

Ж.ПУРЭВ

ОБЪЕКТ ХАНДЛАГАТ  
ТЕХНОЛОГИЙН  
С++ ПРОГРАМЧЛАЛ

Үнэ: 15 000₮

ISBN 999295531-7



9 789992 955314

# ОБЪЕКТ ХАНДЛАГАТ ТЕХНОЛОГИЙН С++ ПРОГРАМЧЛАЛ

Ж.ПҮРЭВ  
А.АНГАР

- С/С++ ХЭЛНИЙ ГОЛ ОЙЛГОЛТ, ЖИШЭЭ ПРОГРАМ, ТАЙЛБАР ЗУРАГ
- С/С++ ХЭЛНИЙ ГОЛ КОМАНДЫН ЛАВЛАХ
- ОБЪЕКТ ХАНДЛАГАТ ПРОГРАМЧЛАЛЫН ГОЛ ОЙЛГОЛТ, ЖИШЭЭ ПРОГРАМ, ТАЙЛБАР ЗУРАГ
- СЭДЭВ БҮРТ МЭДЛЭГЭЭ ШАЛГАХ АСУУЛТ, ЖИШЭЭ БОДЛОГО, ПРОГРАМЧЛАХ БОДЛОГО
- НИЙТ 160 ОРЧИМ ТОМ ЖИЖИГ С++ ПРОГРАМ, 129 ТАЙЛБАР ЗУРАГ
- TURBO C++ ПРОГРАМЫН ТОВЧ ЗААВАР
- СТАНДАРТ ФУНКЦИЙН ЖАГСААЛТ, ХЭРЭГЛЭХ ЗААВАР
- ОБЪЕКТ ХАНДЛАГАТ ШИНЖИЛГЭЭ БА ЗОХИОМЖ, UML ХЭЛНИЙ ҮНДСЭН ОЙЛГОЛТ

ДИКИЙ  
ИЗДАЛ

**DDC**  
**005.262'077**  
**П-915**

*Монгол улсын ирээдүй баг үндэсний  
техники, технологийн дэвшилт байх болно.  
Энэхүү техники, технологийг бүтээлцэж  
монгол улсаа хөгжилт дэвшилт нийт хөтөлбөх  
чадварлаг монгол инженерийг төрүүлэх  
МУИС-даа зориулаа.*

**Зохиогч:** *Ж.Пүрэв*  
**Редактор:** *Ж.Пүрэв*  
**Д.Батсаа**  
**Хэвлэлд бэлдсэн:** *Ж.Пүрэв*  
**Цасны хэмжээ:** *190x240.мм*  
**Хэвлэлийн хэмжээ:** *32 х.х*  
**Номын зурсайг:** *К.Есенбек*  
**Хавтасны зурсайг:** *К.Есенбек*  
**Хэвлэлийн газар:** *Хадан Принтинг ХХК*

**USI books** © “USI Books” хэвлэлийн газраас эрхлэн хэвлэхүүлэв.  
*Ингэхүүгээр хэсэгчлэн авах, хүүхбартан олшируулахыг  
захирагчийн эрх гарчсонд тохиогно.*

**ISBN 978-99929-55-31-7**

## ТАЛАРХАЛ

Оюуны хөрөнгө оруулалт нь дээдийн дээд үр шим өгдөг.

An investment in knowledge pays the highest dividends.

-Ben Franklin

Энэ сурх бичгийг бичиж хэвлэлийн эхийг бэлдэх, хэвлэх ажилд тусалсан бүх хүнд чин сэтгэлийн талархлаа илэрхийль.

Ялангуяа сурх бичгийг хэвлүүлж иштийн хүртээл болгоход анхааран дэмжисж сэтгэл харамгуй туслацаа узүүлсэн Ц.Бүянцогтоо захиралтай USI ХХК-ийн хамт олонд гүнээ талархаж Монгол оронд мэдээллийн технологийг хөгжүүлэхэд чиглэсэн цаашдын бүйтэй үйлс, зорилго нь сэтгэлчэн бүтэж байхын ерөөл тавья.

Мон сурх бичгийн зурагийг зохиогчийн гаргасан эх зохиомжийн дагуу хичээл зүтгэл гаргаж зурсан К.Есенбект талархаж байна. Тэрбээр нүүр хавтасны зохиомжийг мон хийсэн юм.

Энэ сурх бичгийн эхийг анхааран ушииж элдэв алдаа мадгийг ариутган шүүсэн Д.Батгаад гүнээ талархаж байна.

2008 оны 7 дугаар сар 1

 Ж.Пүрэв

2008/12/26

## ГАРЧИГ

### НЭГДҮГЭЭР БҮЛЭГ

#### Объект хандлагат програмчлал, 3

- Объект хандлагат загварчлал, 3
  - Класс, объект гэж юу болох, 4
  - Битүүмжлэл, 6
  - Удамшил, 6
  - Полиморфизм буюу олон хэлбэршил, 7
  - Объект хоорондын уялдаа холбоо, 7
    - Холбоо, 8
    - Бүрдэл холбоо, 8
    - Ерөнхийлөөс нарийсгал, 9
    - Хамтын ажиллагаа, 9
  - Класс зохиомжлох ўйл явц, 9
    - Бодлогын шаардлага тогтоох, 9
    - Нэр үт классын нэр болох нь, 10
    - Үйл үг гишүүн функций болох нь, 10
    - Төсөөлөл зураглал ашиглан үйл үгийг олох, 10
    - Классын ажил үүргийг хуваарилах, 11
  - Объект хандлагат загварчлалын хэл, 11
  - Объект хандлагат программын шинж, 13

### ХОЁРДУГААР БҮЛЭГ

#### C++ хэлний тухай, 17

- C++ хэлний цагаан толгой, 18
- C++ хэлний бүтвэр, 18
- Өгөгдлийн торол, 19
- C++ программын бүтэц, 19
- Утга оноох оператор, 20
- Арифметик оператор, 20
- Харыцуулах оператор, 22
- Логик оператор, 22
- if-else команд, 23
- switch-case команд, 25
- for команд, 27
- while команд, 30
- do-while команд, 31
- Давхар давталт, 32
- Хүснэгт, 34
- Функци, 38
- Хувьсагч, түүний эзэмших ойн ангилал, 41
  - Automatic хувьсагч, 42
  - register хувьсагч, 42
  - Гадаад хувьсагч, 42

static хувьсагч, 43  
Бүтэц, 43  
    Бүтцийн тодорхойлолт, 44  
    Бүтцийн гишүүд рүү хандах, 45  
    Давхар бүтэц, 47  
    Бүтцэн хүснэгт, 48  
    Функц ба бүтэц, 50  
Хэрэглэгч тодорхойлох өгөглийн төрөл, 52  
    Бүтэц ба typedef, 52  
    Нэрлэсэн тогтмолон төрөл, 54  
    Тогтмол, 56  
Хаяг, хаяган хувьсагч, 56  
    Хаяган хувьсагч тодорхойлох, 58  
    Хаяган хувьсагчид утга оноох, 59  
    Хаяг ба тоон утга, 61  
    Дам хандалтын оператор, 61  
    Тогтмолын хаяг, тогмолон хаяг, 63  
    Хаягийн арифметик, 65  
    Хаяг ба хүснэгт, 66  
    Хаяг ба бүтэц, 68  
    Хаяг ба тэмдэгт мөр, 69  
    Хаяг ба тэмдэгт мөрийн хүснэгт, 70  
    Хаягийн хаяг, 70  
    void хаяг, 70  
    Функцийн хаяг, 71  
C++ хэлийг Си хэлтэй харьцуулах нь, 72  
    inline функц, 72  
        Макро, дотоод функц, 74  
    Заалтан хувьсагч, 75  
        Заалтан хувьсагч ба функц, 77  
        Бүтэц ба заалтан хувьсагч, 79  
        Заалт буцаах функц, 79  
    Динамик ойн оператор: new, delete, 80  
        new оператор, 80  
        new оператор ба хүснэгт, 82  
        delete оператор, 83  
        Ойн цоорхой, 85  
    Функцийн параметрт гаралын утга оноох, 86  
    Дахин тодорхойлсон функц, полиморф функц, 87

## ГУРАВДУГААР БҮЛЭГ

### Класс, объект, 95

Объект гэж юу болох, 95  
Класс гэж юу болох, 95  
Классын тодорхойлолт, 96  
Класс зарлах, 96  
Гишүүн өгөгдөл, 97  
Гишүүн функц, 97  
private/public гишүүд, 97  
Өгөгдөл далдлал, битүүмжлэл, 101

Объект байгуулах, 101  
Гишүүн өгөгдөл хандах, 102  
Гишүүн функц рүү хандах, 103  
Гишүүн функц тодорхойлох, 103  
Функц, объектын ялгаа, 106  
private, public гишүүн функц, 106  
Гишүүн функцийг гишүүн функцээс дуудах, 108  
Класс доторх хүснэгт, 109  
Объектон хүснэгт, 111  
Объект, функц, 113

## ДӨРӨВДҮГЭЭР БҮЛЭГ

### Байгуулагч, устгач функц, 121

Байгуулагч функцийн загвар, тодорхойлолт, 121  
Анхдагч байгуулагч, 125  
Байгуулагч функцийн шинж, 127  
Параметртэй байгуулагч, 128  
Гарааны утгыг нь аргумент хэлбэрээр объект руу дамжуулах, 128  
Гарааны утгатай байгуулагч, 130  
Хуулагч байгуулагч, 132  
Устгач функц, 137  
Устгач функцийн тодорхойлолт, 137  
Устгач функцийн шинж, 138  
Устгач функцийн хэрэглээ, 138  
Объектон хүснэгт ба байгуулагч функц, 140

## ТАВДУГААР БҮЛЭГ

### Функц дахин тодорхойлох, 147

Параметрийн тооны ялгаа ба дахин тодорхойлох функц, 147  
Функц зарлах, тодорхойлох, 148  
Параметрийн төрөл ба дахин тодорхойлох функц, 148  
Функцийн төрөл ба дахин тодорхойлох функц, 151  
Дахин тодорхойлох функц хэрэглэхэд хийгдэх үйлдэл, 152  
Загвар функц, загвар класс, 153  
Загвар функц, 153  
Загвар класс, 155

## ЗУРГААДУГААР БҮЛЭГ

### Удамших, удамшуулах, 161

Удамшлын тухай ойлголт, 161  
Эх класс, удамших класс, 162  
Удамшлын төрөл, 162  
Удамших классын тодорхойлолт, 165  
Private, public горимоор удамших класс, 166  
Дан удамшил, 170  
Харагдах байдал: private, public ба protected, 172  
Олон түвшингт удамшил, 174  
Олон-нэг буюу нийлмэл удамшил, 178  
Удамших, удамшуулах үйлдлийн давуу тал, 179  
Хийсвэр функц, 179

Удамшилын давхардлыг арилгах, 182  
Давхар класс, 183  
Удамшил ба байгуулагч функц, 186  
Удамшил ба устгагч функц, 193  
derived\_class (const base\_class &) байгуулагч, 193  
Функц дахин програмчлах, 197  
    Дахин тодорхойлсон функцийг дахин програмчлах, 200  
    Жинхэнэ хийсвэр функц ба хийсвэр класс, 200  
C++ класс тодорхойлоход анхаарах зүйл, 201

## ДОЛООДУГААР БҮЛЭГ

### Класс ба ойн менежмент, 207

Объектын хаяг, 207  
Класс ба new, delete оператор, 209  
Хаяган хувьсагчийг класс дотор хэрэглэх, 211  
    Хаяган ёгөгдлийн гардааны утгатай холбоотой үүсэх алдаа, 215  
    Хаяган ёгөгдлийн утгатай холбоотой үүсэх алдаа, 218  
    Хаяган ёгөгдлийн гардааны утгатай холбоотой үүсэх алдааг засах, 221  
    Хаяган ёгөгдлийн утгатай холбоотой үүсэх алдааг засах, 224  
this хувьсагч, 226  
Статич ёгөгдел, 228  
Статик функц, 231

## НАЙМДУГААР БҮЛЭГ

### Найз функц, оператор дахин тодорхойлох, 239

Найз функц, 239  
Найз класс, 247  
Оператор дахин тодорхойлох, 248  
    Ганц операндын операторыг дахин тодорхойлох, 249  
    Хос операндын операторыг дахин тодорхойлох, 253  
    Хоёр тэмдэгт мөрийг залгах, 256  
    Хоёр объектыг жиших, 257  
    Хоёр объектын тэнцүү эсэхийг шалгах, 259  
    Нэмээд буцааж утга оноох нийлмэл операторыг дахин тодорхойлох, 260  
Найз функц ба дахин тодорхойлох оператор, 261  
<< операторыг дахин тодорхойлох, 265  
Оператор дахин тодорхойлоход анхаарах зүйлс, 269  
Өгөгдөл хөрвүүлэг, 270  
    Суурь ба зохиомол төрөл хоорондын хувиргалт, 271  
    Зохиомол төрлийн объект хоорондын хувиргалт, 271

## ЕСДҮҮГЭЭР БҮЛЭГ

### Оролт гаралт ба файл, 281

Орох гарах урсгал, 281  
Оролт гаралтын буфер, 282  
iostream.h толгой файл, 282  
cout объект, 283  
<< оператор, 284  
void \* төрөл, 286  
Оролт гаралтын чиглэл өөрчлөх, 286

ostream арга put(), 287  
ostream арга write(), 288  
cout << хэрэглээ, 289  
Тооны суурь тоог өөрчилж хэвлэх, 290  
Талбарын өргөний тохируулга, 291  
Ашиглагдаагүй зайл дүүргэх, 292  
Бутархайн нарийвчлал, 292  
Утгат бус тэгийг хэвлэх, 293  
cin объект, 293  
Оруулгын оператор >>, 294  
cin >> нь гарын буферийг хэрхэн унших, 296  
istream арга: get(char &), 301  
istream арга: get(void), 304  
istream арга: get(char \*, int, char ='\n'), 305  
istream арга: getline(char \*, int, char ='\n'), 307  
istream арга: ignore(), 308  
istream арга: read(), 309  
istream арга: peek(), 310  
istream арга: gcount(), 311  
istream арга: putback(), 312  
Файлын оролт гаралт, 313  
fstream.h файл, 313  
Шинэ файл нээж мэдээлэл бичих, 314  
Файлаас мэдээлэл унших, 316  
Тэмдэгтийн оролт гаралт, 318  
Бичгийн файл, 319  
Хоёртын файл, 320  
Файлын горим, 322  
Файлыг унших бичихээр нээх, 323  
Файлын заагуур, 325  
seekg(), tellg() аргууд, 326  
seekp(), tellp() аргууд, 328  
Урсгал шалгах, 329  
Файлын төгсгөлийг хянах, 332  
Файлд нэмж бичих, 332  
Санамсаргүй хандалттай файл, 335  
Объектын мэдээллийг файлд бичих жишээ програм, 336  
Объектын мэдээллийг унших жишээ програм, 337  
Объектын мэдээллийг файлд бичиж унших жишээ програм, 339

Хавсралт А: C++ хэлний оператор, 345  
Хавсралт В: Turbo C++ тусгай угс, 347  
Хавсралт С: Онцгой тэмдэгт, 348  
Хавсралт Д: Том программыг зохион байгуулах, 349  
Хавсралт Е: Толгой файл, 353  
Хавсралт F: Зарим функцийн товч тайлбар, 356  
Хавсралт G: Turbo C++ програм, 363  
Хавсралт H: C++ командын товч лавлах, 373  
Ном зүй, 375  
Индекс, 376

## ЗУРГИЙН ЖАГСААЛТ

- Зураг 1.1: Классын бүтэц, 4  
Зураг 1.2: Уламжлалт програмчлалын арга дахь өвөгдөл ба кодын хамаарал, 4  
Зураг 1.3: Бодит объектын жишээ, 5  
Зураг 1.4: Файлын зориулалттай функцийн хоорондын хамаарал, 6  
Зураг 1.5: Машин, жолооч, зорчигч хоорондын холбоо, 8  
Зураг 1.6: Байшин, гэрийн бүрдэл холбоо, 9  
Зураг 1.7: UML объект диаграммын жишээ, 12  
Зураг 1.8: Сорилын программын хялбаршуулсан UML класс диаграммын жишээ, 12  
Зураг 1.9: Зочид буудлын программын хялбаршуулсан UML класс диаграммын жишээ, 12
- Зураг 2.1: *for* командын хийгдэх дараалал, 27  
Зураг 2.2: *while* командын хийгдэх дараалал, 30  
Зураг 2.3: *do-while* командын хийгдэх дараалал, 32  
Зураг 2.4: 10 бүхэл тоон бүтээртэй а хүснэгт, 34  
Зураг 2.5: а хүснэгтийн дүрслэл, 35  
Зураг 2.6: Ойн ангилал тогтоох загвар, 42  
Зураг 2.7: *sum* хувьсачийн эзэмших ойн дүрслэл, 46  
Зураг 2.8: *etr* хувьсачийн эзэмших ойн дүрслэл, 46  
Зураг 2.9: Ой руу хандах, 57  
Зураг 2.10: Program #2.21-д хэрэглэдэх ойн дүрслэл, 58  
Зураг 2.11: Ойн хаягийн флаггээрх хэлбэр, 58  
Зураг 2.12: Program #2.22-ын ойн дүрслэл, 60  
Зураг 2.13: Дам хандалтын операторыг хэрэглэх\_1, 62  
Зураг 2.14: Дам хандалтын операторыг хэрэглэх\_2, 62  
Зураг 2.15: *double* хаягийн арифметикийн жишээ, 65  
Зураг 2.16: *int* хаягийн арифметикийн жишээ, 66  
Зураг 2.17: *int sum[5]* хүснэгтийн ойн дүрслэл, 67  
Зураг 2.18: *char \*ptr = "Hello World";* командаын үр дүн, 69  
Зураг 2.19: Олон түвшний хаягийн хэрэглээ, 71  
Зураг 2.20: Ердийн Си функцийн ажиллагааны зарчим, 73  
Зураг 2.21: Дотоод функцийн ажиллагааны зарчим, 74  
Зураг 2.22: Хувьсагч, засалтан хувьсагч хоорондын харьцаа, 75  
Зураг 2.23: Засалтакар нь параметр дамжуулах, 78  
Зураг 2.24: Динамик ойц болдэх, 81  
Зураг 2.25: Дам хандалтын операторыг хэрэглэх нь, 81  
Зураг 2.26: *arrayPtr = new int[10];* командаар бэлдэх ойн дүрслэл, 82  
Зураг 2.27: Динамик хүснэгтийн бүтээрт хандах, 83  
Зураг 2.28: *new, delete* операторын уялдаа, 84  
Зураг 2.29: Давталтын *new int[10]* командаын үр дүн, 85  
Зураг 2.30: Хоёрдахь давталтын дараах байдал, 85  
Зураг 2.31: Гурвалдахь давталтын *new int[10]* командаын үр дүн, 85

- 
- Зураг 3.1:* Класс, объект хоорондын харьцаа, 96  
*Зураг 3.2:* employee классын тодорхойлолт, 97  
*Зураг 3.3:* employee классын тодорхойлолт, 98  
*Зураг 3.4:* Классын гаднаас хандах боломж, 99  
*Зураг 3.5:* employee класс ба `main()` функцийн хоорондын интерфейс, 101  
*Зураг 3.6:* ерөнхий объектын дүрслэл, 101  
*Зураг 3.7:* Объект тус бүрийн утга тусдаа ойд байрлана, 102  
*Зураг 3.8:* Классын гишүүн функцийг тодорхойлох загвар, 104
- Зураг 4.1:* Параметртэй байгуулагчаар байгуулсан объект, 129  
*Зураг 4.2:* Гарааны утга бүхий параметртэй байгуулагчаар байгуулах объект, 131
- Зураг 5.1:* Дахин тодорхойлсон функцийг олж дуудах дараалал, 152
- Зураг 6.1:* Нэг-нэг буюу энгийн удамшил, 162  
*Зураг 6.2:* Нэг-нэг удамшил, 163  
*Зураг 6.3:* Олон-нэг буюу нийлмэл удамшил, 163  
*Зураг 6.4:* Олон түвшинт удамшил, 163  
*Зураг 6.5:* Шаталсан удамшил, 164  
*Зураг 6.6:* Халимог удамшил, 165  
*Зураг 6.7:* Удамших классын тодорхойлолт, 165  
*Зураг 6.8:* Private удамшил, 166  
*Зураг 6.9:* Private удамшилын хандалтын хамаарал, 167  
*Зураг 6.10:* Public удамшил, 168  
*Зураг 6.11:* Public удамшилын хандалтын хамаарал, 169  
*Зураг 6.12:* Дан удамшил, 172  
*Зураг 6.13:* Олон түвшинт удамшил, 175  
*Зураг 6.14:* Объект хоорондын "is a" харьцаа, 183  
*Зураг 6.15:* Объект хоорондын "has a" харьцаа, 184  
*Зураг 6.16:* Удамших классын байгуулагчийн тодорхойлалтын хэсэг, 193
- Зураг 7.1:* Объект, хаяган хувьсагч хоорондын харьцаа, 207  
*Зураг 7.2:* Нэргүй объект, хаяган хувьсагч хоорондын харьцаа, 211  
*Зураг 7.3:* Гарааны утгатай нэргүй объект, хаяган хувьсагч хоорондын харьцаа, 211  
*Зураг 7.4:* employee төрлийн е объект, 212  
*Зураг 7.5:* Гишүүн огогдол нь хаяган хувьсагч бүхий е объектын бутэц, 214  
*Зураг 7.6:* Program #7.4-оор байгуулагдах объектууд; хаягийн утга нь захиомол, 217  
*Зураг 7.7:* е объект устсаны дараах байдал, 218  
*Зураг 7.8:* e2 объект устсаны дараах байдал, 218  
*Зураг 7.9:* Объектоор утга оноох үйлдлийн омнөх байдал; хаягийн утгууд захиомол, 220  
*Зураг 7.10:* Объектоор утга оноох үйлдлийн дараах байдал; 220  
*Зураг 7.11:* e объект устсаны дараах байдал; хаягийн утгууд захиомол, 221  
*Зураг 7.12:* e2 объект устсаны дараах байдал; хаягийн утга захиомол, 221  
*Зураг 7.13:* Бүх объект устсаны дараах байдал, 221  
*Зураг 7.14:* Хуулагч байгуулагчийг ажиллуулсны дараах байдал, 222  
*Зураг 7.15:* static гишүүн огогдол тодорхойлох загвар, 229  
*Зураг 7.16:* Ерөнхий статик number огогдол бүхий объектууд, 232

- Зураг 8.1:** Ганц операндын операторыг дахин тодорхойлох загвар, 250  
**Зураг 8.2:** Хос операндын операторыг дахин тодорхойлох загвар, 253  
**Зураг 8.3:** Дахин тодорхойлогдаж << оператор, 268  
**Зураг 8.4:** Нэг сонт объектыг олон string объекттой хэрэглэхэд үүсэх байдал, 268  
**Зураг 8.5:** Нэг сонт объектыг олон string объекттой хэрэглэх, 269
- Зураг 9.1:** Дэлгэцлэх үйлдэл, 282  
**Зураг 9.2:** Урсгал, класс хоорондын хамаарал, 283  
**Зураг 9.3** "Hello World" өгүүлбэрийг дэлгэцлэх байдал, 284  
**Зураг 9.4:** сош объектын ажиллах зарчим, 285  
**Зураг 9.5:** Олон << операторыг хэрэглэх боламж, 286  
**Зураг 9.6:** char \*ptr = "Hello World"; командын үр дүнгийн дүрслэл, 286  
**Зураг 9.7:** cout.put('a'); командын үр дүн, 287  
**Зураг 9.8:** cout.put('a').put('b').put('c'); командын ажиллах зарчим, 288  
**Зураг 9.9:** Program #9.3-ын эхний хоёр командын ажиллах зарчим, 289  
**Зураг 9.10:** cout.write(ptr1,13)<<"\n"; командын ажиллах зарчим, 289  
**Зураг 9.11:** Оруулалх үйлслийн дүрслэл, 294  
**Зураг 9.12:** cin >> name >> age >> height; командын үр дүн, 296  
**Зураг 9.13:** char name1[10] ба char name2[10] хүснэгтүүдийн дүрслэл, 300  
**Зураг 9.14:** cin.get(c1).get(c2).get(c3) >> c4 ; командын ажиллах зарчим, 303  
**Зураг 9.15:** cin.ignore(10, '\*').ignore(5, '\n') командын ажиллах зарчим, 309  
**Зураг 9.16:** cin.read(line1,20).read(line2,20); командын ажиллах зарчим, 310  
**Зураг 9.17:** Урсгалын классын шаталын будүүвч, 314  
**Зураг 9.18:** 0.375 тооны хоёртын дүрслэл, 320  
**Зураг 9.19:** 0.375 тооны тэмдэгтэн (ASCII) дүрслэл, 320  
**Зураг 9.20:** Program #9.47-ийн үр дүнгийн дүрслэл, 325  
**Зураг 9.21:** Орох, гарах файлтай холбогдох заагууруудын дүрслэл, 325  
**Зураг 9.22:** ofstream fout("test.dat"), холбогдох файлын заагуур, 326  
**Зураг 9.23:** ifstream fin("test.dat"), холбогдох файлын заагуур, 326  
**Зураг 9.24:** ofstream fout("test.dat", ios::app), холбогдох файлын заагуур, 326  
**Зураг 9.25:** fstream finout("test.dat", ios::in | ios::out), холбогдох файлын заагуур, 326  
**Зураг 9.26:** fstream finout("test.dat", ios::ate), холбогдох файлын заагуур, 326  
**Зураг 9.27:** Файлын заагуурыг удирдах функцийг хэрэглэх жишээ, 327  
**Зураг 9.28:** Program #9.49-ийн үр дүн, 329  
**Зураг 9.29:** Төловийн байт, түүний битүүдийн зориулалт, 330  
**Зураг 9.30:** Төловийн байт, 331  
**Зураг 9.31:** Төловийн байтын зарим битийн утга, 331
- Зураг D.1:** Толгой файл хэрэглэх, 349  
**Зураг D.2:** Препроцессорын команд хийгдсэний дараах file.cpp файлын бүтэц, 349  
**Зураг D.3:** header1.h файл, 350  
**Зураг D.4:** header2.h файл, 350  
**Зураг D.5:** file.cpp файл, 350  
**Зураг D.6:** Препроцессорын командууд хийгдсэний дараах байдал, 350  
**Зураг D.7:** header1.h файл, 351  
**Зураг D.8:** Препроцессорын команд хийгдсэний дараах байдал\_1, 351  
**Зураг D.9:** header2.h файлын бүтэц\_1, 351  
**Зураг D.10:** header2.h файлын бүтэц\_2, 352  
**Зураг D.11:** Препроцессорын команд хийгдсэний дараах байдал\_2, 352

## **ХҮСНЭГТИЙН ЖАГСААЛТ**

- Хүснэгт 2.1: C++ хэлний стандарт огогдлийн төрөл, 19  
Хүснэгт 2.2: C++ хэлний нийлмэл операторын жишээ, 21  
Хүснэгт 2.3: C++ хэлний арифметик операторын жагсаалт, 21  
Хүснэгт 2.4: Харьцуулах операторын жагсаалт, 22  
Хүснэгт 2.5: Логик операторын жагсаалт, 23  
Хүснэгт 2.6: Дотор давталтын уед k, т хувьсагчуудын авах утса, 35  
Хүснэгт 2.7: Функцийн параметр ба аргумент хоорондын ялгаа, 40  
Хүснэгт 2.8: Хувьсагч ба ойн ангилац, 43
- Хүснэгт 3.1: Бүтэц, класс хоорондын ялгаа, 99
- Хүснэгт 6.1: Класс доторх хандалтын мүж, 161  
Хүснэгт 6.2: Private горимоор удамших классын хандалтын горим, 167  
Хүснэгт 6.3: Private горимоор удамших классын харагдах байдал, 167  
Хүснэгт 6.4: Public горимоор удамших классын хандалтын горим, 168  
Хүснэгт 6.5: Public горимоор удамших классын харагдах байдал, 168  
Хүснэгт 6.6: Удамших классын хандалтын горим, 174  
Хүснэгт 6.7: Удамших классын харагдах байдал, 174
- Хүснэгт 9.1: open() функцийн авах аргумент, 323
- Хүснэгт A.1: C++ хэлний операторын жагсаалт, 345

## ТОВЧИЛСОН ҮГС

OXII		Объект хандлагат програмчлал
ANSI	American National Standard Institution	Америкийн үндэсний стандартын хүрээлэн
BCPL	Basic Combined Programming Language	Програмчлалыйн хэлний компайлэр хөгжүүлэхэд хэрэглэх хэл
CR	carriage return	Морийн эх рүү шилжих
DEC	Digital Equipment Corporation	
DOS	Disk Operating System	Дискийн үйлдлийн систем
eof	end of file	Файлын тогсгол
I/O	Input/Output	Оролт/Гаралт
LF	line feed	Мор хоөх
MFC	Microsoft Foundation Class	Майкрософт классын сан
OO	Object-oriented	Объект хандлагат
OOP	Object-Oriented Programming	Объект хандлагат програмчлал
OS	Operating System	Үйлдлийн систем
PARC	Palo Alto Research Center	
PC	Personal Computer	Компьютер
PDP	Programmable Data Processor	
UML	Unified Modelling Language	Нэгдмэл загварчлалын хэл

## **ЗАРИМ АНГЛИ НЭР ТОМЬЁОНЫ МОНГОЛ ОРЧУУЛГА**

aggregation	бүрдэл холбоо
association	холбоо
accessor	хандагч
abstract class	хийсвэр класс, логик класс
attribute	шинж тэмдэг
behavior	үйл хөдлөл
class	класс, англи, ангилал
compiler	компайлэр, хөрвүүтийч програм
composition	ноги холбоо, ногцолбор
concrete class	бодит(ой) класс
constructor	байгуулагч
containership	класс зооворлолт
copy	хуулагч
data abstraction	өгөгдөл хийсвэрлэл
data field	өгөгдлийн талбар
data hiding	өгөгдөл далдзал
data member	өгөгдлийн гишүүн
data security	өгөгдөл нууцлал
debugger	дебагер, програм зүгшрүүтийч
default	анхдагч, гарцааны
destructor	устсагч
directive	заалт команд
double	давхар нарийвчлалт бодит тоо
editor	эдитор, бичүүр, засуур програм
encapsulation	битүүмжлэл
field	талбар, үзүүлэлт
friend class	найз класс
friend function	найз функци
inheritance	удамшил
inheritance path	удамшлын зам, замнал
inline	дотоод мөр функци, дотоод функци
interaction	хамтын ажиллагаа
generalization specialization	срөнхийллөөс нарийсгал
keyword	түлхүүр үг, тусгай үг

ifecycle	орших хугацаа
library	сан
modifier	өөрчлөгч
modifying function	өөрчлөх функц
member data	гишүүн өгөгдөл
member functions	гишүүн функц
memory leaks	оин цоорхой
method overloading	арга дахин тодорхойлох
method overriding	арга дахин програмчлах
mutator	хувиргач
nesting of classes	давхар класс
object	объект, оногдохуун
object-oriented	объект хандлагат
operator overloading	оператор дахин тодорхойлох
overflow	хэтрэлт
parameter	хэмжигдэхүүн, параметр
parametrized	параметржсэн
polymorphism	олон хэлбэршил, олон хэлбэр
postfix	дагавар
prefix	үгтвар
pressure	дааралт
principal amount	нийт дүн
protected	хязгаарлагдмал
private	хувийн, хаагдмал
public	нийтийн, илэлттэй
pure abstract function	жинхэнэ хийсвэр функц
record	бичлэг
relationship	шүтэлцээ, уялдаа холбоо
reserved keywords	нооц түлхүүр үгс
scenarios	тосөөлөл зураглал, дүр зураг
signed	тэмдэгтэй
stream	урсгал
template	загвар
token	бүтвэр, элемент
type	терөл
unsigned	тэмдэггүй

## ӨМНӨХ ҮГ

Үншигч та энэхүү омнох үгийг хичээнгүйлэн уншиж байгаа аваас Си хэлийг судлахыг, суралхыг хүсч байгаа хэрэг. Та зөв сонголт хийжээ. Янз бүрийн програмчлалын хэл байх ч Си хэл бол хурд (Java програмаас 10-20 дахин хурдтай гэж үздэг.), уян хатан чанар, хүчирхэг байдаг зэрэг олон шинжээрээ миний мэдэх Алгол (оюутан байхдаа анх заалгасан хэл). Фортран, Кобол, ПЛ/1, Ассемблер зэрэг хэлээс давуу хэл мөн. Энэ хэл ихэвчтэн мэргэжлийн програм зохиогч хэрэглэгдэг хэл бөгөөд компьютерийн үйлдлийн систем, тохиоромж удирдах програм гэх мэтийн системийн програм, үйлдвэрлэлийн процессын удирдлагын систем, инженерийн тооцооны систем, тоглоомын програм, машин орчуулгын програм гээд олон арван зүйлийн програмын системийг зөвхөн уг хэрэгжүүлж бичдэг гэдэгтэй уншигч та ч санал нийлэх биз ээ...

Танд толилтуулж буй энэ сурал бичгийг 1990-ээд оны үеэс эдүгээ хүртэлээ сэдэвтэй холбогдох жишээ програм, зураг орсон хэрэглэж "Мэдлээч шалгах асуулт", "Жишээ бодлого", "Програмлах бодлого" зэрэг нь бүлэг бүрийн тогсголд нь хадаж өгсөн байгаа. Нийт 160 орчим том жижиг програм бичиж Turbo C++ version 3.0 дээр шалгаж алдааг засаж зөв болгосноо сурал бичигт оруулсан тул тэдгээрийг ажилтуулж судлах замаар програмчлалын ур чадвараа дээшилүүлэх боломжтой юм. Мөн хавсралтанд байгаатай нийлээд 129 тайлбар зураг, 18 хүснэгт хэрэглэсэн.

Объект хандлагат технологи, C++ програмчлалын нэр томъёог монгол хэлээ буулгаж хэрэглэхийг хичээсний хамт зарим оновчгүй гэсэн газраа мон англиар бичиж өглөө. Turbo C++ програм нь кирилл үсгийг дэмжиж ажиллаж чаддаггүй тул жишээ програмын тайлбар, тэдгээрт хэрэглэсэн уг, өгүүлбэрийг англиар бичлээ.

