=3.14;

2. #define MAX 100

3. Хувьсагчийг зарлах хэсэг

Программд хэрэглэгдэх глобаль хувьсагчдыг энд зарлана.
Бичих хэлбэр : <Хувьсагчийн төрөл> <Хувьсагчийн нэр>;

Жишээ : int i,j;
char c;

Глобаль хувьсагч

Глобаль хувьсагч гэдэг нь тухайн программын аль ч хэсэгт харагдах , утгыг нь өөрчилж болох хувьсагч юм. Глобаль хувьсагчийг дараах 2 шинж чанараар нь ялгана.

- Программын аль ч функц , процедур дотроос түүний утгыг хэрэглэж, өөрчилж болдог.
- Түүний амьдрах хугацаа нь программыг дуусан дуустал үргэлжилнэ.

4. Функцүүдийг зарлах хэсэг

Энд хэрэглэгчийн тодорхойлсон функц , процедуруудыг (дэд программ) байрлуулна. Дэд программ нь өөрийн гэсэн нэртэй ба үндсэн программын хаанаас нь ч нэрээр нь дуудаж хэрэглэж болдог.

Функцын нэр(параметрүүд)
{ <Хувьсагчдыг зарлах хэсэг> → Локаль хувьсагч
<Функцийн бие>
}

Локаль хувьсагч

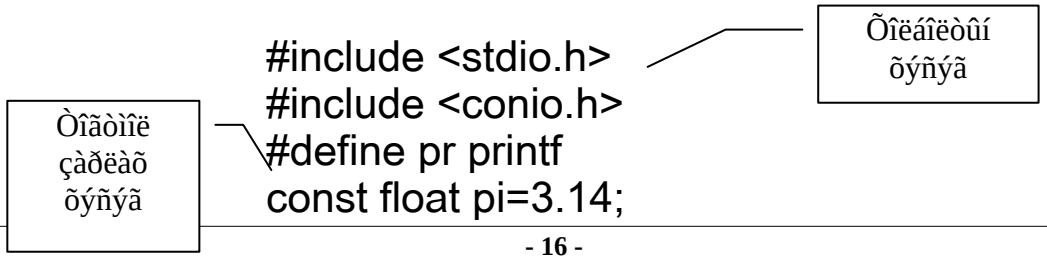
- Зөвхөн тухайн функц дотроо хэрэглэгдэнэ.
- Амьдрах хугацаа нь функцын ажиллах хугацаа юм.

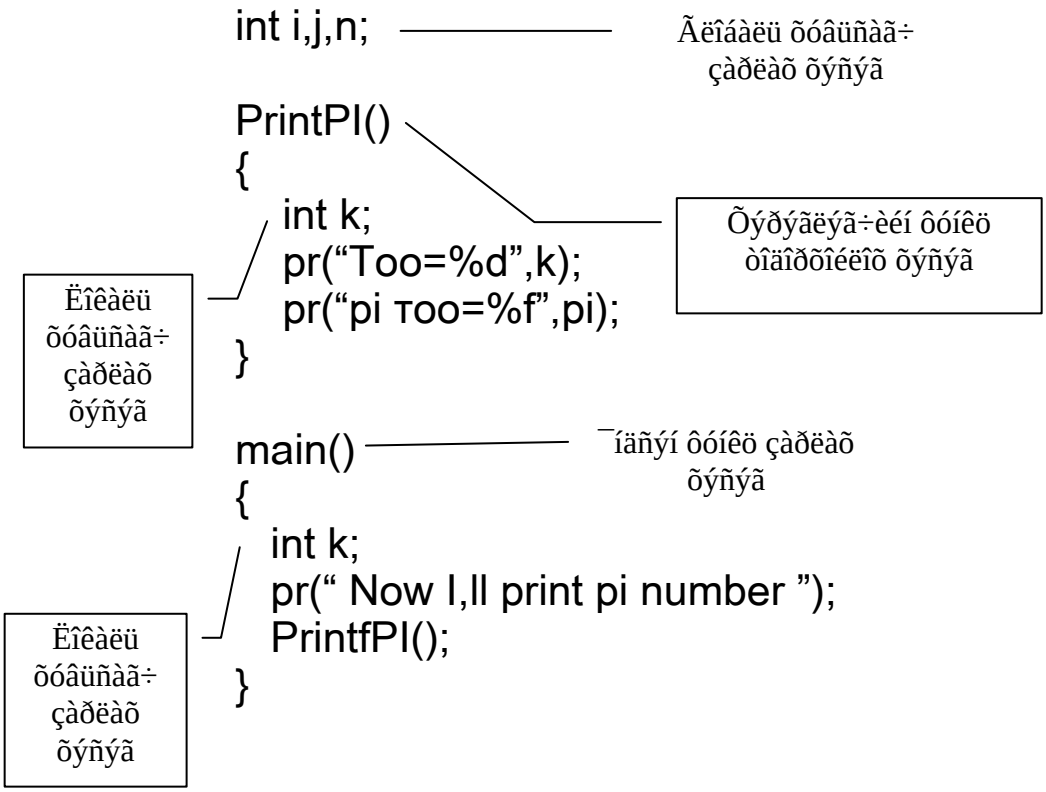
5.Үндсэн функц

Си хэлэнд программын үндсэн функц нь main нэртэйгээр тодорхойлогддог. Үндсэн функцээс программ эхэлж ажиллана. Ө.х Си хэл дээр бичигдсэн программын хамгийн анхны ажиллах функц нь main функц юм.

```
main()
{ <Хувьсагч зарлах хэсэг>
  <Үндсэн программын хэсэг>
}
```

Программын жишээ :





СЭДЭВ 6. ИЛЭРХИЙЛЭЛ БА ҮЙЛДЛҮҮД

6.1 Илэрхийлэл

Бүх төрлийн хувьсагч, тогтмол болон үйлдлүүдийг илэрхийлэлд хэрэглэж болно. Тогтмол болон хувьсагчдыг үйлдлийн тэмдэгээр холбосон дарааллыг илэрхийлэл гэнэ.

(x + y - 10) * 10

Си хэлний илэрхийлэлд хоосон зай ямар нэг нөлөө үзүүлдэггүй. Өөрөөр хэлбэл илэрхийлэлд хичнээн урт хоосон зай ашиглаж болно. Харин хаалт хэрэглэх нь илэрхийллийг ойлгомжтой болгох төдийгүй, үйлдлүүдийн биелэх дарааллыг баттай болгож өгдөг. Илэрхийлэлд хэрэглэгдэх үйлдлүүдийг УНАР , БИНАР гэж ангилдаг.

Унар үйлдэл : Зөвхөн ганц хувьсагч (эсвэл тогтмол) дээр гүйцэтгэгдэх үйлдэл бөгөөд үйлдлийн тэмдэг нь хувьсагчийн өмнө бичигддэг.

Бинар үйлдэл : Хоёр хувьсагчийн хооронд (тогтмолуудын хооронд) үйлдэл хийгдэх ба үйлдлийн тэмдэгийг хувьсагчдын дунд бичдэг.

Жишээ : -i үед (-) нь унар үйлдэл юм.
x + y үед (+) нь бинар үйлдэл юм.

Си хэлэнд арифметик, логик үйлдлүүдийг илэрхийлэлд өргөн ашигладаг. Си хэл нь маш том үйлдлийн олонлогтой. Энэ сэдэвт өргөн хэрэглэгддэг зарим үйлдлүүдийг авч үзэх болно.

6.2 Арифметик үйлдлүүд

Үндсэн 4 арифметик үйлдэл нь +, - , *, / үйлдлүүд болно. Си хэлний үйлдлүүд аль болох цаг хэмнэх , мөн үр дүн нь ашигтай байх зарчмыг баримтлан зохиогдсон байдаг.

Бинар үйлдэл : + - * / %
Унар үйлдэл : - (Turbo C хэлэнд унар + үйлдэл байдаг.)

6.3 Үндсэн үйлдлүүдийн зэрэглэл

Хүснэгт 3.4-т үндсэн үйлдлүүдийн зэрэглэлийг үзүүлэв. Хамгийн эхний мөрд байгаа нь хамгийн өндөр зэрэглэлтэй, мөр доошлох тусам зэрэглэл буурна. Ижил зэрэглэлтэй үйлдлүүдийн хувьд аль талаас нь эхэлж үйлдлүүдийг гүйцэтгэхийг **чиглэл** нь тодорхойлно.

Үйлдлүүд	Чиглэл
( )	Зүүнээс баруун
Унар -	Зүүнээс баруун
* /	Зүүнээс баруун
+ -	Зүүнээс баруун

Одоо дараах илэрхийллийг хэрхэн бодогдохыг авч үзье.
- (-3 * (5+ 2 * 6))+ (3 * 4 + 4)/ 2
- (-3 * (5+ 12))+ (3 * 4 + 4)/ 2
- (-3 * 17)+ (3 * 4 + 4)/ 2
- (-51)+ (3 * 4 + 4)/ 2
- (-51)+ (12 + 4)/ 2
- (-51)+ 16/ 2
51+ 16/ 2
51+ 8
59

/ үйлдлээс өмнө унар - үйлдэл биелэгдэнэ. Хэрэв үйлдлүүдийн зэрэглэл ижил бол чиглэлийнхээ дагуу бодогдоно.

Утга Олгох үйлдлүүд

Утга олгох үйлдлээр аливаа нэг илэрхийллийн эцсийн үр дүнг хувьсагчид олгогдоно. Утга олгох үйлдэл хамгийн бага зэрэглэлтэй үйлдэлд тооцогддог.

Утга олгох үйлдэлийн бичигдэх хэлбэр :

<Хувьсагч>[<үйлдэл>]=<илэрхийлэл>;

<үйлдэл> нь: +, -, *, /, %, <<, >>, &, ^, | эсвэл байхгүй байж болно.

Дараах жишээнд үндсэн үйлдлүүд нь утга олгох үйлдлээс өмнө биелэгдэх болно.

A=2 + 3 * 4; /* Үр дүн : 14 */

Нэгэнт л утга олгох үйлдэл нь **үйлдэл** учраас үйлдэл ашиглаж болох бүх л газар түүнийг ашиглаж болно. Жишээ :

a**=**b**=**c+4; c**=**3 * (d**=**12.0 / x);

Мөн утга олгох үйлдлүүд нь Си хэлний бичлэгийг авсаархан болгоход чухал үүрэг гүйцэтгэдэг. Яаж авсаархан болгосныг доорх жишээнээс үзнэ үү.

= үйлдлээр	+=,-=,*=,/= . . . үйлдлүүд
i = i + 2;	i+=2;
k=k*10;	k *= 10;
x=x*(y+1)	x*=y+1

Жишээ : **x=20; x*=y=z=6;**

Энэ жишээ биелэгдэхдээ :
x*=(y=(z=6)) /* z=6 */
x*=(y=6) /* y=6, z=6 */
x*=6 /* x=x*6 */
x=120; /* Үр дүн 120 */

6.4 Нэмэгдүүлэх, хорогдуулах үйлдэл

Си хэлэнд маш өргөн хэрэглэгддэг үйлдлүүд нь нэмэгдүүлэх (++), хорогдуулах (--) үйлдлүүд юм. i++ гэсэн илэрхийлэл нь i=i+1 гэсэн илэрхийлэлтэй ижил чанартай юм. Нэмэгдүүлэх, хорогдуулах үйлдэл нь обьектын утгыг 1-ээр нэмэгдүүлж, хорогдуулдаг. Энэ үйлдэл нь ихээхэн хэрэглэгддэг үйлдэл боловч түүний дүрэм, зэрэглэлийг сайн ойлгоогүй тохиолдолд хэрэглэхэд хэцүү үйлдэл юм. Дараах илэрхийлэлд

c + i ++

эхлээд i-ийн утга c-дээр нэмэгдээд , дараа нь i-ын утга 1-ээр нэмэгдэнэ.
Дараах хоёр бүлэг командууд ижил чанартай.

b=c + i++;	b=c + i; i=i+1;

++ үйлдэл нь мөн обьектын өмнө бичигдэж болно. Жишээ : c=++i;
++ үйлдэл өмнө нь бичигдэх тохиолдолд c-д i-ын утга олгогдохоос өмнө i-ын утга 1-ээр нэмэгдэнэ.

b=c + ++i;	i=i+1; b=c + i;

Мөн хорогдуулагч (--) үйлдэл нь (++) үйлдэлтэй ижил зарчмаар ажиллана. (++),(--) үйлдлүүд нь унар үйлдэл учир үндсэн 4 үйлдлээс өндөр зэрэглэлтэй байна.

Дүрэм : ++, -- үйлдлийг зөвхөн **обьекттой** хийдэг гэдгийг анхаар. Эдгээр үйлдлийг **тогтмол** утган дээр **хийхгүй. / 5++ , 5 -- , ++5 , --5 бол АЛДАА /**

Үйлдлүүдийн зэрэглэл II
()
++ -- - (Унар үйлдлүүд)
/ * %
+ -
=, +=, -= , …

Мөн Си хэлд өргөн хэрэглэгддэг арифметик үйлдэл нь үлдэгдэл олох (**%**) үйлдэл юм. Энэ үйлдэл нь хуваалт үйлдлийн үлдэгдлийг олдог үйлдэл юм.
y=5; x=12; z=x % y; Үр дүн : z=2
12-ыг 5-д хуваахад 2 ногдож 2 үлдэнэ. Энэ үйлдэл нь float эсвэл double төрлийн утгатай ажиллахгүй, зөвхөн бүхэл тоон төрлийн утгатай ажиллана.