Сурал бичиг бүтцийн хувьд 9 бүлэг, 8 хавсралттай. Нэгдүгээр бүлэгт объект хандлагат програмчлал, объект хандлагат шинжилгээ ба зохиомж, UML хэлний тухай төвч ойлголт; хоёрдугаар бүлэгт C++ хэлний тухай; гуравдугаар бүлэгт класс объект; дөрөвдүгээр бүлэгт байгуулагч, устгагч функц; тавдугаар бүлэгт функц дахин тодорхойлох; зургаадугаар бүлэгт

удамших, удамишуулах; долоодугаар бүлэгт класс ба ойн менежмент; паймдугаар бүлэгт наиз функц, оператор дахин тодорхойлох; есдүгээр бүлэгт оролт гаралт ба файл гэсэн сэдвүүдийг орууллаа. Мөн хавсралтын хэсэгт C++ хэлний оператор, Turbo C++ тусгай үсс, онцгой тэмдэгт, том программыг зохион байгуулах, толгой файл, зарим функцийн товч тайлбар, Turbo C++ програм хийгээд C++ командын товч лавлах зэргийг дэлгэрэнгүй оруулсан.

Сурах бичиг нь Си хэлний суурь мэдлэгтэй хумууст зориулагдсан хэдий ч эл хэлний үндсэн мэдлэгийг "C++ хэлний тухай" хоёрдугаар бүлгээс олж авч болно. Мөн энэхүү сурах бичгийг бүрэн гүйцэд үзэж, жишээ програмуудыг нягтлан судалж, улмаар эдгээр материалынхаа ойлгоцыг бие дааж шалгах,olson мэдлэгээ батжуулах зорилгоор мэдлэгээ шалгах асуулт, жишээ бодлого, програмчлах бодлого зэргийг сайтар хийхийг зөвлөж байна.

Номын талаарх шинэ санаа, зөвлөгөө, санал, шуумжийг МУИС, МТС-ийн Компьютер, мэдээллийн технологийн тэнхимд хүлээн авна.

Зохиогч

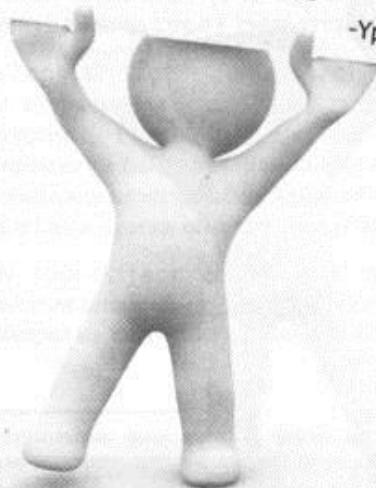
2008 оны 7 дугаар сарын 1

# НЭГДҮГЭЭР БҮЛЭГ

## ОБЪЕКТ ХАНДЛАГАТ ПРОГРАМЧЛАЛ

- Объект хандлагат загварчлал, 3
  - Класс, объект гэж юу болох, 4
  - Битуумжлэл, 6
  - Удамшил, 6
- Полиморфизм буюу олон хэлбэршил, 7
  - Холбоо, 8
  - Бурдмэл холбоо, 8
  - Ерэнхийллөөс нарийсгал, 9
  - Хамтын ажиллагаа, 9
- Класс зохиомжлох үйл явц, 9
  - Боллогын шаардлага тогтоох, 9
  - Нэр үг классын нэр болох нь, 10

-Үргэлжлэл- →



## ТӨГРӨГЧИЙН ТӨГРӨГЧИЙН

### -Үргэлжлэл-

- Үйл үг гишүүн функц болох нь, 10
- Төсөөлөл зураг ашиглан үйл үгийг олох, 10
- Классын ажил үүргийг хуваарилах, 11
- Объект хандлагат загварчлалын хэл, 11
- Объект хандлагат програмын шинж, 13



## НЭГДҮГЭЭР БҮЛЭГ

### ОБЪЕКТ ХАНДЛАГАТ ПРОГРАМЧЛАЛ

Компьютерийн технологид гарч байгаа шинэ нээлт, шинэ санаа, шинэ зохион бүтээх ажлын үр дүнд компьютерийн тооцох чадвар улам нэмэгдэж байгаатай холбоотойгоор компьютерээр хийлгэх, хийх ажлын цар хүрээ улам тэлж урьд нь боломжгүй байсан олон арван бодлогыг бodoх боломжтой болсоор байна. Хүнд хөнгөний хувьд олон янз байх тэдгээр бодлогод зориулж том хэмжээний програм бичих шаардлага байнга гардаг. Гэтэл том программыг бичих, уншиж ойлгох, туршиж зүгшруулэхэд амаргүй байдаг тул үүнийг хөнгөвчлөх үүднээс програмчлалын технологийг байнга хөгжүүлж түүнд олон шинэ боломж нэмж оруулж байдгийн нэг нь OOP (Object-Oriented-Programming) буюу объект хандлагат програмчлал (OХП)-ын технологи юм. Үүнийг бас объект хандлагат технологи гэж болно.

Уlamжлалт<sup>1</sup> програмчлалын арга хэрэглэх тохиолдолд өгөгдлийн бүтцэд орох өөрчлөлтийг тухай бүрд нь программын кодод тусгаж өгөх шаардлага байнга гардаг. Гэтэл программын хэмжээ том, бас олон модультай<sup>2</sup> бол энэ нь ихээхэн хөдөлмөр, цаг хугацаа шаардах нүсэр ажил болдог. Үүнтэй төстэй бэрхшээл программыг олон хүн хувааж бичих үед бас үүсч болно. Энэ нь шинэ арга, технологи бий болгоход хүргэсэн нь ОХП-ын технологи юм. Программын технологид гарсан эргэлт гэж үздэг эл технологи бол өмнөх уlamжлалт аргуудын сайн шинжийг өөртөө шингээж сүл талыг нь арилгасан арга юм.

Програм өөр хоорондоо тодорхой хамаарлтай бүлэг объектоос тогтдог гэх үндэслэлийг ОХП-д хэрэглэх тул бодлогыг модульд биш харин объектод хувааж үздэг. Програмчлах объект, түүнтэй холбоотой ажил үйлчилгээ, хоорондын хамаарлыг нь загварчлах боломжийг энэхүү програмчлалын шинэ технологи хангадаг байна.

#### 1.1 Объект хандлагат загварчлал

Класс (англ. ангилал) гэдэг үгийг гол төлөв нийтлэг үйл хөдлөл (үйлдэл), холбоо хамаарал, утга учирт зорилготой, ижил шинж тэмдэг бүхий бүлэг объектыг дүрслэхэд хэрэглэдэг. Жишээ нь, эдийн засагчид өрх гэрийг орлогоор нь бага, дунд, их орлоготой гэж ангилдаг. Ийм маягаар хэрэглэх класс гэдэг уг нь өөр хоорондоо уялдаа холбоотой бүлэг объектын сөрөнхий нэр болох тул төрөл (type) гэдэг үгтэй ижил утга санааг илэрхийлдэг.

C++ хэлний int, long, unsigned, double, char гэх зэрэг төрлийг сөрөнхийд нь built-in буюу дотоод суурь төрөл гэх бөгөөд тэдгээрийг тоон болон тэмдэгтэн объектыг загварчлахад хэрэглэдэг. Гэвч дотоод төрөл, жишээ нь хүний нэр мэтийн зүйлийг загварчлахад тохиromжгүй байдаг нь хүний нэр олон тэмдэгтээс тогтдог байхад char нь зөвхөн нэг тэмдэгтийн хэмжээстэй байдагтай холбоотой юм. Ийм бодлогын хувьд өгөгдлийн шинэ төрөл болох class үүсгэж объект дүрслэх боломжтой юм.

Объект хандлагат (OO-Object oriented) програм нь классаас тогтоно. Класс бүр модуль болно. Классын шинж тэмдэг нь объектын шинж буюу төлөв байдал, үйл хөдлөл нь функций болно. Хамаарал нь класс хоорондын, объект хоорондын холбоог заана.

<sup>1</sup> Объект хандлагат технологиос өмнө хэрэглэж байсан алгоритман, процедурен, модуль буюу бүтцийн арга, технологийг нийгдэж нь уlamжлалт арга гэж иоршилээ.

<sup>2</sup> Модулийн програмчлалд программыг модуль гэх бие даасан хэсгүүдэд хувааж бичдог.

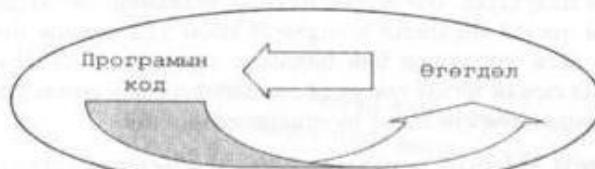
Класс бол биднийг хүрээлэн байгаа юмс, үзэгдлийн хийсвэр загварчлал юм. ОО програмын хувьд класс (class), объект (object), битүүмжлэл (encapsulation), удашил (inheritance), олон хэлбэршил (polymorphism), өгөгдөл далдлал (data hiding), холбоо хамаарал (relationship) гэсэн сурь ойлголтууд байдаг.

### 1.1.1 Класс, объект гэж иштэх болох

Класс бол програмчлалын хэлний ойлголт юм. Түүнийг объект үүсгэхэд загвар болгож хэрэглэдэг. Класс нь ОО програмын гол материал, барилгатай зүйрэзвэл түүний үндсэн түүхий эд болох тоосго юм. C++ класс нь объектын төлов байдал болон үйл хөдлөлийг битүүмжилсэн нэгэн хийсвэр нэгж болгон тодорхойлдог. Классын төлов байдал физик шинжээс нь хамаарна. Классын үйл хөдлөл нь классын хийдэг зүйл юм. Ер нь, үйл хөдлөл үргэлж үйл үзэр тодорхойлогддог. Жишээ нь,

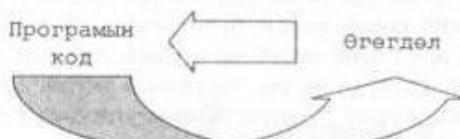
Cats eat, sleep, and play.  
Муур идэж, унтаж бас тоглодог.

Харин ОО програмд классын үйл хөдлөлийг функцийн тодорхойлно. Тэгэхлээр, класс нь түүний объектууд дундаа хэрэглэх төлов байдлын өгөгдэл, түүнийг боловсруулах функцийн<sup>3</sup> нэгдэл болно (Зураг 1.1).



Зураг 1.1: Классын бутец

ОХП нь өгөгдөл, түүнийг боловсруулах аргыг нэгэн нэгдмэл зүйлийн хэлбэрээр авч үзүүлэхээс тэдгээрийг салангид авч үздэг уламжлалт програмчлалын аргаас ялгаатай (Зураг 1.2).



Зураг 1.2: Уламжлалт програмчлалын арга дахь өгөгдөл ба кодын хамаарал

Объект гэж эд юм, нэгж зүйл, нэр үг, шилж сонгож авах ямар нэгэн юм, дүрслэн бодож болох өөрийн гэсэн ялгах шинж тэмдэг бүхий аливаа юмыг хэлнэ. Объект бол тухайн классын нэг жишээ, нэг тохиолдол юм. Зарим объект амьд, зарим нь амьд бишээр байдаг. Мөн объект бодитойгоор оршихоос гадна хийсвэрээр байж болно. Машин тэрэг, хүн амьтан, байшин барилга, нохой, шувуу, ном, дэвтэр, мод гээд бидний эргэн тойронд байдаг зүйлс цөм объект юм (Зураг 1.3).

Объект нь нэртэй, тодорхой утга бүхий шинж тэмдэгтэй байна. Машинд загварын нэр, үйлдвэрлэсэн он, өнгө; нохойд үүлдэр, нас, өнгө гэх мэт шинж заавал байдаг. Объект бас үйл

<sup>3</sup> Зарим сурх бичигт "арга" гэдгээ нэр томъёог хэрэглэдэг. Арга, функции хоёрыг бие биэс харилсан орлож чадахуулж иж ойлголт гэж үзж болох ч зохиомжийн түвшинд арга, кодыны түвшинд функци гэдгээ ойлголт дүйнээ гэж үзлээ.

хөдлөлтэй. Машин нэг газраас нөгөө рүү явдаг бол нохой хуцдаг. Хүн овог нэр, төрсөн огноо болон хаягаа мэдэж байхаас гадна нэрээ, хэдэн настайгаа хэлж чадна. Объект нь бусад объекттой тодорхой шүтэлэцэнд оршино.

ОО програмд аливаа объектыг код руу буулгаж дурслэнэ. Суудлын машины загварчлалын програм бичих бол машин бүрд хамаатай барааны нэр (брэнд), загварын нэр, үйлдвэрлэсэн он, онгө зэрэг нийтлэг шинжийг агуулах *Car* классыг эхлээд тодорхойлно. Ийм шинж бүхий машин бүр, жишээ нь 1999 онд үйлдвэрлэсэн улаан онгийн BMW 525i машин бол *Car* классын нэг тохиолдол, нэг объект болно. Тэгэхлээр класс бол объект үүсгэхэд хэрэглэх логик загвар юм. Машины мэдээллийг түүнд харгалзах *Car* объектын хувьсагч руу хадгалах ба ийм хувьсагчийг *data member* буюу өгөгдлийн гишүүн, *member data* буюу гишүүн өгөгдэл, *data field* буюу өгөгдлийн талбар, товчоор өгөгдөл гэнэ.



Зураг 1.3: Бодит объектын жишээ

*Car* объектын кодыг бичих үед системийн бусад хэсгийг түр хойш тавьж зөвхөн машинтай холбоотой хэсэгт анхаарал төвлөрүүлэх боломж нь програм бичих ажлыг хялбар болгодог. Дараа нь *Car* объект хэрэглэх түлшиний хэмжээгээ мэдэж байх шаардлага гарвал энэ тухай мэдээлэл болон холбогдох үйл хөдлөлийг нь *Car* классын код руу нэмж оруулна. Мен *Car* объектын түлшиний хэмжээг хэрэглэх кодын хэсгийг нь зөвхөн өөрчлөх хэрэгтэй болдог. Бусад кодын хэсэг өөрчлөгдөхгүй хэвээр үлдэнэ. Өөрчлөлт хийх хэсгийн байршлыг олж тогтоо болон хялбар өөрчлөх боломж зэрэг нь ОО програмын чухал шинжийн нэг юм.

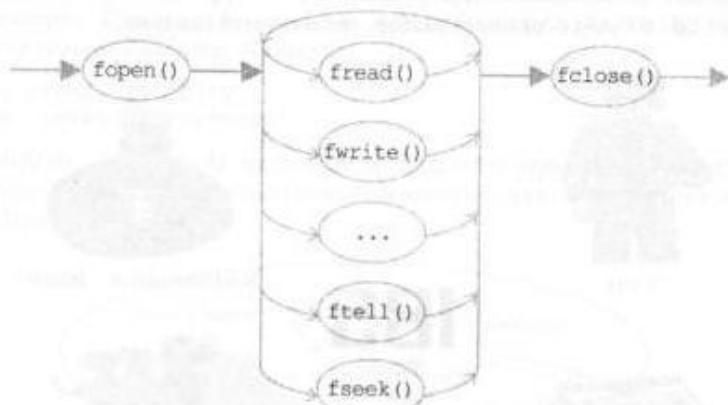
Машины шинж тэмдэг тус бүрийн хувьд хийх олон үйлдэл, боловсруулах олон арга байна. Жишээ нь, тухайн машинтай хамаатай бүх мэдээлэл хэрэгтэй бол түүний өгөгдөл бүрийн утгыг хэвлэх функцийг хэрэглэнэ.

Класс ба объект хоорондын харьцааг заах өөр жишээ үзье. Си (С гэдгийг дуудлагаар нь Си гэж нэршилээ.) нь *fopen()*, *fwrite()*, *fclose()*, *fseek()*, *fclose()* гэх мэт файлын зориулалт бүхий олон функцийн програмчлалын хэл юм. Си програм дотор дискийн файл нээхдээ *FILE*<sup>4</sup> төрлийн ой бэлдэж нээх файлтай холбоод уг файлтай хамаатай мэдээллийг түүнд хадгалж хаягийг нь буцааж өгөх *fopen()* функцийг хэрэглэнэ. Ийм маягаар нэг програмын хүрээнд олон файл зэрэг нээж болно, учир нь *fopen()* функцийн хэрэглэх *FILE* ойг нээх файл тус бүрийн хувьд өөр өөр ойд бэлддэг. Ингэж нээх файл бүр нь ийм олон файлын нэг толоөлөл, *FILE* объект (оий) нь ийм олон объектын нэг юм. Файлыг нээсний

<sup>4</sup> *FILE* нь Turbo C++ програмын *studio.h* толгой файл дотор тодорхойлсон бүтэц юм.

дараагаар файлын зориулалттай бусад функцийг нь хэрэглэнэ. Харин `fopen()` функцийн бэлдсэн FILE объектыг `fclose()` функцийн устгаж, өөрөөр хэлбэл эзэмшиж байгаа ойт нь чөлөөлснөөр тухайн файлд хандах боломжтүй болно (Зураг 1.4).

Програм дотроос файл нээхэд хэрэглэх FILE бүтэц нь өгөгдлийн зохиомол төрөл юм. Энэ нь FILE объектын үндэс болдог. Энд `fopen()` функцийн дуудагдах бүрдээ FILE объектын тухайн нэг тохиолдлыг ойд шинээр зохион байгуулаад хаягийг нь буцааж өгнө. Ийм функцийг ОО ойлголтоор байгуулагч (*constructor*) функцийн гээнэ. Харин `fclose()` нь `fopen()` функцийн устгахад объектыг устгах (ойт нь суллах) учир устгагч (*destructor*) функцийн гээнэ.



Зураг 1.4: Файлын зориулалттай функцийн хамаарал

Дээрхээс өгөгдлийн зохиомол, зохиомол бус төрөл бүр өөр өөрийн байгуулагч, устгагч функцийтэй байдал гэсэн дүгнэлтийг хийж болно. Байгуулагч нь тухайн төрлийн объектыг үүсгэж идэвхжүүлэх бол устгагч нь ийм объектыг устгаж идэвхгүй болгоно.

### 1.1.2 Битүүмжлэл

Аль ч объект дотооддоо хийх ажлыг бусдаас далдалж байхаас гадна түүнтэй бусад объект харьцах, ярилцах интерфейстэй байна. Энэ бол модуль програм ба ОО програм хооронд байх гол ялгаа юм. Тухайн объект дотроо хэрхэн загварчлагдсаныг гадаад объект мэдэхгүй, мэдэх ч шаардлага байдагтүй. Харин гадаад объект ийм объектыг хэрхэн хэрэглэхийг, түүнтэй ямар интерфейсээр харьцахыг мэддэг байх ёстой. Өгөгдөл далдлалтай холбоотой эл ойлголтыг битүүмжлэл гэх ба энэ нь програмын найдвартай байдлыг дээшлүүлэх болон код хөгжүүлэх ажлыг хөнгөвчилдэг арга юм.

### 1.1.3 Удамшил

Класс бол ерөнхий шинжтэй, нийтлэг аргатай объектуудын изгдэл юм. Объект нь тухайн классын нэг төлөөлөл болохын хувьд классынхаа шинж, аргыг өвлөн авдаг. Эл ойлголтыг удамшил гэх ба түүнийг туслах класст бас хэрэглэнэ. Класс өөр классаас үүсч болно. Ингэхдээ эх классынхаа шинж, аргыг өвлөж авна. Жишээ нь, спортын машины *SportCar* бол *Car* классын хувьд түүний дэд класс юм. Харин *Car* нь түүний эх класс, дээд класс болно. Машин бүрд байх нийтлэг шинж, үйл хөдлөл спортын машинд бас байна. Тэгэхлээр, эл туслах класс нь *Car* классаас удамшиж үүснэ.

Удамшил бол урьд нь бичсэн програмын кодыг дахин хэрэглэх боломж бий болгодог, ОО загварчлал болон C++ хэлний чухал ойлголт юм.

#### 1.1.4 Полиморфизм буюу олон хэлбэршил

Хоорондоо өөр классууд ижил аргыг ялгаатайгаар хэрэглэнэ. Жишээ нь, "явах" үйлийг сүл морь, тушаатай морь, чөдөртэй морь өөр өөрөөр хийх тул уг ажил үйлийн үр дүн болох туулах зам нь хоорондоо ялгаатай байдаг. Класс тухайн аргыг өөртөө өөрийнхөө төлөв байдалд нийшүүлэн хэрхэн загварчлахаа мэдэж байдаг. Энэ ойлголт бол полиморфизм (polymorphism) буюу олон хэлбэршил юм. Энд "poly" нь олон, "morph" нь хэлбэр, дүрс гэсэн утгатай грек гаралтай үгс юм. Тэгэхлээр полиморфизм нь объект олон хэлбэрт байх чадвар юм. Үүний гол санаа нь аль нэгэн арга тухайн байдалд эсвэл өөр классст өөрөөр хэрэглэгдэж болохыг илэрхийлнэ. Үүнийг ОО хэлэнд хэрэгжүүлдэг гол механизм бол method overloading буюу арга дахин тодорхойлох, method overriding буюу арга дахин програмчлах, operator overloading буюу оператор дахин тодорхойлох юм.

Арга дахин тодорхойлох бол тухайн байдалд бичдэс<sup>5</sup> нь ялгаатай байхаар тодорхойлогдох ижил нэртэй функцуудын чадвар юм. Өөрөөр хэлбэл, гаднаас авах утгын төрөл болон түүний дараалал, тоо хэмжээ, мөн гарах утгын төрлөөр ялгаатай ижил нэртэй олон функц тодорхойлж хэрэглэхийг функц дахин тодорхойлох гэнэ. Энэ арга нь гол төлөв статик төрлийн програмчлалын хэлэнд байдал. Дахин тодорхойлсон функцуудээс алхиг нь тухайн байдалд хэрэглэхээ компайлдер шийддэг байна.

Функц гаднаас авах параметртэй<sup>6</sup> байж болох ба ийм параметр нь функцийг дуудаж хэрэглэх үед дамжуулж өгөх өгөгдлийн төрлийг заана. Функцийг дуудахдаа өөр өөр төрлийн өгөгдөл гаднаас өгч болно. Ийм функцийг дахин тодорхойлох механизмаар дамжуулан загварчилна.

Арга дахин програмчлах бол эх классын хийсвэр аргыг дахин програмчилж хэрэглэдэг удамших классын чадвар юм. Харин оператор дахин тодорхойлох нь, жишээ нь хоёр тооны нийлбэр олоход хэрэглэдэг '+' нээмэх үйлдлийг дахин тодорхойлох замаар хоёр тэмдэгт мөрийг залгахад хэрэглэж болдог ОО технологийн чухал шинж юм.

#### 1.1.5 Объект хоорондын уялдаа холбоо

Класс бусад класстай, объект бусад объекттой шууд эсвэл шууд бусаар, нягт эсвэл сүл холбогддог. Объектыг хооронд нь холбож өгснөөр тэдгээр нь улам хүчирхэгжиж нэмэлт мэдээлэл, нэмэлт үйл хөдлөл олж авах боломжтой болдог. Жишээ нь, customer объект руу захидал явуулахын тулд түүнтэй холбоотой address объект руу хандаж хаягийг нь авч болно.

Класс хооронд, объект хооронд дөрвөн үндсэн уялдаа холбоо байдаг. Тухайлбал,

- холбоо (association)
- бүрдмэл холбоо (aggregation)
- сронхийллөөс нарийслал (generalization specialization )
- хамтын ажиллагаа (interaction)

<sup>5</sup> Signature буюу бичдэс нь, жишээ нь функцийтэй холбогдох параметрийн төрөл, дараалал, тоо ширхэг болон функцээс буцах утгын тодорхойлолт юм.

<sup>6</sup> Функцийн гаднаас авах хэмжигдэхүүнийг параметр, аргумент гэх мэтээр нэрлэдэг. Функцийн тодорхойлолтод хэрэглэхийг нь параметр, функцийг хэрэглэхэд дамжуулж өгөх утгыг аргумент гэж нэршилээ.

### 1.1.5.1 Холбоо

Холбоо бол уялдаа холбооны энгийн хэлбэр юм. Объект нь бусад зүйлийн нэг хэсэг болж болно. Гэвч тэдгээр нь хоорондоо бүрэн хамааралтай биш. Жишээ нь, машин, түүний жолооч болон хоёр зорчигч хоорондын хамаарлыг авч үзье. Машинд жолооч хоёр зорчигчийн хамт явж байвал тэд хоорондоо холбогдоно, холбоотой болно. Энэ нь сул холбоо юм. Нэг зорчигчийг замдаа буулгаад өөр чиглэл рүү явах бол буусан зорчигч бусадтайгаа хамааралгүй болно. Ийм холбоог Зурагт 1.5-д үзүүлснээр дүрсэлж болно.



Зураг 1.5: Машин, жолооч, зорчигч хоорондын холбоо

Бүх объект тус бүрдээ орших хугацаатай, хооронд нь эзэмшилийн харьцаа байдаггүй. Багш, оюутны хоорондын холбоог авч үзье. Олон оюутан нэг багштай, нэг оюутан олон багштай холбогдож болох ч тэдгээрийн хооронд эзэмшилийн харьцаа байхгүй. Хоёулаа тус тусын орших хугацаатай, тус тусад нь үүсгэж устгаж болно.

Класс хоорондох энэ холбоо хамаарлыг "has a" буюу "-тай" гэдгээр илэрхийлж болдог. Жишээ нь, a car "has a" cellular phone буюу машин үүрэн утас "тай". Энэ холбооны хоёр талын объектын тооны харьцаа янз бүр байна. Жишээ нь, нэг машин нэг үүрэн утастай бол энэ 1-ээс 1, машин олон зорчигчтой бол машин болон зорчигчдын хооронд 1-ээс ОЛОН холбоо тус тус үүсэх жишээтэй.

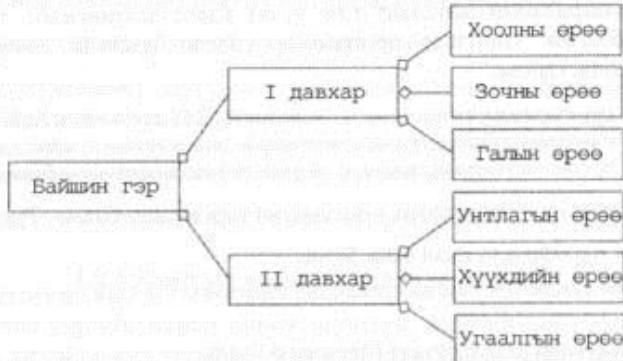
### 1.1.5.2 Бүрдмэл холбоо

Бүрдмэл холбоо нь холбооны тусгай хэлбэр юм. Олон объект хамтдаа өөр объект үүсгэхийг хэлнэ. Гол төлөв бүрдүүлбэр-бүрдмэл гэсэн шатласан байдалтайгаар хэрэгждэг холбоо юм. Бүрдүүлбэр объект нь бүрдмэл объектын чухал хэсэг. Бүрдмэлээс бүрдүүлбэр рүү гэсэн нягт холбоог агуулдаг. Бүх объект өөр өөрсдийн орших хугацаатай, гэхдээ тэдгээрийн хооронд эзэмшилийн харьцаа байдаг. Бүрдмэл объектын нэгэн адиллаар бүрдүүлбэр объектыг бий болгож, хуулбарлаж, улмаар устгаж болно. Бүрдүүлбэр объект нь өөр эх класст харьялагдахгүй. Жишээ нь, багш объект болон танхим объект хоорондын хамаарлын хувьд нэг багш олон танхимд харьялагдахгүй, гэхдээ танхим устахад багш дагаж устдагтүй.

Класс хоорондох энэ холбоог "part of" буюу "-ийн хэсэг" гэдгээр илэрхийлж болдог. Ийм хамаарал, жишээ нь машин, түүний хөдөлгүүр хооронд үүснэ. Car has a part of motor буюу машин хөдөлгүүртэй, хөдөлгүүр машины бүрдүүлбэр хэсэг. Бүрдмэл холбооны хоёр талын объектын тооны харьцаа янз бүр байна. Жишээ нь, машин нэг хөдөлгүүртэй бол 1-ээс 1 гэсэн холбоо үүснэ.

Бүрдмэл холбооны тусгай хэлбэр бол composition буюу цогц бүрдмэл холбоо юм. Охин объект буюу дэд классын объект өөрийн гэсэн орших хугацаагүй. Эх объект уствал бүх охин объект мөн дагаж устана. Жишээ болгож байшин гэр, түүний орое хоорондын хамаарлыг авч

үзье. Байшин гэр олон өрөөтэй. Өрөө хоёр өөр байшин гэрт харьялагдахгүй. Өрөө дангаар бие даасан биш. Байшин гэр уставал өрөө автоматаар устана (Зураг 1.6).



Зураг 1.6: Байшин гэрийн цогц бурдмэл холбоо

Мөн асуулт, сонголт хоорондын холбоо нь цогц бурдмэл холбоо юм. Нэг асуулт олон сонголтой байх ч тухайн нэг сонголт олон асуултанд байж болохгүй. Асуулт уставал түүний сонголтууд бас устана.

#### 1.1.5.3 Ерөнхийлөөс нарийсгал

Энэ холбоо нь дээд болон дэд класс хоорондох дундын төлөв байдлыг, тухайлбал тэдгээр нь нийтлэг шинж тэмдэг, үйл хөдлөл, хамааралтай болохыг заана. Холбоо нь "is a" буюу "type of" терлийных байна. Жишээ нь, машин тээврийн хэрэгслийн нэг төрөл. Энэ холбоог удамшилаар дамжуулан хэрэгжүүлдэг.

#### 1.1.5.4 Хамтын ажиллагаа

Объект бүр хамгийн багадаа нэг өөр объекттой холбогддог. Ийм объектууд хамтран ажиллаж нарийн зүйлийг хийж чадна. Объектууд өөр хоорондоо "зурvas мэдээ" хэлбэрээр үйлдлийн хүсэлт явуулах замаар харилцдаг. Функци дуддаж ажиллуулах хүсэлт явуулна. Функци бусад объектын хүсэлтийн хариу болгож хүссэн үйлдлийг нь хийдэг.

### 1.2 Класс зохиомжлох үйл явц

Уламжлалт програмчлалд урьдчилан боловсруулсан алгоритмын дагуу программын кодыг бичиз. Харин C++ зэрэг ОО хэлний хувьд код бичихээс өмнө бодлогын ОO загварчлалд анхаарах ёстой байлаг. Тухайлбал, бодлогод тохирох классыг эхлээд загварчилж, дараа нь бусад объекттой хэрхэн харьцахыг нь тодорхойлох шаардлагатай. Тэгхэлээр, холбогдох аргыг нь судалж эзэмшихгүйгээр ОO програм бичих боломжгүй юм.

ОО програм ямар класс болон гишүүн функцийтэй байхыг шийдэх нь түвэгтэй процесс юм. Ялангуяа тохирох зөв функци бүхий классуудыг тэдгээр нь нэгдмэл, чөлөөтэй холбогддог, зөв ажилладаг сайн програм үүсгэж бас хялбар өөрчилж болдог байхаар тодорхойлох нь нилээд цаг хугацаа шаардах ажил юм. Классыг зохиомжилж хөгжүүлэх процесс нь бодлогод тавигдах шаардлагыг тодорхойлохоос эхэлнэ.

#### 1.2.1 Бодлогын шаардлага тогтоо

Бодлогын шаардлага бол захиалагч ба хэрэглэгчийн хэрэгцээ, шаардлага юм. Үүнийг олж тогтоохын тулд програм зохиогч нь хэрэглэгчтэй хамтран ажиллаж програмыг хэрхэн

ашиглах, програм юу гүйцэлдүүлэх ёстой, бусад програм болон хэрэглэгчтэй хэрхэн холбогдож хамтран ажиллах талаар олж мэдэх шаардлагатай болдог. Ер нь, бодлогод тавигдах шаардлагыг түүний тавил, тодорхойлолтоос гаргаж авах боломж байх тул ийм тодорхойлолтыг шаардлагын жагсаалт гэж үзээд класс зохиомжлох хэсэг рүү шилжиж болно. Жишээ болгож сорилын программын үндсэн хэсгийг хөгжүүлэхэд тавигдах шаардлагыг томъёолж гаргая.

*Математик, тухух, дуу хөгжим, уран зохиол гэх мэт янз бүрийн асуулттай байх сорилын програм нь олон оюутныг зэрэг шалгадаг байна. Нэг компьютерийг олон оюутан дундаа хэрэглэх бол тэд тус тусдаа асуултанд хариузна. Асуултанд дахин хариуцж болно.*

Дээрх тодорхойлолтод суурilan доорх шаардлагыг гаргаж авч болно. Тухайлбал,

1. Сорил олон төрлийн асуулттай байж болно.
2. Олон оюутан дундаа нэг компьютерийн гар хэрэглэвэл тэд эзлжээр тус тусын асуултанд хариузна.
3. Нэг асуултанд олон удаа хариулах боломжтой байна.

### 1.2.2 Нэр үг классын иэр болох нь

Бодлогын тавил, тодорхойлолтод (орсон) байгаа иэр үг нь гол төлөв зохиомжлох ёстой классын иэр болох магадлалтай байдаг. Дээр тогтоосон шаардлага дотор сорил (quiz), оюутан (student), компьютер (computer), компьютерийн гар (keyword), асуулт (question) зэрэг нэр үг хэрэглэснээс сорил, оюутан, асуулт зэрэг нь зохиомжлох классын иэр болох боломжтой үгс юм.

### 1.2.3 Үйл үг гишүүн функци болох нь

Өмнө олж тогтоосон иэр бүхий классыг зохиомжлох ажлын эхний үе бол классын хийх үйл хөдлөл болон үүрэг даалгаврыг тогтоох явдал юм. Классын үйл хөдлөлийг гишүүн функцийн тодорхойлно. Классыг зохиомжлох ганц зөв арга зам гэж байхгүй, заримдаа эхлээд аль нэг класст хамаатуулж үзсэн аргыг зохиомжлох ажлын явцад өөр класс руу иэгтгэх тохиолдол байнга гардаг.

Классын үүрэг даалгавар бол класстай, класс хоорондох хамтын ажиллагаатай холбогдох арга юм. Классын арга нь бодлогын тодорхойлолтод байгаа үйл үг, үйлт нэрийн хэлбэрээр илрэх магадлалтай. Гэхдээ бүх үүргийг класст оноож өгдөггүй, зарим үүрэг main функцийн доторх код, зарим нь сүл энгийн функци хэлбэрээр байж болдог.

### 1.2.4 Төсөөлөл зураг ашиглан үйл үгийг олох

Зохиомжлох ажлын эхний үед ямар гишүүн функци хэрэгтэй болох нь тодорхойгүй байдаг тул зарим тохиолдолд програм хожим хэрхэн ажиллах талаар scenarios буюу төсөөлөл зураг хийх нь классын үйл хөдлөл болон үүрэг даалгаврыг тогтооход тустай. Төсөөлөл зураг нь хэрэглэгч болон програм эсвэл програм дэх класс хооронд явагдах харилцан яриа адил тодорхойлолт юм.

Өмнөх сорилын программын үйл ажиллагааны төсөөлөл зураг нь доорх зүйлийг агуулж болно. Тухайлбал,

1. Нэг компьютерийг хоёр оюутан дундаа хэрэглэвэл тэд тус бүрээ нэрээ оруулж бургүүлээд авсан сорилынхоо асуултанд эзлжээн хариузна.
2. Оюутан асуултанд хариулж эхлэхийн өмнө сорилд хэдэн асуулт байгааг мэдэж авна. Компьютер хувааж хэрэглэх хоёр оюутны авсан асуултын тоо адил байна.

3. Оюутан асуултанд хоёр удаа хариулж болно. Оюутны хариулт бүрд програм "зөв" эсвэл "буруу" гэж хариулна. Оюутан огт зөв хариулахгүй бол зөв хариут програм буцааж өгнө.
4. Сорил дуусгавар болоход програм оюутан бүрийн авсан нийт оноог дэлгэнэд гаргана.

Дээрх төсөөлөл зургаас дараах үйл үргийг гарган авч болно.

Нэрээ оруулах (EnterName), асуултын төрөл сонгох (ChooseKindOfQuestion), асуултын дугаар сонгох (ChooseNumberOfQuestions), асуулт асуух (AskQuestion), зөв хариут авах (GetCorrectAnswer), асуултанд хариулах (RespondToQuestion), нийт оноог авах (GetScore), програм зөв хариут хэлэх (ProvideFeedback)

Үүний дараагаар гишүүн функцийг класст оноох ажлыг хийнэ.

### 1.2.5 Классын ажил үүргийг хуваарилах

Хийх ёстой бүх ажил үүргийг класст оноож өгдөгтүй. Зарим ажил үүрэг, жишээ нь main функц доторх код, зарим нь сул энгийн функц хэлбэрээр байна. Сорилын програмд классын ажил үүргийг доор үзүүлсэн байдлаар хуваарилж болно. Тухайлбал,

1. Student  
Construct using name (ask student's name)  
RespondTo a question  
GetScore
2. Quiz  
ChooseKindOfQuestion  
AskQuestion or Give a question to student
4. Question  
Construct question type  
AskQuestion  
GetCorrectAnswer

Классын зохиомжийг бэлэн болгосны дараа харгалзах кодыг бичиж турших ажлыг хийнэ. Эн сурх бичгийн жишээ програмд энгийн класс хэрэглэсэн учир класс зохиомжлох ажлыг ёсчлон хийгээгүй болно.

### 1.3 Объект хандлагат загварчлалын хэл

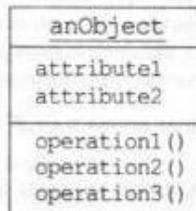
ОО загварчлалд хэрэглэдэг стандарт тэмдэглээний нэг бол UML (Unified Modelling Language) буюу нэгдмэл загварчлалын хэл<sup>7</sup> юм. Түүнийг объект хандлагатай програм бүтээх ажлын эхний алхам болгож хэрэглэдэг. UML нь програмын системийн хамгийн бага бие даасан нэгж болох объект тус бүрийн онцлог шинжийг тодорхойлон гаргаж хийсвэрлэн дүрслэх, зохион бүтээх, баримтжуулах зориулалтын олон хэлний нэг төлөөлөл болдог.

Объекттой ажиллах үед түүнийг тодорхойлон бичих, түүний тухай эргэцүүлэн бодохын тулд объектыг зураглан үзүүлэх арга зам хэрэгтэй. Зураг 1.7-д хэрэглэсэн тэмдэглээ бол UML объект диаграмм юм. Гурван хэсэг бүхий нүдэнд объектын нэр (доогуур нь зурсан), түүний шинж тэмдэг, үйл хөдлөл буюу үйлдэл зэргийг бичнэ. Ингэхдээ шинж тэмдгийн нэрээс ялгахын тулд үйлдлийн нэрийг ардаа нээх хаах бага хаалттай байхаар бичнэ. Үйлдлийг

<sup>7</sup>Тэртээ 1970-аад оны дунд үеэс 1980-вад оны сүүл үе хүртэлх хугацаанд "загварчлалын хэл" гэгдэх олон арван арга технологи бий болжээ. Тухайлбал, Grady Booch, Ivan Jacobson, James Rumbaugh нар ижил зорилгын төлөө тус тусдаа ажилласж Booch арга (Booch method), объект хандлагат програм хангамжийн инженерчлэг (Object oriented software engineering), объектын загварчлалын арга (Object modeling technique)-ыг хөгжүүлж байгаад 1990-ээд оны дунд үеэс, тухайлбал 1995 оноос хамтран ажиллах болсоор UML хэл бий болсон байна.

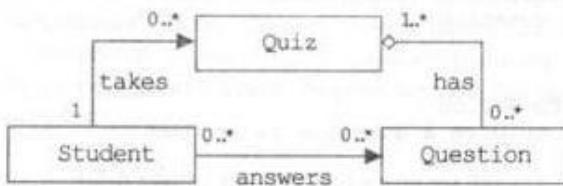
объект диаграмм гол төлов харуулдаггүй. Гэхдээ классын бүтцийг бүхэлд нь дүрсэлж харуулах зорилгоор объектын шинж, үйлдлийг хамтад нь үзүүллээ.

Шинж тэмдэг нь объект дотор далал байдаг. Тэдгээрт үйлдлээр нь хандах ганц зам бий. Энэ нь зурагт радиогийн сувгийг солихдоо дотроо ямар электроник хэрэглэсэн байгааг анзаарагчийгээр, мэдэхгүйгээр зориулалтын товчийг нь хэрэглэдэгтэй адил юм.

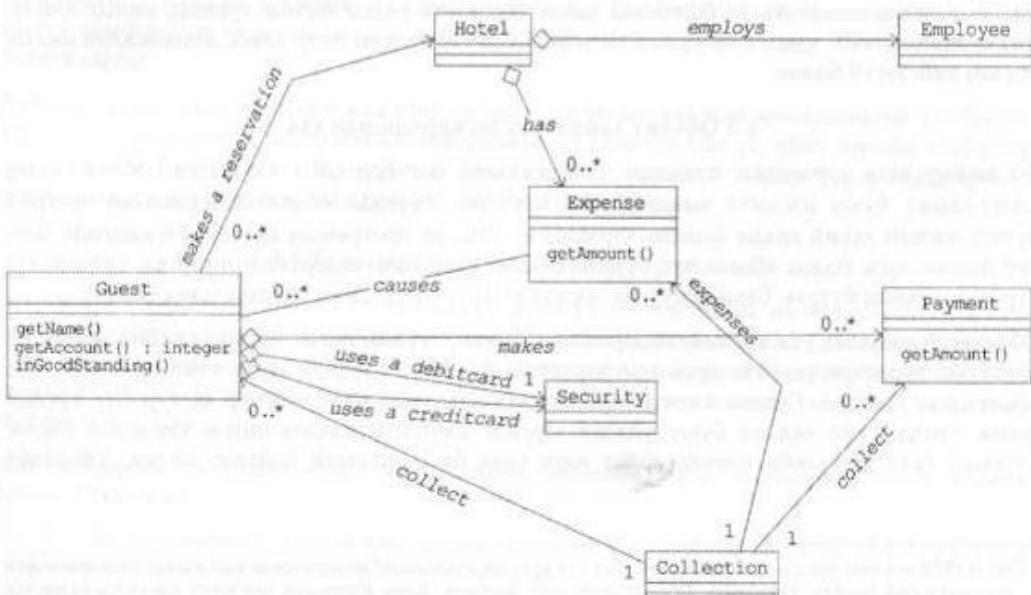


Зураг 1.7: UML объект диаграммын жишээ

UML класс диаграмм бол хоорондоо холбогдсон классууд бүхий жишиг зураг юм (Зураг 1.8-1.9).



Зураг 1.8: Сорилын программын хялбаршуулсан UML класс диаграммын жишээ



Зураг 1.9: Зочид буудлын программын хялбаршуулсан UML класс диаграммын жишээ

Зураг 1.8~1.9-д үзүүлсэн диаграм нь класс тус бүрийн гол гишүүд, класс хоорондын харьцаа, хийгдэх үлийг тодорхойлж үзүүлэх тул түүнийг классын диаграм гэнэ. Класс хоорондын холбоосыг гол төлөв үл үгээр нэрлэдэг. Холбооны тэмдэглээ нь шулуун, бүрдмэлийнх бурдмэл талдаа ромбо бүхий шулуун, удамшлынх дундаа гурвалжин бүхий шулуун байдаг. Бүрдмэл холбооны ромбо цагаан, цогц бүрдмэлийнх хар өнгөтэй. Класс хоорондох холбооны хоёр талын объектын тооны харьцаа олон янз байх бөгөөд түүнийг тэмдэглээний хоёр талд бичдэг. Жишээ болгож сорилын програмын хялбаршуулсан класс диаграмыг Зураг 1.8-д, зочил буудлын программын класс диаграмыг Зураг 1.9-д тус тус үзүүллээ.

UML диаграм боловсруулах зориулалтын олон арван программын систем байдгийн нэг нь Microsoft Visual Modeler юм. Мөн Relational Rose зэрэг системүүд байдаг.

Программын системийг програмчлахаас өмнө загварыг нь эхэлж гаргах нь том программын хувьд түүний ноорог төсөл болдог сайн талтай. Гэхдээ энэ нь энгийн бодлогын хувьд ч загварыг нь эхэлж гарга гэсэн үт биш юм. Загварыг нь эхэлж гаргах эсэхээ програм хөгжүүлэгч<sup>8</sup> өөрөө шийддэг.

#### 1.4 Объект хандлагат программын шинж

Объект бол ажиллагааг нь автомажуулахыг хүсч байгаа бизнестэй нь холбоотой байдаг тул хүн объектыг ойлгоход амархан. Яг ёсоор нь хэрэглэвэл класс нь туршиж алдааг засварлах ажлыг хөнгөвчлох, кодын давхардалтыг багасгах замаар програм хөгжүүлэлтийг хурдасгаж чаддаг. Ер нь, ОО программын олон сайн шинжийг доор үзүүлснээр жагсааж болно. Тухайлбал,

- Программын найдвартай ажиллагаа сайжирдаг; ОО програм уншуур сайтай; ОО програм нь модулийн бүтэцтэй байх учир программын аль ч хэсгийг, аль ч модулийг бусдаас нь салангид хөгжүүлж болох ба энэ нь програмд гарч болох алдааг багасгах сайн талтай.
- Класс нь класс биш кодыг бодвол програмд дахин хэрэглэх, хөгжүүлэхэд хялбар байдаг нь түүний модулийн шинжтэй холбоотой юм. Тодорхой модулийг дангаар нь хөгжүүлж болно.
- Программыг зохион бүтээх хугацааг богиносгох боломжтой байдаг нь урьд бичсэн кодыг өөр програмд дахин хэрэглэж бодлогтой холбоотой, ингэснээр программын оорийн ортог буурах бололцоотой байдаг.

#### Мэдлэгээ шалгах асуулт

- 1.1. Уламжлалт програмчлалын арга гэж юу болох, түүний гол шинжийг нэрлэх
- 1.2. ОХП гэж юу болох, процедур хандлагат програмчлалаас ямар ялгаатай болох
- 1.3. Класс, объект хоорондын холбоог жишээн дээр тайлбарлах
- 1.4. Битүүмжлэл гэж юу болохыг жишээн дээр тайлбарлах
- 1.5. Удамшил гэж юу болохыг жишээн дээр тайлбарлах
- 1.6. Арга дахин тодорхойлох гэж юу болох
- 1.7. Арга дахин програмчлах гэж юу болох
- 1.8. Класс хоорондын, объект хоорондыг уялдаа хамаарлыг жишээн дээр тайлбарлах
- 1.9. ОО загварчлалын программын системийн талаар юу мэдэх
- 1.10. ОО программын шинжийг нэрлэх
- 1.11. UML гэж юу болох

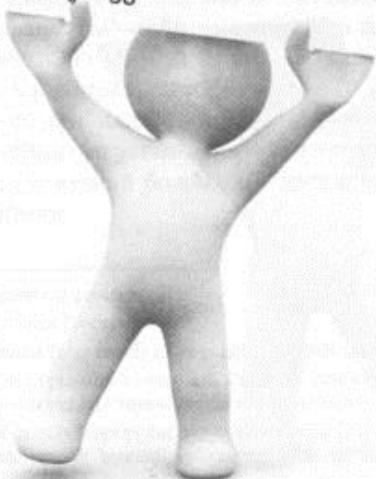
<sup>8</sup> Электрон тооцоолох машин Монголд эрчимтэй нэвтрэх эхэлсэн 1970-аад оны үед компьютерийн програм бичих ажлаа, мэргэжилтий хүнийг програм зохионч гэж нэрлэдэг байсан ажээ. Гэхдээ software developer буюу програм хөгжүүлэгч гэдэг нэрийг хэрэглэлээ.

# ХОЁРДУГААР БҮЛЭГ

## C++ ХЭПНИЙ ТУХАЙ

- C++ хэлний цагаан толгой, 18
  - C++ хэлний бүтвэр, 18
  - Өгөгдлийн төрөл, 19
  - C++ програмын бүтэц, 19
  - Утга оноох оператор, 20
  - Арифметик оператор, 21
  - Харьцуулах оператор, 22
  - Логик оператор, 22
  - if-else команд, 23
  - switch-case команд, 25
  - for команд, 27
  - while команд, 30
  - do-while команд, 31
  - Давхар давталт, 32
  - Хүснэгт, 34
  - Функц, 38

-Үргэлжлэл→



## ЧАГАА ЧАЛХЯЗАХ

### -Үргэлжлэл-

- Хувьсагч, түүний эзэмших ойн ангилал, 41
- Бүтэц, 43
- Бүтцийн тодорхойлолт, 44
- Бүтцийн гишүүд рүү хандах, 45
- Давхар бүтэц, 47
- Бүтцэн хүснэгт, 48
- Функц ба бүтэц, 50
- Хэрэглэгч тодорхойлох өгөгдлийн төрөл, 52
- Хаяг, хаяган хувьсагч, 56
- C++ хэлийт Си хэлтэй харьцуулах нь, 72



## ХОЁРДУГААР БҮЛЭГ

### C++ ХЭЛНИЙ ТУХАЙ

Америкийн Bell Laboratories компанийн ажилтан Ken Thompson 1969 онд Unix үйлдлийн системийг зохиодоо DEC<sup>9</sup> фирмийн PDP<sup>10</sup>-7 машины ассемблер хэдийг хэрэглэсэн байна. Дараа нь уг машины дараагийн үе болох PDP-11 машин зохион бүтээгдэхэд түүнд тохируулан Unix системийг дахин бичих шаардлага гарсан нь PDP-7 машины ассемблер хэл шинэ машинд тохирхгүй байсантай холбоотой юм. Ингэж шинэ машин бүтээх бүрд Unix системийг ассемблер хэлээр дахин дахин бичих нүсэр ажлыг байнга хийхгүйн тулд энэ системд сайн тохирох дээд түвшний хэл хэрэглэх шаардлага гарчээ. Энэ ажилд тохирох хэл тухайн үеийн програмчлалын салбарт хэрэглэж байсан олон арван хэл дотроос одоогүй тул Ken Thompson тэр үеийн BCPL<sup>11</sup> хэлд түшиглэн В хэлийг шинээр 1972 онд зохиожээ. Гэвч шинэ хэл нь Unix системд төдийлон сайн тохирхгүй байсан тул Bell Laboratories компанийн ажилтан Dennis Ritchie уг хэлийн шинэ хувилбар болох C<sup>12</sup> хэлийг бүтээж түүгээр Unix системийн 90 орчим хувийг бичсэн байна. Уг хэл удалгүй их, дээд сургуулийн хүрээнд ихэд тархаж улмаар компьютерт зориулсан, хямд төсөр Си хэлний compiler буюу компайлер програм<sup>13</sup> бий болсноор олон арван хэрэглээний програмыг уг хэлийн буулгахад хүргэсэн байна. Гэхдээ Си хэдий хүчирхэг хэл боловч орчин цагийн програмчлалын технологид үл ийцх 2 үндэсн шинж түүнд байсан нь нэгдүгээрт, Си хэлний хэрэглэгч програмчлалын зохих туршлагатай байхыг шаарддаг; хоёрдугаарт, програмчлалын технологи нь Си хэл зохиогдсон 1970 оныхоос ихэд өөрчлөгдсөн зэрэг юм.

Бодитойгоор оршин буй юмс, үзэгдлийн тухай мэдээлэл, түүнийг боловсруулах аргыг хамтад нь загварчилж болох уу гэсэн санаа анх 1960-аад оны үес гарч Норвегийн компьютерийн мэргжилтэн Kristen Nygaard ба Ole Johan Dahl нар 1976 онд бодитой байгаа юмсыг симуляц хийхэд зориулан Simula-67 хэлийг зохиожээ. Уг хэл нь юмсын хийсвэр загварчлалаа хэрэглэх class хэмээх шинжлэй бөгөөд түүнийг удамшилын механизмаар дамжуулан өргөтгөх боломжтой байхаар бүтээгдсэн байна. Програм хөгжүүлэгчид уламжлалт програмчлалын арга хэмээн нэршигдэх болсон бүтцийн аргаас салах дургүй байсан зэрэг бодит учир шалтгааны улмаас шинэ хэл төдийлон сайн тархаж чадаагүй ажээ. Гэвч Simula-67 хэлний class гэх шинж хожим OOP-ын үндэс болжээ. Объектыг класс ба удамшилаар дамжуулан загварчлах шинэ технологийг 1970 оноос эхлэн хэрэглэх болсноор програмыг шинэчлэх, хөгжүүлэхтэй холбоотойгоор гарах зардлыг багасгах боломжтой нь илэрхий болжээ. Гэхдээ 1980 онд Xeroх компанийн Palo Alto Research Center (PARC буюу Palo Alto судалгааны төв)-ийн мэргжилтийн жинхэнэ утгаараа OOP хэл болох Smalltalk-80 хэлийг бүтээжээ. Энэ цаг үед буюу 1980-аад онд Bell Laboratories компанийн ажилтан Bjarne Stroustrup Си хэлэнд классын шинжийг нэмж оруулах төслийг хэрэгжүүлж шинэ хэлийг "C with class" (класстай Си) гэж нэрлэж байсныг 1983 оноос C++ гэх болжээ. Шинэ нэр нь Си объектын утгыг нэгээр нэмэгдүүлэх үүрэгтэй ++ оператороос үүдэлтэй гэж үздэг. Класс бол Си хэлний боломжийг нэгээр ахиулсан шинж гэдгээс улбаалан C++ гэж нэрлэх болжээ. Иймээс

<sup>9</sup>DEC - Digital Equipment Corporation

<sup>10</sup>PDP - Programmable Data Processor

<sup>11</sup>BCPL – Basic Combined Programming Language: Програмчлалын хэлний компайлер хөгжүүлэхэд хэрэглэдэг хэл

<sup>12</sup>Си нь доод болон дээд түвшний хэлний албаны нь шинжийг өөртөө шинигэсэн хэл тул түүнийг дунд түвшний хэл гэж компьютерийн ухааны зарим судлаачид үздэг байна.

<sup>13</sup>Compiler нь Монгол хэлийн "хөрвүүлэх, хөрвүүлэгч" гэж буух ч дуудлагаар нь компайлер гэж нэршилээ. Хойшид "компайлэр програм" гэхийг "компайлэр" гэж товчилж хэрэглэнэ.

C++ = Си + Класс (Объект)

гэж томъёолж болно.

Шинэ хэл нь Си хэлний бүх боломжийг өөртөө шингээсэн байх тул Си хэлний командыг хэрэглэн C++ хэлээр процедур програм бичиж болох ч уг хэл нь ОХП технологид илүү тохиирсон, түүнийт дэмжиж ажилладаг програмчлалын хэл болсон байна. C++ мөн өөрийн гэсэн онцлог шинж, боломжтой хэл юм. Тухайлбал, гарааны утгатай функцийн параметр, дотоод мөр функци, дахин тодорхойлсон функци болон оператор, ойн менежментийн оператор зэрэг Си хэлэнд байдаггүй олон боломж түүнд байдал.

## 2.1 C++ хэлний цагаан толгой

C++ хэлний цагаан толгой дараах гурван хэсэгтэй.

- Латин цагаан толгойн жижиг, том үсэг : a~z, A~Z
- Цифр: 0~9
- Тусгай тэмдэг, тэмдэгт: + - \* / % = ? < > & | \ # ( ) [ ] { } . . . ; ^ !

Зарим онцгой тэмдэгтийн жагсаалтыг Хавсралт С-д үзүүлсэн. Зайн (Spacebar) товchoor оруулах зайлгасан тэмдэгт гэнэ.

### 2.1.1 C++ хэлний бүтвэр

C++ хэлэнд хэрэглэх үгийг элемент буюу бүтвэр гэнэ. Хоёр төрөл зүйлийн тэмдэглээ үгийг хэрэглэдэг. Тухайлбал,

- Тусгай (түлхүүр) үг
- Ялгац (ялгац) иэр

**Тусгай үг.** int, cout, for гэх мэт нь C++ хэлний тусгай түлхүүр үгс юм. Ийм үг бүр илэрхийлэх ганц утгатай байна. C++ хэлний команд бол тусгай үг юм. Тусгай үгийг бас reserved keywords буюу иөөц тусгай үг гэнэ. Turbo C++ програмд хэрэглэгдэх тусгай үгийн жагсаалтыг Хавсралт В-д үзүүллээ.

**Ялгац иэр.** Програм хөгжүүлэгч нь ялгац иэрийг тодорхойлох ба ийм иэр нь хувьсагч, функци, хүснэгт, бүтэц, класс, объект гэх зэрэг хэрэглэгчийн зохиомол төрөл зүйлтэй холбоотой байна. Жишээ нь, sum, showdata гэх мэт.

Тусгай үгийн зориулалт, илэрхийлэх утга нь түүнийт ямар програмд хэрэглэж байгаагаас хамаарахгүйгээр тогтмол байх бол ялгац иэр нь програмд ямар зориулалтаар хэрэглэж байгаагаас хамаарч харилцан адилгүй утга илэрхийлдэг байна. Тухайлбал, `r` нь зарим програмд (principal amount) нийт дунг, зарим програмд (pressure) дараалтыг заах жишээтэй байж болно. Эндээс үзэхэд програмд хэрхэн хэрэглэхээс ялгац иэрийн илэрхийлэх утга хамаарах бол тусгай үгийнх програмаас үл хамаардаг байна.

Тусгай үгийг ялгац иэрийн зориулалтаар хэрэглэж болохгүй. Ингэхлээр, жишээ нь `int` иэртэй хувьсагч, хүснэгт, класс байж болохгүй.

C++ хэлэнд тоон, тэмдэгтэн ба тэмдэгтэн мөр гэсэн 3 өөр төрлийн тогтмол байж болдог. Тоон тогтмол нь бүхэл тоон (123) эсвэл бодит тоон (3.45) төрлийнх байж болно. Хоёр даан хашилт ("") хооронд бичих нэг тэмдэгтийг тэмдэгтэн тогтмол (жишээ нь "r") гэнэ. Харин давхар хашилт ("") хооронд бичигдэх нэг эсвэл бүлэг тэмдэгт бол тэмдэгтэн мөр, тэмдэгтэн цуваан тогтмол (жишээ нь "C++ is" эсвэл "1") болно.

### 2.1.2 Өгөгдлийн төрөл

Өмнөх бүлэгт ялгах нэрийн талаар товч үзсэн билээ. Ийм нэрийн илэрхийлэх төрөл нь түүнд хадгалах өгөгдлийн төрөл зүйлээр тодорхойлогдоно. C++ бол стандарт өгөгдлийн төрөлтэй, ийм төрөл тус бүрд харгалзах тусгай үгтэй хэл юм. Тухайлбал, тэмдэгт өгөгдлийн хувьд `char`, бүхэл тоон өгөгдлийн хувьд `int` эсвэл `long`, болит тоон өгөгдлийн хувьд `float`, `double` гэх үгийг хэрэглэнэ. Энэ хэлний стандарт өгөгдлийн төрлийн жагсаалтыг Хүснэгт 2.1-д үзүүллээ. Тэмдэггүй тоон өгөгдлийн хувьд `unsigned` гэдэг үгийг тохиорох төрөл үгийн нь хамт хэрэглэнэ.

Хүснэгт 2.1: C++ хэлний стандарт өгөгдлийн төрөл

Өгөгдлийн төрлийг заах ург	Шаардах ойн хэмжээ, байтаар	Авах утгын доод, дээд хязгаар	
		доод	дээд
<code>char</code>	1	-128	127
<code>unsigned char</code>	1	0	255
<code>short</code>	2	-32,768	32,767
<code>unsigned short</code>	2	0	65,535
<code>int</code>	2	-32,768	32,767
<code>unsigned int</code>	2	0	65,535
<code>long</code>	4	-2,147,483,648	2,147,483,647
<code>unsigned long</code>	4	0	4,294,967,295
<code>float</code>	4	$3.4 \times 10^{-38}$	$3.4 \times 10^{+38}$
<code>double</code>	8	$1.7 \times 10^{-308}$	$1.7 \times 10^{+308}$
<code>long double</code>	10	$3.4 \times 10^{-4932}$	$3.4 \times 10^{+4932}$

### 2.1.3 C++ програмын бүтцэ

C++ програм нь функцийн тогтолцоогүй. Програм бүрд `main()` функцияа заавал байх ёстой. C++ програм энэ функцийн эхэлж ажилладаг. Энгийн C++ програмын жишээг Program #2.1-д үзүүллээ.

```
//Program #2.1
//Displays "C++ programming is very interesting" message

#include <iostream.h>
#include <conio.h>

void main()
{
    cout << "C++ programming is very interesting" ;
    getch() ;
}
```



C++ programming is very interesting

Дээрх энгийн програмын команд бүрийг авч үзье. Тайлбарын мөр ( // ) хос гэдрэг зураасаар эхлийн C++ компайлдер ийм мөрийг командын мөр гэж үзэхгүй, түүнийг зөвхөн програмын баримтжилтанд хэрэглэнэ.

Удаах хоёр мөр команд бол C++ компайлерт зориулсан препроцессорын заалт команд юм. Эдгээр нь программын бичилтийг шалгахын өмнө програмд нэмж оруулах шаардлагатай `iostream.h`, `conio.h`<sup>14</sup> толгой файлуудыг системд<sup>15</sup> хэлж өгөх үүрэгтэй юм. `cout`

<sup>14</sup> "header" гэдэг англи үгийн эхийн үсгийг эдгээр файлын нэрийн өргөтгэлдэг. Энэ англи үгийг "толгой" гэж монгол хэлийн буулгалаа.

объектын харьялгахаа классын эх загвар нь `iostream.h` файлд тодорхойлогдсон байх тул уг файлын нэрийг препроцессорын `include` командаар C++ компайлерт хэлж өгнө. Хэрэв энэ мөр командыг бичиж өгөөгүй бол `cout` нь чухам юу болохыг C++ компайлэр тайлж уншиж чадахгүй. Үүнтэй адил, `getch()` функцийг програмд хэрэглэхэд `copio.h` файл хэрэгтэй болно. Дараагийн мөрөөс нь `main()` функцийн тодорхойлолт эхэлий. Функцийн нэрийн араас залгаж бичих () дугуй хаалтыг функцийн оператор гэнэ. Хоосон функцийн оператор нь функци гаднаас ямар нэгэн зүйл (аргумент) авахгүйг, функцийн нэрийн омнех `void` тусгай үг нь функцизэс ямар нэгэн утга буцахгүй болохыг тус тус заана.

Програмын `main()` функцийн дотор байх бүлэг командыг баги командын [] их (нуман) хаалт хооронд бичнэ. Функцийн тодорхойлолт нээх их хаалтаар эхэлж, хаах их хаалтаар төгсөнө. Програмын эх кодыг уншууртай болгох үүднээс нээх, хаах их хаалтын болон командын морийн өмнө хэсэг догол зай, мөн командын морийн дээр доор хоосон мөр орхиж болно.

`main()` функцийн командын хэсэг нь ардаа (<<) гаргах<sup>16</sup> оператор бүхий `cout` объекттой мөрөөр эхэлж байна. Энэ нь гаргах операторын баруун талд бичиж өгөх тэмдэгт морийг дэлгэцлэх<sup>17</sup> командын мөр юм. Гаргах оператортой холбогдох тэмдэгт морийг ("") давхар хашилтад бичнэ. Морийн эцэст байх (;) цэгтэй таслал нь командын морийн төгсгөлийг заана.

Програм өгөгдсөн морийг DOS<sup>18</sup> дэлгэц рүү бичээд, жишээ нь Turbo C++ ажлын цонх руу буцаж очих тул програмын ажлын цонх харагдахгүй болно. Харин `getch()` функцийг програмд хэрэглэсэнээр дэлгэцэлсэн мөр бүхий ажлын цонхыг ямар нэгэн товч дээр товших хүртэл харж болно. Товч дээр товшсоноор Turbo C++ ажлын цонх руу буцаж орно.

#### 2.1.4 Утга оноох оператор

C++ програмд (=) тэнцүүгийн тэмдгийг утга оноох оператороор хэрэглэх бөгөөд програмчлалын хэрэглээ нь түүний математик хэрэглээнээс эрс ялгаатай байдаг. Жишээ нь, Си програмын

```
x = y ;
```

команд нь "x нь у-тэй тэнцүү" гэсэн утга илэрхийлэхгүй, харин "у-ийн утгыг x рүү хийх, оноох" гэсэн утгатай юм.

C++ илэрхийллийн хувьд утга оноох операторын баруун талд ямар нэгэн хариутай (any expression) дурын илэрхийлэл, зүүн талд нь (any variable) ямар нэгэн хувьсагч байна. Үүнийг ерөнхийд нь

```
variable = any expression ;
```

гэж бичиж болно. Энд `variable` нь C++ хэлний дотоод суурь төрлийн эсвэл хэрэглэгчийн зохиомол төрлийн ямар нэгэн объект байна. Жишээ нь,

```
N = 3 ;
N = N*N+N-1 ;
```

<sup>15</sup> Систем гэдэгт C++ компайлэр, түүнтэй хамтран ажиллах бусад программыг хамруулж ойлгоно.

<sup>16</sup> insertion оператор, жишээ нь дэлгэц рүү гаргах зүйлийг холбогдох гаралтын урсгал руу нь нэмж оруулах үүрэгтэй юм. Иймд түүнийг оруулжын, оруулах оператор гэж ирэлж болох ч "гаралтын, гаргах урстаг" гэдгийг гол болгох гаралтын, гаргах оператор гэж иоршилоо.

<sup>17</sup> "Компьютерийн дэлгэц рүү гаргах" гэгийг "дэлгэцлэх" гэж товчоор нэршилээ.

<sup>18</sup> DOS -Disk Operating System-Дискийн үйлдлийн систем

Хоёр операндын арифметик операторыг утга оноох оператортой хамт хэрэглэх шаардлага гарвал тэдгээрийн оронд утга оноох нийлмэл операторыг хэрэглэхэд тохиromжтой байдаг. Тухайлбал, ор нь хоёр операндын оператор бөгөөд

```
exp1 = exp1 op exp2 ;
```

хэлбэрийн илэрхийлэл байвал түүнийг

```
exp1 op= exp2 ;
```

гэж хураангуйлж болно. Ийм нийлмэл операторыг арифметик оператор дээр түшиглэн үүсгэж болох ба жагсаалтыг нь Хүснэгт 2.2-т үзүүллээ.

*Хүснэгт 2.2: C++ хэлний нийлмэл операторын жишиг*

Оператор	Хураангуйгаар	Дэлгэрэнгүйгээр
$*=$	$x *= y$	$x = x * y$
$-=$	$y -= z + 1$	$y = y - z + 1$
$/=$	$a /= b$	$a = a / b$
$+=$	$x += y / 8$	$x = x + y / 8$
$\% =$	$y \% = 3$	$y = y \% 3$

### 2.1.5 Арифметик оператор

Арифметик үйлдлийн (+) нэмэх, (-) хасах, (\*) үржих, (/) хуваах зэрэг операторыг бүхэл болон бодит тоон, харин (%) модуль олох операторыг зөвхөн бүхэл тоон өгөгдөл тус тус хэрэглэдэг. Бүхэл тоог бүхэл тоонд хуваахад гарах үлдэгдлийг олоходоо модулийн операторыг хэрэглэнэ.

Арифметик операторын жагсаалтыг Хүснэгт 2.3-т үзүүллээ. Эдгээр оператор нь бүгд хос операндынх юм.

*Хүснэгт 2.3: C++ хэлний арифметик операторын жагсаалт*

Оператор	Зориулалт	Жишээ
+	Нэмэх	$7 + 2 \rightarrow 9$
-	Хасах	$7 - 2 \rightarrow 5$
*	Үржих	$7 * 2 \rightarrow 14$
/	Хуваах	$7 / 2 \rightarrow 3$
%	Үлдэгдэл олох	$7 \% 2 \rightarrow 1$

(a) хос операндын оператор

Оператор	Зориулалт	Жишээ
$++$	Нэгээр нэмэгдүүлэх	$++x, x++$
$--$	Нэгээр хорогдуулах	$--x, x--$

(b) ганц операндын оператор

Ихэнх програмд аль нэгэн хувьсагчийн уттыг нэгээр нэмэгдүүлэх эсвэл хорогдуулах шаардлага байнга гарах ба ийм тохиолдолд объектын уттыг нэгээр нэмэгдүүлэх эсвэл хорогдуулах операторыг хэрэглэвэл тохиromжтой байдаг (Хүснэгт 2.3(b)). Эдгээр нь цөм нэг операндынх юм. Иймд

```
++x ;
x++ ;
--y ;
y-- ;
```

командуудыг

```
x = x+1 ;  
y = y-1 ;
```

командуудаар оруулах боломжтой юм.