6.5 Бит үйлдлүүд

Бит үйлдэл гэдэг нь тооны битүүд дээр хийгддэг үйлдлийг хэлнэ.
Си хэлэнд дараах 6 бит үйлдлийг ашигладаг.
1. **&** бит БА үйлдэл
2. **|** бит БУЮУ үйлдэл
3. **^** бит XOR үйлдэл
4. **<<** бит дүрслэлийг нь зүүн тийш шилжүүлэх үйлдэл
5. **>>** бит дүрслэлийг нь баруун тийш шилжүүлэх үйлдэл
6. **~** үгүйсгэл

1 бит нь 0 ба 1 гэсэн 2 л утгыг авна. Эдгээр утгууд дээр бит үйлдлүүд хэрхэн үйлчилэхийг авч үзье.

x1	x2	x1 & x2	x1 | x2	x1^x2	~x1
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

Хүснэгтээс харахад :

& үйлдлийн утга нь : Хоёр утга хоёулаа 1 бол эцсийн үр дүн 1 байна, бусад тохиолдолд 0 байна.
| үйлдлийн утга нь : Хоёр утга хоёулаа 0 бол эцсийн үр дүн 0 байна, бусад тохиолдолд 1 байна.
^ үйлдлийн утга нь : Хоёр утга хоёулаа ижил бол эцсийн үр дүн 0 байна, ялгаатай бол 1 байна.
~ үйлдлийн утга нь : Утга 1 бол үр дүн 0 байна, эсрэг тохиолдод 1 байна.

XOR үйлдэл нь 2 бит нь ялгаатай бол 1 , адил бол 0 гэсэн үр дүн буцаадаг.

Жишээ 1 :

```
x1=1;  x2=2;
printf(" %d ", x1&x2);
```

```
00000001
    &
00000010
------------
00000000 = 0
```

<< үйлдэл нь тухайн тооны бит дүрслэлийг тодорхой тоогоор зүүн тийш шилжүүлдэг. Энэ нь тухайн тоог 2-ын зэрэгтээр үржүүлж буйтай утга адил болно. Харин >> үйлдэл нь баруун тийш шилжүүлэх ба энэ нь тухайн тоог 2-ын зэрэгтэд хувааж байгаатай утга адил болно.

Жишээ 2 :

```
x=1;
x=x<<2;          /*    x = x * 2^2 = x * 4      */
```

00000001
<< 2

00000100 = 4

Жишээ 3 :

```
x=8;
x=x>>2;          /*    x = x  / 2^3= x  / 4     */
```

00001000
>>2

00000010 = 2

Жишээ 4 :

```
x=10;
x=x<<3;      printf("%d",x);
```

00001010
<<3

01010000 = 64+16=80

Жишээ 5 :

```
x=3; y=2; z=1;
printf("%d",x^y&~z);                    /*      (x^ (2  & -1 ))=(3 ^ 2)=1       */
```

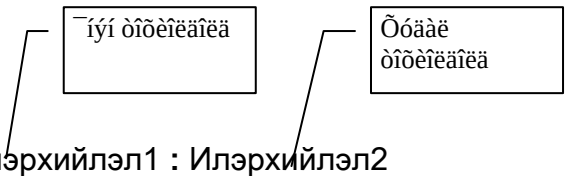
~z= **00000001** y&~z= **00000010** x^(y&(~z))= **00000011**
~ & **11111110** ^ **00000010**
--- --- ---
11111110 **00000010** **00000001**

буюу эцсийн үр дүн **00000001** = 1

Нөхцөлт илэрхийлэл (? :) үйлдэл

Энэ үйлдэл нь илэрхийлэл дотор сонголтын механизмыг хэрэгжүүлдэг үйлдэл юм.

Бичигдэх хэлбэр :



ШалгахИлэрхийлэл **?** Илэрхийлэл1 **:** Илэрхийлэл2

ШалгахИлэрхийлэл -ийн утга Үнэн байвал илэрхийлэл1, харин Худал байвал илэрхийлэл2 нь энэ илэрхийллийн үр дүн буюу утга болдог.

Жишээ 1 :

```
z=(a>b) ? a : b;       /*      z=MAX(a,b)      */
```

Жишээ 2 :

```
x=3,  y=3,  z=1
z+=x<y ? x++ : y++;
---------------------------
(z+=((x>y) ? (x++) : (y++)))
(z+=(y++))
(z+=3)
(z=z+3)
( Үр дүн  x=3, y=4, z=4  )
```

СЭДЭВ 7. СОНГОЛТЫН КОМАНДУУД

7.1 Харьцуулах үйлдлүүд

Си хэл нь үйлдлүүдийн арвин олонлогтой. Бид өмнө нь арифметик, утга олгох зэрэг үйлдлүүдийг үзэж байсан. Удирдлага шилжүүлэх командуудад одоо бидний үзэх харьцуулах үйлдлүүд өргөн ашиглагддаг.

Хүснэгт 7.1

Үйлдлүүд	Утга
==	Тэнцүү
!=	Тэнцүү биш
<	Бага
>	Их
<=	Бага буюу тэнцүү

>=	Их буюу тэнцүү

Дээрх харьцуулах үйлдлүүдийн хувьд үйлдлийн зэрэглэл нь 2 түвшинд хуваагдана. <, >, <=, >= нь дээд түвшний буюу өндөр зэрэглэлтэй үйлдлүүд юм. Харин доод түвшний үйлдлүүд нь ==, != болно. Өөрөөр хэлбэл ==,!= үйлдлүүд нь бусад 4 өөсөө сүүлд бодогдоно. Жишээ : x=5; y=2; z=0;

a. x>y+z /* Үнэн гэсэн утгатай */
b. x+z<=y /* Худал гэсэн утгатай */
c. z<x==y+z>=x /* Худал гэсэн утгатай */

a жишээнд + үйлдэл эхэлж биелэх бөгөөд 5>2 нь үнэн юм. b жишээнд нэмэх үйлдэл эхэлж биелэх бөгөөд 5 <=2 нь худал юм. c жишээнд эхлээд нэмэх үйлдэл, дараа нь <,>= үйлдлүүд биелж, эцэст нь == үйлдэл биелнэ.

0 < 5 == 2 >=5
1 == 0

Тоон төрлийн утгуудыг жишихдээ утгынх нь их багаар жишдэг. Харин тэмдэгт утгыг жишихдээ тэдгээрийн ASCII дугаараар нь жишдэг.
Жишээ нь ASCII дугааруудын заримаас нь :

A-65 B-66 C-67 D-68
. . .
X-88 Y-89 Z-90
a-97 b-98 c-99 d-100
. . .
x-120 y-121 z-122

ASCII дугаараас нь харвал эдгээр тэмдэгтүүдийн эрэмбэ нь :

‘A’<’B’<’C’ . . . ‘Z’<’a’<’b’. . .<’z’

‘A’>’z’ (Худал)
‘Z’>’a’ (Худал)
‘z’>’Z’ (Үнэн)

7.2 if команд

if нөхцөлт командын хамгийн хялбар хэлбэр нь :

if(Нөхцөлт илэрхийлэл)
команд;

```
if (n >1000)
    printf(“ Óòãà õÿçàààðààñ õýòýðëýý … ”);
```

Си хэлэнд Үнэн ба Худал гэсэн ут[illegible] нөхцөлт илэрхийлэлийн утга нь 0 бол түүнийг Худал гэж ойлгоно, харин 0-ээс ялгаатай утгыг үнэн гэж тооцдог.

0	0 утгыг ХУДАЛ гэж ойлгоно.
… -10,-9 … -1,1,2 … 10 , 11 …	0-ээс ялгаатай бүх утгыг ҮНЭН гэж ойлгоно.

Жишээ нь :

. . .
done=1
if(done) printf(“ Үнэн ”);
…
Энэ тохиолдолд дэлгэцэнд **Үнэн** гэж хэвлэгдэнэ. Учир нь 1 бол үнэн гэсэн утгыг төлөөлнө. Дээрх жишээн дэх илэрхийллийг өөрөөр дүрсэлбэл :

if(done ! =0) printf(“ Үнэн ”);

Хэрэв if **команд -**д нэгээс олон команд бичих хэрэгтэй бол тэдгээрийг хос хаалтанд бичиж өгнө.

```
if(нөхцөлт илэрхийлэл)
   {   команд1;
       команд2;
       команд3;
   }
```

Си хэлэнд дээрх нөхцөлт илэрхийлэл нь дурын илэрхийлэл байж болно. Харин бусад программчилалын хэлэнд зөвхөн үнэн, худал утга буцаах илэрхийллийг л зөвшөөрдөг. Жишээ :

if((ch=getchar())==’\n’) команд;

Дээрх жишээг тайлбарлая. Эхлээд getchar функц дуудагдан, үр дүн нь ch хувьсагчид олгогдоно. Дараа нь ch –ын утгыг ‘\n’-тэй тэнцүү эсэхийг шалгаж байна. Энд утга олгох илэрхийллийг заавал хаалтанд хийж өгөх хэрэгтэй. Учир нь == үйлдэл нь = үйлдлээс өндөр зэрэглэлтэй учир түрүүлээд биелэгдчихнэ. Илэрхийллийг бичих ийм боломж нь Си хэлд байдаг дэвшилттэй талуудын нэг юм. Ихэнх программчилалын хэлүүдэд дээрх жишээг дараах байдлаар бичдэг.

Гараас нэг тэмдэгт унших;
if (Тэмдэгт нь Enter товч бол) then команд;

if – else команд

if else –ийн үндсэн формат нь :

```
if(Илэрхийлэл)
      команд;
else
     команд;
```

Энэ команд нь 2 сонголт хийх боломжийг бүрдүүлдэг. Хэрэв илэрхийлэл худал бол удирдлага нь else түлхүүр үгийн дараах командад шилжинэ. Жишээ нь кв тэгшитгэлийн язгуур олох программ бичье.

```
d=b*b – 4*a*c;
if(d>=0)
  {     x1=(-b + sqrt(d)) / (2*a);
        x2=(-b - sqrt(d)) / (2*a);
        printf(“ Язгууруудын нь :   %f , %f”,x1, x2 );
  }
else
   printf(“  Энэ кв тэгшитгэлд язгуур байхгүй. ”);
```

Энэ жишээнд нөхцөл шалган, сонголттой үйл ажиллагаа хийж байна.

7.3 Давхар if команд

Та if командыг хэдэн ч давхраар нь бичиж хэрэглэж болно. Доорх жишээнд хэрэв илэрхийлэл1 –ын утга үнэн бол удирдлага дараагийн if-д шилжих ба илэрхийлэл2-ын утга үнэн бол команд биелнэ. Өөрөөр хэлбэл илэрхийлэл1 ба илэрхийлэл2 хоёр хоёул үнэн утгатай байхад л **команд** биелнэ.

```
if(илэрхийлэл1)
   if(илэрхийлэл2 )  команд;
```

Текст дахь үгийн тоог олдог программ зохиоё. Өгүүлбэр нь цэгээр эсвэл ? ! тэмдэгтүүдээр төгсөнө гэж үзье.

```
if(өмнөх тэмдэгт нь үсэг бол)
  if(Одоогийн тэмдэгт нь үгийн төгсгөл тэмдэгт бол )   үгийн_тоо++;
```

Энэ жишээнд үгийг үсэг бүрээр нь шалгаад үгийн төгсгөл байвал түүнийг тоолж байна. Тэгвэл дээрх жишээг арай өргөтгөе.

```
if(ch==' ')
     {   if(өмнөх тэмдэгт нь үсэг бол) num++;
     }
else
     {  if(ch тэмдэгт нь өгүүлбэрийн төгсгөл бол ) num++;
     }
```

Энд бид давхар if-үүдийг хэрэглэсэн жишээ үзлээ. Давхар if-үүдийг хэрэглэж байхдаа хаалтыг зөв хэрэглэх ёстой. Жишээ :

```
if(илэрхийлэл1)
      if(илэрхийлэл2)
            команд1;
else
      команд2;
```

Хэрэв илэрхийлэл1, илэрхийлэл2 хоёулаа Үнэн бол команд1 биелнэ. Хэрэв илэрхийлэл1 нь үнэн, илэрхийлэл2 нь худал бол команд2 биелнэ. Жишээгээ дахин харцгаая. Харагдах байдлаас нь үзвэл илэрхийлэл1 худал байхад команд2 биелэх ч юм шиг. Тэгэхээр энд “Энэ else аль if-д хамаарах вэ ?” гэсэн асуулт гарч ирнэ. Компиляторт хоосон зай тэмдэгтийн тоо нөлөөлдөггүй гэдгийг санах хэрэгтэй. Өөрөөр хэлбэл бид else-г аль ч if-ын харалдаа бичсэн компилятор нэг л янзаар ойлгоно. Иймээс энд Си хэл тодорхой дүрэм баримталдаг байж таарна.