Дээр үзүүлснээр ганц операндын арифметик оператор нь утгвар, дагавар гэсэн хоёр янз байх ба тэдгээр нь хоорондоо эрс ялгаатай юм. Тухайлбал, нийлмэл илэрхийлийн хувьд `++` команда нь а хувьсагчийн уттыг нэгээр нэмэгдүүлсний дараагаар ээлжит үйлдлийг хийх бол `++` команда нь ээлжит үйлдлийг хийсний дараагаар а хувьсагчийн уттыг нэгээр нэмэгдүүлийн. Жишээ нь,

```
x = 10 ;  
y = ++x ;
```

гэсэн хоёр команд хийгдсний дараагаар `x` ба `y` нь харгалзан

```
x=11, y=11
```

болно. Харин

```
x = 10 ;  
y = x++ ;
```

гэсэн хоёр командын хувьд `x` ба `y` нь харгалзан

```
x=11, y=10
```

болно. Үүнтэй төстэйгээр `(--)` оператор нь хувьсагчийн уттыг нэгээр хорогдуулдаг.

### 2.1.6 Харьцуулах оператор

Хоёр зүйлийн хооронд харьцуулалт<sup>19</sup> хийх замаар тэдгээрийн хооронд өгөгдсөн харьцаа байгаа эсэхийг шалгаж тогтоох шаардлага програмд байнга гардаг. Хоёр operand (утга) хооронд харьцуулалт хийхдээ Хүснэгт 2.4-д жагсаасан харьцуулах операторуудаас аль тохирохыг нь хэрэглэнэ.

Хүснэгт 2.4: Харьцуулах операторын жагсаалт

Оператор	Илэрхийлэх утга
<code>&gt;</code>	Их
<code>&lt;</code>	Бага
<code>==</code>	Тэнцүү
<code>!=</code>	Тэнцүү биш
<code>&gt;=</code>	Их эсвэл тэнцүү; бага биш
<code>&lt;=</code>	Бага эсвэл тэнцүү; их биш

### 2.1.7 Логик оператор

Юмс хооронд тодорхой харьцаа байгаа эсэхийг шалгаж тогтооход хэрэглэх харьцааны операторын талаар өмнөх бүлэгт үзсэн билээ. Олон харьцааг нэг дор зэрэгцүүлэн шалгах бол логик операторыг хэрэглэнэ.

C++ хэлэнд хоёр operandын (`&&` ба `||`), нэг operandын (`!`) логик оператор байдгийг Хүснэгт 2.5-д үзүүллээ.

<sup>19</sup> Хоёр юмны хоорондын харьцааг тогтооход хэрэглэх операторыг бас харьцааны оператор гэж болно.

Хүснэгт 2.5: Логик операторын жагсаалт

Оператор	Илэрхийлэх утга	Тайлбар
&&	ба, бөгөөд	Олон харьцаа зэрэг биесэлж байгаа эсэхийг шалгахад хэрэглизнэ.
	эсвэл, буюу	Олон харьцаанаас ядаж нэг нь биесэлж байгаа эсэхийг шалгахад хэрэглизнэ.
!	Урвуу	Урвуу утгыг олох

### 2.1.8 if-else команд

C++ объектын тухайн утга ямар байгаагаас хамаарч өөр өөр үйлдэл хийх бол тохиорх сонголтыг хийхдээ if-else командыг хэрэглэнэ. Энэ командыг хэрэглэх олон хувилбарыг доор үзүүллээ.

- a. if(шалгах зүйл)
 

```
командын мөр ;
```
- b. if(шалгах зүйл)
 

```
командын мөр ;
      else
      командын мөр ;
```
- c. if(шалгах зүйл)
 

```
{
      командын мөр_1 ;
      //-----
      //-----
      командын мөр_N ;
  }
```
- d. if(шалгах зүйл)
 

```
{
      командын мөр_1 ;
      //-----
      //-----
      командын мөр_N ;
  }
  else
  {
      командын мөр_1 ;
      //-----
      //-----
      командын мөр_N ;
  }
```

Дээрхэс a, c нь шалгаж байгаа зүйлийн хариу зөвхөн үнэн байхад хийх командын хэсэгтэй байхад b, d нь шалгаж байгаа зүйлийн хариу үнэн, худал байх аль ч тохиолдолд хийх командын хэсэгтэй байна. Иймд a, c нь алгоритмын гүйцэд бус; b, d нь гүйцэд салаалах алхамд тохиорх хувилбарууд юм.

Шалгах зүйлийн хариу яаж гарахаас хамаарч ганц үйлдэл хийх бол эхний хоёр, олон үйлдэл хийх бол сүүлийн хоёр загварын аль тохиорохыг нь тус тус хэрэглэнэ.

Дээрх if-else командыг хэрхэн хэрэглэхийг гурван бүхэл тооноос хамгийн багыг нь олох Program #2.2-оор үзүүллээ.

```
//Program #2.2
//Finding the smallest integer (first method)
```

```

#include <iostream.h>
#include <conio.h>

void main()
{
    int a, b, c ;
    cout << "\nEnter three integers: " ;
    cin >> a >> b >> c;

    if (a < b && a < c)
        cout << "\nThe smallest integer is " << a ;

    if (b < c && b < a)
        cout << "\nThe smallest integer is " << b ;

    if (c < a && c < b)
        cout << "\nThe smallest integer is " << c ;
    getch();
}

Enter three integers: 6..J
8..J
3..J
The smallest integer is 3

```

Дээрх програмыг Program #2.3-т үзүүлснээр өөрчлөн бичиж болох ба энэ аргыг н тооны хувьд хэрэглэхээр өргөтгөх боломжтой тул дээрх аргаас давуу юм. Мен гарас унших гурван тоог дээрх програмын үр дүнгийн хэсэгт үзүүлсэн шигээр тус бүрд оруулж болохоос гадна хооронд нь зайгаар зааглаж оруулж болохыг Program #2.3-т харууллаа.

```

//Program #2.3
//Finding the smallest integer (second method)

#include <iostream.h>
#include <conio.h>

void main()
{
    int a, b, c, m ;
    cout << "\nEnter three integers: " ;
    cin >> a >> b >> c;

    if (a < b)
        m = a ;
    else
        m = b ;
    if (c < m)
        m = c ;

    cout << "\nThe smallest integer is " << m ;
    getch();
}

```

```

Enter three integers: 6 8 3..J
The smallest integer is 3

```

Дээрх програмын if-else командын

```
if (a < b)
```

```

    m = a ;
else
    m = b ;
хэсгийг доор үзүүлснээр (?:) нөхцөлт оператороор
    m = (a < b) ? a : b ;
гэж оруулан бичиж болно. Энд (a < b) илэрхийлийн хариуг шалгаж (?), хариу нь үнэн
гарвал a-г, хариу нь худал гарвал b-г тус тус буцаах бөгөөд түүнийг (=) оператороор м рүү
хийнэ. Дээрх командын мөрийг өөрөөр
    (a < b) ? m = a : m = b ;
гэж бичиж болно.

```

### 2.1.9 switch-case команд

Шалгах зүйл нь нэгээс цөөнгүй хариутай бол тухайн хариу утга ямар байхаас хамаарч хийгдэх бололцоотой олон бүлэг командаас аль тохиолдолд нь хийгдэхийг тогтоож өгөхөд switch-case командаыг хэрэглэж болно. Өөрөөр хэлбэл, тухайн хувьсагчийн утга ямар байгаагаас хамааран ялгаатай үйлдэл хийх бол энэ командаыг хэрэглэхэд тохиромжтой байдаг. Командаыг хэрэглэх загварыг доор үзүүллээ.

```

    ↓           char эсвэл int-терлийн хувьсагч, илэрхийлэл
switch(expression)
    ↓           бүхэл тоон эсвэл тэмдэгтэн тогтмол
    case template_1:
        ↓           эхний template_1 тохиолдолд хийх бүлэг команд
        statement;
        statement;
        break;      } break командаар switch командаас гарах
    case template_2:
        ↓           удаах template_2 тохиолдолд хийх бүлэг команд
        statement;
        statement;
        break;
        :
        :
        :
    case template_n:
        ↓           n-дүгээр template_n тохиолдолд хийх бүлэг команд
        statement;
        statement;
        break;      } бусад тохиолдолд хийх бүлэг команд
    default:
        ↓           default: команда
        statement;
        statement;
    }

```

Шалгах expression нь эрхийллийг switch тусгай үгийн дараах () дугуй хаалтанд, програмд авч үзүүх ёстой бүх case тохиолдлыг () багц командын хаалтанд тус тус бичнэ.

Дээрх загварын хувьд expression нь char, int, long гэх мэтчилэнгийн бүхэл тоон хариутай дурын илэрхийлэл байна. switch команд нь expression илэрхийлийн хариу утлыг case тусгай үгсийн ард байх template\_1, ... template\_n утгуудтай харьцуулж шалгана. Илэрхийлийн хариу нь аль нэгэн case-ийн template\_x (x∈{1..n})-тэй дүйж байвал тухайн template\_x-ийн давхар цэгийн дараах командуудыг, харин аль ч template-тэй дүйхгүй бол default хэсгийн командаыг тус тус хийнэ. Харин default

хэсгийг тодорхойлж өгөөгүй байвал програм нь хаах их хаалтын дараах командаас үргэлжлэн ажиллана.

Загварын дагуу тодорхойлсон switch-case командаын хувьд шалгах зүйлийн харин нь аль иэтэн template\_x-тэй тэнцэж байвал түүнд харгалзах давхар цэгийн дараах командаас гадна түүний дараах бүх case-д харгалзах командууд хийгдэнэ. Иймд хийгдэх командыг case хооронд нь зааглаж өгөхийн тулд програмын удирдлагыг switch-case дотроос гаргах break командааг case тус бүрд харгалзах командын сүүлийн команд болгож хэрэглэнэ.

switch-case командаыг хэрхэн хэрэглэхийг Program #2.4-д үзүүллээ. Энэ програм нь компьютерийн гараас оруулах тоонд харгалзах сарын нэрийг дэлгэцлэнэ.

```
//Program #2.4
//Finding the month name

#include <iostream.h>
#include <conio.h>

void main()
{
    int n;
    cout << "\nEnter an integer [1..12]: ";
    cin >> n ;
    cout << "\nThe corresponding month is ";
    switch(n)
    {
        case 1: cout << "January" << endl20 ;
                  break ;
        case 2: cout << "February" << endl ;
                  break ;
        case 3: cout << "March" << endl ;
                  break ;
        case 4: cout << "April" << endl ;
                  break;
        case 5: cout << "May" << endl ;
                  break ;
        case 6: cout << "June" << endl ;
                  break ;
        case 7: cout << "July" << endl ;
                  break ;
        case 8: cout << "August" << endl ;
                  break ;
        case 9: cout << "September" << endl ;
                  break ;
        case 10:cout << "October" << endl ;
                  break ;
        case 11:cout << "November" << endl ;
                  break ;
        case 12:cout << "December" << endl ;
                  break ;
        default: cout << "\nIt is out of range" << endl ;
    }
}
```

<sup>20</sup> endl бол дэлгүүрч хэсгийн зуулгийн араас шинэ морийн '\n' тэмдгийг ижмэж биччээлэвэл үүр албадан гаргах үүрэгтэй объект. Эх загвар нь ostream & endl юм.

```

        getch() ;
    }
    Enter an integer [1..12]: 10
    The corresponding month is October

```

Дээрх програмын case тус бүрд break команда хэрэглээгүй үед дараах үр дүн гарна.

```

    Enter an integer [1..12]: 10
    The corresponding month is October
    November
    December

```

### 2.1.10 for команда

Програмд зарим командыг олон удаа давтах хэрэгцээ гарах ба ийм үед for команда хэрэглээгүй үед дараах загварын дагуу хэрэглэнэ.

- Нэг команд давтахад

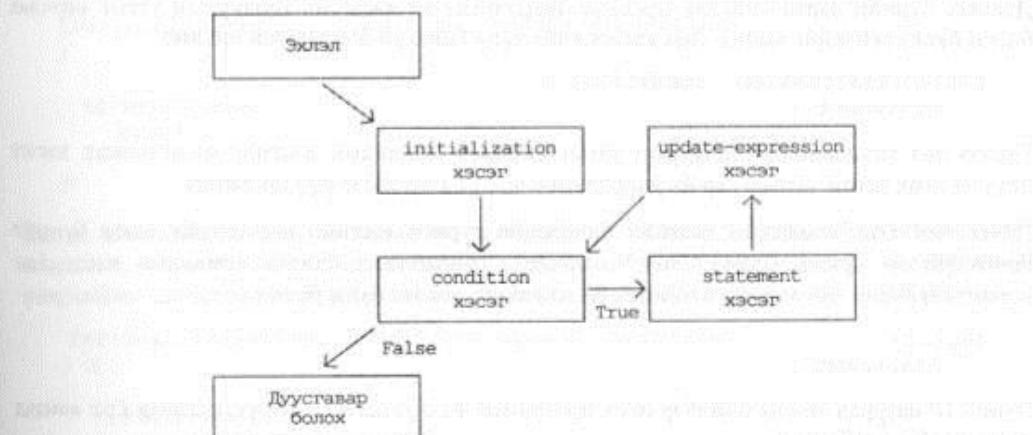
```
for(initialization; condition; update-expression)
    statement ;
```

- Олон команд давтахад

```
for(initialization; condition; update-expression)
{
    statement_1 ;
    .
    .
    statement_n ;
}
```

Дээрх хоёр загварын хувьд for командын initialization; хэсэг нь давталтын тоолуур болон бусад зүйлсийг ажилд бэлдэх, condition; хэсэг нь давталтыг үргэлжлүүлэх нөхцөл биелэгдэж байгаа эсэхийг шалгах, update-expression нь тоолуурын утгыг өөрчлөх үүрэгтэй юм. Эдгээрийн хийгдэх дараалал Зураг 2.1-д үзүүлсэн шиг байна.

```
for(initialization; condition; update-expression)
    statement;
```



Зураг 2.1: for команда хийгдэх дараалал

Эхэлж initialization хэсэг хийгдэх ба ихэвчлэн давталыг тоолох тоолуурын хувьсагчийг ажилд бэлдэж утга оноох командыг энд бичиж өгнө. Олон команд бичих шаардлагатай бол доор үзүүлсэн шигээр хооронд нь таслалаар зааглана. Энэ хэсгийн сүүлийн командын ард ( ; ) цэгтэй таслал заавал байх ёстай.

```
for(init_1, init_2; condition; update-expression)
    statement ;
```

Нэг ч команд энэ хэсэгт бичих шаардлагагүй бол зөвхөн цэгтэй таслалыг доор үзүүлсэн шигээр бичнэ.

```
for(; condition; update-expression)
    statement ;
```

Давталтын бэлтгэл хэсэг болох initialization хэсгийн дараагаар condition хэсэг хийгдэж давталтыг үргэлжлүүлэх нөхцөлийг шалгана. Хариу нь үнэн (тэгээс ялгаатай) гарч байвал давталт үргэлжилинэ; бусад тохиолдолд давталт дуусгавар болж програмын удирдлага for командын дараах команд руу шилжиж очно (Зураг 2.1). Олон зүйлийг шалгах бол аль тохирох логик оператороор, жишээ нь доор үзүүлсэн шигээр холбож өгнө. Энэ хэсгийн сүүлийн илэрхийлэл нь ( ; ) цэгтэй таслалаар төгсөх ёстай.

```
for(initialization; cond_1 && cond_2; update-expression)
    statement ;
```

Ямар ч шалгах илэрхийлэл шаардлагагүй бол зөвхөн цэгтэй таслалыг бичнэ.

```
for(initialization; ; update-expression)
    statement ;
```

Шалгах хэсгийн хариу зөв гарах үед хийгдэх statement хэсэг нь ихэвчлэн нэг, түүнээс олон командтай байна. Олон командыг заавал () багц командын хаалтанд бичнэ. Харин давтах нэг ч команд байхгүй бол зөвхөн ( ; ) цэгтэй таслал бичиж өгнө.

```
for(initialization; condition; update-expression) ;
```

Програм дотор түр зогсолт хийхэд энэ хувилбарыг доорх шигээр хэрэглэх боломжтой юм.

```
for(int i=1; i<101; i++) ;
```

Давталт бүрийн дараа хийгдэх update-expression хэсэг нь тоолуурын утгыг өөрчлох болон бусад зүйлсийг хийнэ. Энэ хэсэгт хийх зүйл байхгүй бол хоосон орхино.

```
for(initialization; condition; )
    statement ;
```

Гэхдээ энэ тохиолдолд тоолуурын утгыг өөрчлох командын хэсгийг statement хэсэгт оруулж өгөх ёстай. Дараагаар нь удирдлага condition хэсэг руу шилжинэ.

Давталтын for командын толгойн бичдэсийн гурван хэсгийн аль нэгийг эсвэл бүгдийг бичихгүйгээр орхиж болно. Энэ тохиолдолд давталтанд бэлдэх командын хэсэг for командын өмнө, бусад хоёр хэсгийнх statement хэсэгт байж болно.

```
for(;;)
    statement ;
```

Эхний 10 натурал тооны нийлбэр олох програмыг Program #2.5-д үзүүлснээр for команд хэрэглэн бичиж болно.

```
//Program #2.5
//Finding sum of first 10 natural numbers (for loop)
```

```

#include <iostream.h>
#include <conio.h>

void main()
{
    int n, s = 0 ;
    for (n=1; n<=10; n++)
        s += n ;

    cout << "\n1+2+3+...+10 = " << s ;
    getch();
}

```



1+2+3+...+10 = 55

Дээрх програмын `s+=n;` командааг өөрөөр `s=s+n;` гэж бичиж болох ба эл команд нь п-ийн утга 10-аас бага эсвэл тэнцүү байвал түүнийг с дээр нэмж хуримтлуулах үйлдлийг давтан хийнэ. Харин п-ийн утга 11 болоход `for` давталт дуусгавар болж улмаар түүний дараах `cout` команд

```

cout << "\n1+2+3+...+10 = " << s ;
хийгдээз.
```

for давталтын тоолуурын утга дээрх програмд хэрэглэсэн шиг есч болохоос гадна доор үзүүлсэн шигээр буурч болно.

```
    for (n=10; n>=1; n--)
```

Мен тоолуурын хувьсагчийн утyg зөвхөн нэгээр бус, харин түүнээс их алхмаар (тоогоор) доор үзүүлсэн шигээр

```
    for (n=1; n<=10; n+=2)
```

нэмэгдүүлж, хорогдуулж болно. Энд 2 бол п-ийн утyg өөрчлөх алхам юм.

Давталт дуусгавар болоогүй байхад програмын удирдлагыг тодорхой нөхцөлийн улмаас for команд дотроос гаргахад `break` командааг доорх загварын дагуу хэрэглэнэ.

```

for(initialization; condition; update-expression)
{
    ...
    if(expression)
        break ;
    ...
}
```

Давтах командаан хэсгээс заримыг нь тодорхой нөхцөлийн улмаас хийхгүйгээр алгасах шаардлага гарвал `continue` командааг хэрэглэх бөгөөд энэ команд нь програмын удирдлагыг давталтын эх рүү, тухайлбал `update-expression` хэсэг рүү шилжүүлнэ.

```

for(initialization; condition; update-expression)
{
    ...
    if(expression)
        continue ;
    ...
}
```

Дээр үзсэн `break` болон `continue` командыг давталтын `while`, `do-while` командаад дээрх жишгээр бас хэрэглэж болно.

### 2.1.11 while команд

Командыг давтах тоог урьлаас мэдэхгүй ч давталтыг зогсоох нөхцөл тодорхой байх тохиолдолд нөхцөлт давталтын `while` командаға дараах загварын дагуу хэрэглэнэ.

- Ирг команд давтахад

```
while (condition)
    statement ;
```

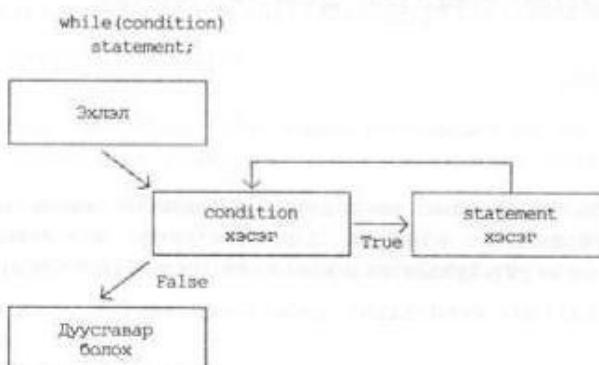
- Олон команд давтахад

```
while (condition)
{
    statement_1 ;
    ...
    statement_n ;
}
```

Давталтын `for` командаад хэрэглэдэг шиг тоолуурыг ажилд бэлдэх `initialization`, тоолуурыг өөрчлох `update-expression` хэсгүүдийг `while` командаад доор үзүүлснээр нэмж болно.

```
initialization ;
while (condition)
{
    statement ;
    update-expression ;
}
```

Зураг 2.2-т үзүүлсэн дарааллын дагуу `while` командаын бүрдэл хэсгүүд хийгдэнэ. Тухайлбал, `condition` илэрхийлийн хариу үнэн (тэгээс ялгаатай) эсэхийг шалгаж зөв хариу гарвал `statement` хэсгийг давтаж хийгээд шалгах `condition` илэрхийлийн хэсэг рүү буцаж очно. Шалгах илэрхийлийн хариу тэгтэй тэнцүү болоход давталт дуусгавар болж програмын удирдлага `while` командаын дараах команд руу шилжиж очно.



Зураг 2.2: `while` командаын хийгдэх дараалал

Өмнөх Program #2.5-д бага зэргийн өөрчлөлт хийх замаар Program #2.6-г доор үзүүлснээр `while` команда хэрэглэн бичиж болно.

```

//Program #2.6
//Finding sum of first 10 natural numbers (while loop)

#include <iostream.h>
#include <conio.h>

void main()
{
    int n = 1, s = 0 ;
    while (n<=10)
    {
        s += n ;
        n++ ;
    }
    cout << "\n1+2+3+...+10 = " << s ;
    getch();
}

```



1+2+3+...+10 = 55

Дээрх програмын үр дүн нь Program #2.5-ынхтай адилхан байна.

### 2.1.12 do-while команд

Давталтын do-while нь өмнө үзсэн while командтай төстэй команд юм. Гэхдээ тэдгээрийн хооронд зарчмын гэмэр ялгаа байдаг. Тухайлбал, while командын эхэнд байх шалгах нөхцөл биелэхгүй байвал давтах командын хэсэг огт хийгдэхгүйгээр түүний дараах командаас програм үргэлжлэн ажиллана. Гэтэл do-while командын хувьд шалгах хэсэг давтах хэсгийн дараагаар байрших учир давтах хэсэг нь шалгах зүйл ямар байгаагаас хамаарахгүйгээр ядаж нэг удаа заавал хийгдэнэ. Энэ командаға доор үзүүлсэн загварын дагуу хэрэглэнэ.

- Ганц команд давтахад

```

do
    statement ;
    while (condition);

```

- Олон команд давтахад

```

do
{
    statement_1 ;
    .
    .
    .
    statement_n ;
} while (condition) ;

```

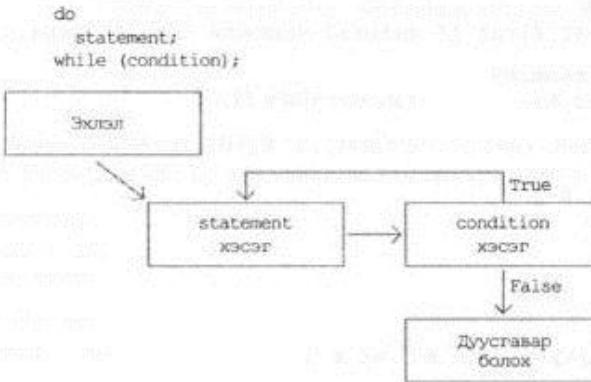
Энэ командаын хэсэг тус бүрийн хийгдэх дарааллыг Зураг 2.3-т үзүүллээ.

Давталтанд бэлдэх initialization, тоолуурыг өөрчлөх update-expression гэсэн хэсгүүдийг do-while командаад доор үзүүлснээр нэмж болно.

```

initialization ;
do
{
    statement ;
    .
    .
    .
    update-expression ;
} while (condition) ;

```



Зураг 2.3: do-while командын хийгдэх дараалал

do-while команда хэрэглэн өмнөх Program #2.6-г бага зэрэг өөрчлөх замаар Program #2.7-г гарган авч болно.

```

//Program #2.7
//Finding sum of first 10 natural numbers (do-while loop)
#include <iostream.h>
#include <conio.h>
void main()
{
    int n = 1, s = 0 ;
    do
    {
        s += n ;
        n++ ;
    } while (n<=10) ;
    cout << "\n1+2+3+...+10 = " << s ;
    getch();
}

```

1+2+3+...+10 = 55

Дээрх програмын үр дүн нь Program #2.5 болон Program #2.6-гийнхтай адил байна.

### 2.1.12 Давхар давталт

Давталт дотроо өөр давталт агуулж байвал тэдгээрийг (багталисан) давхар давталт гэх байм давталтын энгийн тохиолдлуудыг дараах байдлаар бичиж болно.

```

for(initialization; condition; update-expression)
{
    //----- Киймдэлийн хэсэгт байж болно
    for(initialization_2; condition_2; update-expression_2)
    {
        //-----
    }
    //----- Киймдэлийн хэсэгт байж болно
}

```

```

while (condition)
{
    //----- команлын эзслэгийн байх болно
    while (condition_2)
    {
        //-----
    }
    //----- команлын эзслэгийн байх болно
}

while (condition)
{
    for(initialization; condition; update-expression)
    {
        //-----
    }
}

```

Ихэнх програмд давтхар давталтыг хэрэглэх шаардлага гардаг. Ийм програмын нэг толоелол бол 100-ас бага бүх анхны тоог доорх програм юм.

```

//Program #2.8
//Finding all prime numbers below 100

#include <iostream.h>
#include <conio.h>
#include <math.h>
#include <iomanip.h>

void main()
{
    int a, c, k, m ;
    cout << 2 ;
    for (a=3; a<100; a++)
    {
        c=1 ;
        m = sqrt(a);
        for(k=2; k<=m; k++)
            if(a%k == 0)
                c=0 ;
        if(c == 1)
            cout << setw(3) << a ;
    }
    getch() ;
}

```

 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89  
97

Програмын эхэнд байгаа `cout << 2`; командаар 2 гэсэн анхны тоог шууд дэлгэшлэнэ. Гаднах давталтын а хувьсагч  $3+99$  хооронд байх утга авах ба түүний утга бүр анхны тоо мөн эсэхийг доторх давталтанд шалгана. Гаднах давталт 97 ( $99-3+1$ ) удаа хийгдэнэ. Доторх давталт эхлэхийн өмнө с хувьсагчийг 1 болгоно. Энэ давталт хэдэн удаа хийгдэх нь т-ээс хамаатай. Жишээ нь,  $a=37$  үед 37 гэсэн тооны хувьд квадрат язгуур нь 6 байх тул т=6 болно. Иймд

```

if(a%k == 0)
    c=0 ;

```

командаар 37 нь 2, 3, 4, 5, 6 гэсэн таван тоонд үлдэгдэлгүй хуваагдах эсэхийг шалгахад тэдгээрийн алинд ч үлдэгдэлгүй хуваагдахгүй тул с хувьсагчийн утга өөрчлөгдхгүй 1 хэвээр байна. Доторх давталт дуусгавар болсны дараа

```
if(c == 1)
    cout << setw(3) << a ;
```

командаар 37 гэсэн тоог дэлгэцлэнэ. Энд setw(3) функц нь а хувьсагчийн утгыг 3 оронгоор хэвлэхийг заана.

### 2.1.13 Хүснэгт

Бараг програм бүрд ямар нэгэн хувьсагч хэрэглэдэг. Зарим тохиолдолд хоорондоо логик холбоотой нэгэн төрлийн олон хувьсагч хэрэглэх шаардлага гарах тул тэдгээрийг хүснэгт хэлбэрээр зохион байгуулах нь илүү тохиромжтой байдаг.

Хүснэгт бүр өөрийн гэсэн нэртэй байна. Хүснэгт бүтвэртэй байх ба түүний нэр нь хүснэгтийн нэрийн араас залгуулан хүснэгтийн оператор гэгдэх [] дунд хаалтанд бичиж өгөх эзэрг бүхэл тоон дугаараас тогтоно. Арван бүхэл тоон бүтвэртэй а хүснэгтийг доор үзүүлснээр байгуулна.

```
int a[10] ;
```

Энэхүү а хүснэгт нь a[0], a[1], a[2], ...a[9] гэсэн нэр бүхий 10 бүтвэртэй. C++ компайлер дээрх командын дагуу 10 бүхэл тоонд зориулан дараалан байрших ой бэлдэнэ (Зураг 2.4).

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]

Зураг 2.4: 10 бүхэл тоон бүтвэртэй а хүснэгт

Дунд хаалтанд бичигдсэн 0, 1, 2, ..., 9 гэсэн 10 ширхэг тоо бол бүтвэр бүрд харгалзах дугаар<sup>21</sup>, индекс юм. Тухайн бүтвэр хүснэгтийн эхнээс ямар зайд оршин байгааг түүний дугаар нь заана. Дээрх а хүснэгт нь нэг хэмжээстэй хүснэгт юм.

Өгөгдсөн гурван тооны багыг олох Program #2.3-ыг 10 бүхэл тоонд тохируулан Program #2.9-д үзүүлснээр өөрчилж болно.

```
//Program #2.9
//Finding the smallest from given 10 integers
#include <iostream.h>
#include <conio.h>
void main()
{
    int a[10], k, m ;
    cout << "\nEnter 10 integers: " ;
    for (k=0; k<10; k++)
        cin >> a[k] ;
    m = a[0] ;
    for (k=1; k<10; k++)
        if (a[k]<m) m = a[k];
    cout << "\nThe smallest integer is " << m << endl ;
    getch();
}
```

<sup>21</sup> Н бүтвэртэй хүснэгтийн эхийн бүтвэрийн дугаар 0, сүүлийн бүтвэрийн дугаар N-1 байна.

 Enter 10 integers: 45 67 23 90 12 89 56 23 78 34.  
 The smallest integer is 12

Гараас унших 10 тоог тус бүрд нь оруулж ↓ товчоор дуусгаж болдог. Эхний `for` команд хийгдэх явцад 10 тоог гараас оруулах бөгөөд тэдгээр нь а хүснэгтэнд Зураг 2.5-д үзүүлснээр хадгалагдана.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
45	67	23	90	12	89	56	23	78	34

Зураг 2.5: а хүснэгтийн дүрслэл

Олох бага тоог хадгалах түүхийн хувьсагч руу хүснэгтийн эхний бүтвэрийг хийх команд бол `m=a[0];` юм. Дараагаар нь т-ийг бусад бүтвэртэй жишиэ. Ингэхдээ хэрэв т-ээс бага утга бүхий бүтвэр олдвол түүнийг бага тоо болгоно. Дотор давталт хийгдэх үед т, к хоёр хувьсагчийн авах утгыг Хүснэгт 2.5-д харгалзуулан харууллаа.

Хүснэгт 2.6:Дотор давталтын үед k, т хувьсагчудын авах утга

k	m
0	45
1	45
2	23
3	23
4	12
5	12
6	12
7	12
8	12
9	12

Бизнесийн холбогдолтой баримт бичиг дотор тоог үгээр бичих шаардлага байнга гардаг. Дараах програмаас `switch-case`, `for` хоёр командыг хүснэгттэй хэрхэн холбож хэрэглэхийг харж болно.

```
//Program #2.10
//converting an integer into words
#include <iostream.h>
#include <conio.h>

void main()
{
    int a[5], k;
    long n;
    cout << "\nEnter an integer (up to five digits) : ";
    cin >> n;
    for (k=0; k<5; k++)
    {
        a[k] = n % 10;
        n = (n - a[k]) / 10;
        if (k == 1 || k == 4)
        {
            if (a[k] > 1)
                a[k] *= 10;
            if (a[k] == 1)
                {
                    if (k == 1)
                        cout << "one ";
                    else
                        cout << "ten ";
                }
            else
                cout << a[k] << " ";
        }
    }
}
```

```

        a[k] = 10 + a[k-1] ;
        a[k-1] = 0;
    }
}
for (k=4; k>=0; k--)
{
    switch (a[k])
    {
        case 1: cout << "One " ; break ;
        Case 2: cout << "Two " ; break ;
        case 3: cout << "Three " ; break ;
        case 4: cout << "Four " ; break ;
        case 5: cout << "Five " ; break ;
        case 6: cout << "Six " ; break ;
        case 7: cout << "Seven " ; break ;
        case 8: cout << "Eight " ; break ;
        case 9: cout << "Nine " ; break ;
        case 10: cout << "Ten " ; break ;
        case 11: cout << "Eleven " ; break ;
        case 12: cout << "Twelve " ; break ;
        case 13: cout << "Thirteen " ; break ;
        case 14: cout << "Fourteen " ; break ;
        case 15: cout << "Fifteen " ; break ;
        case 16: cout << "Sixteen " ; break ;
        case 17: cout << "Seventeen " ; break ;
        case 18: cout << "Eighteen " ; break ;
        case 19: cout << "Nineteen " ; break ;
        case 20: cout << "Twenty " ; break ;
        case 30: cout << "Thirty " ; break ;
        case 40: cout << "Forty " ; break ;
        case 50: cout << "Fifty " ; break ;
        case 60: cout << "Sixty " ; break ;
        case 70: cout << "Seventy " ; break ;
        case 80: cout << "Eighty " ; break ;
        case 90: cout << "Ninety " ; break ;
    }
    if ((k == 3) && (a[4] !=0 || a[3] !=0))
        cout << "Thousand " ;
    if (k == 2 && a[2] !=0)
        cout << "Hundred " ;
    getch();
}

```

 Enter an integer (up to five digits) : 74915.  
Seventy Four Thousand Nine Hundred Fifteen

Дээрх програмд эхний for нь давтарт компьютерийн гараас оруулах тоог шифруулээр нь салгаж а хүснэгт рүү хийх юм. Ингэхдээ a[0]-д иżгийн оронг, a[1]-д аравтын оронг гэх мэтээр хийнэ.

Гараас оруулах тоо хамгийн ихдээ таван оронтой байх тул хоёрдахь **for** команд 5 удаа хийгдэй. Энэ давталт дотор нэг **switch**, хоёр **if** командыг а хүснэгтийн уттанд харгалзах угсийг дэлгэшлэхэд хэрэглэй.

Хүснэгт нь нэг, хоёр, гурав гэх зөргээр олон хэмжээтэй байж болно. Хүснэгт нэг хэмжээтэй бол нэг хүснэгтийн оператор, хоёр хэмжээтэй бол хоёр хүснэгтийн оператор гэх мэтчилэнгээр шаардагдана. Зарим мэдээллийг олон мөр, баганатай хүснэгт хэлбэрээр дүрслэхэд тохиromжтой байх тул хоёр хэмжээтэй хүснэгтийг их хэрэглэдэг. Ийм хүснэгтийг, жишээ нь таван хичээлийн дүнгийн голчийг дөрвөн юуutan тус бүрээр олох боллогод хэрэглэж болох талтай. Үүнийг доор үзүүлснээр програмчилна.

```
//Program #2.11
//Finding average scores of four students

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>

void main()
{
    int j, k;
    float a[4][6] ;
    for (j=0; j<4; j++)
    {
        cout << endl ;
        for (k=0; k<5; k++)
        {
            cout << "\nEnter marks for student " << (j+1);
            cout << " and subject " << (k+1) << ":" ;
            cin >> a[j][k];
        }
    }
    for (j=0; j<4; j++)
    {
        a[j][5] = 0 ;
        for (k=0; k<5; k++)
            a[j][5]+= a[j][k];
        a[j][5]/=5;
    }

    cout << endl ;
    cout << setw(9) << "Student" << setw(10) << "Subj.1";
    cout << setw(10) << "Subj.2" << setw(10) << "Subj.3";
    cout << setw(10) << "Subj.4" << setw(10) << "Subj.5";
    cout << setw(10) << "Average" << endl;

    for (j=0; j<4; j++)
    {
        cout << setw(8) << "Student " << (j + 1);
        for (k=0; k<6; k++)
            cout << setw(10) << a[j][k];
        cout << endl;
    }
    getch();
}
```

Enter marks for student 1 and subject 1: 85.  
 Enter marks for student 1 and subject 2: 85.  
 Enter marks for student 1 and subject 3: 85.  
 Enter marks for student 1 and subject 4: 85.  
 Enter marks for student 1 and subject 5: 90.  
  
 Enter marks for student 2 and subject 1: 95.  
 Enter marks for student 2 and subject 2: 75.  
 Enter marks for student 2 and subject 3: 75.  
 Enter marks for student 2 and subject 4: 95.  
 Enter marks for student 2 and subject 5: 85.  
  
 Enter marks for student 3 and subject 1: 65.  
 Enter marks for student 3 and subject 2: 95.  
 Enter marks for student 3 and subject 3: 95.  
 Enter marks for student 3 and subject 4: 85.  
 Enter marks for student 3 and subject 5: 60.  
  
 Enter marks for student 4 and subject 1: 75.  
 Enter marks for student 4 and subject 2: 75.  
 Enter marks for student 4 and subject 3: 85.  
 Enter marks for student 4 and subject 4: 85.  
 Enter marks for student 4 and subject 5: 95.  
  

Student	Subj.1	Subj.2	Subj.3	Subj.4	Subj.5	Average
Student 1	85	85	85	85	90	86
Student 2	95	75	75	95	85	85
Student 3	65	95	95	85	60	80
Student 4	75	75	85	85	95	83

C++ компайлэр main() функцийн доторх float a[4][6]; командаар хоёр хэмжээстэй а хүснэгтийг байгуулна. Энд 4 нь оюутны тоог, удаах 6 нь хичээлийн тоог тус тус заана. Нийт таван хичээл байгаа ба сүүлийн зургаадугаар багана нь голч дүнгийн болно. Програмд for командыг 3 удаа хэрэглэсэнээс эхний for давталтаар оюутан тус бүрийн 5 хичээлийн дунг компьютерийн гараас оруулана. Удаах for командин хэсэг нь оюутан бүрийн голч дунг бодож гаргах юм. Харин сүүлийн for команд нь оюутнуудын дунг хүснэгтлэн дэлгэцэлдэг.

#### 2.1.14 Функц

Том програмыг ихэвчлэн олон функцийд хувааж бичдэг. Өмнө үзсэн Program #2.1 нь зөвхөн main() функцтэй програм юм. Програм хөгжүүлэгчийн тодорхойлох функцийг хэрэглэгчийн функц гэх ба ийм функц нь нэр, буцаах утга, авах параметртэй байна. Функцийг, жишээ нь доор үзүүлснээр зарлана.

```
float sum(int m) ;
```

Дээрх команд нь функц sum нэртэй бөгөөд int төрлийн m параметр авч бодолт хийгээд float төрлийн утга буцаана гэдгийг компайлерт мэдээлнэ. Хэрэв функц утга буцаахгүй бол void тусгай үгийг хэрэглэнэ. Жишээ нь, дээрх sum(int m) функц ямар ч утга буцаахгүй тохиолдолд түүнийг дараах байдлаар зарлах ёстой.

```
void sum(int m) ;
```

Дээрх хоёр тохиолдолд sum() функцийн гаднаас авах параметрийн утга m рүү дамжиж хуулагдана. Ямар ч параметргүй, буцаах утгагүй функцийг, жишээ нь

```
void sum(void);  
void sum();
```

гэж зарлана. Энэ функцийг ямар ч void үргүйгээр sum(); гэж бичвэл энэ нь гаднаас утга авдаггүй бүхэл тоон функци болно.

Эхний n ширхэг натурал тооны нийлбэр олох програмыг хэрэглэгчийн функци хэрэглэн дараах байдлаар бичнэ.

```
//Program #2.12  
//Finding the sum of first n natural numbers (first method)  
  
#include <iostream.h>  
#include <conio.h>  
  
int sum(int);  
void main()  
{  
    int n;  
    cout << "\nTo what extent, the sum is needed ? ";  
    cin >> n;  
    cout << "\n1 + 2 + ... + " << n << " = " << sum(n);  
    getch();  
}  
  
int sum(int m)  
{  
    int k, s=0;  
    for (k=1; k<=m; k++)  
        s += k;  
    return s;  
}
```



To what extent, the sum is needed ? 11

1 + 2 + ... + 11 = 66

Хувьсагчийг эхэлж зарлахгүйгээр хэрэглэж болдоггүйн адил зарим тохиолдолд функцийг зарлахгүйгээр хэрэглэх боломжгүй юм. Иймээс дээрх програмд sum() функцийг програмын эхэнд int sum(int); гэж зарласан бөгөөд түүнийг функцийн эх загвар гэнэ. C++ компайлер програмын эх кодын хaa нэгтээ ийм sum() функци тодорхойлогдсон байх ёстойг энэ командаар мэдэж авна. Харин main() функцийн дерөвдэх командын мөрийн төгсгөл хэсэгт бичиж өгсөн sum(n) бол уг функцийг дуудаж ажиллуулахдаа n хувьсагчийн утгыг аргумент болгон дамжуулж өгснийг заана. Дээрх програмын үр дүнгээс үзүүлэхдээ n=11 байсан ба эл утга т рүү дамжих орно. Функцийг дуудаж хэрэглэхдээ дамжуулж өгч байгаа т бол бодитой, жинхэнэ аргумент, функцийг тодорхойлоходоо зааж өгч байгаа т бол бодит бус, хийсвэр аргумент, параметр юм. Функци ажиллаж эхлэхэд т-д зориулан ой бэлтгэж түүнд 11-ийг хадгална. Дараагаар нь эхний 11 ширхэг натурал тооны нийлбэрийг for давталт хэрэглэн олж (энэ тохиолдолд нийлбэр нь 66 болно.) s рүү хадгална. Эл функци int төрлийнх, өөрөөр хэлбэл int төрлийн утга буцаадаг нь түүний тодорхойлолтын хэсгээс тодорхой харагдана. Функцийн сүүлийн команд болох return s; нь функцийн ажиллагааг дуусгавар болгож s-ийн утгыг буцаах үүрэгтэй команд юм. Тэгэхээр, main() функцияа дотроос sum() функцийг дуудсан байршил руу 66 гэсэн тоо буцаж очно.

Функцийг урьдчилан зарлахгүйгээр бас шууд тодорхойлж болдгийг дараах програмаар үзүүллээ.

```
//Program #2.13
//Finding the sum of first n natural numbers (second method)
#include <iostream.h>
#include <conio.h>
int sum(int m)
{
    int k, s = 0;
    for (k=1; k<=m; k++)
        s += k;
    return s;
}

void main()
{
    int n;
    cout << "\nTo what extent, the sum is needed ? ";
    cin >> n;
    cout << "\n1 + 2 + ... + " << n << " = " << sum(n) ;
    getch();
}
```

 To what extent, the sum is needed ? 11.
 1 + 2 + ... + 11 = 66

Дээрх хоёр програмаас гарах үр дүн адил байна. Сүүлийн програмын эхэнд `sum()` функцийг тодорхойлсон байна. Ер нь, функцийн тодорхойлолт нь түүнийг дуудаж хэрэглэх командын мөрийн өмнө хийгдсэн байвал функцийг урьдчилан зарлах шаардлагагүй. Хэмжээ багатай (цеөн командтай) функцийн хувьд ийм аргыг хэрэглэх нь тохиромжтой. Харин програм олон функцигтэй эсвэл функциүүд хэмжээ томтой (олон командын мөртэй) бол урьдчилан зарлах аргыг хэрэглэх нь тохиромжтой. Ингэснээр, C++ компайлер бүх функцийн нэр, тэдгээрээс бувах утгын төрөл, тэлгээрийн авах аргументийн тоо, дараалал болон төрөл зэргийг програмын эхэнд бүрэн мэдэж авах боломжтой болдог.

Өмнө үзсэн Program #2.12+13 доторх `sum()` функцияг рүү аргумент утгаар дамжина. Функцияг гаднаас авах олон утга, аргументтэй байж болох боловч зөвхөн ганц утга буцааж чадна. Функцийн олон утга буцаах шаардлагатай бол тэдгээрт харгалзах олон аргументийг заалтаар<sup>22</sup> нь дамжуулж авах ёстой. Функцийн параметр утгаар, заалтаар дамжих аргуудын ялгааг Хүснэгт 2.7-д үзүүллээ.

Хүснэгт 2.7: Функцийн параметр ба аргумент хоорондын ялгаа

	Утгаар дамжих	Заалтаар дамжих
1	Параметрийн өөрчлөлт нь бодит аргументтэй нэлээлэхгүй.	Параметр өөрчлөгдвэл бодит аргумент бас өөрчлөгднө.
2	Функцияг зөвхөн ганц утга буцааж чадна.	Функцияг олон утга буцааж чадна.
3	Функцияг нь утга оноох операторын зүүн гар талд байж болдоггүй.	Функцияг нь утга оноох операторын зүүн талд байж болдог.

Функцийн авах уттыг заалтаар дамжуулах аргыг хэрхэн програмчилж болохыг доорх квадрат тэгшитгэлийн язгуур олох програмаас үзэж болно.

<sup>22</sup> Заалт, заалтан хувьсагчийн талаар "C++ хэлний давуу тал" дэл бүлэгээс үзэж болно.

```

//Program #2.14
//Finding the roots of quadratic equation

#include <iostream.h>
#include <conio.h>
#include <math.h>

void roots(int, int, int, float &, float &);

void main()
{
    int a, b, c, d;
    float x1, x2;
    cout << "\nEnter the values of a, b and c: ";
    cin >> a >> b >> c ;
    d = b*b-4*a*c;

    if (d < 0)
        cout << "\nThe roots are not real" ;
    else
    {
        roots(a, b, d, x1, x2) ;
        cout << "\nThe roots are " << x1 << " and " << x2 ;
    }
    getch();
}

void roots(int m, int n, int p, float &r1, float &r2)
{
    r1 = (float) (-n + sqrt(p))/(2*m);
    r2 = (float) (-n - sqrt(p))/(2*m);
}

```

Enter the values of a, b and c: 1 -5 6.  
 The roots are 3 and 2

Программын эхэн хэсэгт дараах

```
void roots(int, int, int, float &, float &);
```

бичдэстүй функцийг зарласан байна. Өгөгдсон тоонуудын хувьд д-ийн утга 1 болно, учир нь  $(-5x-5)-(4x^2)=1$ . Иймд д нь тэгээс бага биш тул if командын else хэсгийн бүлэг команд хийгдэнэ. Энд roots() функцийг дуудаж хэрэглэхдээ түүнд a, b, d гурван хувьсагчийн утгууд болон x1, x2 хоёр хувьсагчийн заалтуудыг дамжуулж өгнө. Энэ тохиолдолд x1 ба r1, x2 ба r2 нь харгалзан тус бүрдээ дундаа ерөнхий нэг ойг зэмшинэ. Тэгшитгэлийн язгууруудыг олж r1, r2-д харгалзуулан хийнэ. Харин main() функцийн сүүлийн cout объектоор x1, x2 гэсэн хоёр хувьсагчийн уттыг дэлгэцлэнэ. Энэ объектоор x1, x2 хоёр хувьсагчийн оронд r1, r2 хоёр хувьсагчийн уттыг main() функц дотроос шууд хэвлэх боломжтүй, учир нь тэдгээр нь roots() функцийн дотоод хувьсагчид юм.

### 2.1.15 Хувьсагч, түүний зэмших ойн ангилал

Хувьсагчийн эзэмших ойн ангилал нь тухайн хувьсагч руу програмын аль хэсгээс хандаж болох, уг хувьсагч хир удаан устахгүй оршихыг заадаг байна. Энэ ангиллыг Зураг 2.6-д үзүүлсэн загварын дагуу тогтоох бөгөөд automatic, register, static гэх зэрэг тусгай үгсийн аль тохирохыг нь хэрэглэнэ.

### 2.1.15.1 Automatic хувьсагч

Энэ төрлийн хувьсагчийг `auto` тусгай үгээр тодотгож зарлана. Өөрөөр зааж өгөөгүй бол функцийн дотоод хувьсагч нь `auto` төрлийнх байна. Тэгэхлээр, уг тусгай үгийг заавал хэрэглэх шаардлагагүй байдаг. Өмнөх бүх жишээ програмд хэрэглэж байгаа хувьсагчууд цөм `auto` төрлийнх юм.



Зураг 2.6: Оин ангилал тоогтоо загвар

Ер нь, ямар ч төрлийн хувьсагчийн хувьд үйлчлэх хүрээ, орших хугацаа гэсэн хоёр ойлголтыг хэрэглэдэг. Доор үзүүлсэн хоёр загварын хувьд хувьсагчийн үйлчлэх хүрээ, орших хугацаа нь адилхан байна.

```
int a, b, c, d;  
ба  
auto int a, b, c, d;
```

- (i) *Үйлчлэх хүрээ*. Хувьсагчийн үйлчлэх хүрээ нь тухайн хувьсагч руу хандаж чадах програмын хэсэгтэй холбоотой ойлголт юм. Дотоод хувьсагч руу зөвхөн тодорхойлогдсон функцияа дотроос нь хандаж болно. Ийм хувьсагч руу өөр функцийн хандаж чадахгүй.
- (ii) *Орших хугацаа*. Функцийн дотоод (`auto`) хувьсагч нь функцийг дуудаж хэрэглэх бүрд автоматаар байгуулагдана. Ийм хувьсагч тодорхойлогдсон функция нь ажиллаж дуусахад мөн устана. Дотоод хувьсагч байгуулагдах, устах хоорондох хугацааг уг хувьсагчийн орших хугацаа гэнэ. Дотоод хувьсагчийн орших хугацаа бол тодорхойлогдсон функцийн нь ажиллах хугацаа юм.

### 2.1.15.2 register хувьсагч

Дотоод хувьсагчид байх бүх шинж `register` хувьсагчид бас байна. Энэ хувьсагч нь компьютерийн үндсэн ойд биш харин CPU `register`<sup>23</sup>-т түшиглэн байгуулагдах тул түүнд хурдтай хандах боломжтой болдог. Ийм хувьсагчийг зарлахдаа `register` тусгай үгийг хэрэглэж, жишээ нь доорх шигээр

```
register int x;
```

гэж зааж өгнө. Систем боломжтой бол x хувьсагчийн оронд CPU регистрийг хэрэглэнэ. Бусад тохиолдолд түүнийг ерийн хувьсагч шигээр байгуулна. Энэ хувьсагчийн үйлчлэх хүрээ, орших хугацаа нь `auto` хувьсагчийнхтай ижил байна.

### 2.1.15.3 Гадаад хувьсагч

Өмнө үзсэн `auto`, `register` хоёр хувьсагч нь ямар нэгэн функция дотор, гадаад (`external`, `global`) хувьсагч нь бүх функцийн гадна талд тодорхойлогдоно. Ийм

<sup>23</sup> CPU нь түйцүүгээ зэлжит команцыг барийнхое мэдэлд байх тусгай ойд хийж авсны дараагаар боловсруулах ба ийм ойг регистр (`register`) гэнэ.

хувьсагч руу програмын бүх функцийн хувьсагчид хэрэглэх тусгай үг байдаггүй. Энэ хувьсагчийг бас срөнхий хувьсагч гэнэ. Түүний үйлчлэх хүрээ, орших хугацаа нь доор үзүүлсэн шиг байна.

- (i) **Үйлчлэх хүрээ.** Гадаад хувьсагч нь тодорхойлогдсон файлынхаа бүх функцийн үйлчилнэ.
- (ii) **Орших хугацаа.** Програм ажиллаж эхлэх мөчид гадаад хувьсагч автоматаар байгуулагдаж програм ажиллаж дусахад устана. Гадаад хувьсагч байгуулагдах, устах хоорондох хугацааг уг хувьсагчийн орших хугацаа гэнэ. Гадаад хувьсагчийн орших хугацаа бол програмын ажиллах нийт хугацаа юм.

#### 2.1.15.4 static хувьсагч

Статик хувьсагчийг байгуулахад static тусгай үгийг хэрэглэнэ. Статик хувьсагч нь static automatic, static external гэсэн хоёр зүйлийн байна. Эхний зүйлийн хувьсагч нь үйлчлэх хүрээний хувьд auto хувьсагчтай, орших хугацааны хувьд global хувьсагчтай тестэй. Тэгэхлээр, уг зүйлийн хувьсагч руу зөвхөн тодорхойлогдсон функцийн хандаж болох боловч ийм хувьсагч нь програмын ажиллагаа дуусгавар болтол хадгалагдаж байдаг. Харин удаах зүйлийн хувьсагчийг эх кодын олон файлтай програмд хэрэглэдэг байна.

Статик хувьсагчийг доор үзүүлснээр байгуулж болно.

```
static int s = 10 ;
static char a = 'A' ;
static float f = 3.5 ;
```

Өмнөх үр дүнгээ байнга хэрэглэх функцийд статик хувьсагчийг ихэвчлэн хэрэглэх тул уг хувьсагчийг байгуулж байхдаа дээр үзүүлсэн шигээр заавал гаралааны утга онооно.

Ойн ангиллын нэгдсэн харьцуулалтыг Хүснэгт 2.8-д үзүүлээ.

*Хүснэгт 2.8: Хувьсагч ба ойн ангилал*

Ангилал	Үйлчлэх хүрээ	Орших хугацаа	Цэвэр-лэгдэх эсэх	Аль сегментых <sup>24</sup>	Зориулалт
Auto	функция	функция	үгүй	стек	Функцийн дотоод хувьсагч
register	функция	функция	үгүй	стек	auto хувьсагчтай адил, гэхдээ түүнийг бодвол хурдан хандалттай
гадаад	файл	программа	0	өгөгдлийн сегмент	Олон функцийнх
static auto	функция	программа	0	өгөгдлийн сегмент	auto хувьсагчтай тестэй, гэхдээ утгаа хадгалж үлдэнэ.

#### 2.1.16 Бүтэй

Өмнөх бүлгүүдэд char, int, float гэх мэт C++ хэлний дотоод суурь төрөл, бас хүснэгтийн талаар судалж програмд хэрэглэж үзсэн билээ. Хүснэгт бол нэгэн ижил төрлийн өгөгдлийн цувааг хадгалж боловсруулалт хийхэд сайн тохирдог зохиомол төрөл юм. Гэвч дурьдсан төрлүүдээр дүрслэж болдоггүй эсвэл дүрслэхэд нэн түвэгтэй олон арван бодлого

<sup>24</sup> Компьютерийн ойг логикоор 64 KB хэмжээтэй олон сегментэд хувааж үзэх бөгөөд үүнийг ассемблер програмчлалд өргөн хэрэглэдэг. Стек (stack), өгөгдэл (data), код (code) гэсэн 3 төрлийн 4 сегмент байна.

байдаг. Жишээ нь, 500 ажилчнтай үйлдвэрийн газрын хувьд ажилчин бүрийн иэр, цалин, тэтгэлэг зэргийг дараах байдлаар жагсааж гаргах шаардлага байнга гардаг.

Ананд Дорж	9000	8000
Бат Энхээ	8000	7000
.	.	.
.	.	.
.	.	.

Энд гурван багана мэдээлэл байгаагаас эхнийх болох ажилчны иэр нь `char`, бусад хоёр нь `int` төрлийн мэдээлэл байна. Өгөгдлийн санд тэдгээрийг өрөнхийд нь талбар (`field`) гэнэ. Бүх ажилчдынх 500 мөр мэдээлэл болох ба ийм мөр бүрийг бичлэг (`record`) гэнэ. Тэгхэлээр, дээрх жагсаалт тус бүрдээ гурван талбар үзүүлэлт бүхий 500 мөртэй байна. Тэдгээрийг хүснэгтэнд хадгалахад, жишээ нь Зөвхөн хүснэгтийг доор үзүүлснээр байгуулах шаардлагатай болох ба энэ нь хүснэгт холимог төрлийнх байж болдогтүйтэй холбоотой юм.

```
char name[500][13] ;
int basicpay[500] ;
int allowance[500];
```

Гэхдээ дээрх шиг хоорондоо логик уялдаатай холимог үзүүлэлтийг нэгтгэж бүтэц гэх нэгэн хийсвэр нэгжээр тодорхойлох боломж C++ хэлэнд байдаг.

#### 2.1.16.1 Бүтцийн тодорхойлолт

Бүтэц бол энгийн хувьсагчуудын хийсвэр нэгдэл юм. Тэдгээр хувьсагч янз бүрийн төрлийнх байж болно. Бүтцэд багтаж өгөгдлийг бүтцийн гишүүн бүтвэр, гишүүн өгөгдөл, товчоор гишүүн гэнэ. Бүтцийг тодорхойлох өрөнхий загварыг доор үзүүлээ.

```
struct structure-name
{
    member_1 ;
    member_2 ;
    .
    .
    member_n ;
};
```

Бүтцийн тодорхойлолт `struct` үгээр эхлэх бөгөөд түүний дараагаар бүтцийн иэрийг (`structure-name`) бичиз. Харин `member_1`, `member_2`, ..., `member_n` бол бүтцийн гишүүд болох хувьсагчууд юм.

Бүтцийн гишүүн нь энгийн хувьсагч, хүснэгт, хаяг болон бүтэц байж болдог. Бүтцийн иэр бусдынхаас ялгаатай байх ёстой. Бүтцийн тодорхойлолт дотор гишүүн бүтвэрт гаранаы утга олгож болдогтүй.

Ажилчны иэр (`name`), суурь цалин (`basicpay`), тэтгэлэг (`allowance`) гэсэн гурван гишүүн бүтвэртэй `employee` бүтцийг дээрх загварын дагуу доорх шигзэр тодорхойлж болно.

```
struct employee
{
    char name [20] ;
    int basicpay ;
    int allowance ;
};
```

Тодорхойлолтын толгойн хэсгийн бичдэс болох `struct employee` бол бүтцийн тодорхойлогч мөр юм. Энд `struct` нь `employee` гэсэн нэртэй бүтэц тодорхойлогдсоныг

C++ компайлерт хэлж “танилцуулах” үүрэгтэй тусгай үг юм. Бүтцийн гишүүд болох name, basicpay, allowance зэрэг хувьсагчуудыг нээх, хаах их хаалт дотор бичнэ. Ардаа цэгтэй таслал бүхий хаах хаалт нь бүтцийн тодорхойлолтын төгсгөлийг заана. Бүтцийн тодорхойлолтоор ямар нэгэн ой бэлдэх үйлдэл хийгдэхгүй, харин түүгээр хувьсагч<sup>25</sup> байгуулах тохиолдолд уг хувьсагчид зориулан ой бэлдэж гарынан утга онооно.

### 2.1.16.2 Бүтцийн гишүүд рүү хандах

Бүтцийн гишүүд рүү variable.member гэсэн загварын дагуу хандана. variable нь бүтцин хувьсагчийн нэр, member нь бүтцийн гишүүн бүтвэрийн нэр юм. Эдгээрийг цэгээр хооронд нь зааглаж бичнэ. Ажилчдын нийт цалинг бодох програмыг бүтэц хэрэглэн дараах байдлаар бичиж болно.

```
//Program #2.15
//finding the gross pay of an employee

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <stdio.h>

struct employee
{
    char name[20];
    int basicpay;
    int allowance;
};

employee emp;

void getdata();
void heading();
void showdata();
void main()
{
    clrscr();
    getdata();
    heading();
    showdata();
    getch();
}

void getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets (emp.name);
    cout << "Enter the Basic Pay:   ";
    cin  >> emp.basicpay;
    cout << "Enter the Allowance:  ";
    cin  >> emp.allowance;
}

void heading()
{
    cout << endl;
    cout << setw(20) << "Employee Name" ;
}
```

<sup>25</sup> Ийм хувьсагчийг бас бүтцин хувьсагч гэнэ.

```

        cout << setw(8) << "Basic" ;
        cout << setw(12) << "Allowance" ;
        cout << setw(8) << "Gross" ;
    }

    void showdata()
    {
        int grosspay ;
        grosspay = emp.basicpay + emp.allowance ;

        cout << endl ;
        cout << setw(20) << emp.name ;
        cout << setw(8) << emp.basicpay ;
        cout << setw(12) << emp.allowance ;
        cout << setw(8) << grosspay ;
    }
}

Enter the Employee Name: Anand Kumar.
Enter the Basic Pay      : 9000.
Enter the Allowance      : 8000.

Employee Name  Basic   Allowance   Gross
          Anand Kumar    9000       8000     17000

```

Дээрх employee бүтцийн тодорхойллын дараах мөрөнд байгаа employee emp; командын emp хувьсагчийг байгуулна. Ингэснээр, ийм хувьсагч руу getdata(), showdata() гэсэн хоёр функции дотроос хандах боломжтой болдог.

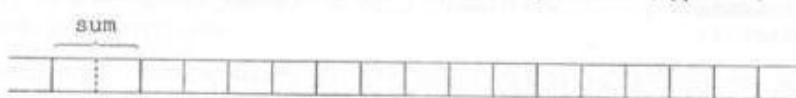
C++ хэлний хувьд char, int, float мэтийн дотоод төрлийн хувьсагчийг доор үзүүлсэн загварын дагуу байгуулж болдгийг өмнө үзсэн.

```
data_type name variable_1, variable_2, . . . ;
```

Энэ загварын дагуу, жишээ нь int төрлийн хувьсагчийг доорх шигээр байгуулна.

```
int sum;
```

Энд int бол C++ хэлний дотоод төрөл, бухэл тоон өгөгдлийг заах тусгай үг тул C++ компайлер sum хувьсагчид зориулан 2 байт ойг автоматаар бэлддэг (Зураг 2.7).



Зураг 2.7: sum хувьсагчийн эзэмших ойн дүрслэл

Дээрхтэй төстэйгээр, employee emp; командаар emp хувьсагч байгуулагдана. Гэхдээ employee нь char, int төрлийн 3 гишүүнтэй бүтэц тул тэдгээрт шаардагдах ойг emp хувьсагчид зориулж бэлдэнэ. Иймд emp бүтцэн хувьсагчийн name гишүүн 20 байтын урттай бол basicpay, allowance нь тус бүрдээ 2 байтынх байх ба emp хувьсагч нь нийтдээ 24 байт хэмжээтэй ойг эзэмшидэг байна (Зураг 2.8).



Зураг 2.8: emp хувьсагчийн эзэмших ойн дүрслэл

Бүтцэн хувьсагч тодорхойлсон бол түүний бүтээрт бүтцийн нэрээр биш, харин бүтцэн хувьсагчийн нэрээр emp.name, emp.basicpay, emp.allowance гэж хандаж болдог. Жишээ нь, showdata() функц дотор нийт цалингийн нийлбэрийг олохдоо

```
grosspay = emp.basicpay + emp.allowance ;  
гэсэн бодолт хийж болно.
```

### 2.1.16.3 Давхар бүтэц

Бүтэц дотор өөр бүтцийн бүтцэн хувьсагч тодорхойлж болдог. Үүнийг өмнөх Program #2.15-д бага зэрэг өөрчлөлт хийх замаар гаргаж авах доорх програмд үзүүлэв.

```
//Program #2.16  
//finding gross pay using nested structures  
  
#include <iostream.h>  
#include <stdio.h>  
#include <conio.h>  
#include <iomanip.h>  
  
struct empname  
{  
    char firstname[10] ;  
    char lastname[10] ;  
};  
  
struct employee  
{  
    empname name ;  
    int basicpay ;  
    int allowance ;  
};  
  
employee emp ;  
void getdata() ;  
void heading() ;  
void showdata() ;  
  
void main()  
{  
    clrscr() ;  
    getdata() ;  
    heading() ;  
    showdata() ;  
    getch() ;  
}  
  
void getdata()  
{  
    cout << "\nEnter the First Name: " ;  
    gets (emp.name.firstname) ;  
    cout << "Enter the Last Name: " ;  
    gets (emp.name.lastname) ;  
    cout << "Enter the Basic Pay: " ;  
    cin >> emp.basicpay ;  
    cout << "Enter the Allowance: " ;  
    cin >> emp.allowance ;  
}
```

```

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8) << "Gross" ;
}

void showdata()
{
    int grosspay ;
    grosspay = emp.basicpay + emp.allowance ;
    cout << endl ;
    cout << setw(10) << emp.name.firstname ;
    cout << setw(10) << emp.name.lastname ;
    cout << setw(8) << emp.basicpay ;
    cout << setw(12) << emp.allowance ;
    cout << setw(8) << grosspay ;
}

 Enter the First Name: Anand.
Enter the Last Name : Kumar.
Enter the Basic Pay : 9000.
Enter the Allowances: 8000.

Employee Name  Basic  Allowance  Gross
      Anand Kumar     9000       8000     17000

```

Дээрх програмд доор үзүүлснээр тодорхойлсон employee бүтцийн хувьд

```

struct employee
{
    empname name ;
    int basicpay ;
    int allowance ;
} ;

```

name бол empname бүтцэн төрлийн гишүүн тул дараах

```

struct empname
{
    char firstname [10] ;
    char lastname [10] ;
} ;

```

empname бүтцийг employee бүтцийн өмнө заавал тодорхойлох ёстай. Иймд employee emp; командаар байгуулсан emp хувьсагчийн хувьд employee бүтцийн name бүтвэрээр дамжуулан empname бүтцийн firstname, lastname гэсэн бүтвэрүүд рүү хандахлаа getdata() функцийн gets(emp.name.firstname); командын мөрөнд хэрэглэсэн шигээр бичик огно. Энд firstname бүтвэр рүү хүрч очихын тулд шууд сонголтын цэг операторыг хоёр удаа хэрэглэсэн байна.

#### 2.1.16.4 Бүтцэн хүснэгт

Хүснэгтийн бүх бүтвэрийн төрөл ижил байдаг талаар өмнө үзсэн билээ. Тэдгээр бүтвэр нь char, int, float гэх зэрэг дотоод төрлийнх байхаас гадна бүтцэн төрлийнх байж болно.

Program #2.15-д бага зэргийн өөрчлөлт хийж турван ажилчны нийт цалинг бодох зориулалттай болгож болно.

```
//Program #2.17
//finding gross pay of 3 employees using array of structures

#include <iostream.h>
#include <stdio.h>
#include <iomanip.h>
#include <conio.h>

struct employee
{
    char name [20];
    int basicpay;
    int allowance;
};

employee emp[3];
void getdata();
void heading();
void showdata();

void main()
{
    clrscr();
    getdata();
    heading();
    showdata();
    getch();
}

void getdata()
{
    for (int k=0; k<3; k++)
    {
        cout << "\nEnter Name of Employee " << k+1 << ":" ;
        gets (emp[k].name);
        cout << "Enter Basic Pay: ";
        cin >> emp[k].basicpay;
        cout << "Enter Allowance: ";
        cin >> emp[k].allowance;
    }
}

void heading ()
{
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
}

void showdata()
{
    int grosspay, k;
    for (k=0; k<3; k++)
    {
        grosspay=emp[k].basicpay+emp[k].allowance;
        cout << endl;
    }
}
```

```

        cout << setw(20) << emp[k].name ;
        cout << setw(8)  << emp[k].basicpay ;
        cout << setw(12) << emp[k].allowance ;
        cout << setw(8)  << grosspay ;
    }
}

 Enter Name of Employee 1: George.
Enter Basic Pay: 9000.
Enter Allowance: 8000.

Enter Name of Employee 2: Bill.
Enter Basic Pay: 8000.
Enter Allowance: 7000.

Enter Name of Employee 3: Enkhbayar.
Enter Basic Pay: 7000.
Enter Allowance: 5000.

Employee Name      Basic   Allowance   Gross
    George        9000     8000    17000
        Bill         8000     7000    15000
    Enkhbayar      7000     5000    12000

```

Гараас оруулах мэдээллийг програмын эхнд тодорхойлсон бүтцэн хүснэгтэд хуримтлуулж үр дүнг хүснэгт хэлбэрээр дэлгэцлиэ. Бүтцийн тодорхойлолтын дараагийн мөрөнд байгаа employee emp[3]; нь бүтцэн хүснэгт байгуулах команд юм. Энд emp[3] бол int, char гэх мэт төрлийн хүснэгттэй адил бичдэстэй. Эл хүснэгт employee төрлийнх. Хүснэгтийн бутвэрүүд болох бүтцийн гишүүд рүү доор үзүүлсэн загвараар хандана.

array\_name[index].member

Дээрх нь array\_name хүснэгтийн index-дүгээр бутвэрийн member гишүүнийг заана. Өмнөх програмын хувьд ажилчдын нэр эхнээсээ emp[0].name, emp[1].name, emp[2].name-д харгалzan байршина. Үүнтэй адил аар ажилчдын цалин, тэтгэлэг нь emp[i] (i=0..2) бутвэрийн basicpay, allowance-д тус тус хадгалагдана. Ажилчдын нийт цалинг grosspay хувьсагч руу хуримтлуулна.