Дүрэм : Си нь else –г түүнд хамгийн ойр бөгөөд түүнээс өмнө орших, өөрийн гэсэн else –гүй if-д харгалзуулан ойлгодог. Өөрөөр хэлбэл хаалтны баланстай яг ижил зарчмаар ойлгодог. /* […. { …. (…..) … } …] */

Одоо дээрх жишээ ойлгомжтой боллоо. else –г яаж ч бичсэн түүнд хамгийн ойр орших if-д буюу if(илэрхийлэл2)- т хамаарах else нь байна гэсэн үг. Харин ийм ойлгомжгүй байдлаас зайлсхийх 2 арга байдаг. Жишээ :

Хаалт хэрэглэх	Бүх if-д харгалзах else-г нь бичих

if(илэрхийлэл1)
{ if(илэрхийлэл2)
команд1;
}
else
команд2;
	if(илэрхийлэл1)
	if(илэрхийлэл2)
	команд1;
	else
	;
	else
	команд2;

Харин энд ; буюу хоосон командыг дүрсэлсэн байна. Хоосон команд нь программд их хэрэглэгддэг бөгөөд үүнийг хэрэглэх нь программын зөв бичлэг, уншигдах хэлбэрийг сайжруулдаг. Гэхдээ хаалт хэрэглэх нь илүү өргөн хэрэглэгддэг зохимжтой арга юм.

```
if(ch==' ')
    {  if(өмнөх тэмдэгт нь үсэг бол) num++;
    }
else
    {  if(ch тэмдэгт нь өгүүлбэрийн төгсгөл бол )
         if(өмнөх тэмдэгт нь үсэг бол)  num++;
    }
```

7.4 Логик үйлдлүүд

Олон давхар if хэрэглэхээс зайлсхийх өөр нэг арга нь логик үйлдлийг хэрэглэж сурах явдал юм. Си хэлэнд логик && (БА) , || (БУЮУ) гэсэн холбоосууд байдаг. Эдгээр логик үйлдлүүд нь харьцуулах үйлдлүүд болох <,>- аас ч бага зэрэглэлтэй үйлдлүүд юм. Харин && үйлдэл нь || -ээсээ өндөр зэрэглэлтэй. Харин зарим хэлэнд логик үйлдлүүд нь харьцуулах үйлдлээс өндөр зэрэглэлтэй байдаг бөгөөд энэ тохиолдолд хаалт хэрэглэдэг: Жишээ нь : Pascal хэлэнд a<b OR c>d илэрхийлэл нь алдаатай илэрхийлэл болно. Учир нь OR нь эхэлж биелэх бөгөөд үр дүнг нь илэрхийлэлд орлуулахад энэ илэрхийлэл алдаатай илэрхийлэл болно. Харин Си хэлд a<b || c>d нь зөв илэрхийлэл болно.

Логик (БА) буюу && үйлдэл

Шалгагдаж байгаа нөхцөлүүд бүгд үнэн бол эцсийн үр дүн үнэн байна. Үр дүн нь int төрлийн утгатай ба Үнэн=1, Худал=0 гэсэн тус тусын харгалзах утгатай байна.

Бичигдэх хэлбэр : **илэрхийлэл1 && илэрхийлэл2 . . .**

Дээр бичиж байсан жишээгээ одоо && үйлдэл ашиглан бичье.

```
if((ch=' ')&&(өмнөх тэмдэгт нь үсэг)) num++;
else
  {  if((ch нь өгүүлбэрийн төгсгөл)&&( өмнөх тэмдэгт нь үсэг)) num++;
  }
```

Энэ жишээг үйлдлийн зэрэглэл ойлгосон хүн бол хаалтгүй бичнэ.

```
if(ch=' ' && өмнөх тэмдэгт нь үсэг)
```

Гэхдээ харахад ойлгомжтой байх үүднээс хаалт хэрэглэхийг зарим хүн илүүд үздэг.

Жишээ 1 :

```
#include <stdio.h>

int a=1,b=2;
char c='k';

main()
{
  if (a=1 && b=2 && c='k')
     printf(" Бүх шалгалтууд үнэн байна");
  else
     printf("Аль нэг нь, магадгүй бүгд худал байж ч болно.");
}
```

Логик (БУЮУ) буюу || үйлдэл

Шалгагдаж байгаа нөхцлүүдийн ядаж нэг нөхцөл нь ҮНЭН бол үр дүн нь ҮНЭН байна. Харин нөхцлүүд нь бүгд ХУДАЛ тохиолдолд л үр дүн ХУДАЛ байна.

Бичигдэх хэлбэр : **илэрхийлэл1 || илэрхийлэл2 . . .**

Жишээ нь :

```
#include <stdio.h>
int a=1,b=2;
char c='k';

main()
{   if (a=1 || b=1 || c='a')
       printf(" Нөхцөлүүдийн аль нэг нь үнэн байна");
    else
       printf("Нөхцөлүүд бүгд худал байна.");
}
```

Өмнөх үг тоолдог жишээгээ БУЮУ үйлдэл ашиглан кодчилолыг улам багасгая.

Жишээ :

```
if((ch=' ' && өмнөх тэмдэгт нь үсэг) ||
   (ch нь өгүүлбэрийн төгсгөл && өмнөх тэмдэгт нь үсэг))
       num++;
```

&& үйлдэл нь ||-аасаа өндөр зэрэглэлтэй учраас энд мөн хаалт хэрэглэхгүй байж болно. Гэхдээ ингэж кодчилолыг багасгах нь түүнийг ойлгоход улам төвөгтэй болгож байна .
if(ch=' ' && өмнөх тэмдэгт нь үсэг || ch нь өгүүлбэрийн төгсгөл && өмнөх тэмдэгт нь үсэг) num++;

Логик Үгүйсгэл буюу ! үйлдэл

Логик илэрхийлэл нь үнэн, худал гэсэн 2 л утгатай байдаг. Тэгвэл энэ үйлдлээр илэрхийллийн үр дүнг урвуугаар хөрвүүлнэ. Өөрөөр хэлбэл ҮНЭН утгыг ХУДАЛ, ХУДАЛ утгыг ҮНЭН болгоно.

Жишээ : ! (5 < 7) Энэ илэрхийлэл нь худал утгатай байна.

Логик үгүйсгэл үйлдэл нь унар үйлдэл юм. Тиймээс && ба || үйлдлүүдээс өндөр зэрэглэлтэй үйлдэл юм.

!(5<7) || (3>2) -ын үр дүн ҮНЭН
!(5<7 || 3>2) -ын үр дүн ХУДАЛ

Үйлдэл	Тайлбар	Илэрхийлэл	X	Y	Үр дүн
!	Үгүйсгэл	!x	Үнэн		Худал
			Худал		Үнэн
&&	БА	x&&y	Үнэн	Үнэн	Үнэн
			Үнэн	Худал	Худал
			Худал	Үнэн	Худал
			Худал	Худал	Худал
||	БУЮУ	x||y	Үнэн	Үнэн	Үнэн
			Үнэн	Худал	Үнэн
			Худал	Үнэн	Үнэн
			Худал	Худал	Худал

7.5 Үйлдлүүдийн зэрэглэл

Тайлбар	Үйлдлүүд	Чиглэл
Хаалт	( )	Зүүнээс
Массивын элемент	[]	Зүүнээс
Бүтцийн элемент	->	Зүүнээс
Бүтцийн элемент	.	Зүүнээс
Нэмэгдүүлэх, хорогдуулах	++ --	Баруунаас
Бит “Үгүйсгэл” үйлдэл	~	Баруунаас
Логик “Үгүйсгэл” үйлдэл	!	Баруунаас
Хаяг авах үйлдэл	&	Баруунаас
Заагчийн утга үйлдэл	*	Баруунаас
Төрөл хувиргах үйлдэл	(Төрөл)	Баруунаас
Унар хасах үйлдэл	-	Баруунаас
Хэмжээ авах үйлдэл	sizeof	Баруунаас
Үржүүлэх үйлдэл	*	Зүүнээс
Хуваах үйлдэл	/	Зүүнээс
Үлдэгдэл олох үйлдэл	%	Зүүнээс
Нэмэх үйлдэл	+	Зүүнээс
Хасах үйлдэл	-	Зүүнээс

Зүүн тийш шилжүүлэх Баруун тийш шилжүүлэх	<< >>	Зүүнээс Зүүнээс
Бага, бага буюу тэнцүү,их,их буюу тэнцүү	<, <= , > ,>=	Зүүнээс
Тэнцүү, тэнцүү биш	== !=	Зүүнээс
Бит ‘БА’ үйлдэл Бит ‘XOR’ үйлдэл Бит ‘БУЮУ’ үйлдэл	& ^ |	Зүүнээс Зүүнээс Зүүнээс
Логик ‘БА’ үйлдэл Логик ‘БУЮУ’ үйлдэл	&& ||	Зүүнээс Зүүнээс
Нөхцөлт илэрхийлэл	?:	Баруунаас
Утга олгох үйлдлүүд	= += -= /= %= &= ^= |= <<= >>=	Баруунаас
Таслал үйлдэл	,	Зүүнээс

7.6 switch команд

Си хэлэнд тухайн нэг илэрхийлсэн утгаас шалтгаалж олон сонголт хийх шаардлагыг хэрэгжүүлдэг команд байдаг.
Бичигдэх хэлбэр :

```
switch (Илэрхийлэл) {
        case утга1 : Командууд; Командууд; . . . break;
        case утга2 : Командууд; Командууд; . . . break;
        . . .
        default : Командууд; break;
    }
```

Энд **Илэрхийллийн** утгаас олон командуудын аль нь биелэгдэх нь шалтгаална. Хэрэв илэрхийллийн утга нь утга1-тэй тэнцүү бол утга1-ын коммандууд биелэгдэнэ. Хэрэв утга2-той тэнцүү бол түүний командууд биелэгдэх гэх мэт …

Харин илэрхийллийн утга тухайн сонголтуудад байхгүй бол default түлхүүр үгийн дараах командууд биелэгдэнэ. switch командыг тухайн хэрэглэх шаардлагаас хамаарч янз янзаар бичиж ашигладаг. Үүнд :

- ❖ break командыг бичихгүй байж болно.
- ❖ default сонголт байхгүй байж болно.
- ❖ Хамгийн сүүлчийн сонголтын ард break бичих шаардлагагүй.

Гэхдээ дараах дүрмийг зайлшгүй баримтлах ёстой.

- илэрхийлэл нь int төрлийн утга буцаах ёстой.
- case түлхүүр үгийн хойно зөвхөн тэмдэгтэн болон тоон тогтмол л бичигдэнэ.
- Сонголтын утгууд нь хоорондоо ялгаатай утга байх ёстой.

---

switch-ээс тасалдлын оператор break-ийн тусламжтайгаар гардаг.

```
a=2;
switch (a) {
        case 1 : printf(“ Утга 1 \n”); break;
        case 2 : printf(“ Утга 2 \n”);
        case 3 : printf(“ Утга 3 \n”);
                 break;
    }
```

Үр дүн нь : **Утга 2**
Утга 3

Энд 2 гэсэн сонголтыг break –ээр төгсгөөгүй учир шууд дараачийн сонголт руу орж байна. Олон утгаас шалтгаалах командыг биелүүлэхийн тулд switch-ийн зарим сонголтод break бичдэггүй.

```
switch(x)
 {        case2 : case 4: case 6 : case 8:  printf(“ Тэгш тоо \n”);   break;
 }
```

switch, if хоёр нь хоёулаа сонголтын командууд бөгөөд олон сонголттой үед switch ашиглах нь илүү тохиромжтой байдаг.

```
if(score>=90) grade=’A’;
else if(score>=80) grade=’B’;
     else if(score>=70) grade=’C’;
          else if(score>=60) grade=’D’;
               else grade=’F’;
```

Эдгээр if-үүдийг нэг л switch орлож чадна.

```
int n;
n=score/10;

switch(n)
 {
      case 10:
      case 9: grade=’A’; break;
      case 8: grade=’B’; break;
      case 7: grade=’C’; break;
      case 6: grade=’D’; break;
      default: grade=’F’; break;
 }
```

Жишээ :

```
switch(c)
{       case ‘a’ :  printf(“ a үсэг \n”); break;
        case ‘b’ :  printf(“ b үсэг \n”); break;
        case ‘c’ :  printf(“ c үсэг \n”); break;
 }
```

СЭДЭВ 8. ДАВТАЛТ

Нэг буюу хэд хэдэн үйлдлийг олон дахин гүйцэтгэх зорилгоор давталтын командуудыг ашигладаг. Си хэлэнд дараах 3 давталтын командыг ашигладаг.

1. Тодорхой давталт (for)
2. Нөхцөлт давталт (while)
3. Нөхцөлт давталт (do . . . while)

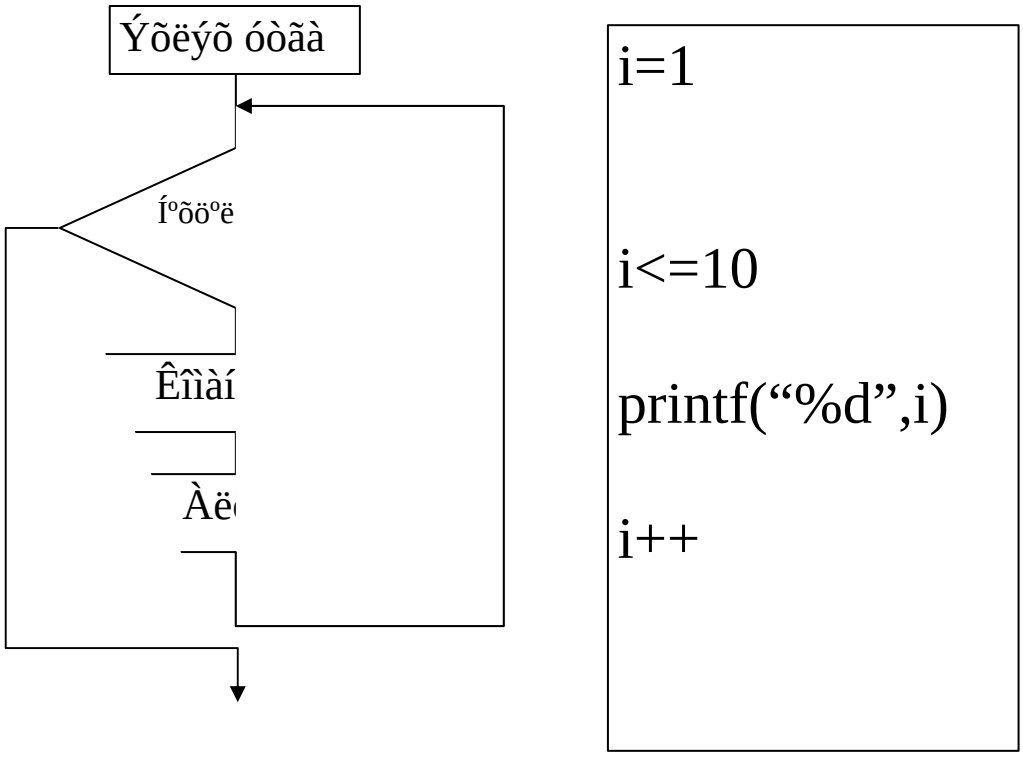
7.1 Тодорхой давталт буюу for давталт

Давтах давталтын тоо тодорхой бол энэ операторыг ашигладаг. Энэ давталтыг тоолуурт давталт ч гэж нэрлэдэг.