#### 2.1.16.5 Функц ба бүтэц

Бүтцийг энгийн хувьсагчийн нэгэн адил аар функц рүү дамжуулж болдог. Өмнө үзсэн Program #2.16-гийн showdata() функцийг гаднаас бүтэц авахаар доор үзүүлснээр сөрчлөн бичиж болно.

```

//Program #2.18
//finding gross pay by passing structure to function
#include <iostream.h>
#include <stdio.h>
#include <iomanip.h>
#include <conio.h>

struct employee
{
    char name[20];
    int basicpay;
    int allowance;
};

```

```

employee emp ;
void getdata() ;
void heading() ;
void showdata(employee) ;

void main()
{
    clrscr() ;
    getdata() ;
    heading() ;
    showdata(emp) ;
    getch();
}

void getdata()
{
    cout << "\nEnter the Employee Name: " ;
    gets(emp.name) ;
    cout << "Enter the Basic Pay : " ;
    cin  >> emp.basicpay ;
    cout << "Enter the Allowance : " ;
    cin  >> emp.allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Gross" ;
}

void showdata(employee e)
{
    int grosspay ;
    grosspay = e.basicpay + e.allowance ;
    cout << endl ;
    cout << setw(20) << e.name ;
    cout << setw(8)  << e.basicpay ;
    cout << setw(12) << e.allowance ;
    cout << setw(8)  << grosspay ;
}

```

 Enter the Employee Name: George.  
 Enter the Basic Pay : 9000.  
 Enter the Allowance : 8000.

Employee Name	Basic	Allowance	Gross
George	9000	8000	17000

Программын код нь өмнөх Program #2.17-гийнхоос бага зэрэг ялгаатай боловч үр дүн нь ижил болохыг дээрх үр дүнгээс харж болно. Бүтцийн тодорхойлолтын дараах employee emp;<sup>26</sup> бол emp бүтцэн хувьсагчийг байгуулах команд юм. Иймд name, basicpay, allowance зэрэг бутвэрийн утга emp бүтцэн хувьсагч дотор хадгалагдана. Бүтцэн

---

<sup>26</sup> ANSI Си хэлний хувьд struct үгийг хэрэглэн struct employee emp; гэж бичиж огох ёстой.

хувьсагчийн тодорхойлолтын дараа `getdata()`, `showdata(employee)`, `heading()` гэсэн гурван функцийг эхэлж зарлаад `main()` функцийн дараагаар тодорхойлсон байна. Харин `main()` функцийн дотор тэдгээрийг дуудаж хэрэглэхдээ `showdata(employee)` функции рүү `emp` бүтцэн хувьсагчийг дамжуулж огнен. Энэ тохиолдолд `emp` хувьсагчийн утга е бүтцин хувьсагч руу хуулагдана. Иймд `showdata()` функцийн командууд `name`, `basicpay`, `allowance` гэсэн бүтвэрүүдийн утлыг дэлгэшэхийн тулд е хувьсагчийн гишүүд рүү хандана.

### 2.1.17 Хэрэглэгч тодорхойлох өгөгдлийн төрөл

C++ хэлний дотоод суурь өгөгдлийн төрлийг түшиглэн шинэ өгөгдлийн төрөл тодорхойлох боломж бий. Ингэж тодорхойлсон төрлийг хэрэглэгчийн өгөгдлийн төрөл гэх ба ийм төрлийн хувьсагч, хүснэгт, бүтцэн хувьсагчийг байгуулж болдог. Шинэ өгөгдлийн төрлийг үүсгэхдээ доорх дүрмийг баримтална.

```
typedef data-type variable ;
```

Энд `typedef` нь тусгай уг, `data-type` нь байгаа өгөгдлийн төрөл, `variable` нь хэрэглэгчийн тодорхойлох шинэ өгөгдлийн төрлийн нэр юм. Шинэ төрөл нь `data-type`-д шаардагдахтай адил хэмжээний ойг эзэмшина. Өмнөх дүрмийн дагуу бичигдсэн

```
typedef int pay ;
```

командын мерийг авч үзье. Энд `pay` нь `int` төрөлтэй адилхан зориулалт бүхий шинэ өгөгдлийн төрөл болно. Иймд `pay` төрлийн хувьсагчийг доорх жишгээр

```
pay basicpay, allowance ;
```

гэж байгуулж болох ба энэ нь

```
int basicpay, allowance ;
```

гэдэгтэй ижил юм. Эл `basicpay`, `allowance` хоёр хувьсагч нь `pay` төрлийнх боловч үнэндээ `int` төрлийнх юм. Шинэ өгөгдлийн төрөл тодорхойлж түүнд түшсэн хувьсагч байгуулах өөр жишээг доор үзүүллээ.

```
typedef char name[10];  
name n1, n2;
```

Дээрх жишээнд `name` нь `char` төрлийн 10 бүтвэртэй шинэ өгөгдлийн төрөл гэж тодорхойлогдсон тул `n1, n2` хоёр мөн `char` төрлийн 10 бүтвэртэй хүснэгтүүд болно.

#### 2.1.17.1 Бүтэц ба `typedef`

Хэрэглэгчийн тодорхойлсон бүтцийг шинэ өгөгдлийн төрөл болгоходоо `typedef` үгийг доор үзүүлснээр хэрэглэнэ.

```
typedef struct emp_struct  
{  
    char name[13] ;  
    int basicpay ;  
    int allowance ;  
} employee ;  
  
employee emp ;
```

Дээрх тодорхойлолтод цэгтэй таслалыг хаах их хаалтын араас залгуулж бичсэгүй, түүний өмнө бичигдсэн `employee` бол шинэ өгөгдлийн төрлийн нэр болох ба ийм төрлийн `emp`

хувьсагч байгуулсныг удаах командаас нь харж болно. Харин struct үгийн дараа байгаа emp\_struct бол бүтцийн нэр юм. Энхүү шинэ өгөгдлийн төрлийг хэрэглэх жишээ програмыг доор үзүүллээ.

```
//Program #2.19
//finding the gross pay of an employee using typedef

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <stdio.h>

typedef struct
{
    char name[13] ;
    int basicpay ;
    int allowance ;
} employee ;

employee emp;

void getdata();
void heading();
void showdata();

void main()
{
    clrscr() ;
    getdata() ;
    heading() ;
    showdata() ;
    getch() ;
}

void getdata()
{
    cout << "\nEnter the Employee Name: " ;
    gets(emp.name) ;
    cout << "Enter the Basic Pay      : " ;
    cin  >> emp.basicpay ;
    cout << "Enter the Allowance     : " ;
    cin  >> emp.allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(13) << "Employee Name" ;
    cout << setw(8)   << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)   << "Gross" ;
}

void showdata()
{
    int grosspay;
    grosspay = emp.basicpay + emp.allowance ;
    cout << endl ;
    cout << setw(13) << emp.name ;
```

```
    cout << setw(8) << emp.basicpay ;
    cout << setw(12) << emp.allowance ;
    cout << setw(8) << grosspay ;
}
```



Программын үр дүн Program #2.15-ынхтай ижил.

### 2.1.17.2 Нэрлэсэн тогтмолон төрөл

Өмнөх дэд бүлэгт `typedef` тусгай түлхүүр үгийг хэрэглэн шинэ өгөгдлийн төрөл тодорхойлох талаар үзсэн билээ. Шинэ өгөгдлийн төрөл үүсгэж болох удаах арга бол нэрлэсэн тогтмолон төрөл тодорхойлох арга юм. Утга нь, жишээ нь өнгө, хүйс, долоо хоногийн нэр нь байгаагаараа хэрэглэгдэх объект зарлахдаа бүхэл тоон утгуудтай уяж хэрэглэх боломж, механизм C/C++ хэлэнд байдаг ба үүнийг нэрлэсэн тогтмол гэнэ. Ийм шинэ өгөгдлийн төрөл тодорхойлсаноор түүнийг хэрэглэх энгийн, уншууртай програм бичих боломжтой болно. Нэрлэсэн тогтмолон торлийг дараах загварын дагуу үүсгэнэ.

```
enum list {member1, member2, . . . , membern} ;
```

Энд enum бол түлхүүр үг; list бол нээх хаах их хаалтанд бичиж өгсөн member1, member2, . . . , membern гэсэн нэр бүхий гишүүдийн жагсаалтын нэр юм. Гишүүдийн нэр хоорондоо ялгаатай байх ёстой. Гишүүдийн нэрийг list төрлийн хувьсагчид утга оноход хэрэглэж болно.

Өгөгдсөн тоо нь хоёроос их зэрэг тоо бол түүнийг анхны тоо мөн эсэхийг тогтоох програмыг доор үзүүлснээр нэрлэсэн тогтмол хэрэглэн бичиж болно.

```
//Program #2.20
//testing a positive integer for primeness

#include <iostream.h>
#include <conio.h>
#include <math.h>

enum boolean {false, true};
void main()
{
    int a, m, k ;
    boolean c ;
    c = true;
    cout << "\nEnter a positive integer greater than 2: " ;
    cin  >> a ;

    m = sqrt(a);
    for (k=2; k<=m; k++)
        if (a % k == 0)
            c = false;

    if (c == true)
        cout << a << " is a prime number" ;
    else
        cout << a << " is not a prime number" ;
    getch();
}
```



Enter a positive integer greater than 2: 97  
97 is a prime number

Програмын эхэн хэсэгт байгаа boolean {false, true}; бол boolean гэх нэрлэсэн тогтмолон төрөл тодорхойлох командын мөр юм. Ийм төрлийн өгөгдлийн гишүүд нь false, true хоёр болно. Эдгээрт 0-ээс эхлэх дараалсан бүхэл тоон утга автоматаар олгогдоно. Иймд false нь 0, true нь 1 гэсэн утгыг тус тус заана.

Нэрлэсэн тогтмолон өгөгдөл бол бүхэл тоон утгуудын жагсаалт юм. Гэхдээ энэ нь int төрөлтэй төстэй биш. Харин байж болох утгын жагсаалт хэлбэрээр өгөгдөх ба ийм утга бурд оноосон нэр заавал байна.

Нэрлэсэн тогтмолон жагсаалт тодорхойлсноор уг төрлийн хувьсагчийг байгуулж болдог. Дээрх програмд boolean c; гэсэн командаар boolean төрлийн с хувьсагч байгуулна. Иймд энэ хувьсагчийн утга нь нэрлэсэн тогтмол байна. Програмд c = true; командаар с хувьсагчид true утга онооно. Нэрлэсэн тогтмолыг тодорхойлох явцад true рүү 1 гэсэн утга оноогдох тул с хувьсагчийн утга мөн 1 болно. Өгөгдсөн а тоо нь 2 ба түүний (а тооны) квадрат язгуур хооронд орших аль нэгэн тоонд үлдэгдэлгүйгээр хуваагдаж байвал с хувьсагч ruy false (0) утга хийнэ. Эцэст нь с хувьсагч true (1) эсэхийг шалгаж хэрэв тийм бол өгөгдсөн а тоог анхны тоо гэж үзнэ.

Доорх

```
enum IP Codes {12531, 14405, 21724, 30081};  
enum Grades {A, A-, B+, B, B-, C+, C, C-, D+, D, D-, F};
```

гэсэн тодорхойлолтууд цөм алдаатай, учир нь тэдгээрт багтах нэрс зөв бичих дүрмийн хувьд алдаатай байна.

Нэрлэсэн тогтмолон жагсаалтын гишүүдэд 0-ээс эхэлсэн бүхэл тоон утгыг автоматаар оноодог тухай өмнө үзсэн билээ. Эл дэс тоон дугаарыг өөрчлөх боломж бий. Тухайлбал, жагсаалтын гишүүн бүрд өөр өөр утга оноож болдгийг дараах жишээгээр харууллаа.

```
enum days {Sun=1, Mon=2, Tue=3, Wed=4, Thu=5, Fri=6, Sat=7};
```

Бодит байдалд, жагсаалтын гишүүн бүрд утга оноох шаардлага байдаггүй. Өөрөөр зааж өгөөгүй байвал тухайн нэрлэсэн тогтмолд харгалзах тоон утга өмнөх тогтмолынхoo утгаас нэгээр их байх дүрэм байдаг. Иймд өмнөх жагсаалтыг доор үзүүлснээр өөрчлөн бичиж болно.

```
enum days {Sun=1, Mon, Tue, Wed, Thu=3, Fri, Sat};
```

Энэ командаар жагсаалтын гишүүн бүрд дараах утгууд харгалзан оноогдоно.

```
Sun 1  
Mon 2  
Tue 3  
Wed 4  
Thu 3  
Fri 4  
Sat 5
```

Энэ нэрлэсэн тогтмолын хувьд Sun, Thu гишүүдэд 1 ба 3 гэсэн утга харгалзуулан оноосон байгааг дээрх үр дүнгээс харж болно. Харин тэдгээрээс дараалан өссөн утгууд бусад гишүүдэд оноогдоно. Нэрлэсэн тогтмолын зарим гишүүдийн утга давхардсан байна. Ийм гишүүд бол харгалзан Tue, Thu болон Wed, Fri юм.

Дээр авч үзсэн нэрлэсэн тогтмолын жишээнүүд нь тоог тодорхой иэртэй "уяж" болдог Си хэлний шинжийг харуулна. Тэдгээр дээр үндэслэн нэрлэсэн тогтмолыг тодорхойлох загварыг доорх шигээр бичиж болно.

```
enum typename {List};
```

Энд `typename` бол шинэ өгөгдлийн төрлийн нэр, `List` бол таслалаар зааглаж жагсаасан, шинэ төрлийн авах утгын цуваа юм. Ийм цуваа нь тоон утгад харгалзах нэр, түүнд гарсаны утга оноох илэрхийллээс тогтоно. Ийм шинэ төрлийн объект (хувьсагч) байгуулж гарсаны утга оноож болдгийг жишээ болгож Program #2.20-д үзүүлсэн.

### 2.1.17.3 Тогтмол

C++ хэлэнд тогтмолыг `const`, `enum` зэрэг тусгай үгсээр тодорхойлдог. Жишээ нь, 100 гэсэн утгатай бүхэл тоон тогтмолыг доор үзүүлснээр тодорхойлно.

```
const int max = 100 ;
```

C++ хэлэнд объект байгуулахдаа өөрөөр зааж өгоогүй бол `int` төрлийг шууд авдаг тул омнохийг хураангуйлан дараах байдлаар бичнэ.

```
const max = 100 ;
```

Дээрхтэй төстэйгээр бодит тоон эсвэл тэмдэгтэн тогтмолыг доорх шиг тодорхойлно.

```
const float pi = 3.14;
```

Эсвэл

```
const char b = 'y' ;
```

Гэвч `const int pi=3.14;` бол логик тай, учир нь `int` төрлийн хувьсагч руу бодит тоон утга оноож болдогтүй. Иймд `pi` нь 3 гэсэн утгатай болно. Үүнтэй адиллаар, `const char b='yes';` нь логик тай, учир нь тэмдэгтэн `b` хувьсагч руу гурван тэмдэгт оноож болохгүй. Иймд эхний хоёр тэмдэгт нь тайрагдаж `b` нь 's' гэсэн утгтай болно.

Тэмдэгтэн тогтмол тодорхойлох өөр арга бол `enum` үгийг хэрэглизн изрлэсэн тогтмол тодорхойлох юм. Жагсаалтын гишүүн бүрд ялгаатай утга оноож болно. Ийм утгууд заавал өсөх эсвэл буурах дарааллтай дэс тоо байх шаардлагагүй байдаг. Жишээ нь, 20, 8, 15, 17, 3, 11 ба 9 гэсэн харгалзсан утгууд бүхий нэрлэсэн тогтмолон жагсаалтыг доорх шигээр

```
enum days{Sun=20,Mon=8,Tue=15,Wed=17,Thu=3,Fri=11,Sat=9};
```

гэж үүсгэж болно.

### 2.1.18 Хаяг, хаяган хувьсагч

C++ компайлер функци дуудагдахад түүний дотоод хувьсагч бүрд ой хуваарилдаг. Гэхдээ хуваарилах байтын тоо хувьсагчийн төрлөөс, тухайлбал `char`, `int`, `float`, `double` гэх мэтийн тусгай үсийн алийг хэрэглэж байгаагаас хамаарна. Жишээ нь, `char` хувьсагч<sup>27</sup> нэг, `int` хувьсагч хоёр, `float` хувьсагч дөрөв, `double` хувьсагч найман байт ойг тус тус зэмшдэг байна. Дараах командыг авч үзье.

```
int num ;
```

Хувьсагч зарлах команд нь хувьсагчийн төрлийг заах тусгай уг, хувьсагчийн нэр хоёроос тогтоно. Энэ дагуу тодорхойлсон дээрх команд нь дараах зүйлсийг хийдэг. Зураг 2.9-д үзүүлсэн шигээр 2 байт ой хуваарилж<sup>28</sup> түүнд пим гэдэг "бэлгэдэл" нэр онооно. Тэгэхлээр, пим нь зөвхөн тэрхүү хоёр байт ойг бүхэлд нь заах бөгөөд систем тэдгээрийг өөр хувьсагчид

<sup>27</sup> Жишээ нь, `char` төрлийн хувьсагч гэхийг товчоор `char` хувьсагч гэж болно.

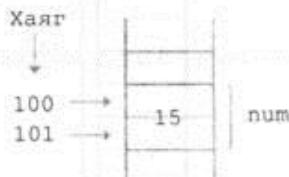
<sup>28</sup> Ой хуваарилахыг бас ой ноших, ой бэлдх гээ.

зориулан дахин хуваарилдаггүй. Бүхэл тоон утгыг тэдгээр байтад хадгалж болно. Жишээ нь, дараах утга оноох команд

```
num=15;
```

хийгдсэнээр num хувьсагчид харгалзах ойн утга 15 болно.

Хувьсагчийн иэрээр түүний утга руу хандахаас гадна хаягаар нь хандаж болно. Ой бэлдэх нь тодорхой ойн хаягтай холбоотой явагддаг үйл юм. Зураг 2.9-д үзүүлсэн ой бэлдэх үйл нь 100 гэсэн дугаартай ойд хийгдсэн байна.



Зураг 2.9: Ой руу хандах

Иймд num хувьсагчаар 100, 101 гэсэн дугаартай хоёр байт ой руу бүхэлд нь хандана. Гэхдээ 100 нь энэ хувьсагчийн хаяг болдог. Үнэндээ, програм хөгжүүлэгч нь систем хувьсагчид ямар ой бэлдсэнийг мэддэгтүй, түүнийг хянадаггүй. Сул ойн боломжоос хамааран аль ойг хуварилаахаа C++ компайлер өөрөө шийддэг. Гэхдээ ямар ч хувьсагчийн зээмшиж байгаа ойн хаягийг (&) оператороор<sup>29</sup> мэдэж болдгийг дараах програмаас үзэж болно.

```
//Program #2.21
//address of a variable
#include <iostream.h>

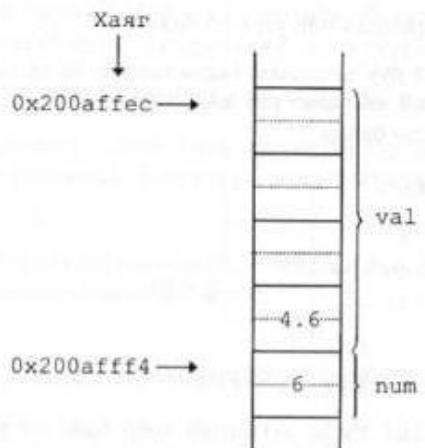
void main(void)
{
    int num = 6 ;
    double val = 4.6 ;
    cout << "\nValue of the int variable num = " << num ;
    cout << "\nAddress of the variable num = " << &num ;
    cout << "\nValue of the double variable val = " << val ;
    cout << "\nAddress of the variable val = " << &val ;
}

Value of the int variable num = 6
Address of the variable num = 0x200afff4
Value of the double variable val = 4.6
Address of the variable val = 0x200affec
```

Ойн хуваарилалт нь Зураг 2.10-д үзүүлсэн шигээр явагдсан болохыг дээрх үр дүн харуулна. Энэ зурагт үзүүлсэн байгаа 0x200afff4 ба 0x200affec гэсэн хоёр хаягийн зөрөө (0x200afff4-0x200affec=8) 8 байгаа нь double val хувьсагч дараалсан 8 байт ой хэрэглэдэг, num хувьсагчийн хаяг нь val хувьсагчийнхыг бодвол өндөр дугаартай байгаа зэрэгтэй холбоотой юм. Ер нь, ойн хаяг бол дээрээс доошлох өөрөөр хэлбэл хойшлох тутам всдэг тоон дугаар юм. Програмын main() болон бусад функцийн дотоод хувьсагчийн хувьд ойн хуваарилалт доороос дээшээ чиглэлтэйгээр явагдана. Эхний хувьсагчид ихэнхдээ их дугаартай ой хуваарилагдана. Дараагийн хувьсагч омнихөөсөө бага дугаартай ой

<sup>29</sup>Энэ операторыг хаягийн оператор гэнэ.

эзэмшинэ. Харин срөнхий хувьсагчийн хувьд ой хуваарилалт дээрхээс эсрэгээр явагддаг байна. Тухайлбал, эхний хувьсагчид хуваарилсан ойн дугаар түүний арын хувьсагчийнхаас бага байх жишээтэй.

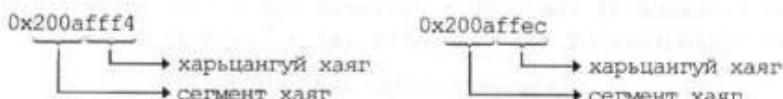


Зураг 2.10: Program #2.21-д хэрэглэгдэх ойн дүрслэл

Дээрх програм нь хувьсагчийн хаягийг доор үзүүлсэн шигээр 4 байт (32 бит) урттайгаар дэлгэцлэсэн байна.

```
Address of the variable num = 0x200afff4
Address of the variable val = 0x200afffec
```

Систем ойг тус бүр нь 64 KB бүхий логик сегментэд хувааж үзэх учир аль ч объектын хаяг нь сегментийн хаяг, сегментийн доторх харьцангуй хаяг гэсэн 2 хэсгээс тогтдог. Дээр үзүүлсэн хоёр хаягийн хувьд 0x200a бол сегментийн хаяг, харин 0xffff4 болон 0xffec бол уг сегмент доторх харьцангуй хаягууд юм. Иймд cout объект нь хаягийг Зураг 2.11-д үзүүлсэн шигээр дэлгэцлэхдээс эхлээд сегмент хаягийг, дараа нь харьцангуй хаягийг залгаж хэвлэнэ<sup>30</sup>.



Зураг 2.11 Ойн хаягийг дэлгэцлэх хэлбэр

### 2.1.18.1 Хаяган хувьсагч тодорхойлох

Хаяг бол тоон утга тул түүнийг хувьсагч руу хадгалж болох ба хаяг хадгалах хувьсагчийг хаяган хувьсагч гэнэ. Програм нь хаягийн зааж байгаа утга ямар төрлийнх болохыг мэдэж байх ёстой. Учир нь, (char) тэмдэгтийн хэрэглэж байгаа ойн хаяг бүтцийн хувьд (double) бодит тооныхтой адил байвч тэлгээрт шаардлагдах ой нь тоо хэмжээ болон компьютерийн дотоод дүрслэлийн хувьд хоорондоо ялгаатай байдаг. Иймд хаяган хувьсагчийг тодорхойлоходо түүнд ямар утга хадгалахыг зааж өгөх ёстой.

Бүхэл тоон утга хадгалах ойн хаягийн хаяган хувьсагчийг доор үзүүлсэн шигээр тодорхойлно.

<sup>30</sup> Гэхдээ хаягийг ямар хэлбэрээр хэвлэх нь C/C++ компайлерээс хамаардаг.

```
int *ptr ;
```

Үг хувьсагч хаяган төрлийнх болохыг од (\*), хаяг нь бүхэл тоон утгынх болохыг int тус тус заана. Харин ptr бол хаяган хувьсагчийн нэр юм.

Хаягийн заах ойд ямар төрлийн утга хадгалахаас хамаарахгүйгээр хаяган хувьсагч бүр дараалсан 2 байтынх байна. Янз янзын хаяган хувьсагчийг доорх шигээр тодорхойлж болно.

```
char      *char_ptr ;
float    *float_ptr ;
double   *double_ptr ;
long     *long_ptr ;
```

Хаяган хувьсагчийг тодорхойлохдоо дараах гурван бичвэрийн алийг ч хэрэглэж болно.

```
int *ptr ;
```

Эсвэл

```
int* ptr ;
```

Эсвэл

```
int * ptr ;
```

Мөн хаяган хувьсагч тодорхойлох үүрэгтэй дараах

```
int *ptr1, *ptr2 ;
```

командыг доор үзүүлсэн шигээр хоёр хувааж бичиж болно.

```
int *ptr1 ;
```

```
int *ptr2 ;
```

Гэхдээ доорх хэлбэрийн командыг

```
int* ptr1, ptr2 ;
```

дээрхтэй адилгаар

```
int *ptr1 ;
```

```
int *ptr2 ;
```

гэж бичдэггүй. Харин ptr2 нь бүхэл тоон хувьсагч тул доор үзүүлсэн шигээр

```
int *ptr1 ;
```

```
int ptr2 ;
```

гэж бичиж болно.

### 2.1.18.2 Хаяган хувьсагчид утга оноох

Бүхэл тоон утга заах хаяган хувьсагчийг доорх шигээр тодорхойлсон гэж үзье.

```
int *ptr ;
```

Энэ ptr хувьсагчийн заах ойд бүхэл тоон утга хадгалах ёстой. Иймд дараах тодорхойлолтыг авч үзье.

```
int num = 288 ;
```

Бүхэл тоон num хувьсагчийн хаягийг хаягийн оператороор &num гэж авч ptr хувьсагч руу хийж болно. Үүнийг Зураг 2.12-т үзүүлснээр дүрсэлж Program #2.22-т байгаа шигээр програмчилж болно.

```

//Program #2.22
//pointer variable declaration and initialization
#include <iostream.h>

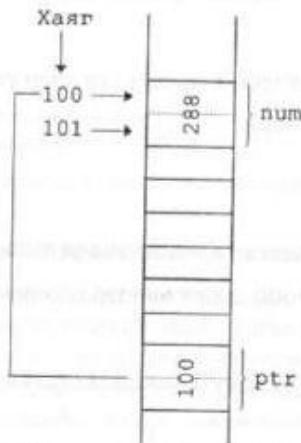
void main(void)
{
    int *int_ptr ;
    int var = 288 ;
    int_ptr = &var ;

    cout << "\nValue of var           = " << var ;
    cout << "\nAddress of var         = " << &var ;
    cout << "\nValue of int_ptr       = " << int_ptr ;
    char *char_ptr ;
    float *float_ptr ;
    double *double_ptr ;
    long *long_ptr ;

    cout << "\nSize of int_ptr        = " << sizeof(int_ptr) ;
    cout << "\nSize of char_ptr       = " << sizeof(char_ptr) ;
    cout << "\nSize of float_ptr      = " << sizeof(float_ptr) ;
    cout << "\nSize of double_ptr     = " << sizeof(double_ptr) ;
    cout << "\nSize of long_ptr       = " << sizeof(long_ptr) ;
}

```

 Value of var = 288  
 Address of var = 0x1b97ffff2 (зохиомол)  
 Value of int\_ptr = 0x1b97ffff2  
 Size of int\_ptr = 2 bytes  
 Size of char\_ptr = 2 bytes  
 Size of float\_ptr = 2 bytes  
 Size of double\_ptr = 2 bytes  
 Size of long\_ptr = 2 bytes



Зураг 2.12: Program #2.22-ын ойн дүрслэл

Хаяган хувьсагч `int_ptr` нь 2 байтын учир түүнд 4 байтын хэмжээтэй `0x1b97ffff2` хаягийг хэрхэн хадгалах вэ? Хаягийг `cout` объектоор дэлгэцлих үед хаяг нь сегментийн болон харьцангуй хаяг гэсэн хоёр хэсэгтэйгээр хэвлэгддэг тухай өмнө үзсэн билээ. Иймд

0xb97ffff2 хаягийн 0xb97 нь сегментийн хаяг, 0xffff2 нь уг сегментийн доторх харьцангуй хаяг болно. Хувьсагч int\_ptr зөвхөн харьцангуй хаягийг хадгалж байдаг.

#### 2.1.18.3 Хаяг ба тоон утга

Компьютер хаягийг тэмдэггүй бүхэл тоо шигээр авч үзнэ. Гэхдээ хаяг үнэндээ ерийн тэмдэггүй бүхэл тоо биш бөгөөд тэдгээрийн хооронд зарчмын ялгаа бий. Нэмэх, хасах, үржих, хуваах гэх мэт олон үйлдлийг бүхэл тооны хувьд хийж болно. Гэтэл хаяг нь компьютерийн ойн аль ишгэн байршилыг заах үүрэгтэй тул дээрх шигээр үржих, хуваах үйлдэл хийх нь утгагүй юм. Хаягийн хувьд доор үзүүлсэн шиг үйлдлийг бас хийх боломжгүй байдаг.

```
int *ptr;  
ptr = 0xb800;
```

Дээрх команд нь 0xb800 бол чухамхүү хаяг юм гэдгийг илэрхийлж чаддагтүй. Иймд бүхэл тоог (жишээ нь 0xb800) хаяг гэж зааж өгөхдөө

```
int *ptr;  
ptr = (int *)0xb800 ;
```

эсвэл

```
char *ptr;  
ptr = (char *)0xb800 ;
```

шигээр төрөл хувиргалт хийх ёстой.

#### 2.1.18.4 Дам хандалтын оператор

Од (\*) оператор нь хаягийн хувьсагчийн эсвэл хаяган тогтмолын зүүн талд хэрэглэгдэж байвал түүнийг дам хандалтын, дам утгын оператор гэнэ. Хэрэв ptr нь хаяган хувьсагч бол дам хандалтын оператор бүхий \*ptr илэрхийлэл нь уг хувьсагчийн заах ойн утгыг авчирч өгнө. Түүнийг 4 янзаар хэрэглэж болохыг доор үзүүллээ.

1. x = \*ptr ;

Энэ командын хувьд x нь хувьсагч, ptr нь хаяган хувьсагч юм. Уг команда ptr хувьсагчийн заах ойн утгыг x хувьсагч руу хийх үүрэгтэй юм (Зураг 2.13). Дээрх шиг командыг хэрхэн хэрэглэж болохыг дараах програмын жишээнээс харж болно.

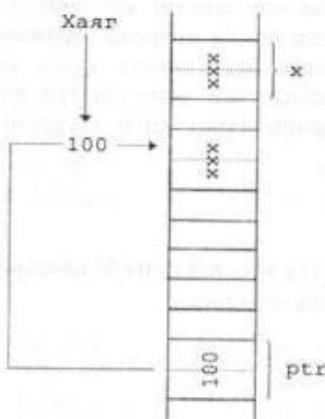
```
//Program #2.23  
//the indirection operation x=*ptr  
#include <iostream.h>  
  
void main(void)  
{  
    int x ;  
    int y ;  
    int *ptr ;  
    y = 258 ;  
    ptr = &y ;  
    x = *ptr ;  
    cout << "\nvalue of x = " << x ;  
}
```

 value of x = 258

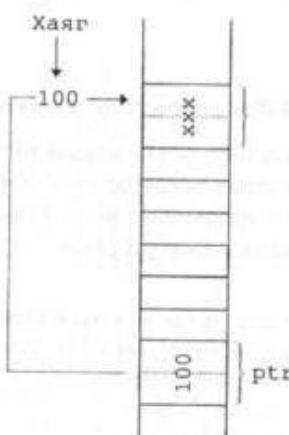
2. x = \*CONST\_PTR ;

CONST\_PTR, жишээ нь (`int*`) `0xb8000000` гэсэн хаяган тогтмол бол энэ байршилд байгаа утлыг хүү дээрх командаар хийнэ.

3. `*ptr = xxx ;`  
xxx нь ptr хувьсагчийн утгаар хаяглагдах ойн утга болохыг Зураг 2.14-д дүрслэн үзүүллээ.



Зураг 2.13: Дам хандалтын операторыг хэрэглэх\_1



Зураг 2.14: Дам хандалтын операторыг хэрэглэх\_2

4. `*CONST_PTR = xxx ;`  
CONST\_PTR хувьсагчийн заах ойд xxx гэсэн утлыг хадгалина. Үүнийг доорх програмд үзүүлсэн шигэр програмчилна.

```
//Program #2.24
//the indirection operation *ptr=xxx
#include <iostream.h>
void main(void)
{
    int x ;
    int y ;
    int *ptr ;
    x = 238 ;
    ptr = &y ;
    *ptr = x ;
```

```
    cout << "\nvalue of y = " << y ;  
}
```

```
value of y = 238
```

### 2.1.18.5 Тогтмолын хаяг, тогтмолон хаяг

Тогтмол тодорхойлоходоо `const` тусгай үгийг хэрэглэнэ. Жишээ нь,

```
const int age ;
```

нь бичлэгийн хувьд зөв команд юм. Гэхдээ `age` хувьсагчийг байгуулж байхдаа утга оноож өгөөгүй тул түүнийг юунд ч хэрэглэх боломжгүй болдог. Мөн тогтмолын утгыг програмаар өөрчилж болохгүй. Тэгэхлээр, дараах хоёр командын нийлэмж нь зөв бичих дурмийн алдаатай учир C++ компайлдер энэ тухай алдааны мэдээлэл өгнө.

```
const int age ;  
age = 40 ; //буруу
```

Харин дараах команд ямар ч алдаагүй тул түүнийг хэрэглэх боломжтой байдаг.

```
const int age = 40 ;
```

Тогтмолон хувьсагчийг зарлах явцдаа заавал утга ононо. Дараах хоёр командын нийлэмж мен алдаатай, учир нь тогтмолыг програмын аргаар доорх шигээр өөрчилж болдоггүй.

```
const int age = 40 ;  
age += 1 ; //буруу
```

Ерийн хувьсагчийн хаягийг хаяган хувьсагч руу хийж болох учир доор үзүүлсэн шиг команд ямар ч алдаагүй юм.

```
int age = 40 ;  
int * ptr = &age ;
```

Харин тогтмолон хувьсагчийн хаягийг ерийн хаяган хувьсагч руу хийж болохгүй тул доорх шиг команд зөв бичих дурмийн алдаатай юм.

```
const int age = 40 ;  
int * ptr = &age ; //буруу } seg-A
```

Хаягийн хувьд `const` тусгай үгийг хоёр янзаар хэрэглэх боломжтойгоос эхний арга нь хаяган хувьсагч тогтмолон объект заадаг байхаар, удаах арга нь хаяган хувьсагчийг өөрийг нь тогтмол гэж тус тус програмчлах юм.

Хаяган хувьсагч тогтмолон объектыг заадаг байх командыг доорхын адилаар бичиж болно.

```
int age = 40 ;  
const int *ptr = &age ; } seg-B
```

Энд хоёрдахь команд нь `ptr` хувьсагч тогтмолыг хаягладаг болохыг заана. Иймд `age` хувьсагчийг өөрчлөх зэргээр түүнд хандахдаа `ptr` хувьсагчийг хэрэглэх боломжгүй учир дараах командууд буруу юм.

```
int age = 40 ;  
const int *ptr = &age ; //буруу  
*ptr = 41 ; //буруу  
cin >> *ptr ; } seg-C
```

Тогтмолон хаяган хувьсагч нь түүний хаяглаж байгаа утга тогтмолон байх ёстойг илтгэж чаддаггүй. Зөвхөн дээр үзүүлсэн шигээр ptr хувьсагчаар дамжуулан бодолт хийж болохгүй заана. Иймд ptr хувьсагчийн хаяглаж байгаа ойн уттыг өөрчлох шаардлагатай бол, жишээ нь age хувьсагчийг өөрийг нь хэрэглэнэ.

```
int age = 40 ;
const int *ptr = &age ;
age = 50 ;                                //зөв
*ptr = 50 ;                               //буруу }
```

} seg-D

Тогтмолон хувьсагчийн хаягийг ерийн хаяган хувьсагч руу хадгалж болдоггүй талаар дээр (seg-A) үзсэн билээ. Харин энэ асуудлыг доор үзүүлсэн шигээр шийдэж болно.

```
const int age = 40 ;
const int * ptr = &age ;                  //зөв }
```

} seg-E

age нь тогтмолон хувьсагч тул түүний уттыг өөрчилж болохгүй. Үүнтэй адилгаар ptr нь тогтмолыг хаяглах хувьсагч учир түүгээр дамжуулан age хувьсагчийг өөрчлөх боломжгүй байдаг. Иймд дээрх хоёр команд өөр хоорондоо бүрэн нийцдэг байна. Үүнийг дараах жишээгээр үзүүлж болно.

```
const int age = 40 ;
const int * ptr = &age ;
age = 41 ;                                //буруу
*ptr = 41 ;                               //буруу }
```

} seg-F

seg-A хэсгийн хоёрдахь командыг дүрмийн хувьд зөв команд гэж үзвэл энэ нь age хувьсагчийн уттыг ptr хувьсагчаар дамжуулан өөрчилж болно гэсэн үг бөгөөд гэтэл энэ нь age хувьсагчийн (тогтмолон) шинжтэй харшилдаж байгаа юм. Бусад seg-B, seg-C, seg-D, seg-E, seg-F зэрэг жишээний хувьд ptr хувьсагч нь тогтмолон биш тул түүний уттыг доорх жишгээр өөрчилж болно.

```
int age = 40 ;
int page = 100 ;
const int * ptr = &age ;
ptr = &page ;                            //зөв
```

const тусгай үгийг хэрэглэх хоёрдахь аргын хувьд хаяган хувьсагчийг өөрийг нь тогтмолоор тодорхойлдог. Үүнийг доорх шигээр бичиж өгнө.

```
int age = 40 ;
int * const ptr = &age ;
```

Дээрх командын age хувьсагчийг утга оноох команд хэрэглэн эсвэл ptr хувьсагчаар дамжуулан доорх шигээр өөрчлөх боломжтой юм.

```
int age = 40 ;
int page = 100 ;
int * const ptr = &age ;
age = 45 ;                                //зөв
*ptr = 50 ;                               //зөв
ptr = &page ;                            //буруу
```

Сүүлийн командын хувьд ptr нь зөвхөн age хувьсагчийг хаяглах тул буруу хэрэглээ болно. Тогтмолон объект хаяглах хувьсагчийг const гэж тодорхойлох боломжтойг доор үзүүллээ.

```
int age = 40 ;
const int * const ptr = &age ;
```

`ptr` зөвхөн `age` хувьсагчийг хаяглах ч түүнийг `ptr` хувьсагчаар дамжуулан өөрчилж болдоггүй. Гэхдээ `age` хувьсагчийг доор үзүүлсэн шигээр утга оноох командаар өөрчлох бололцоотой юм.

```
int age = 40 ;
int page = 100 ;
const int * const ptr = &age;
age = 41;                                //зөв
page = 101;                               //зөв
ptr = & page ;                            //буруу
* ptr = 50;                               //буруу
```

Одоо доорх хоёр командыг авч үзье.

```
const int age = 40 ;
const int * const ptr = &age ;
```

Энд `age`, `ptr` нь хоёулаа тогтмолон хувьсагч тул тэдгээрийг өөрчилж болдогтүйг дараах жишээ үзүүлнэ.

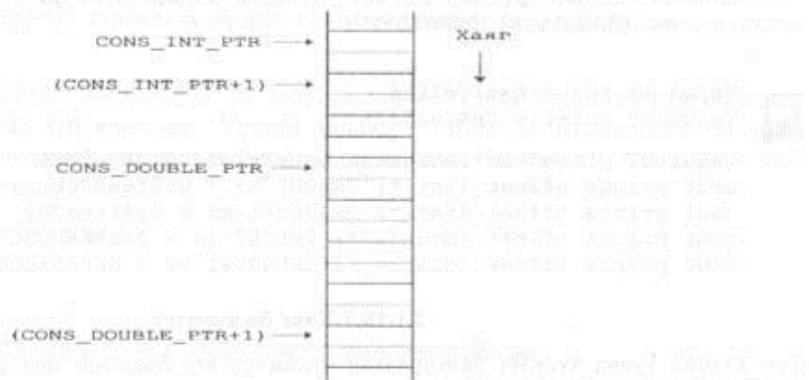
```
int page = 100 ;
const int age = 40 ;
const int * const ptr= &age ;
age = 41 ;                                //буруу
*ptr = 41 ;                               //буруу
ptr = &page ;                            //буруу
page = 101;                               //зөв
```

Бас доор үзүүлсэн шиг командын нийлэмжийг C++ компайлер зөвшөөрдөггүй байна.

```
const int age = 40 ;
int * const ptr = &age ;      //буруу
```

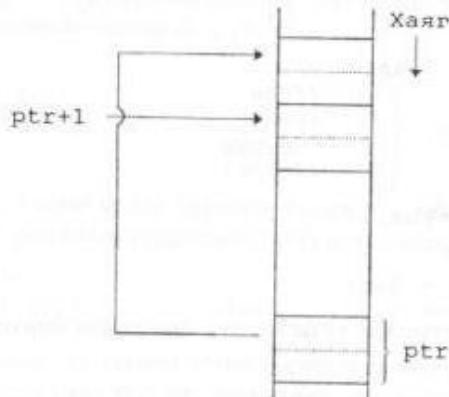
### 2.1.18.6 Хаягийн арифметик

`CONST_INT_PTR` бол бүхэл тоон өгөгдлийг хаяглах хувьсагч бөгөөд түүний утга нь `(int*)0xa000` гэсэн хаяг байна гэж үзье. Тэгвэл `(CONST_INT_PTR+1)` хаяг нь `CONST_INT_PTR` хувьсагчийн хаяглах өгогдлийн байт хэмжээгээр `CONST_INT_PTR`-аас алсад орших ойг заана. Жишээ нь, `CONST_DOUBLE_PTR` бол `double` утгыг хаяглах хаяган тогтмол, түүний утга нь `(double *)0xb000` гэсэн хаяг байна гэж үзье. Тэгвэл `(CONST_DOUBLE_PTR+1)` нь `CONST_DOUBLE_PTR`-аас 8 байтаар их байх харьцангуй хаягийг заана. Үүнийг Зураг 2.15-д харууллаа.



Зураг 2.15: `double` хаягийн арифметикийн жишээ

Бүхэл тоон өгөгдлийг хаяглах ptr хувьсагчийн хувьд ptr+1 илэрхийллийн хариу нь ptr хувьсагчийн хаяглахаас 2 байтаар их байх харьцангуй хаягийг заана (Зураг 2.16). Иймд ptr нь 200 гэсэн утгатай гэж үзвэл ptr++ нь ptr хувьсагчийн утгыг 1-ээр нэмж 202 болно.



Зураг 2.16: int хаягийн арифметикийн жишээ

Дараах програм нь хаяган хувьсагчийн зарим хэрэглээг харуулна.

```
//Program #2.25
//points arithmetics

#include <iostream.h>

void main(void)
{
    int x ;
    int *ptr ;
    ptr = &x ;
    cout << "\nValue of ptr = " << ptr ;
    cout << "\nValue of ptr+1 = " << ptr+1 << "\n" ;
    ptr++ ;
    cout << "\nValue of ptr after the increment operation = " << ptr ;
    cout << "\ncout prints offset (int *)0xa000 as = " << (int *)0xa000 ;
    cout << "\ncout prints offset (int *)0xa000+1 as = " << (int *)0xa000+1 ;
    cout << "\ncout prints offset (double *)0xb000 as = " << (double *)0xb000 ;
    cout << "\ncout prints offset (double *)0xb000+1 as = "
         << (double *) 0xb000+1 ;
}

Value of ptr = 0x8f6afff4
Value of ptr+1 = 0x8f6afff6

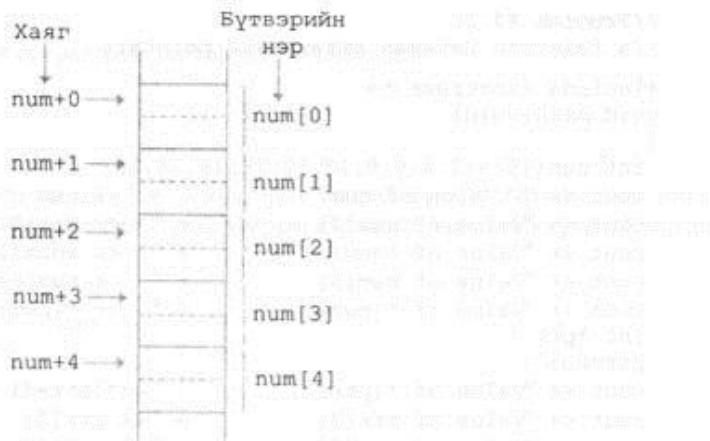
Value of ptr after increment operation =0x8f6afff6
cout prints offset (int *) 0xa000 as = 0x8f6aa000
cout prints offset (int *) 0xa000+1 as = 0x8f6aa002
cout prints offset (double *) 0xb000 as = 0x8f6ab000
cout prints offset (double *) 0xb000+1 as = 0x8f6ab008
```

#### 2.1.18.7 Хаяг ба хүснэгт

C++ хэлний хувьд хүснэгт байгуулахад чухамхүү юу болдгийг авч үзэхийн тулд дараах хүснэгтийг байгуульяа. Тухайлбал,

```
int num[5] ;
```

Уг команд тус бүр нь 2 байтаас тогтох 5 нүдтэй хүснэгтэд зориулан Зураг 2.17-д үзүүлсэн шигээр ой бэлдэнэ.



Зураг 2.17: int num[5] хүснэгтийн ойн дүрслээ

Индекс хэрэглэн тэдгээр нүд рүү num[0], num[1], num[2] гэх зэргээр нэрээр нь хандаж болох ба num[0], num[1], num[2]... нь тухайн нүдний утгыг зааж байдаг.

C++ компайлер хүснэгтийн нэрийг (num) хүснэгтийн эхлэл хаяг гэж үзэх учир түүнтэй холбогдох дараах онцлог шинж түүнд байна. Тухайлбал,

- num бол хүснэгтийн эхлэл хаяг
- num бол тогтмол утгат учир num++ бол алдаатай үйлдэл
- num = some\_value бол зөв бичих дүрмийн алдаатай, учир нь num бол програмаас утга оноож болох хувьсагч биш
- num нь (int \*) төрлийнх, иймд бүхэл тоо агуулах ойн хаяг
- num нь хүснэгтийн эхний бүтэвэрийн хаяг, иймд num, &num[0] хоёр адилхан утгатай

Өмнөх хүснэгтийн хувьд num өөрөөр хэлбэл num+0 нь хүснэгтийн эхлэл хаягийг заана. Иймд энэ нь хүснэгтийн эхний нүдийг заана. Хаягийн арифметик хэрэглэвэл (num+1) хоёрдах нүдийг, (num+2) гуравдах нүдийг гэх мэтчилэнгээр заадгийг хялбархан мэдэж болно (Зураг 2.17 ).

Дээрхээс үзвэл \* (num+0) ба num[0] нь хоёулаа эхний нүдний утгыг авчирч огох тул \* (num+0) нь num[0]-тэй адил юм. Үүнтэй адилаар \* (num+1) нь num[1]-тэй дүйх жишээтэй. Иймд \* (num+n) нь num[n]-тэй адилхан, бие биеэ орлох боломжтой гэж еренхийлон бичиж болно.

Дараах хоёр командыг авч үзье.

```
int *ptr ;  
int num[5] ;
```

Хүснэгтийн нэрээр ptr хувьсагч руу доорх шигээр утга оноож болно.

```
ptr = num ;
```

Иймд `*(ptr+n)` нь `*(num+n)-тэй`, `*(ptr+n)` нь `num[n]-тэй` адил юм. C++ хэлний хувьд `*(ptr+n)` илэрхийллийг `ptr[n]` гэж бичиж өгөх боломжтой байдаг. Тэгэхлээр, хэрэв `xuz` нь хүснэгтийн нэр эсвэл хаяган хувьсагч бол `*(xuz+n)` ба `xuz[n]` нь адил юм. Үүнийг дараах програмаас үзэж болно.

```
//Program #2.26
//a relation between arrays and pointers

#include <iostream.h>
void main(void)
{
    int num[10]=(2,4,6,8,10,12,14,16,18,20) ;
    cout << "\nValue of num          = " << num      << "\n";
    cout << "Value of num[0]       = " << num[0]   << "\n";
    cout << "Value of &num[0]       = " << &num[0]  << "\n";
    cout << "Value of num[5]       = " << num[5]   << "\n";
    cout << "Value of *(num+5)     = " << *(num+5)<< "\n";
    int *ptr ;
    ptr=num ;
    cout << "Value of *(ptr+3)     = " << *(ptr+3) << "\n";
    cout << "Value of ptr[3]        = " << ptr[3]   << "\n";
    cout << "Value of num[3]        = " << num[3]   << "\n";
}

Value of num          = 0x103ffffe2
Value of num [0]       = 2
Value of & num [0]       = 0x103ffffe2
Value of num [5]       = 12
Value of *(num+5)     = 12
Value of *(ptr+3)     = 8
Value of ptr[3]        = 8
Value of num[3]        = 8
```

#### 2.1.18.8 Хаяг ба бүтэц

Дараах тодорхойлолт бүхий бүтэц

```
struct person
{
    char name[20] ;
    int age ;
    double height ;
} ;
```

егегдсэн бол `person` төрлийн бүтцэн хувьсагчийг доорх шигээр тодорхойлж гарсаны утга оноож болно.

```
person p = {"Bill", 40, 170.5} ;
```

Ийм бүтцэн хувьсагчийн хаягийг авахын тулд (`&`) хаягийн операторыг хэрэглэж болох ба р хувьсагчийн хаягийг авахдаа `&p` гэж бичиж өгнө. `person` төрлийн бүтцэн өгөгдлийг заах хаяган хувьсагчийг

```
person * ptr ;
```

гэж байгуулах ба энд `ptr` нь хувьсагчийн нэр юм. Уг хувьсагч 2 байтын хэмжээтэй. Түүнд дараах байдлаар утга онооно.

```
ptr = &p ;
```

Бүтцэн хувьсагчийг хаяглах хувьсагчаар нь дамжуулан бүтцийн гишүүн өгөгдлөд хандаадаа дам сонголтын (->) сумтай операторыг доорх шигээр хэрэглэнэ.

```
ptr->name  
ptr->age  
ptr->height
```

Мөн шууд сонголтын ( . ) цэг операторыг бас хэрэглэх боломжтой.

```
(*ptr).name  
(*ptr).age  
(*ptr).height
```

Дам хандалтын (\*) оператор нь цэг (.) операторыг бодвол түвшин багатай тул дээрх жишээнд хаалт хэрэглэсэн байна. Хаяг ба бүтэц хоёрын хэрэглээг дараах програм тодорхой харуулна.

```
//Program #2.27
//a pointer and a structure
#include <iostream.h>

struct person
{
    char name[20];
    int age;
    float height;
};

void main(void)
{
    person p={"Bill", 40, 170.5};
    person *ptr;
    ptr = &p;
    cout << "\nName of the person : " << ptr->name << "\n";
    cout << "Age of the person : " << (*ptr).age << "years\n";
    cout << "Height of the person : " << ptr->height << " cm\n";
}
```

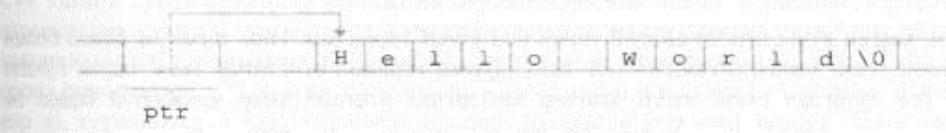
Name of the person : Bill  
Age of the person : 40 years  
Height of the person : 170.5 cm

#### **2.1.18.9 Хаяг ба тэмдэгт мөр**

Тэмдэгт мөрийг хаяган хувьсагчтай хамтруулан доор үзүүлсэн шиггэрхэрэглэж болно.

```
char *ptr = "Hello World";
```

Дээрх командаар `ptr` хувьсагч байгуулна. Ингэхдээ “Hello World” морийг нэргүй ойд байгуулж төгсгэлд нь тэмдэгт морийн төгсгөл заах хоёртын тэгийг (“\0”) нэмж хийгээд ойн эхэлж хаягийг `ptr` рүү хийнэ. Үүнийг Зураг 2.18-аас үзэж болно.



Зураг 2.18: `char *ptr = "Hello World";` командаын үр дүн.

### 2.1.18.10 Хаяг ба тэмдэгт мөрийн хүснэгт

"Тооцох" гэдэг уг тоон өгөгдөл дээр арифметик тооцоо хийх санааг гол төлөв илтгэнэ. Иймд компьютер бол зөвхөн тоон мэдээлэл боловсруулах техник гэсэн дүгнэлт хийж болох ба эн нь эхэн үеийн тооцоолох машины хувьд ихээхэн үнэнд ойр байсан. Хэдий тийм ч хэрэглэгч ба програм хоорондоо ярилцах ажлыг хөнгөвчлөхийн тулд компьютерт үсгэн мэдээллийг дүрсэлж боловсруулах боломжтой байх кодыг зохион бүтээх хэрэгцээ гарчээ. Уг бол дараалсан олон тэмдэгтээс, тэмдэгтийн цуваанаас бүрдэх тул ийм цувааг ойд хадгалж боловсруулах асуудал зайлшгүй гарах болсон байна. Үүний шийдэл бол тэмдэгт мөр юм.

Тэмдэгт мөрөн хүснэгтийг доор үзүүлсэн шигээр байгуулж бүтвэр бүрд гарааны утга оноож болно.

```
char *day[7]={"Sunday",
               "Monday",
               "Tuesday",
               "Wednesday",
               "Thursday",
               "Friday",
               "Saturday"
} ;
```

Энэ хүснэгтийн хувьд, жишээ нь day[0] нь "Sunday" тэмдэгт мөрийн эхлэл хаягийг агуулж байх тул

```
cout << day[0] ;
```

командаар "Sunday" гэсэн үгийг дэлгэнэлнэ.

### 2.1.18.11 Хаягийн хаяг

C++ нь хаягийн хаяг өөрөөр хэлбэл хаяган хувьсагчийн хаяг хадгалах хаяган хувьсагч хэрэглэхийг зөвшөөрдөг. Ийм хаяг нь хоёрдах<sup>31</sup> түвшнийх, гуравдах түвшнийх гэх мэтээр олон түвшнийх байж болно. Үүнийг програмчлахын тулд хаягийн түвшний тоогоор нь од (\*) операторыг хаяган хувьсагчийн нэрийн өмнө бичиж өгөх ба үүнийг доорх жишэнээс үзж болно.

```
char a ;
char *b ;
char ** c ;
a = 'z' ;
b = &a ;
c = &b ;
```

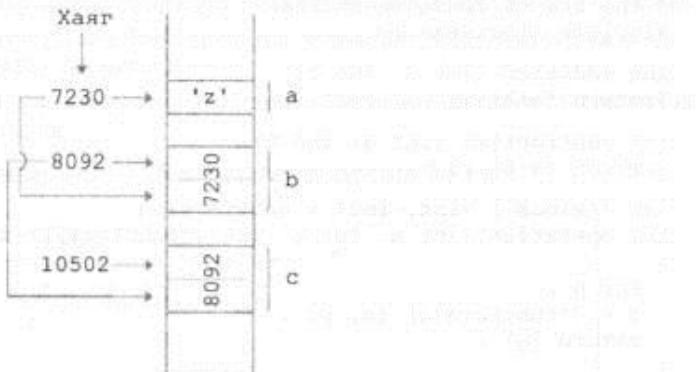
Хувьсагч бүрд 7230, 8092, 10502 гэсэн дугаар бүхий ой харгалзан хуваарилагдсан гэж үзвэл Зураг 2.19-д үзүүлсэн шиг байдал үүснэ. Зурагт үзүүлснээр с бол 8092 гэсэн утга бүхий (char \*\*) төрлийн хувьсагч, \*с бол 7230 гэсэн утга бүхий (char \*) төрлийн хувьсагч, \*\*с бол 'z' гэсэн утга бүхий (char) төрлийн хувьсагч болно.

### 2.1.18.12 void хаяг

Хаягийн, хаяган хувьсагчийн онцгой терөл бол void төрөл юм. Ийм төрөл нь бүхэл болон бодит тоон эсвэл тэмдэгт, тэмдэгт мөр гээд дурын төрлийн өгөгдлийг зааж чадна. Гэхдээ түүний гол дутагдал гэвэл void хаягаар хаяглагдах өгөгдөл ямар хэмжээтэй болох нь

<sup>31</sup> C++ хэлний ямар ч объектын хаяг нэгдэх түвшнийх болно.

хаягаас үл мэдэгдэх тул дам хандалтын операторыг шууд хэрэглэх боломжгүй байдаг. Иймд төрөл хувиргалтыг заавал хэрэглэх ёстой. Үүнийг дараах програмаас үзэж болно.



Зураг 2.19: Олон түвшиний хаягийн хэрэглэлийн схема

```
//Program #2.28
//void pointer

#include <iostream.h>
void increase (void* data, int type)
{
    switch (type)
    {
        case sizeof(char) : (*((char*)data))++ ; break;
        case sizeof(short): (*((short*)data))++ ; break;
        case sizeof(long) : (*((char*)data))++ ; break;
    }
}

int main()
{
    char a = 5 ;
    short b = 9 ;
    long c = 12 ;
    increase(&a, sizeof(a)) ;
    increase(&b, sizeof(b)) ;
    increase(&c, sizeof(c)) ;
    cout << (int) a << ", " << b << ", " << c ;
    return 0 ;
}
```



6, 10, 13

### 2.1.18.13 Функцийн хаяг

C++ хэлний хувьд функцийг хаягаар нь дуудаж хэрэглэж болно. Функцийг өөр функцийн параметрээр хэрэглэх тохиолдолд функцийн хаягийг мэдэх шаардлагатай юм. Функцийн хаяган хувьсагч тодорхойлох нь функцийг зарлахтай еронхийдөө төстэй. Функцийн нэрийн оронд өмнөө одтой (\*) хаяган хувьсагчийн нэрийг дугуй хаалтанд бичиж өгнө. Хүснэгтийн нэр нь түүний эхлэл хаяг болдгийн адиллаар функцийн нэр мөн түүний эхлэл хаяг юм. Функцийн хаяг хадгалах хувьсагчийг функцийн хаяган хувьсагч гэнэ.

Функцийн хаяган хувьсагч хэрэглэх жишээ програмыг доор үзүүллээ.

```
//Program #2.29
//the use of function pointer
#include <iostream.h>

int addition (int a, int b)
{return (a+b) ;}

int subtraction (int a, int b)
{return (a-b) ;}

int (*minus ) (int, int) = subtraction ;
int operation (int x, int y, int (*functocall)(int, int))
{
    int g ;
    g = (*functocall) (x, y) ;
    return (g) ;
}
int main()
{
    int m, n ;
    m = operation (7, 5, addition) ;
    n = operation (20, m, minus) ;
    cout << n ;
    return 0 ;
}
```



Дээрх програмын `minus` бол `int` төрлийн хоёр параметртэй функцийн хаяг хадгалах өрөнхий хувьсагч бөгөөд түүнд `subtraction` функцийн хаягийг гарцааны утга хэлбэрээр оноож өгсөн байна.

## 2.2 C++ хэлийг Си хэлтэй харьцуулах нь

C++ бол ОХП-ын технологийг дэмждэг гол хөдөлгүүр хэл юм. Гэхдээ Си хэлэнд байх бүх шинж, боломж түүнд байдаг. Мөн түүнд Си хэлнээс ялгагдах өөрийн гэсэн шинж, боломж байдгийг дараах байдлаар жагсааж болно. Тухайлбал,

- Дотоод мөр функци (inline) хэрэглэж болно,
- Заалтан хувьсагч хэрэглэнэ,
- Тэмдэгт мөрийн тэмдэгтийн зориулгатай `cout` ба `cin` объекттой,
- Динамик ойн `new` ба `delete` оператортой,
- Гараани утгатай параметр хэрэглэж болно,
- Функцийг дахин тодорхойлохыг зөвшөөрнө,
- Операторыг дахин тодорхойлож болно

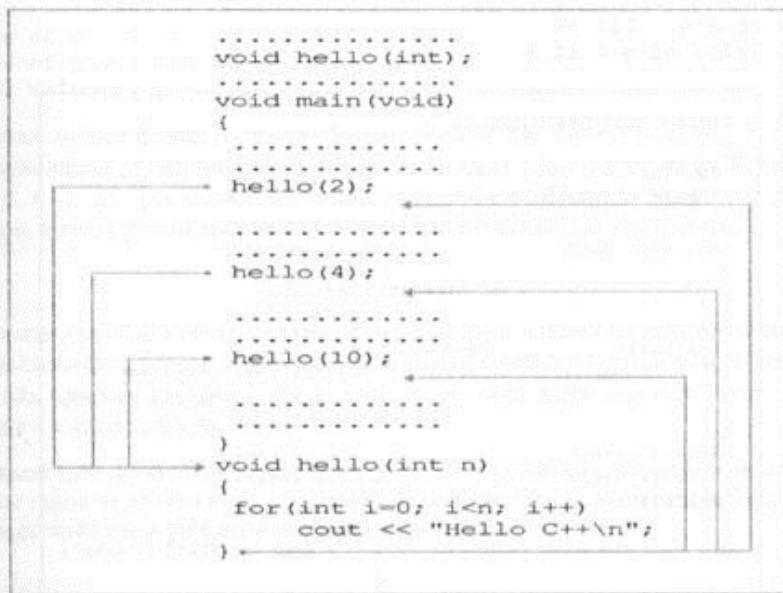
### 2.2.1 inline функци

C++ хэлний `inline` функци бол програмын хурдыг нэмэгдүүлэх зориулалтаар бүтээгдсэн боломж юм. Дотоод мөр функци ба ерийн Си функци хооронд байх үндсэн ялгаа нь тэдгээрийг програмд хэрхэн холбож өгдөгт оршино. Си функцийг өөр функци дотроос дуудах үед дараах үйлдлүүд дараалан хийгддэг. Тухайлбал,

- Дараагийн сэлжинд хийгдэх командын хаягийг хадгалах,

- CPU дотоод регистрийн утгыг хадгалах,
- Дуудагдсан функц рүү дамжуулж огч байгаа аргументийг стек рүү хадгалах,
- Программын удирдлага дуудагдсан функц рүү шилжин очиж улмаар аргументээ стекээс авах,
- Функц ажиллаж дуусахад түүнээс буцах утгыг үндсэн програм, эх функц нь гаргаж авахаар CPU регистр эсвэл стек рүү түр хадгалах,
- Программын удирдлага эх функц рүү шилжихдээ омно хадгалсан буцах хаяг, CPU регистрийн утгыг стекээс буцааж гаргах.

C++ функцийн ажиллах зарчмыг Зураг 2.20-д үзүүлснээр дүрсэлж болно.



Зураг 2.20: Ерийн C++ функцийн ажиллагааны зарчим

Функц хэрэглэх үед дээрх бүх үйлдэлд нэмэлт CPU цаг шаардагдана. Дотоод функц хэрэглэвэл түүний командуудыг функцийг хэрэглэсэн командын мөр бүрд хувилж үүсгэнэ. Энэ бол дотоод функцийн сүл тал, учир нь программын хэмжээ ихсэх талтай. Гэхдээ буцах хаяг хадгалах мэтээр дээр дурдсан бүх үйлдлийг хийх шаардлагагүй болох тул нэмэлт CPU цаг шаардахгүй, ингэснээр программын хурд нэмэгдэх боломжтой болдог.

Хэрэглэгчийн функцийг дотоод функц болгоходо `inline` тусгай үгийг хэрэглэнэ. Зураг 2.20-д үзүүлсэн жишээ программын `hello()` функцийг Зураг 2.21-д үзүүлсэн байдлаар дотоод функц болгож өөрчилье. Энэ зургаас `inline` үгийг хэрхэн хэрэглэх, дотоод функцийн командууд функцийг дуудаж хэрэглэсэн тохиолдол бүрд хуулагддагийг харж болно. Энэ бүх ажлыг C++ компайлер хийнэ.

Дотоод функц хэрэглэсэн өөр жишээ програмыг Program #2.30-д үзүүллээ.

```

//Program #2.30
//the use of inline function
#include <iostream.h>
inline double square(double n)
{
    return n*n ;
}
  
```

```

void main(void)
{
    double a=10.6 ;
    double b ;
    cout << "\nValue of a = " << a ;
    b = square(a++) ;
    cout << "\nValue of a*a = " << b ;
    cout << "\nFinal Value of a = " << a ;
}

```

Value of a = 10.6  
 Value of a\*a = 112.36  
 Final Value of a = 11.6

```

inline void hello(int n)
{
    for(int i=0; i<n; i++)
        cout << "Hello C++\n";
}

void main(void)           void main(void)
{
    .....                   .....
    .....                   .....
    hello(2);      =====>  (n=2;
                           for(int i=0; i<n; i++)
                               cout << "Hello C++\n";
                           )
    .....                   .....
    .....                   .....
    hello(4);      =====>  (n=4;
                           for(i=0; i<n; i++)
                               cout << "Hello C++\n";
                           )
    .....                   .....
    .....                   .....
    hello(10);     =====>   (n=10;
                           for(i=0; i<n; i++)
                               cout << "Hello C++\n";
                           )
    .....                   .....
    .....                   .....
}

```

Зураг 2.21: Дотоод функцийн ажиллагааны зарчим

Функци бүр inline функци болж чадахгүй. Тухайлбал, C++ компайлер нь хэт том функци, бас рекурсив функцийг дотоод функци болж чадахгүй. Мөн C++ компайлерт inline шинж байхгүй бол ямар ч функци дотоод функци болж чадахгүй.

### 2.2.1.1 Макро, дотоод функци

Макро функцийг C++ хэлний препроцессорын #define командаар тодорхойлох ба ийм функцийг Program #2.31-д үзүүлсэн шигээр бичиж болно.

```

//Program #2.31
//the use of macro definition
#include <iostream.h>

```

```

#define Square(x)      x*x

void main(void)
{
    double a=10.6 ;
    double b ;
    cout << "\nValue of a = " << a ;
    b=Square(a++) ;
    cout << "\nValue of a*a = " << b ;
    cout << "\nFinal Value of a = " << a ;
}

Value of a = 10.6
Value of a*a = 112.36
Final Value of a = 12.6

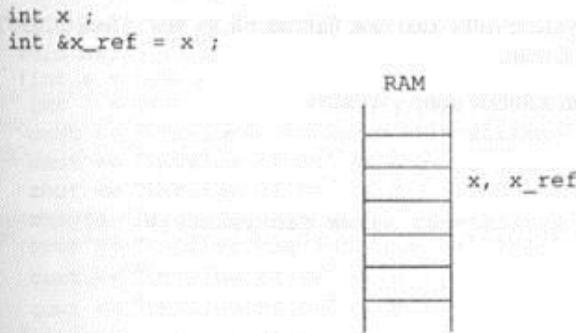
```

Өнгөц харахад дотоод функций нь макро функцийтэй адил юм. Гэвч `Square(10.6+3.1)` гэж хэрэглэх тохиолдолд уг макро функций `10.6+3.1*10.6+3.1` гэсэн илэрхийллийн хариуг, `square(10.6+3.1)` функций нь `13.7*13.7` гэсэн илэрхийллийн хариут тус тус буцаана. Эндээс макро, дотоод функций хооронд байх зарчмын ялгааг байгааг харж болно.

### 2.2.2 Заалтан хувьсагч

Заалтан хувьсагч нь Си хэлэнд байдагтүй, зөвхөн C++ хэлний шинэ боломжийн нэг юм. Заалтан хувьсагчийг тодорхой нэр бүхий хувьсагчтай холбож үүсгэх бөгөөд хэрэв `x_ref` нь `x` хувьсагчийн заалтан нэр бол `x` ба `x_ref` нь нэг ойн хоёр өөр нэр болох учир эдгээр нэрийн алийг ч хэрэглэж болно (Зураг 2.22).