Бичигдэх хэлбэр :

for([<ЭхлэхИлэрхийлэл>];[<ШалгахИлэрхийлэл>];[<АлхамИлэрхийлэл>])
команд;

Энэ давталт нь <ШалгахИлэрхийлэл>-ийн утга худал болтол биелнэ. Хэрэв <ЭхлэхИлэрхийлэл> нь <ШалгахИлэрхийлэл>-ийн утгаас хальсан тохиолдолд цикл ганц ч удаа биелэхгүй. <АлхамИлэрхийлэл> нь давталтын тоолуурын утга хэрхэн өөрчлөгдөхийг тодорхойлно.



1. Хэрэв Шалгах илэрхийллийн утга үнэн бол оператор биелнэ. Дараа нь алхам тодорхойлсон илэрхийлэл бодогдоно.
2. Хэрэв Шалгах илэрхийлэл байхгүй бол түүний утгыг үнэнд тооцож, дээрх маягаар давталт үргэлжилнэ. Ийм тохиолдолд давталт төгсгөлгүй үргэлжилнэ.
3. Хэрэв нөхцөлт илэрхийллийн утга худал бол **for** операторын хийх үйлдэл төгсөж удирдлага түүний дараагийн операторт шилжинэ.

Жишээн дээр тайлбарлавал илүү ойлгомжтой байх болно.

```
printf(“ Тоолж байна . . . \n”);
for(i=1; i<=10; i++)   printf(“%d \t”,i);
```

Одоо дээрх программын хэсгийг тайлбарлая. Давталт нь ЭхлэхИлэрхийллээс эхлэх ба i-д утгыг 1 утгыг олгоно. ЭхлэхИлэрхийлэл нь давталтад ердөө ганц удаа буюу хамгийн эхэнд биелэгдэнэ. Дараа нь Шалгах илэрхийлэл бодогдоно. Ингээд 1<=10 нь үнэн учир давталт цааш үргэлжилж, printf функц ажиллан дэлгэцэнд 1 гэж хэвлэнэ. Дараа нь АлхамИлэрхийлэл бодогдон i–ын утга 2 болно. Тэгээд Шалгах Илэрхийлэл бодогдон i<=10 эсэхийг шалгах гэх мэтчилэн давталт үргэлжилнэ. Шалгах Илэрхийллийн утга худал болсон үед давталтаас гарна.
Дээрх жишээний үр дүн :

```
Тоолж байна . . .
1       2       3       4       5       6       7       8       9       10
```

Жишээ 2 :

```
for(i=10; i>0; i--)   printf(“%d \t”,i);
```

Жишээ 3 : Давталтын алхам нь 1- ээс ялгаатай байж болно.

```
for(i=1; i<18; i+=3)   printf(“%d \t”,i);
```

Мөн for давталтыг хэдэн ч давхраар нь ашиглаж болно.

```
for(i=1, k=1; i<=3; i++)
  {    for(j=1; j<=3; j++)  printf(“%d \t”, k++);
       printf(“\n”);
  }
```

Жишээ 4 :

```
for(i=0,i=1000; j>i;  i++, j/=10)  printf(“%d  %d”,i,j);
```

Эхлэхдээ i=0, j=1000 дуусахдаа i=3, j=1 утгатай болно.

Жишээ 5: 1-ээс 10 хүртэлх тооны нийлбэр ол.

```
#include<stdio.h>
int i, sum=0;
main()
{   for(i=1; i<=10; i++)  sum+=i;
    printf(“ %d”, sum);
}
```

Зөвлөгөө :

- for давталтыг хувьсагчийн утга тодорхой утгаар нэмэгдэх, хасагдах давталтанд ашигла.
- for давталтын ШалгахИлэрхийлэл нь зөв илэрхийлэл эсэхийг анхаарч бай.
- for давталтан дотор ;-аар илэрхийллүүд хоорондоо зааглагддагийг анхаар.

8.2 Нөхцөлт while давталт

Хамгийн хялбар төрлийн давталт нь while давталт юм.
Бичигдэх хэлбэр :

while (<илэрхийлэл>)
{ нэг буюу хэд хэдэн команд; }

Илэрхийллийн утга үнэн л байвал давталт үргэлжилнэ. while –ын арын ( ) хаалтны ард ; тавьж болохгүй. Хэрвээ тавьбал while командууд хэзээ ч биелэгдэхгүй. while-ын нөхцөлт илэрхийлэл анх удаагаа шалгагдахдаа буруу бол while-ын командууд ерөөсөө биелэгдэхгүй.

(i<10)

printf(. . . .)

Жишээ :

```
/*        Дэлгэц  цэвэрлэх          */
void cls(void)
{   int i=0;
    while(i<25)        /*      25 хоосон мөр хэвлэснээр дэлгэц цэвэрлэгдэнэ       */
     {    printf("\n");              /*      Нэг хоосон мөр хэвлэх       */
           i++;
     }
}
```

Бараг бүх Си компилятор дэлгэц цэвэрлэдэг функцтэй байдаг. Энэ арга нь хэдийгээр муу боловч дэлгэц цэвэрлэдэг аргуудын нэг юм. Харин Turbo Си- д clrscr (conio.h-д байдаг) функцээр дэлгэц цэвэрлэнэ. Дээрх жишээнд i-ын утга 0-ээс 25 хүртэл давтана.Учир нь 25 –нь 25 –аас бага биш юм. (Тэнцүү) Хэрэв дээрх жишээнд i-ын утгыг өөрчилж өгөөгүй бол давталт мөнхийн болох магадлалтай.

8.3 Нөхцөлт do while давталт

while нь do командтай хамт нэгэн давталтыг үүсгэдэг. do … while нь while давталттай бараг адилхан юм.

Бичигдэх хэлбэр :

```
do
   {  нэг буюу хэд хэдэн команд;  }
while(<илэрхийлэл>);
```

do давталтын команд нь илэрхийллийн утга худал болтол давтагдана.

Давталтын командууд нь дор хаяж нэг удаа биелэгдэх давталтыг do - while ашиглан хийдэг. Нөхцөл нь давталтын командууд биелэгдсэний дараа шалгагддаг.

Дараах программд 0 … 9 тоо ба тэдний нийлбэрийг хэвлэж байна.

```
main()
{ int count=0;
  int total=0;
  do {  total+=count;
        printf(“ тоо = %d ,  Нийлбэр = %d  \n” , count++, total);
      }
  while (count<10);
}
```

8.4 Үргэлжлүүлэх команд

Бичигдэх хэлбэр : **continue;**

Энэ нь **do, for, while** давталтын командуудтай хамтарч хэрэглэгдэнэ. Өөрөөр хэлбэл энэ команд нь давталтын командуудын блок дотор хэрэглэгдэнэ. Программ биелж байгаад тухайн давталтын командын блок доторх **continue** –д удирдлага шилжвэл түүний дараагийн үйлдлүүд биелэхгүй бөгөөд дараагийн цикл шууд эхэлнэ.

Жишээ :

```
i=1;
while(i<100)
{ if (i % 2==0)
      {  i++;
         continue;
      }
  printf(“ %d ”, i++);
}
```

Энэ жишээнд 1 … 99 гэсэн завсраас сондгой тоонуудыг хэвлэж байна. Тэгш утгууд дайралдахад if –ын нөхцөл биелэн, continue команд биелэгдэн дараагийн цикл эхлэнэ. i –ын утга нэмэгдсэн байх тул энэ циклд i сондгой тоо байна. Гэх мэтчилэн бүх сондгой тоог хэвлэнэ.

Жишээ : Өмнөх жишээг for давталтаар гүйцэтгэе.

```
for(i=1;i<100;i++)
 {  if(i % 2==0)  continue;
    printf(“ %d ”,i);
 }
```

8.5 Тасалдлын команд

Программ бичиж байх явцад циклээс ердийн бус аргаар (Давталтын нөхцөлөөс өөр аргаар) гарах шаардлага тулгардаг. Энэ зорилгыг хэрэгжүүлэх **break** команд Си хэлэнд байдаг.

Бичигдэх хэлбэр : **break;**

Энэ нь мөн л давталт дотор агуулагдах ба давталтаас гарах үүрэгтэй оператор юм. Үүнийг мөн **for, while, do** давталтуудаас, мөн **switch** сонголтоос гарах зорилгоор ашигладаг. Давталт биелж байгаад тасалдлын команд тааралдан биелмэгц удирдлага давталтаас гарна. Өөрөөр хэлбэл break команд биелснээр давталт дуусна. Жишээ : Өмнөх жишээний continue-г break-ээр сольё.

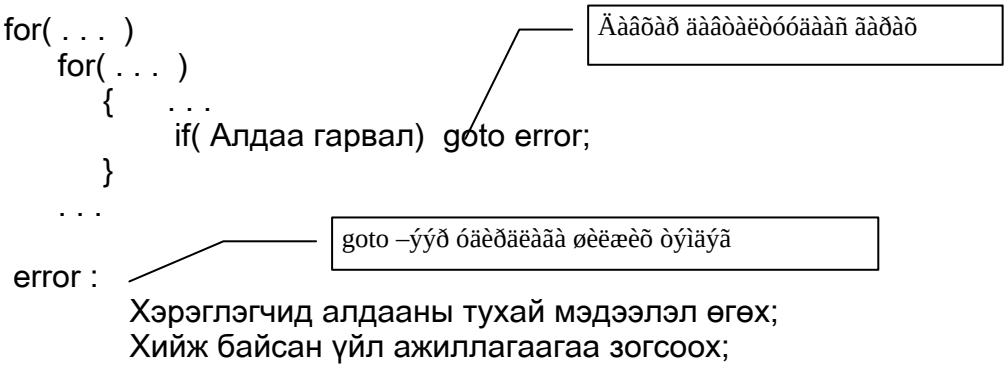
```
i=1;
while(i<100)
{ if (i % 2==0)  { i++; break; }
   printf(" %d ", i++);
}
```

Тэгвэл энд зөвхөн 1 гэсэн утгыг хэвлээд л давталт дуусна. Учир нь i-ын утга 2 болсноор break комманд биелэгдэнэ.

8.6 Удирдлага шилжүүлэх goto комманд

Энэ командыг ашиглан программын удирдлагыг нэг цэгээс нөгөөд шилжүүлдэг. Гэхдээ ямар ч программыг энэ коммандыг ашиглахгүйгээр бичиж болдог. Ер нь энэ командыг их хэрэглэх хэрэггүй юм. Учир нь үүнийг их хэрэглэж бичсэн программ зөв бүтэцтэй алгоритм болоход их түвэгтэй. Үүнийг улам ихээр хэрэглэх тусам шийдэхэд төвөгтэй асуудлууд гарч ирдэг.

Гэхдээ үүнийг зайлшгүй хэрэглэх зарим тохиолдлууд байдаг. Олон давхар хэрэглэгдсэн давталтаас гарахын тулд энэ командыг ашиглана. Учир нь break командаар давхар давхар давталтаас гарахгүй, зөвхөн дотор талын давталтаас гарна.



goto командаар удирдлага шилжүүлэхийн тулд удирдлага шилжиж очих газар тэмдэг тавих шаардлагатай. Тэмдэгийг нэрлэхдээ хувьсагч яаж нэрлэдэг шиг нэрлэдэг .

Тэмдэг тавих хэлбэр :

<Тэмдгийн нэр> :

Удирдлага шилжүүлэхдээ :

goto <Тэмдгийн нэр>;

Тэмдгийн үйлчлэх хүрээ нь зөвхөн нэг функцээр хязгаарлагдана. Өөрөөр хэлбэл өөр функцд тавьсан тэмдэг рүү **goto** –г ашиглан удирдлага шилжиж болохгүй.

СЭДЭВ 9. ФУНКЦ

Си хэлэнд функцуудыг стандарт ба хэрэглэгчийн гэж 2 ангилдаг. Стандарт функцууд нь урьдчилан тодорхойлогдсон (зохиогдсон) функцууд байдаг ба *.h файлуудад тодорхойлогддог. Харин хэрэглэгчийн функцийг программ зохиогч зохионо. Функц нь программд олон давтан хийгдэх үйлдлүүдийг багасгаж, үйлдлүүдийг бүлэглэж өгдөг.

Хэрэглэгчийн функц тодорхойлох

Үндсэн хэлбэр:

```
<буцаах утгын төрөл> <ФункцНэр>(<Аргументийн жагсаалт>)
<Аргументуудын зарлалт>
{
  <Локаль хувьсагчдыг зарлах>;
   . . .
  <Функцын үндсэн бие буюу  Командууд >;
   . . .
   return (буцаах утга);
}
```

1. Хэрэв функц утга буцаах шаардлагатай бол функцын нэрийн өмнө тухайн функцээс **буцаах утгын төрлийг** бичиж өгнө. Хэрэв функц **утга буцаадаггүй бол** түүний нэрийн өмнө нь **void** түлхүүр үгийг бичиж өгнө эсвэл ерөөсөө түлхүүр үг бичихгүй.
2. Хэрэв функц руу аливаа утга дамждаг бол түүнийг хүлээж авах хувьсагчдыг **аргументууд гэж нэрлэдэг** ба хувьсагчдыг функцын нэрийн ард ( ) хаалтанд **таслалаар зааглан** бичиж өгнө. Хэрэв функц **утга хүлээн авдаггүй бол** ( ) хаалтанд юу ч бичихгүй. Аргументуудын төрлийг буюу тэдний зарлалтыг дараагийн мөрд хувьсагч зарладаг шигээр зарлаж өгнө.
3. Локаль хувьсагчдыг зарлах хэсэгт тухайн **функц дотор л хэрэглэгдэх** хувьсагчдыг зарлаж өгнө.
4. Харин return үйлдлийг ашиглан тухайн **функцын үр дүнг** түүнийг дуудсан газарт **буцаана.**

Функцын биеийг { } хаалтанд бичнэ.

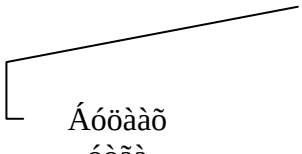
Àðãóìåíòóóä

Жишээ :

```
int Max(a,b)
```

Àðãóìåíòóóäûí çàðëàëò

Ëîêàëü õóâüñàã÷




```
int a,b;
{       int k;
        k=(a>b) ? a : b;
        printf(“ Max= %d”, k);
        return k;
}
```

Òòãà áóöààõ ¿éëäýë

Функц нь аргументгүй байж болно. Мөн заавал утга буцаах албагүй. Pascal хэлэнд утга буцаадаггүй функцыг процедур ч гэж нэрлэдэг. Харин Си хэлд бол ялгаа байхгүй бүгд функц юм.

```
void notice()
{  printf(“ Та түр хүлээнэ үү , бодолт хийж байна . . . \n”);
}
```

Хэдийгээр тухайн функц ямар ч аргументгүй ч байсан түүнийг тодорхойлохдоо нэрийнх нь ард () хос хаалтыг заавал бичдэг. Мөн дээрх функц нь утга буцаахгүй учраас **return –ыг бичээгүй байна**. Утга буцаадаггүй функцыг тодорхойлохдоо функцын нэрийнх нь өмнө **void** түлхүүр үгийг бичиж өгдөг.

Функцыг дуудахдаа нэрээр дуудах ба хэрэв функц аргумент авдаг бол түүнийг хаалтанд жагсааж өгнө. Дээрх жишээнд үзүүлсэн Max функцыг дуудъя.

```
Max(i,n);
```

Мөн функцыг илэрхийлэлд ашиглаж болно.

```
k = Max(i,n);
if (Max(k,100)==100)  printf(“ Max = 100 ”);
```

Функцээс утга буцаах үйлдэл

Си хэлэнд функцээс утга буцаахдаа **return** түлхүүр үгийг хэрэглэдэг.

Бичигдэх хэлбэр :

return [( ] <илэрхийлэл> [ )];

Жишээ :
```
return a+b;
return (a+b);
return a;
return (a);
```

return үйлдэл гүйцэтгэгдэхэд удирдлага тухайн функцээс гарна. Өөрөөр хэлбэл нэг блокт түүний ард бичигдсэн үйлдлүүд биелэгдэхгүй.

```
int Sum(int a, int b)
{  int k;
   k=a+b;
   return k;
   printf(“Sum=%d”,k);   ← Үйлдэл хэзээ ч биелэгдэхгүй
}
```

Функцыг зарлах

Функц нь char, float, double гэх мэт янз бүрийн утга буцаадаг. Хэрэв функц int-ээс ялгаатай утга буцаадаг бол түүнийг заавал зарлах шаардлагатай. Функцыг зарлахдаа тухайн функц рүү дамжих аргументын болон буцаах утгын төрлийг зааж өгдөг.

```
float round_n();                /*      Функцыг зарлах      */
main()
{   round_n(123.6789, 2);       /*      Функцыг дуудах нь   */
}
/*      Функцыг тодорхойлолт        */
/* Энэ функц нь x тоог n оронгоор нарийвчлана.    */
float round_n(x,n)
float x;
int n;
{   float factor=1.0;
    int i;
    for(i=1;i<=n;i++)  factor*=10.0;
    return ((float)((int)(factor*x+0.5))/factor);
}
```

Дээрх жишээнд x тоог таслалаас хойш n оронгоор нарийвчилж байна. Жишээг тоон туршиж үзвэл: 54.6789 тоог 2 оронгоор нарийвчлая. factor –д 10*10 буюу 100 гэсэн утга олгогдоно. Тоогоо үүгээр үржүүлснээр 5467.89 утга гарна. 0.5 –ийг нэмснээр 5468.39 болох ба (int) хувиргалт хийснээр 5468 болно. Энэ утгаа factor –д хуваасnаар эцсийн үр дүн 54.68 гарна.

Функц нь дараах байдлаар дуудагдаж болно.

```
a)  x=round_n(x,2);
b)  m=3;
    z=round_n(x,m);
c)  if(round_n(cost,2)>100.0)
    командууд;
```

Функцыг Си хэлний дурын илэрхийлэлд дотор дуудаж ашиглаж болно. Гэхдээ төрлийн зохицуулалтыг зөв хийх ёстой. Жишээ : power функц int төрлийн утга буцаадаг. Тэгвэл түүнийг дараах илэрхийлэлд ашиглахдаа төрлийн зохицуулалт хийх ёстой.

```
float x;
x=(float)power(a,b);
```

Харин x=power(a,b) байдлаар дуудаж болохгүй. Хэрэв функцыг дуудахдаа түүний нэрийн өмнө буцаах утгын төрлийг заагаагүй бол Си энэ функцын буцаах утгыг int төрөлтэй гэж ойлгоно.

Өөрөөр хэлбэл default өгөгдлийн төрөл нь int юм Хэрэв функц үнэхээр юу ч буцаадаггүй бол түүнийг зарлахдаа нэрийн өмнө void түлхүүр үгийг хэрэглэдэг.

```
void print_it(n)
```

Ингэж void гэж тодорхойлж өгснөөр функц зарлалт ба функц дуудалтыг ялгаж өгнө . Жишээнээс үзвэл :

```
int  i,j;
wizard();
```

Энэ жишээнд тухайн функцыг ажиллуулахаар дуудаж байгаа юу эсвэл түүнийг зарлаж байгаа юм уу гэдгийг ялгахад түвэгтэй болжээ. Энэ функц нь утга буцаадаггүй учраас түүнийг зарлахдаа төрөл бичихгүй гэвэл энэ нь функц дууддаг бичлэгтэй адилхан болох юм. Иймээс утга буцаадаггүй функцыг зарлахдаа void түлхүүр үгийг хэрэглэдэг.

```
int i,j;
void wizard();
```

Одоо ийм бичлэг хэрэглэсэн тохиолдолд ямар ч түвэгтэй асуудал байхгүй болно. Void төрлийг мөн ямар ч аргумент авдаггүй функцын зарлалт болон тодорхойлолтонд хэрэглэдэг. Ингэж void төрлийг хэрэглэх нь аливаа хоёрдмол шийдлээс зайлсхийдэг сайн талтай. Мөн аргумент авдаггүй функцыг зарлахдаа ч void түлхүүр үгийг ашигладаг.

```
void Sum(void);
main()
{    Sum();
}
Sum()
{    printf(" Sum = %d", a+b);       }
```

Функцын аргументууд

Функцын тодорхойлолтонд тухайн функцын аргументуудыг жагсааж тодорхойлж өгдөг.

```
print_n(x,y,k)
int x,y,k;
{
}
```

Энэ жагсаалтанд хэдэн ч хувьсагч байж болно эсвэл бүр хоосон ч байсан болно. Функцын тодорхойлолтонд байгаа аргументуудыг тодорхойлсон аргументууд гэж нэрлэнэ. Функцыг дуудахдаа мөн аргументуудыг зааж өгөх бөгөөд тэдгээрийг жинхэнэ аргументууд гэж нэрлэнэ.

Тодорхойлсон болон жинхэнэ аргументууд тоогоороо мөн төрлөөрөө харгалзан тэнцүү байх ёстой боловч Си хэл нь яг ийм хатуу дүрэм баримталдаггүй. Энэ нь Си хэлний бас нэг өргөн боломж юм. Гэтэл бусад хэлэнд харгалзан тэнцүү байх дүрмийг баримталдаг. Си хэлэнд жинхэнэ аргументуудын тоо нь тодорхойлсон тооноосоо бага байж болно. Жишээ нь зарим стандарт функц буюу printf , scanf нь хувьсах тооны жинхэнэ аргументуудтай байдаг.

Төрөл шалгах

Си хэл нь хэлний дүрэм нь уян хатан байдаг. Иймд жинхэнэ аргументуудын төрөл нь тодорхойлсон төрөлтэйгээ ижил байх албагүй. Энэ нь тийм ч муу тал

биш юм. Жишээ нь 2 тооны ихийг олдог функц зохиоё. Энэ тохиолдолд тоо гэсэн ойлголтонд бодит тоо, бүхэл тоо багтана. Иймээс хатуу дүрэмт хэл Паскал дээр ийм функц бичье гэвэл бодит бүхэл тоон төрөл тус бүр дээр ажиллах 2 өөр функц бичих хэрэгтэй болно. Харин Си хэлэнд ингэх шаардлагагүй.

```
main()
{ float x,y,z,w,max();
  i=5; j=7; m=2;
  n=(int)max((float)i,(float)j);    /*     1      */
  z=max(x,y);
  printf("%d  %f",n,z);
  w=max((float)i,x);
  m=(int)max(y,(float)j);
  printf("%f %d",w,m);
}
float max(a,b)
float a,b;
{ if(a>b) return a;
  else return b;
}
```

Энд max функц нь float төрлийн утгууд аван, float төрлийн утга буцаасан байна. Иймээс n,m мэтийн бүхэл тоон төрлийн хувьсагчид функцын үр дүнг олгохдоо төрлийн хувиргалт хийх шаардлагатай. Мөн аргументэд нь бүхэл тоон төрлийн хувьсагч олгохдоо (float) төрлийн хувиргалт хийх хэрэгтэй.

Дараах жишээнд sqrt функц нь double төрөлтэй аргумент авдаг. Тэгвэл дараах байдлаар дуудсанаар буруу үр дүн үзүүлж болно.

```
int n;
double x;
n=7;
x=sqrt(n);
```

Учир нь n аргумент нь sqrt –ийн аргументаас өөр төрөлтэй байна. Тиймээс төрлийн хувиргалтыг дараах 2 аргын аль нэгээр хийх ёстой.

a.
```
y=(double)n;
x=sqrt(y);
```

b.
```
x=sqrt((double)n);
```

Функцээс утга буцаах

Бидний өмнө үзсэн байдлаар функцэд аргумент дамжуулахад тухайн утгууд нь функцын мужид дахин нэг хувь хуулагддаг. Иймээс энэ аргументийн утгыг өөрчилсөн ч тухайн функцэд утгаа дамжуулсан үндсэн хувьсагчийн утга өөрчлөгддөггүй. Ийм байдлаар функцэд аргумент дамжуулахыг утгаар дамжуулах гэж нэрлэдэг. Харин хувьсагчийн утгыг функц дотроос өөрчлөхийн тулд хаягаар дамжуулах гэдэг аргыг хэрэглэдэг. Энэ аргын гол мөн чанар нь функцэд хувьсагчийн утга дамжуулахын оронд хувьсагчийн хаягийг дамжуулдаг.

Дараах жишээнд хоёр бүхэл тоон хувьсагчийн утгыг сольдог функцыг үзүүлэв.

```
swap(m,n)
int  *m,*n;
{  int temp;
   temp=*m;
   *m=*n;
   *n=temp;
}
```

Энэ функцыг дуудахдаа **swap(&m,&n)** байдлаар дуудна. Өөрөөр хэлбэл тухайн функцэд хувьсагчдын утгыг биш тэдгээрийн хаягийг дамжуулах ёстой учраас **&** үйлдлийг ашиглан хаягийг нь авч функцэд дамжуулж байна.

Харин функц дотроо эдгээр хувьсагчдын утгыг уншихдаа заагч хувьсагч дээр ажилладаг **унар *** үйлдлийг ашиглан тэдгээрийн утгыг уншиж, өөрчилж байна.

Дээрх функцын үйл ажиллагааг тайлбарлавал :

```
temp=*m;     /*  m –ын агуулж байгаа хаягт байгаа утгыг temp –д хийж байна  */
*m = *n;     /*  n –ын агуулж байгаа хаягт байгаа утгыг m хаягт хийж байна  */
*m=temp;     /*  temp-ын утгыг m –ын агуулж байгаа хаягт  хийж байна  */
```

Харин **int *m,*n;** гэсэн бичлэг нь int төрлийн обьект заах n,m хувьсагч зарлаж байна гэсэн үг.

Хадгалах ангилал

Бид урьд нь хадгалах ангилалын талаар бага зэрэг үзэж байсан. Хувьсагч зарлахад түүний хадгалах ангилал нь тодорхойлогддог. Хувьсагч зарлах :

[ХадгалахАнгилал] [Хувиргагч] Төрөл Идентификатор;

Хадгалах ангилал нь хувьсагчийн ажиллах хүрээ ба амьдралын хугацааг тодорхойлдог. default буюу анхдагч хадгалах ангилал нь auto ангилал юм. Өөрөөр хэлбэл хувьсагчийг зарлахдаа хадгалах ангилалыг нь зааж өгөөгүй бол тэр хувьсагч нь шууд auto хадгалах ангилалаар зарлагдана. Ийм хадгалах ангилалтай зарлагдсан хувьсагчийн ажиллах хүрээ нь локал буюу функцын хүрээнд л ажиллана. Мөн амьдралын хугацаа нь функцын ажиллах хугацаагаар хязгаарлагдана. Функцээс удирдлага шилжихэд хувьсагч санах ойгоос арилна.

Утга нь тухайн программын амьдралын хугацаанд хадгалагддаг, харин ажиллах хүрээ нь нэг функцын хүрээгээр тодорхойлогддог хувьсагчийг статик хувьсагч гэнэ. Статик хувьсагчийг **static** түлхүүр үгээр зарлана. Зөвхөн хувьсагч ч биш, функцыг статик ангилалаар зарлаж болно. Хэрэв таны бичсэн С программ хэд хэдэн файлаас тогтдог бол статик ангилалаар зарлагдсан функцыг тухайн файлд л хэрэглэх буюу бусад файлаас энэ функцыг ашиглах боломжгүй болно.

Бичигдэх хэлбэр : **static <төрөл> <хувьсагч>;**

Жишээ :

```
#include <stdio.h>
Count()
{  static int  k=1;
   printf("k=%d\n", k++);
}
main()
{  Count();          /*  k = 1  */
    Count();         /*  k = 2  */
    Count();         /*  k = 3  */
    Count();         /*  k = 4  */
    Count();         /*  k = 5  */
}
```

Гадаад хувьсагч болон гадаад функцыг ашиглахдаа түүнийг **extern** түлхүүр үгээр зарладаг. Өөрөөр хэлбэл таны ашиглах гэж буй функц , хувьсагч нь энэ файлд биш өөр бусад файлд байрладаг үед энэ түлхүүр үгийг ашиглан зарласнаар тэдгээрийг ашиглах боломжтой болно. Зөвхөн глобаль хувьсагчдыг л extern хадгалах ангилалаар тодорхойлж өгнө. Доорх жишээнд хувьсагчдын хадгалах ангилалуудын ялгааг үзүүлжээ.

Хүснэгт 9.1 Хадгалах ангилал, ажиллах хүрээ, хугацаа

Түлхүүр үг	Ажиллах хүрээ	Амьдралын хугацаа
auto	Локаль	Функцын хугацаа
static	Локаль	Программын туршид
register	Локаль	Программын туршид
external	Бүх файлуудад	Байнга
external static	Нэг файлд	Байнга

Дараах жишээнд external хувьсагчийг хэрхэн тодорхойлох болон зарлахыг үзүүлжээ.

Prg1.c

```
#include <stdio.h>
int max,sum;
Sum(int a,int b)
{ sum=a+b;  }
Max(int a,int b)
{ max=a>b?a:b; }
```

Prg2.c

```
#include <stdio.h>
extern int max,sum;
extern int Sum(int,int),Max(int,int);
main()
{ Sum(2,3); Max(2,3);
  printf(" Sum : %d\n",sum);
  printf(" Max : %d\n",max);
}
```

Дээрх 2 файлыг нэгтгэн нэг программ болгохын тулд PRJ файл тодорхойлох шаардлагатай.

Блок бүтэц

Бидний өмнө үзсэнчлэн auto , register, static хувьсагчид нь функц дотор тодорхойлогдвол локаль ажиллах хүрээтэй болдог. Тэгвэл блок дотор локаль ажиллах хувьсагч зарлах боломж Си хэлд байдаг. Ийм хувьсагч нь зөвхөн тухайн блок дотор л ажиллах ба блокоос гарахад устаж үгүй болно.

```
int exam(a1,a2)
int a1,a2;
{   int i, n;
    . . .
     {   int i,temp;
          for(i=1;i<n;i++)   temp+=i;
     }
    . . .
}
```

Тухайн блок дотор л i, temp хувьсагчид ашиглагдах бөгөөд блокын гадна эдгээр хувьсагчид ашиглагдах боломжгүй юм. Функц дотор зарлагдсан i хувьсагч, блок дотор зарлагдсан i хувьсагч хоёр нь ялгаатай хувьсагчид юм.

Рекурс

Хэрэглэгчийн функц нь шууд болон шууд бус аргаар өөрөө өөрийгөө дуудах боломж Си хэлэнд байдаг ба үүнийг рекурс гэж нэрлэдэг. Тоог тэмдэгт мөр болгон хэвлэдэг функц дээр жишээ авч үзье. Ингэж хэвлэхийн тулд тооны цифрүүдийг олж хэвлэх шаардлагатай ба цифрүүдийг олоход тэдгээр нь урвуу дарааллаар буюу хойноосоо эхэлж олддог. Иймээс тэдгээрийг зөв дараалалд оруулан хэвлэх шаардлагатай.

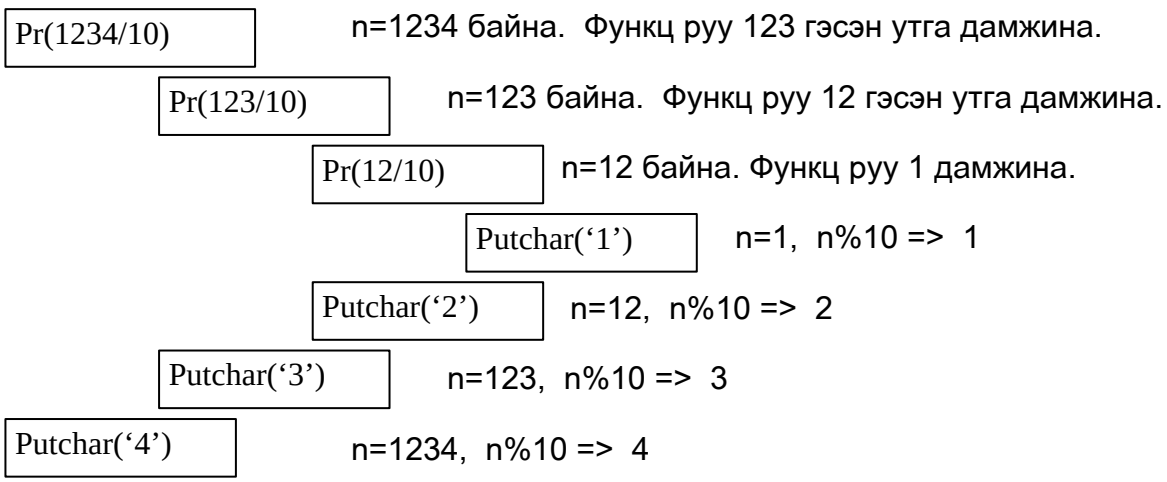
Үүнийг 2 аргаар гүйцэтгэж болно. Нэгд цифрүүдийг нь хадгалж аваад (массивт) эргүүлэн хэвлэх арга байж болно. Хоёрт рекурс функц бичиж болно.

```
#include <stdio.h>
void pr(long n)
{ if(n<0)
   { putchar('-');  n=-n; }

  if(n/10)  pr(n/10);

  putchar(n%10+'0');
}
main()
{ clrscr();
  pr(1234);         printf("\n");
  pr(7654321);   printf("\n");
  pr(-1234567); printf("\n");
}
```

Функц нь өөрийгөө дуудах тохиолдолд шинээр дуудагдах функц нь өөрийн бүх локаль хувьсагчдаа дахин анхны утгыг нь тогтоон үүсгэдэг ба энэ үүссэн хувьсагчид нь өмнөх функцын хувьсагчдаас ялгаатай юм. Хамгийн эхний дуудсан жишээг буюу pr(1234) –г тайлбарлая.



Үүнийг мөн алхамаар үзүүлбэл :

1. pr(1234);	8. if(1/10) /* Худал */
2. if(1234/10)	9. putchar(1%10+'0'); Функцээс буцах;

3.	pr(123);	10. putchar(12%10+'0'); Функцээс буцах;
4.	if(123/10)	11. putchar(123%10+'0'); Функцээс буцах;
5.	pr(12);	12. putchar(1234%10+'0'); Функцээс буцах;
6.	if(12/10)	13. РЕКУРС дуусав.
7.	pr(1);	

Дээр үзүүлсэн жишээ нь рекурсын нэг төрөл буюу функц нь шууд өөрийгөө дуудсан байна. Харин шууд бус аргаар рекурс үүсгэж болдог. Энэ нь тухайн функц нь өөрийгөө шууд дуудах бус түүний дуудсан өөр нэг функцээс тухайн функцыг дууддаг арга юм.

```
Func1()
{ . . .
    Func2()
   . . .
}

Func2()
{  . . .
   Func1()
   . . .
}
```

Æèøýýíä Func1 ôóíêö íü àæèëëàõ ÿâöääà Func2 ôóíêöûã äóóääàõ áà òýð ôóíêö íü ýðãýýä Func1 –èéã äóóääàõ çàìààð ðåêóðñûã ¿¿ñãýæ áàéíà.

Ингэж рекурс байдлаар функц бичих нь дараах ач холбогдолуудтай байдаг.

1. Программын кодчилолыг бага хэмжээтэй болгох
2. Бичих, ойлгоход хялбар болох
3. Мод гэх мэтийн шатласан зохион байгуулалттай бүтцүүдтэй ажиллах боломжийг бүрдүүлж өгдөг.

TYPEDEF–ээр төрөл тодорхойлох

Та энэ командыг ашиглан шинээр төрөл тодорхойлох боломжтой.

Бичигдэх хэлбэр : **typedef <Төрөл> <Шинэ төрөл>;**

Жишээ : **typedef float real;**

Энэ команд нь хаана зарлагдсанаас шалтгаалж ажиллах хүрээ нь тодорхойлогддог. Өөрөөр хэлбэл функц дотор зарлагдвал ажиллах хүрээ нь локаль, функцээс гадна зарлагдвал глобаль байна.

```
void ff()
{  typedef  short int  integer;  }
```

СЭДЭВ 10. МАССИВ

Программист нь ижил зорилгоор хэрэглэгдэх олон утгыг ашиглах, хадгалахдаа дараах байдлаар олон хувьсагчид зарлаж болно.

```
int name1=101;
int name2=202;
int name3=303;
```

Хувьсагчдын тоо ихсэх тохиолдолд тэдгээртэй ажиллахад маш их хүндрэлтэй болдог ба энэ тохиолдолд массив гэж нэрлэгдэх нийлмэл төрлийг ашигладаг.

Ижил төрлийн олон утгыг хадгалах боломжтой , өгөгдлийн нийлмэл төрлийг массив гэнэ.

Массив нь олон элементтэй матриц мэтээр төсөөлөгдөх бөгөөд элементэд нь хандахын тулд тусгай дугаараар хандана. Өөрөөр хэлбэл массивын элемент бүр тодорхой дугаартай байна. Си хэлэнд массивын дугаарлалт нь 0 гэсэн утгаас эхэлнэ.

Массивыг зарлахын тулд хувьсагчийн нэрийн ард [ ] буюу хос хаалт ашиглана. Хос хаалтанд тухайн массивын элементүүдийн тоог бичиж өгнө.

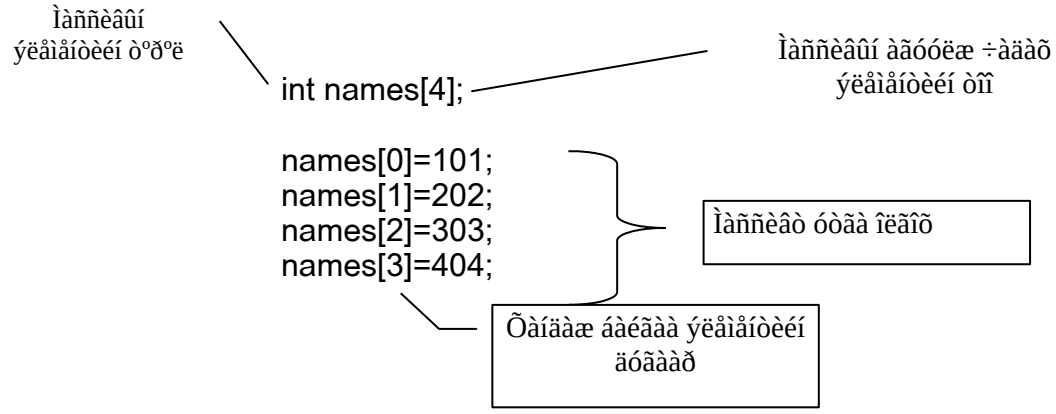
Нэг хэмжээст массивыг зарлах

Нэг хэмжээст массив гэдэг нь шугаман массив юм.

<төрөл> <нэр>[<индекс>]

Жишээ : int a[5];
int b[100];

Дээрх жишээг массиваар зарлавал :



Массивын элементэд хандахдаа [ ] хос хаалтыг ашиглана. names[2] гэж хандахад тухайн массивын 3-р элементэд хандана.

Массив санах ойд байрлах

Нэг хэмжээст массив нь санах ойд дараалсан байрлалтай байна.

int A[5] массив санах ойд байгаа нь :

A[0]	A[1]	A[2]	A[3]	A[4]	. . .	. . .

Хэрэв sizeof үйлдлийг ашиглан A-гийн хэмжээг олбол 10 гэсэн үр дүнг үзүүлнэ. Учир нь int төрөл нь 2 byte хэмжээтэй ба нийт 5 элементтэй тул 5*2=10 byte юм.

Массивын анхны утгыг олгон зарлах

Массивын зарлахдаа түүний элементүүдийн анхны утгыг олгож зарлаж болно. Массивын элементүүдийн анхны утгыг { } хос хаалтанд зарладаг.

```
int val[5]={ 10, 40, 70, 90, 120 };
```

Ингэж зарласан нь массивын элементүүдэд дараах байдлаар утга олгосонтой ижил юм.

```
val[0] = 10;
val[1] = 40;
val[2] = 70;
val[3] = 90;
val[4] = 120;
```

Эндээс **Си хэлэнд массивын элементүүдийг 0-ээс эхлэн дугаарладаг** нь харагдаж байна.

```
int a[5]={1,2,3,4,5};   /*  Эхний 5 элементэд утга олгох  */
int b[20]={0,1,2,3,4,5,6,7,8,9};   /* Эхний 10 элементэд нь утга олгож байна.
                                      Бусад элементүүд нь 0 утгатай болно. */
float money[10]={ 6.23, 2.45, 8.01, 2.97, 6.41 };
char grades[5]={‘A’,’B’,’C’,’D’,’\0’};
```

Жишээнд char төрлийн 5 элементтэй массивт анхны утгыг олгон зарлажээ. Үүнийг мөн дараах аргаар зарлаж болно.

```
char grades[5]=”ABCD”;
```

Массивыг зарлахдаа хос хаалтанд үргэлж элементийн тоог бичиж бай. Харин массивт анхны утгыг олгож зарлахдаа л тоог бичихгүй байж болно.

```
int ages[5]={ 5,27,40,65,92 };          /*  Зөв  */
int ages[];                             /*  Буруу */
int ages[] ={ 5,27,40,65,92 };          /*  Зөв  */
```

Хэрэв та массивын бүх элементийг 0-ээр дүүргэхийг хүсвэл, дараах бичлэгийг ашиглана.

```
float  A[100]={0.0};   /*  Бүх  элементийг тэглэх  */
```

Жишээнд массивын эхний нэг элементэд л утга олгож байна. Гэвч Си бусад элементийг нь 0-ээр дүүргэдэг. Өөрөөр хэлбэл массивын ядаж нэг элементэд анхны утгыг нь олговол Си бусад элементийг нь 0-ээр дүүргэдэг гэсэн үг.

Массивын элементэд утга олгох түүнийг илэрхийлэлд ашиглах

```
a[1]=20;  a[6]=a[2];  a[0]=i;  a[1]=b[i];

x=y+a[3]*a[i];    printf(“ 3-р элемент : %d”,a[3]);
```

Массивын элементэд утга олгох мөн элемент дээрх үйлдлүүд нь ердийн хувьсагчид утга олгох, илэрхийлэлд оролцох хэлбэртэй ижил болно.

Массивын элементүүдийг гараас унших

Массивын элементийг хэрэглэх нь хувьсагч ашиглах хэлбэртэй ижил учир элементүүдийг гараас уншихдаа дараах байдлаар уншина.

```
scanf(“%d”,&a[1]);
scanf(“%d”,&a[2]);
scanf(“%d”,&a[3]);
```

Массивын элементүүдийг гараас уншихдаа цикл ашиглах нь тохиромжтой.

```
int a[10], i;
for(i=0;i<10;i++)   scanf(“%d”,&a[i]);
```

Жишээнд циклийн тоолуур 0 -оос 9 хүртэлх утгыг авах бөгөөд scanf функц нь a[0]. . . a[9] элементүүдийн утгуудыг гараас уншина.

```
int a[10],i;
for(i=0;i<10;i++)
  {  printf(“ A[%d] =”,i);
     scanf(“%d”,&a[i]);
  }
```

Үр дүн нь :

```
A[0]= 101
A[1]= 122
. . .
A[9]= 222
```

Массивын элементүүдийг хэвлэх

Массивын зарим элементийг хэвлэхдээ дараах байдлаар хэвлэнэ.

```
printf(“ %d ”,a[0]);
printf(“ %d ”,a[1]);
printf(“ %d ”,a[2]);
```

Массивын бүх элементүүдийг хэвлэхдээ цикл ашиглах нь тохиромжтой.

```
int a[10], i;
for(i=0;i<10;i++)  printf(“ A[%d]=%d \n”, i, a[i]);
```

Үр дүн нь :

```
A[0]= 101
A[1]= 122
. . .
A[9]= 222
```

Хоёр хэмжээст массив

Си хэлэнд нэгээс олон хэмжээст массив зарлан ашиглах боломжтой байдаг. Олон хэмжээст массив дээр хэрхэн ажиллахыг 2 хэмжээст буюу хүснэгт маягийн

массив дээр авч үзье. Ийм массивын элементэд мөр, багана гэсэн 2 дугаараар хандана. Дугаарлах утга нь мөн л 0-ээс эхэлнэ.

А массив (3 мөр 4 баганатай)

A[0][0]	A[0][1]	A[0][2]	A[0][3]
A[1][0]	A[1][1]	A[1][2]	A[1][3]
A[2][0]	A[2][1]	A[2][2]	A[2][3]

2 хэмжээст массив санах ойд байрлах

A[0][0]	A[0][1]	A[0][2]	A[0][3]	A[1][0]	A[1][1]	A[1][2]	. . .

Өөрөөр хэлбэл :

1-р мөр	2-р мөр	3 мөр

Хоёр хэмжээст массивыг зарлах

<төрөл> <нэр>[<индекс>][<индекс>]
эсвэл
<төрөл> <нэр>[][<индекс>]

Жишээ :

```
int a[3][4];
int b[2][5];
```

Массивын анхны утгыг олгох

2 хэмжээст массивын анхны утгыг олгохдоо 1 хэмжээст массивын анхны утга олгодог байдалтай төстэй буюу энэ массивыг нэг хэмжээст массивуудын массив байдлаар ойлгохоор бичлэгтэй байдаг.

```
int a[3][2]={{1,2},{3,4},{5,6}};
int b[2][3]={{1,2,3},{4,5,6},{7,8,9}};
int A[3][3]={1,2,3,4,5,6,7,8,9};
```

2 хэмжээст массивын элементэд утга олгох түүнийг илэрхийлэлд ашиглах

```
a[1][1]=20;  a[6][2]=a[2][2];  a[0][2]=i;  a[1][1]=b[i][1];
x=y+a[3][1]*a[i][1];    printf("[1,3]-р элемент : %d",a[1][3]);
```

2 хэмжээст массивын элементүүдийг гараас унших

Массивын элементийг хэрэглэх нь хувьсагч ашиглах хэлбэртэй ижил учир элементүүдийг гараас уншихдаа дараах байдлаар уншина.

```
scanf(“%d”,&a[1][1]);
scanf(“%d”,&a[2][1]);
scanf(“%d”,&a[3][1]);
```

2 хэмжээст массивын элементүүдийг гараас уншихдаа давхар цикл ашиглах нь тохиромжтой. Нэг цикл нь мөрийн дугаарыг тодорхойлно. Нөгөө нь баганын дугаарыг тодорхойлно.

```
int a[2][2], i,j;
for(i=0; i<2; i++)
for(j=0; j<2; j++)
      { printf(“ A[%d][%d]=”,i,j);
        scanf(“%d”,&a[i][j]);
      }
```

Үр дүн нь :

```
A[0][0]= 101
A[0][1]= 122
A[1][0]= 122
A[1][1]= 222
```

2 хэмжээст массивын элементүүдийг хэвлэх

2 хэмжээст массивын элементүүдийг хэвлэхдээ мөн л давхар цикл ашиглах нь тохиромжтой. Нэг цикл нь мөрийн дугаарыг тодорхойлно. Нөгөө нь баганын дугаарыг тодорхойлно.

```
int a[2][2], i,j;
for(i=0; i<2; i++)
{  for(j=0; j<2; j++)  printf(“%7d”, a[i][j]);
   printf(“\n”);
}
```

Үр дүн нь :

```
101        122
122                200
```

Тэмдэгт мөр төрлийн хувьсагч

Тэмдэгт мөр нь ASCII системээр кодлогдсон тэмдэгтүүдийн дараалал юм.

Тэмдэгтүүд нь :
үсэг : ABC. . .XYZ, abc. . . xyz
цифр : 0123456789
тусгай тэмдэгт : + - * / | \ $ # . . .

Тэмдэгт мөр төрлийн хувьсагчийг Си хэлэнд **char** түлхүүр үгээр , массив хэлбэрээр зарлана.

char a[15];

Си хэлэнд тэмдэгт мөр хамгийн ихдээ 256 byte эзэлдэг. Дээр зарлагдсан a хувьсагч санах ойд хэдэн byte эзэлж буйг тодорхойлохдоо sizeof үйлдлийн тусламжтайгаар мэднэ.

Жишээ : printf(“ %d ”, sizeof(a));
Үр дүн : 15

Тэмдэгт мөрт дараах хэлбэрүүдээр анхны утга олгож зарладаг.

char a[8]=”HELLO”;
char Name[25]=”Batbold”;
char grades[5]={‘A’,’B’,’C’,’D’,’\0’};

Тэмдэгт мөрийг гараас уншихдаа & тэмдэгтийг scanf-д ашигладаггүй.

scanf(“%d”,a);

Си хэлэнд ‘\0’ тэмдэгт нь мөрийн төгсгөлийг заана. Дээрх жишээнд grades массивын 4-р элементэд ‘\0’ элементийг олгосон нь тухайн тэмдэгт мөр дуусч буйг тодорхойлж байна.

Тэмдэгт мөр төрлийн хувьсагчийг илэрхийлэлд ашиглах

Тэмдэгт мөр төрлийн хувьсагчдыг шууд илэрхийлэлд оролцуулах боломж байдаггүй. Өөрөөр хэлбэл тэмдэгт мөр төрөл нь олон элементээс бүрдэх нийлмэл төрөл учир нэг элемент нь л илэрхийлэлд оролцохоос биш бүхлээрээ илэрхийлэлд оролцдоггүй.

Дараах үйлдлүүд нь боломжгүй үйлдлүүд юм.

char a[15];
a=”Hello !!!”; a=a+” World”;

Иймд эдгээр үйлдлүүдийг функцээр гүйцэтгэдэг. Өөрөөр хэлбэл утга олгох үйлдэл, тэмдэгт мөр залгах үйлдлийг гүйцэтгэдэг функцууд байдаг.

Тэмдэгт мөр хувьсагчид утга олгох strcpy() функц

Бичигдэх хэлбэр : char *strcpy(char *s1, const char *s2);
/ s2 тэмдэгт мөрийн утгыг s1 тэмдэгт мөрт олгох /

Жишээ 1 :

```
#include<stdio.h>
char str[]="ПХ-ын 1",str1[]="ПХ-ын 2";
main()
{  strcpy(str,str1);        /*   str=str1   */
    printf(" Тэмдэгт мөр = %s \n",str);
}
```

Үр дүн : Тэмдэгт мөр = ПХ-ын 2

Жишээ 2 :

```
#include<stdio.h>
char str[]="ПХ-ын 1";
main()
{  strcpy(str,"ПХ-ын 2");
    printf(" Тэмдэгт мөр = %s \n",str);
}
```

Тэмдэгт мөрүүдийг залгах strcat функц

Бичигдэх хэлбэр : **char *strcat(char *s1, const char *s2);**

Жишээ :

```
#include <string.h>
char str[100],str1[100];
main()
{ printf(" str= "); scanf("%s",&str);    /*  ПХ-ын   */
   strcat(str,"2a");                      /* s1=s1+s2 */
   printf("%s\n",str);
}
```

Үр дүн : ПХ-ын 2a

Мөр төрлийн хувьсагчдыг харьцуулах нь

strcmp() функц

Бичигдэх хэлбэр : int strcmp(const char *s1, const char *s2);

Хэрэв буцаах утга нь :

==0 бол s1==s2 (Тэнцүү)
>0 бол s1>s2 (s1 нь их)
<0 бол s1<s2 (s2 нь их)

Жишээ :

```
. . .
i=strcmp("aac","aab");
switch(i)
```

```
{       case 0: printf("Тэнцүү тэмдэгт мөр"); break;
        default: printf("Ялгаатай тэмдэгт мөр");
}
. . .
```

Тэмдэгт мөрийн уртыг тодорхойлох

strlen функц

Бичигдэх хэлбэр : **int strlen(char *s);**

Тэмдэгт мөрийн уртыг тооцохдоо төгсгөлийн "\0" тэмдэгтийг үл тооцно.

Жишээ :

```
#include<string.h>
char str[]="ПХ-ын 1";
main()
{ int count;
  count=strlen(str);
  printf(" Тэмдэгт мөрийн урт = %d", count);
}
```

Үр дүн : Тэмдэгт мөрийн урт = 7

СЭДЭВ 12. БҮТЭЦ ТӨРӨЛ

Бүтэц төрөл нь утгын хувьд хоорондоо уялдаа холбоотой олон төрлийн хувьсагчдын олонлогоос тогтох нийлмэл төрөл юм.

Бүтэц гэж нэрлэгдэх өгөгдлийн хэлбэр нь янз бүрийн төрөлтэй өгөгдлүүдийг хамтатган тодорхойлж, хэрэглэгчдэд шинэ өгөгдлийн хэлбэрийг үүсгэхэд хэрэглэгдэнэ.

Бүтэц төрлийг тодорхойлох хэлбэр нь :

```
struct [<Бүтцийн нэр>] {
                [<төрөл1> <хувьсагчийн нэр1>]
                [<төрөл2> <хувьсагчийн нэр2>]
                [<төрөл3> <хувьсагчийн нэр3>]
                  . . . . .
                [<төрөл n> <хувьсагчийн нэрn>]
        };
```

Оюутны мэдээллийг хадгалах бүтэц үүсгэх жишээ авч үзье.

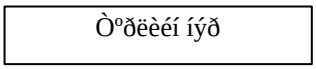
```
struct student {
                 char name[20];
                int age;
                float GPA;
         };
```

Бүтэц төрлийг тодорхойлохдоо struct түлхүүр үгийг ашигладаг. Бүтцэд агуулагдах хувьсагчдыг бүтцийн гишүүн гэнэ.

Бүтэц төрлийн хувьсагч зарлах хэлбэр :

struct <бүтцийн нэр> <хувьсагчийн нэр>;

Жишээ :



1. struct student sum; Õóâüñàã÷èéí íýð

2. struct student { char name[15];
int age;
}sum;

3. struct { char name[15];
int age;
}sum;

Бүтэц төрлийн хувьсагчид анхны утга олгох

Бүтэц төрлийн хувьсагчид дараах хэлбэрээр анхны утгыг олгоно.

struct student sum={ “Бат”, 25 };

Бүтцийн гишүүн элементэд хандах

Бүтэц доторхи элементэд хандахын тулд . тэмдгийг хэрэглэнэ.

student . age = 18;

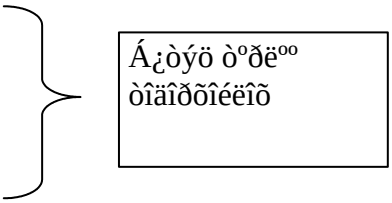
Бүтэц төрөлтэй массив

Бүтэц төрөлтэй массивыг зарлах хэлбэр :

struct <Бүтэц төрөл> <Массивын нэр>[хэмжээ] ;

Жишээ нь :

```
struct student {
        char name[20];
        int grade;
       };
```



struct student sw102[100]; Ìàññèâàà çàðëàõ

sw102[0].name	sw102[0].grade
sw102[1].name	sw102[1].grade
sw102[2].name	sw102[2].grade

sw102[3].name	sw102[3].grade
. . .	. . .
sw102[100].name	sw102[100].grade

Бүтэц төрлийн массив санах ойд байрлах

name1	grade1	Name2	grade2 . . .

Бүтэц төрөл ашигласан жишээ программ :

```
#include<stdio.h>
struct student {
                char name[20];
                int age;
               float GPA;
        };

struct student Who;       /*  Who нь student  бүтэцтэй  */

int i,j;

main()
{   printf(“ Оюутны нэр : “); scanf(“%s”,Who.name);
    printf(“ Оюутны нас : “); scanf(“%d”,&Who.age);
    printf(“ Оюутны дүн : “); scanf(“%f”,&Who.GPA);
    printf(“ Оюутны нэр :  %s \n”,Who.name);
    printf(“ Оюутны нас :  %d \n”,Who.age);
    printf(“ Оюутны дүн :  %f \n”,Who.GPA);
}
```

Бит талбар

Утгыг түүний авч болох хязгаарт нь тохируулж хадгалах зорилгоор бит талбар гэж нэрлэгдэх талбарыг ашигладаг.

Жишээ нь Оюутны хүйсийг тэмдэглэхэд 0 ба 1 гэсэн хоёр утга л хангалттай . 0 -эм , 1-эр гэж тооцоё. Тэгвэл бид түүнийг хадгалахдаа хамгийн багаар 1 byte хэмжээтэй хувьсагч буюу char төрлийн хувьсагч зарлана.

```
char Sex;
```

Тэгвэл зөвхөн 0 ба 1 гэсэн утгыг хадгалахын тулд 8 бит зарцуулж байна. Хэрвээ хувьсагчдыг битээр зарлаж болдог бол үүнд зөвхөн 1 л бит хангалттай.

Си хэлэнд ийм байдлаар бит талбарыг зарлах боломж байдаг.

Жишээ :

```
#include <stdio.h>

struct  AAAAA {
          int Sex : 1;                          /*  0..1   */
```

```
        int k : 2;                          /*  0..3   */
        int l : 5;                          /*  0..31*/
        int s : 2;                          /*  0..3   */
        int m : 6;                          /*  0..63  */
} AA;


main()
{
  AA.Sex=1;   printf(“%d\n”,AA.Sex);
  AA.k=3;      printf(“%d\n”,AA.k);
  AA.l=31;     printf(“%d\n”,AA.l);
  AA.s=3;      printf(“%d\n”,AA.s);
  AA.m=63;    printf(“%d\n”,AA.m);
  printf(“%u\n”,AA);
}
```

ENUM төрөл

Си хэлэнд олонлогийг enum түлхүүр үгийн тусламжтайгаар тодорхойлдог. Дараах жишээнд долоо хоногийн өдрүүдийг төлөөлсөн нэрүүдтэй Weekdays олонлогийг тодорхойлов.

Жишээ :

```
#include <stdio.h>

enum DAYS
   {  Dabaa=1, Migmar=2, Lhagva=3, Purev=4, Baasan=5,
      Bimba=6, Nim=0
   } Weekdays;

main()
{  int i;
  printf(“ Day = ”); scanf(“%d”,&i);
  switch(i)
   {   case Dabaa   : printf(“Даваа”);break;
       case Migmar : printf(“Мягмар”);break;
       case Lhagva  : printf(“Лхагва”);break;
       case Purev     : printf(“Пүрэв”);break;
       case Baasan    : printf(“Баасан”);break;
       case Bimba     : printf(“Бямба”);break;
       case Nim       : printf(“Ням”);break;
      default : printf(“Долоо хоногийн өдөр биш ”);
   }
}
```

UNION төрөл

Энэ төрлийг ашиглан ялгаатай төрлүүдийн хувьсагчдыг хугацааны өөр агшинд хадгалах хувьсагчийг зарладаг. Өөрөөр хэлбэл хугацааны өөр өөр агшинд тухайн үүрэнд өөр өөр төрлийн өгөгдлүүдийг хадгалах бололцоотой гэсэн үг.

Түр зуур утга хадгалах зорилгоор хэрэглэгдэх temp хувьсагч дээр жишээ авч үзье.

```
#include<stdio.h>
union TEMPERARY
  { int n;
    float m;
  }temp;
int i=10,j=20;
float k=10.1,l=20.1;
main()
{ temp.n=i; i=j; j=temp.n; printf("%d %d\n",i,j);
  temp.m=k; k=l; l=temp.m; printf("%f %f\n",k,l);
}
```

Энэ программыг харвал ердийн struct ашигласнаас нэг их ялгараад байх зүйлгүй байна. Өөрөөр хэлбэл үүнийг дараах хэлбэрээр бас бичиж болно.

```
#include<stdio.h>
struct TEMPERARY
  { int n;
    float m;
  }temp;
int i=10,j=20;
float k=10.1,l=20.1;
main()
{ temp.n=i; i=j; j=temp.n; printf("%d %d\n",i,j);
  temp.m=k; k=l; l=temp.m; printf("%f %f\n",k,l);
}
```

struct болон union –ны гол ялгаа нь санах ойг хэрхэн ашиглаж байгаа явдал юм.

Өмнөх жишээнд зарласан temp нэртэй хувьсагч нь санах ойд 2byte эзэлнэ. Учир нь union –ны гишүүн элементүүд санах ойг ээлжлэн, хамтарч эзэмшдэг. Харин сүүлийн жишээнд буй temp хувьсагч нь санах ойд 4byte эзэлнэ. Яагаад гэвэл энэ нь struct төрөл юм. Иймд union төрлийн гишүүн элементүүд нь санах ойг хамтарч эзэмшдэг.

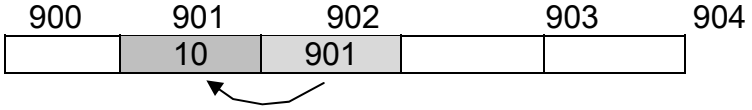
СЭДЭВ 13. ЗААГЧ ТӨРӨЛ

Заагч гэдэг бол обьектыг заах хувьсагч юм. Заагч төрлийн хувьсагчийг зарлахдаа өмнө нь (*) од тавина. Заагч хувьсагч нь санах ойд байрласан

хувьсагчийн хаягийг нь агуулна. Заагч хувьсагч ба ердийн хувьсагчийн ялгааг дараах жишээн дээр авч үзье.

```
int  Var=10;
int *pVar=&Var;
```

Дээрх хоёр хувьсагч нь санах ойд дараах байдлаар байрлана.



901- р хаяг нь Var хувьсагчийн байрлах санах ойн үүр
902- р хаяг нь pVar хувьсагчийн байрлах санах ойн үүр

Эндээс харвал ердийн хувьсагч нь утга агуулдаг, харин заагч хувьсагч нь санах ойд байрласан хувьсагчийн хаягийг нь агуулна. Ө.х тухайн хувьсагчийн утганд шууд бусаар хандах арга юм.

Заагч хувьсагч нь төрөлт ба төрөлт биш гэсэн хоёр янз байдаг.

Төрөлт заагч

Төрөлт заагч гэдэг нь тодорхой төрөлтэй хувьсагчийг заах заагчийг хэлнэ.

```
int  i=10;              /*  int төрлийн хувьсагч зарлах */
int  *k;        /*  int төрлийн заагч          */
```

Бүхэл тоон төрлийн заагч нь зөвхөн бүхэл тоон (int) төрлийн хувьсагчийг, обьектийг заана. Заагчид утга олгоогүй байх үед тэр ямар ч обьектийг заахгүй, харин түүнд ямар нэг хувьсагчийн хаягийг олгохдоо & үйлдлийг ашиглана.

Заагчид анхны утга олгох

Заагч хувьсагчид анхны утга олгохдоо дараах байдлаар олгоно.

```
int  i;
int *p=&i;
/* p заагчид i хувьсагчийн хаягийг олгох буюу i хувьсагчийг заалгах  */
```

Заагч хувьсагчийг ямар нэг обьектийг заалгахдаа дараах байдлаар заалгана.

```
int  i;
int *p;
p=&i;
/* p заагчид i хувьсагчийн хаягийг олгох буюу i хувьсагчийг заалгах  */
```

Заагчийн зааж байгаа обьектод хандах

Заагчийн зааж байгаа обьектод хандахдаа * тэмдгийг ашиглана.
Жишээ :

```
int  i=3,  *p;
p=&i;
/* p заагчид i хувьсагчийн хаягийг олгох буюу i хувьсагчийг заалгах  */

printf(" i =%d", *p);     /*  3 гэсэн утга хэвлэгдэнэ  */
*p=10;                    /* i=10 гэсэнтэй ижил */
```

```
printf(“ i=%d”, i);                /*  10 гэсэн утга хэвлэгдэнэ */
```

Дараах программд char төрөлт заагч ашигласан байна.

```
#include <stdio.h>
main()
{     char c = 'Q';
       char *ch = &c;
       printf("%c %c\n", c, *ch);
       c = 'Z';
       printf("%c %c\n", c, *ch);
       *ch= 'Y';
       printf("%c %c\n", c, *ch);
}
```

Заагч дээрх үйлдлүүд

1. Заагчийг олгох

Нэг заагчийг нөгөөд нь олгох боломжтой.
Жишээ нь :

```
        int i=100,*p1,*p2;
        p1=&i;                  /* i-г заалгах */
        p2=p1;
/* p1-ийн обьектийг заалгах  */

printf(“%d \n”,*p1);/* 100 -г хэвлэх */
printf(“%d \n”,*p2);/* 100 -г хэвлэх */
```

2. Заагчийг харьцуулах

Хоёр заагчийг хооронд нь харьцуулж болно.
Жишээ:

```
int x=2,y=1;
int *p1, *p2;

p1=&x;    p2=&y;
if(p1!=p2)
        printf(“2 заагч тэнцүү биш”);
else printf(“2 заагч тэнцүү”);
```

3. Арифметик үйлдлүүд

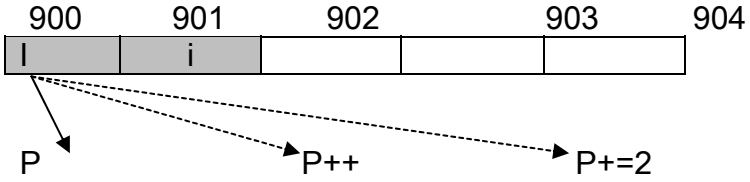
Заагч дээр нэмэгдүүлэх (++), хорогдуулах (--), нэмэх (+),
хасах (-) үйлдлүүдийг гүйцэтгэж болно. Төрөлт заагч ашиглаж байгаа үед эдгээр үйлдлүүдийг гүйцэтгэхэд заагчийн төрлийн хэмжээгээр нэмэгдэж, хорогдоно.

```
int i, *p1=&i;
p1++, p1--, p1+=2,  p1-=2 . . . гэх мэт үйлдэл хийж болно.

int *p1=&i;
p1++;          /*  Хаяг = хаяг + 2  */
p1--;          /*  Хаяг = хаяг - 2   */

p1+=2;         /*  Хаяг= хаяг + 4   */
p1-=2;/*  Хаяг= хаяг - 4   */
```

Дараах зурагт 900 –р хаягт байрлах i хувьсагчийг заасан p заагч дээр үйлдэл хийхэд заагч хэрхэн шилжихийг харуулав.



Заагч дээр хийгдэх эдгээр үйлдлүүд нь зөвхөн заагчийн агуулж байгаа хаягийг өөрчилнө. Түүнээс биш хаягт байгаа өгөгдөлд хандахгүй. Хаягт байгаа өгөгдлийг өөрчлөхийн тулд дараах байдлаар хандах хэрэгтэй.

```
int *p1=&i;

++*p1;         /*  i=i+1  */
 (*p1)++;              /*  i=i+1  */

*p1=*p1+2;             /*  i=i+2  */
*p1-=2;                /*  i=i-1   */
```

Жишээ :

```
#include <stdio.h>
main()
{       int i=10, *p1=&i;
printf("%d \n", ++*p1);      /*    11   */
printf("%d \n", i)             /*    11   */
}
```

Төрөлт заагч дээр үйлдэл хийхэд түүний хаяг нь төрлийнх нь хэмжээгээр нэмэгдэж буурдаг.

Жишээ :

```
#include <stdio.h>
main()
{       char c[10]="abababababab";
        int i, *p1=c;
        for(i=0;i<5;i++)
        printf("%c  ",*p1++);
}
```

Үр дүн:

```
a a a a a
```

Жишээ :

```
#include <stdio.h>
main()
{       char c[10]=”abababababab”;
        int i, *p1=c+1;
for(i=0;i<5;i++)
printf(“%c  ”,++*p1++);
}
```

Үр дүн:

```
c c c c c
```

Дараах үйлдлүүдийг заагчид зөвшөөрдөггүй.

- Хоёр заагчийг нэмэх
- Ялгаатай обьект дээрх 2 заагчийг хасах
- Бодит тоотой заагчийг нэмэх
- Заагчийг үржүүлэх
- Заагчийг хуваах

Заагчийг шилжүүлэх үйлдэл