Заалтан хувьсагчийг функцийн параметрээр хэрэглэхэд иэн тохиромжтой байдаг. Ингэснээр ой хэмнэхээс гадна эх өгөгдөл рүү нь шууд хандан бодолт хийх боломжтой болно. Хувьсагч, заалтан хувьсагчийг доор үзүүлснээр холбож өгнө.



Зураг 2.22: Хувьсагч, заалтан хувьсагч хоорондын харьцаа

Энд `x_ref` хувьсагчийн өмнөх `&` бол хаягийн эсвэл логик үржихийн оператор биш, харин `x_ref` бол `x` хувьсагчтай холбоотой заалтан хувьсагч гэдгийг заах үүрэгтэй оператор юм.

Заалтан хувьсагчийг нэр бүхий хувьсагчтай холбооны тулд түүнийг зарлахдаа дээр үзүүлсэн шигээр гарааны утга оноохоос биш програм дотроос утга оноож болдогтүй. Иймд дараах үйлдэл алдаатай юм.

```

int x;
int &x_ref ;
x_ref = x ;

```

Хувьсагч, түүний заалтан хувьсагч өрөнхий нэг ойн өөр өөр нэр болохыг доор үзүүлэх жишээ програмын үр дүнгээс харж болно.

```
//Program #2.32
//the use of reference variables
#include <iostream.h>
void main(void)
{
    int x = 100 ;
    int &x_ref = x ;
    cout << "\nValue of x = " << x ;
    cout << "\nValue of x_ref = " << x_ref ;
    cout << "\nAddress of x = " << &x ;
    cout << "\nAddress of x_ref = " << &x_ref ;
    x++ ;
    cout << "\nValue of x = " << x ;
    cout << "\nValue of x_ref = " << x_ref ;
    x_ref++ ;
    cout << "\nValue of x = " << x ;
    cout << "\nValue of x_ref = " << x_ref ;
}
```



```
Value of x = 100
Value of x_ref = 100
Address of x = 0x2000fff4
address of x_ref = 0x2000fff4
Value of x = 101
Value of x_ref = 101
Value of x = 102
Value of x_ref = 102
```

Дээрх програмын x\_ref, x хоёр хувьсагчийн хаяглаж байгаа ой нь нэг (0x2000FFF4) болохыг програмын үр дүнгээс харж болно.

Заалтан хувьсагчийг буруу хэрэглэсэн жишээг доор үзүүллээ.

```
int x;
int &x_ref=x;
int y;
int &x_ref=y; //заалтан хувьсагчийг дахиж байтулахгүй.
```

Харин дараах хэрэглээ зөв юм.

```
int x = 50;
int &x_ref = x;
int y = 100;
x_ref = y; //зөв, заалтаар нь дамжуулан x хувьсагчийн утгыг
//100 болгож өөрчилж болно.
```

Нэг хувьсагчийн хувьд хэдэн ч заалтан хувьсагч тодорхойлж болохыг доорх жишээ үзүүлнэ.

```
int x ;
int &x_ref1 = x ;
int &x_ref2 = x ;

int &x_refn = x ;
```

### 2.2.2.1 Заалтан хувьсагч ба функци

Заалтан хувьсагчийг функцийн параметрээр хэрэглэхэд тохиромжтой байдал талаар өмнө товч дурдсан билээ. Функци рүү аргумент дамжуулж өгөх энэ аргыг заалтаар нь параметр дамжуулах арга, аргументийг нь заалтаар дамжсан аргумент гэнэ. Уг аргыг хэрэглэхдээ дараах үйлдлийг дараалуулан хийнэ.

- Функци зарлах

```
void func_name(int &x) ;           //ил арга
```

ЭСВЭЛ

```
void func_name(int &) ;           //ил биш, далд арга
```

Хоёрдахь аргын хувьд & оператор нь параметр заалтаар дамжина гэдгийг C++ компайлерт мэдээлж байгаа болохос биш параметр болох дотоод хувьсагчийн нэр ямар байх нь тодорхойгүй байна.

- Функцийг програмаас дуудаж хэрэглэх

```
int a = 100 ;  
func_name(a) ;
```

- Функци тодорхойлох

Функцийн тодорхойлолт нь функцийг илээр зарлах тохиолдолд хэрэглэх бичдэстэй адилхан мөрөөр эхэлдэг.

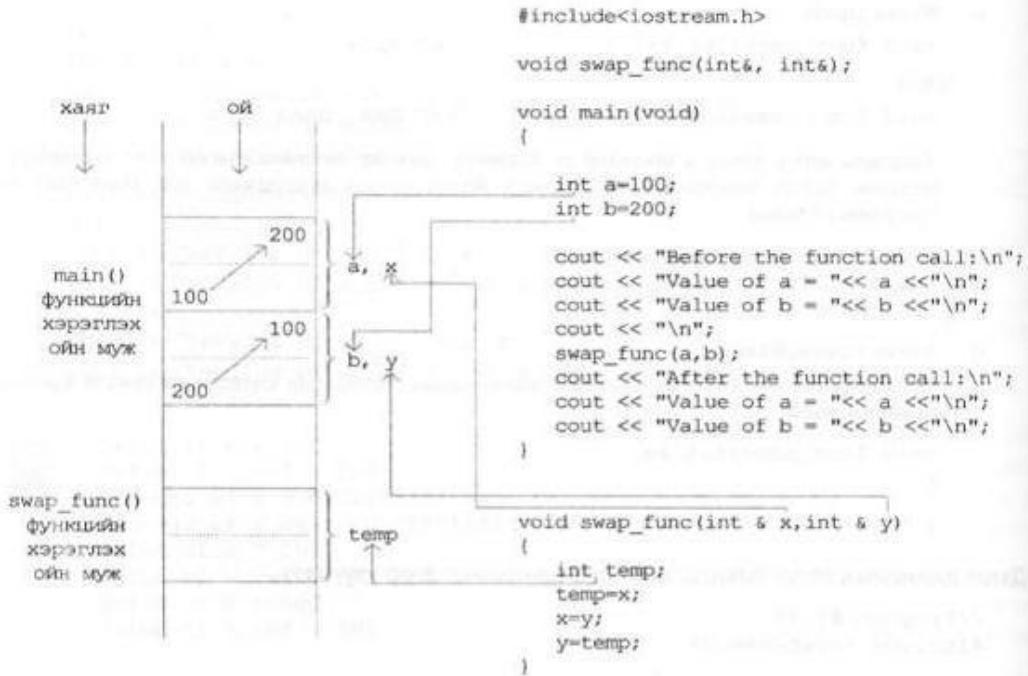
```
void func_name(int &x)  
{  
    //  
}
```

Дээрх дарааллын дагуу бичигдсэн жишигээ програмыг доор үзүүллээ.

```
//Program #2.33  
#include <iostream.h>  
  
void swap(int &, int &) ;  
void main(void)  
{int a = 100 ;  
int b = 200 ;  
cout << "\nBefore the function calling:" ;  
cout << "\nValue of a=" << a ;  
cout << "\nValue of b=" << b ;  
swap(a, b) ;  
cout << "\nAfter the function calling:" ;  
cout << "\nValue of a=" << a ;  
cout << "\nValue of b=" << b ;  
}  
  
void swap(int &x, int &y)  
{  
    int temp ;  
    temp = x ;  
    x = y ;  
    y = temp ;  
}  
  
Before the function call:  
Value of a=100  
Value of b=200  
After the function call:
```

Value of a=200  
Value of b=100

Дээрх программын ажиллах зарчмыг Зураг 2.23-д үзүүллээ. Функцийг зарлахдаа зөвхөн хуурмаг аргументийн төрөл, тоог нь зааж өгөхөд хангалттай байдгийг дээрх программаас харж болно.



Зураг 2.23: Заалтаар нь параметр дамжуулах

Функцийн параметр заалтаар өгөгдсөн тохиолдолд тухайн функци дотроос эх өгөгдэл шууд хандаж бодолт хийх боломжтой тул өөрчлөгдхөх ёсгүй ойн (хувьсагчийн) уттыг санамсаргүйгээр өөрчлөхөөс сэргийлж const үгийг хэрэглэдэг. Үүнийг доор үзүүлсэн жишээнээс харж болно.

#### Параметрийн утгыг өөрчлөх

```

int square(int &);

int square(int &x)
{
    int z=x*x ;
    x=100 ;
    return z ;
}

```

#### Параметрийн утгыг өөрчлөхгүй

```

int square(const int &);

int square(const int &x);
int square(const int &x)
{
    int z = x*x ;
    //x=100 ;
    return z ;
}

```

Функцийн параметрийг const үзээр тодотгож өгснөөр түүнийг функци нь өөр дотроосоо өөрчлөх боломжгүй болох тул босоо зураасын баруун талын кодын хэсгийн x=100; командын мөрийг тайлбар болгож тэмдэглэсэн.

### 2.2.2.2 Бүтэц ба заалтан хувьсагч

Бүтэн хувьсагчид заалтын оператор (&) хэрэглэж болохыг гаднаас person төрлийн өгөгдэл авах func\_name(person &) функцийн жишээн дээр үзүүллээ.

```
//Program #2.34
#include <iostream.h>
#include <string.h>
struct person
{
    char name[20] ;
    int age ;
    float height ;
};
void func_name(person &) ;
void main(void)
{
    person p = {"Bill", 30, 180.5} ;
    func_name(p) ;
    cout << "\nName = " << p.name ;
    cout << "\nAge = " << p.age ;
    cout.precision(2) ;
    cout << "\nHeight = " << p.height ;
}
void func_name(person &pp)
{
    strcpy(pp.name, "A") ;
    pp.age = 20 ;
    pp.height = 170.6 ;
}
Name = A
Age = 20
Height = 170.6
```

### 2.2.2.3 Заалт буцаах функци

C++ функци заалтан төрлийнх байж болдгийг доор үзүүлсэн жишээ програмаас харж болно.

```
//Program #2.35
#include <iostream.h>

int & func_name(int &) ;
void main(void)
{
    int x=10 ;
    int y=func_name(x) ;
    cout << "\nValue of x = " << x ;
    cout << "\nValue of y = " << y ;
}
int & func_name(int &z)
{
    return z;
}
Value of x = 10
Value of y = 10
```

Дээрх програмын `int y=func_name(x);` гэсэн мөрийг өөрөөр `func_name(x)=30;` гэж бичиж болно.

### 2.2.3 Динамик ойн оператор: new, delete

Си програмд хүснэгт байгуулахад бичиж өгөх доорх шиг

```
const int CAPACITY = 10 ;  
double arrName[CAPACITY] ;
```

команд нь авсан (`double`) давхар нарийвчлалт бодит тоонд хүрэлцэхүйц хэмжээний ойг бэлдэж эхлэл хаягийг нь `arrName` нэргэй холбох үргийг компайлерт өгнө. Ийм маягаар байгуулагдах хүснэгтийн хувьд түүний хэмжээтэй холбоотойгоор гарч болох 2 дутагдал байдаг нь

- хэмжээ нь огогдлийн тоо, хэмжээнээс бага байвал хэтрэлт (`overflow`) үүсэх
- хэмжээ нь огогдлийн тоо, хэмжээнээс хэт их байвал ойг үр ашиг багатай хэрэглэх

зэрэг юм. Эдгээр дутагдал нь хүснэгтийн хэмжээ програмыг машины хэл рүү буулгах шатанд тодорхой байх ёстой байдагтай холбоотойгоор гардаг. `CAPACITY` хувьсагчийн гарвааны утгыг өөрчлөөд програмыг машины хэл рүү дахин буулгах замаар `arrName` хүснэгтэд зориулан бэлдэх ойн хэмжээг ихэсгэж багасгаж болно.

C++ хэлний динамик ойн механизмын хэрэглэснээр дээрх асуудлыг оновчтойгоор шийдэж болно. Товчоор хэлбэл, ийм механизмын нь ой бэлдэх, дараа нь чөлөөлөх гэсэн 2 үйлдэл хийнз. Энэ механизм бол Си хэлний хувьд `malloc()`, `free()` функциүүд бөгөөд эхний функцизэр нь ой бэлдэж хэрэглээд буцааж чөлөөлөхдөө удаах функцийг нь хэрэглэнэ. Эдгээрийг програмд хэрэглэхийн тулд эх загвар нь байгаа `alloc.h`, `stdlib.h` толгой файлуудын аль нэгийг препроцессорын `#include` командаар зааж өгөх ёстой. Мөн `malloc()` нь хэрэглэх ойн хэмжээг тодорхой зааж өгөх, нэгэнт бэлдсэн ойг хаяган хувьсагчтай холбоодоо (ялангуяа үүсмэл төрлийн хувьд) төрөл хувиргалт хийх зэрэг нэмэлт ажил хийхийг шаарддаг түвэгтэй функци юм.

Дээрх хоёр функцийг C++ хэлэнд шууд хэрэглэж болно. Мөн `malloc()` функцийг орлох `new`, харин `free()` функцийг орлох `delete` гэсэн хоёр оператор C++ хэлэнд байдаг. Эдгээр нь хэрэглээ энгийнтэй операторууд юм. Тухайлбал, `new` нь `malloc()` функци шиг олон нэмэлт зүйл шаардлагтүй байна.

#### 2.2.3.1 new оператор

`new` нь програмын эхэнд толгой файл зарлах, ойн хэмжээг урьдчилан тооцох, хаяган хувьсагчид тохируулан төрөл хувиргалт хийх зэрэг нэмэлт ажил шаардахгүйгээрээ `malloc()` функцийг бодвол энгийн хэрэглээтэй оператор юм. Эл оператор сүл ойгоос иөөцөлж авсан ойн эхлэл хаягийг буцаана.

Програм ажиллах явцдаа (OS-Operating System) үйлдлийн системээс нэмэлт ой хүсэхдээ `new` операторыг хэрэглэж болно. Ийм хүсэлтийг C++ хэлнээ доорх шигээр буулгана.

```
new Type ;
```

Туре төрлийн объектод хүрэлцэх ойг бэлдэх хүсэлтийг системд тавих ба систем түүнийг биелүүлж чадвал бэлдсэн ойнхоо эхлэл хаягийг, ой бэлдэх боломжгүй бол хоёртын тэгийг `new` операторт буцааж өгнө. Энэ оператороос буцаж ирэх хаягийг төрөл нь тохирох хаяган

хувьсагчид хадгалж болох учир уг операторыг голдуу хаяган хувьсагчтай холбож хэрэглэнэ.  
Жишээ нь,

```
int *intPtr ;  
intPtr = new int ;
```

команд хийгдэх үед new int нь бүхэл тоонд хүрэлцэхүйц хэмжээтэй (sizeof(int) байт) ойг авч хэрэглэх хүснэгтийг системд тавина. OS хүснэгтийг билүүлэх боломжтой бол нооцолсон ойн хаягаа буцааж түүнийг нь intPtr хувьсагч руу хадгална. Харин OS хүссэн хэмжээний ойг гаргах боломжгүй бол хоёртын 0 (null) буцаасныг мөн intPtr хувьсагч руу хадгална. Иймд new оператороор ой бэлдэж чадсан, оороор хэлбэл буцааж ирсэн хаяг тэгээс ялгаатай эсэхийг хэрэглэхийн өмнө байнга шалгаж байх ёстой. Ингэхдээ сонголтын if команд юм уу assert()<sup>32</sup> функцийн аль нэгийг доор үзүүлсний дагуу хэрэглэж болно.

```
assert(intPtr!=0) ;
```

ЭСВЭЛ

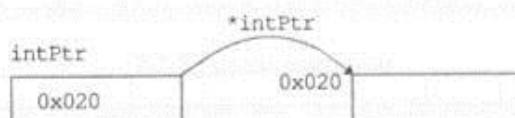
```
if(intPtr == 0 )  
{  
    cerr << "\n*** Not enough memory ***" ;  
    exit(1) ;  
}
```

Хэрэв intPtr хувьсагч хоосон биш бол түүнд байгаа хаяг бүхий ой нь нэргүй хувьсагч болно (Зураг 2.24). Ой бэлдэх new оператороос буцах хаяг, жишээ нь 0x020 нь intPtr хувьсагчийн утга болно.



Зураг 2.24: Динамик ой бэлдэх

Шинээр бэлдэх ойд тодорхой нэр оноохгүй учир нэрээр ойд хандах аргыг хэрэглэх боломжгүй тодлог. Гэвч ойн хаяг нь intPtr хувьсагчийн утга болох тул ийм нэргүй хувьсагч руу intPtr хувьсагчийн утгаар дамжуулан хандаж болно (Зураг 2.25).



Зураг 2.25: Дам хандалтын операторыг хэрэглэх нь

Нэргүй хувьсагчийн хувьд дараах үйлдлийг хийж болно.

```
cin >> *intPtr ; //Гараас оруулах тоог уншиж авах  
if(*intPtr < 100) //Харьцуулах үйлдэл хийх  
    (*intPtr)++ ; //Арифметик үйлдэл хийх  
*intPtr=100 ; //Шинэ утга оноох
```

Товчоор хэлбэл, ерийн бүхэл тоон хувьсагчийн хувьд хийж болох бүх үйлдлийг нэргүй хувьсагчийн хувьд intPtr хувьсагчаар дамжуулан хийж болно.

<sup>32</sup> void assert(int expression); бол өгөгдсөн expression илэрхийллийн хариу утга false/худал байвал энэ тухай мэдээллийг дэлгэнцээд программыг зогсоох стандарт функц юм.

Ойн new операторыг бусад төрлийн хувьд хэрэглэх жишээг доор үзүүллээ.

```
char *cvar = new char ;
int *ivar = new int ;
float *fvar = new float ;
```

#### 2.2.3.2 new оператор ба хүснэгт

Бодит байдал бүхэл тоо мэтийн ганц утга хадгалахад зориулан new оператороор ой бэлдэх нь ховор байдаг. Үнэндээ, энэ операторыг нэргүй нийлмэл объектод, жишээ нь хүснэгтэнд зориулан ой бэлдэхэд хэрэглэж болно.

Бидний мэдэх уламжлалт аргаар, жишээ нь арван бүхэл тоо хадгалах intArray хүснэгтийг дараах командаар байгуулаа.

```
int intArray [10] ;
```

Хүснэгтийн нэртэй уягдах утга бол уг хүснэгтийн эхлэл хаяг, оорөөр хэлбэл эхний бүтвэрийн нь хаяг байдаг. Иймээс хүснэгтийг хүснэгт рүү хуулахад утга оноох операторыг хэрэглэж болдоггүй байна. Жишээ нь, доорх alpha, beta нь хобулаа хүснэгт гэвэл

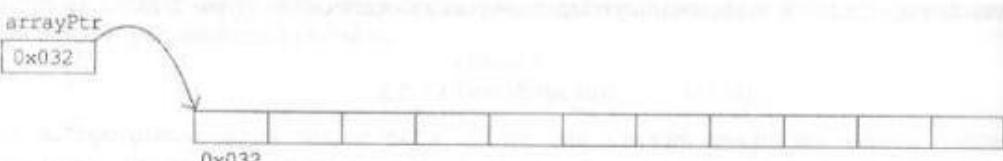
```
alpha = beta ;
```

команд beta хүснэгтийн бүтвэрүүдийг хуулахын оронд уг хүснэгтийн эхлэл хаягийг alpha руу хуулах оролдлого хийж улмаар алдаа гардаг байна<sup>33</sup>.

Дээрх хүснэгт intArray нь int[10] төрлийнх байна. Ийм хүснэгтэнд хэрэглэх ойг new оператор хэрэглэн динамикаар бэлдэж болно.

```
int *arrayPtr ;
arrayPtr = new int[10] ;
```

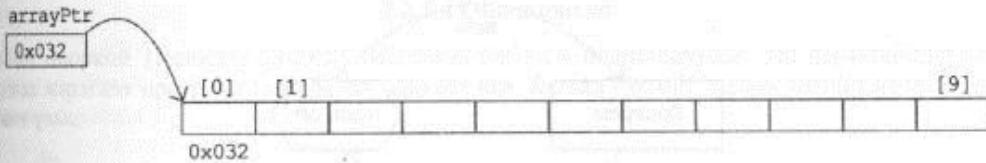
Эдгээр команд арван бүхэл тооны хүснэгтэнд хэрэглэх ойг бэлдэнэ. Хоёрдахь команда хийгдэх хүртэл arrayPtr бол утга нь тодорхойгүй ерийн хаяган хувьсагч байх ба ут команда хийгдсэний дараагаар, хангалттай сүл ой байсан тохиолдолд, arrayPtr нь шинээр байгуулсан ойн эхлэл хаягийг агуулж байдаг. Үүнийг Зураг 2.26-д байгаа шигээр үзүүлж болно.



Зураг 2.26: arrayPtr = new int[10]; командаар бэлдэх ойн дүрслэл

Си програмыг машины хэл рүү буулгах үед байгуулагдах хүснэгтийн эхлэл хаяг түүний нэртэй уялсан байдаг тухай өмнө үзсэн билээ. Статик аргаар байгуулагдсан хүснэгтийн бүтвэр рүү түүний нэрээр хандаж болдгийн адил динамик хүснэгтийн бүтвэр руу хаяган хувьсагчаар дамжин хандахдаа бүтвэрийн нэрийг бас хэрэглэж болно. Учир нь, ийм хүснэгтийн эхлэл хаяг нь хаяган хувьсагчид хадгалагдаж байдаг. Тэгэхлээр, дээрх хүснэгтийн эхний бүтвэрт arrayPtr[0], удаах бүтвэрт arrayPtr[1], сүүлийн бүтвэрт arrayPtr[9] гэх мэтээр хандах ба үүнийг Зураг 2.27-д үзүүллээ.

<sup>33</sup> Статик хүснэгтийн нэрийг хаяган тогтолцой зүйрлэж болох ба ийм тогтолцын уттыг програмаас оөрчилж болохгүй.



Зураг 2.27: Динамик хүснэгтийн бүтвэрт хандах

Дээр өгүүлснээр `arrayPtr` хувьсагчийн утга нь хүснэгтийн эхлэл хаягийг заах тул түүний `i`-дүгээр бүтвэрт хандах `arrayPtr[i]` нь `(arrayPtr+i)` дугаартай ойг хаяглана. Энэ талаар "Хаяг ба хаяган хувьсагч" бүлгээс дэлгэрүүлэн үзж болно.

Бусад төрлийн хүснэгтэнд зориулан ой бэлдэхэд `new` операторыг мөн хэрэглэнэ. Жишээ нь, бодит тоон 50 бүтвэртэй хүснэгтэнд ой бэлдэх командыг

```
float *farr = new float[50];
```

гэж бичиз. Энд `new` оператор нь 50 бүтвэртэй бодит тоон хүснэгтэнд ой бэлдэж түүний эхийн бүтвэрт харгалзах ойн хаягийг буцаасныг `farr` хаяган хувьсагч руу хадгална. Захиалсан ойг `new` оператор бэлдэж чадахгүй бол `null` буюу хоёртын тэг (\0) буцаадаг.

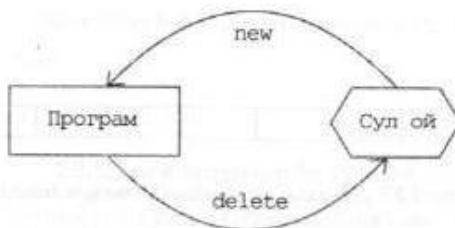
Хүснэгтэнд динамик ой хэрэглэхийн давуу тал нь машины програм гаргаж авах шатанд хүснэгтийн хэмжээг урьдчилан мэдэж байх шаардлагагүй байдагт оршино. Иймд дараах шиг жишиг бичиж болно.

```
cout << "How many entries ? " ; //хүснэгтийн хэмжээг авах
cin >> numEntries ;
assert (numEntries > 0) ; //numEntries > 0 эсэхийг
                           //шалгах
double *dPtr = new double[numEntries] ; //ой бэлдэх
assert (dPtr != 0) ; //хүснэгт байгуулсан
                           //эсэхийг шалгах
cout << "\nEnter your values: " ;
for (int i=0; i < numEntries; i++) //хүснэгт дүүргэх
    cin >> dPtr[i];
```

Динамик хүснэгтийн хэмжээ түүнд байх өгөгдлийн тоо хэмжээтэй тохирч байдаг тул статик хүснэгтийн хэрэглээтэй холбоотойгоор гарч болох асуудлыг бурэн шийдэх боломжтой юм.

### 2.2.3.3 delete оператор

Програм ажиллах явцдаа сул ойн сангаас авч хэрэглэх боломж бий бөгөөд үүний тулд хэрэгцээтэй ойн мужийг сул ойн сангаас хасч түүнийг програмд холбож егех хүсэлтийг `new` оператороор OS-д тавьдаг талаар өмнөх бүлэгт үзсэн билээ. Ийм аргаар бэлдэж авсан ойн хаягийг хаяган хувьсагчид хадгалсны дараагаар програм дотроос хэрэглэж болно. Сул ойн хэмжээ хязгаартай бөгөөд хэмжээ нь хэрэглэх `new` оператор бүрийн дараагаар багасаж байдаг. Иймд ойг хэрэглэж дуусад `delete` оператороор чөлөөлж сул ойн санд буцааж оруулж байх ёстой. `new` нь сул ойн сангаас тодорхой хэмжээний ойн мужийг авч хэрэглэх хүсэлт бол `delete` нь ойн мужийг сул ойн санд буцааж бүртгэхтэй холбоотой хүсэлт юм. Ингэхлээр, `delete` бол `new` оператороор бэлдсэн ойг чөлөөлөх үүрэгтэй оператор юм. Түүгээр чөлөөлсөн ойн хэсэг эзлжит `new` оператороор дахин неөшлөгдж болно. Иймд `new` ба `delete` нь харилцан уялдаатай, бие биенийхээ гүйцээлт болдог операторууд юм (Зураг 2.28).



Зураг 2.28: *new, delete* операторын уялдаа

*delete* операторыг хэрэглэх загварыг доор үзүүллээ.

```
delete pointerVariable ;
delete [] arrayPointerVariable ;
```

Эхнийх нь хаяган хувьсагчид хаяг нь байх объектын эзэмшиж байсан ойг чөлөөлнө. Жишээ нь,

```
int *intPtr = new int ;
 бол intPtr хувьсагчийн утгаар хаяглагдах ойг
 delete intPtr ;
```

гэж чөлөөлнө. Энэ үйлдлийн дараагаар intPtr хувьсагчийн утга тодорхой бус болох тул \*intPtr мэтийн үйлдэл хийвэл алдаа гарах шалтгаан болно. Ийм байдлаас сэргийлэх сайн арга бол доор үзүүлсэн шигээр хаяган хувьсагчийн утгыг 0 болгох юм.

```
delete intPtr ;
intPtr = 0 ;
```

Мен хаяган хувьсагчийн утгыг шалгах командыг хэрэглэж болно.

```
if (intPtr != 0)
{
    // intPtr хувьсагч руу хандаж болно.
}
else
{
    //intPtr хоосон
}
```

Эхлэл хаяг нь arrayPointerVariable хаяган хувьсагчид байгаа хүснэгтийн ойг дээр дурьдсан хоёрдахь загварын *delete* оператороор чөлөөлнө. Үүнийг хэрэглэх жишээ командыг доор үзүүллээ.

```
int *iptr=new int[50];
if(iptr == 0)
{
    error_message();
    exit(1);
}
. . .
delete [50] iptr ;
iptr = 0 ;
```

Дээрх жишээнд хэрэглэж байгаа iptr бол бүхэл тоон 50 бүтвэртэй хүснэгтийн эхний бүтвэрийн хаягийг хадгалах хаяган хувьсагч юм.

#### 2.2.3.4 Ойн цоорхой

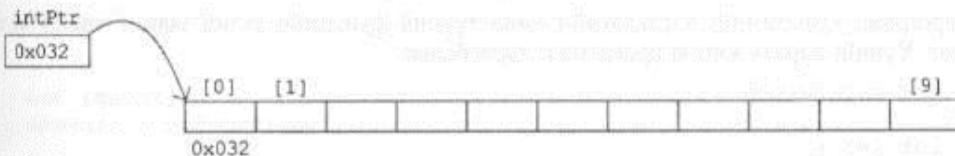
Ойн цоорхой (Memory leaks) бол `new`, `delete` операторуудыг дэс дараатайгаар олон удаа хэрэглэх програмд гарч болох ойлголт юм. Яагаад гэдгийг дараах энгийн жишээн дээр авч үзье:

```
do
{
    int *intPtr = new int[10] ;
    assert (intPtr!=0) ;

    . . .

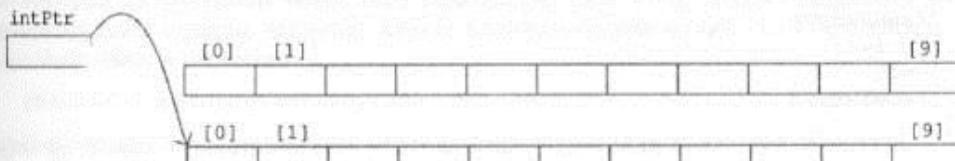
    cout << "\nDo another (y or J)?" ;
    cin >> answer ;
} while (answer!='\n') ;
```

Эхний давталтаар арван бүхэл тооны хүснэгт байгуулагдана (Зураг 2.29).



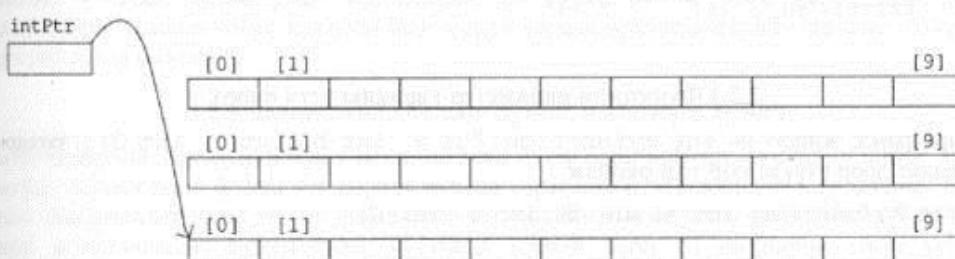
Зураг 2.29: Давталтын `new int[10]` командын үр дүн

Удаах давталтаар өмнөхөөс тусдаа хүснэгт үүсгэж хаягийг нь `intPtr` руу хийхийн өмнө эхний хүснэгтийг чөлөөлөхөөр `delete` операторыг хэрэглээгүй тул энэ ойн хэсэг програмд харьялгадсан хэвээр үлддэг. Гэхдээ хоёрдахь хүснэгтийн хаягийг `intPtr` хувьсагч руу хийснээр эхний хүснэгтийн хаяг арчигдах тул эхний хүснэгт эзэнгүй хаягдаж программаас хандах юм уу сүл ойн санд буцаах боломжгүй болно (Зураг 2.30).



Зураг 2.30: Хоёрдахь давталтын дараах байдал

Гуравдахь давталтаар ээлжит хүснэгтийг байгуулж хаягийг нь `intPtr` руу хийх үед өмнөх хоёрдахь хүснэгт бас эзэнгүйдэнэ (Зураг 2.31).



Зураг 2.31: Гуравдахь давталтын `new int[10]` командын үр дүн

Ийнхүү ээлжит давталт бүрд int[10] хэмжээтэй ой эзэнгүйдсээр тодорхой давталтын дараагаар сүл ой хүрэлцэхгүйн улмаас intPtr!=0 биелэхгүй болж програм дуусгавар болно. Ийм програм ойг эзэнгүйдэхэд хүргэх ба үүнийг ойн цоорхой гэнэ.

Дээрх байдалд хүрэхгүйн тулд хаяган хувьсагчид шинэ утга оноож өгөхийн өмнө түүний хаяглах ойг доор үзүүлсэн шигээр чөлөөлж байх ёстой.

```
do
{
    int *intPtr = new int[10] ;
    assert (intPtr != 0) ;
    ...
    delete [10] intPtr ;
    cout << "\nDo another (y or n) ?" ;
    cin >> answer ;
} while (answer != '\n') ;
```

#### 2.2.4 C++ хувьсагч зарлах

Си програмд хувьсагчийг хэрэглэхийн өмнө түүний функцийн эхэнд заавал зарлах ёстой байдаг. Үүнийг дараах жишээ програмаас харж болно.

```
void main(void)
{
    int i=3 ;
    //командууд
    i+=1;
    //командууд
}
```

Харин C++ програмын хувьд хувьсагчийг заавал функцийн эхэнд бус харин хэрэглэх үедээ зарлаж болно. Тэгэхлээр, өмнөх жишээ програмыг дараах байдлаар өөрчлен бичиж болно.

```
void main(void)
{
    //командууд
    int i=3;
    i+=1;
    //командууд
}
```

Өөр жишээ програмыг доор үзүүллээ.

```
void main(void)
{
    //командууд
    for(int i=0; i<=n; i++)
        printf("%d ", i);
}
```

#### 2.2.4 Функцийн параметрт гарсаны утга оноох

Си програмд, жишээ нь int myfunction(int a, int b, int c, int d); загварын функцийг доор үзүүлснээр тодорхойлж

```
int myfunction( int a, int b, int c, int d)
{
    ...
}
```

`main()` функц дотроос, жишээ нь `myfunction(1, 4, 3, 7);` гэж дуудаж хэрэглэнэ. C++ бол Си хэлийг бодвол чөлеөтэй бичлэгтэй, програмчлах ажлыг хялбар болгох талдаа уян хтан хийгдсэн хэл юм. Ийм шинжийн нэг бол `myfunction()` функцийн авах дөрвөн параметр бүгд эсвэл зарим нь ихэнх тохиолдолд тогтмол нэг утга авдаг бол тэдгээр утыг параметрийн гарааны утга хэлбэрээр зааж өгөх боломж юм. Иймд дээрх `myfunction()` функцийг тодорхойлохдоо түүний параметруүдийг гарааны утгатай байхаар зааж огч болно. Тухайлбал,

```
int myfunction(int a=10, int b=20, int c=30, int d=40) ;
```

тэж зарласан функцийн тодорхойлолтыг доор үзүүлснээр бичнэ.

```
int myfunction(int a=10, int b=20, int c=30, int d=40)
{
}
// ...
}
```

Ийм функцийг `main()` зэрэг бусад функц дотроос дараах байдлаар дуудаж болно.

```
void main(void)
{
    int iResult;
    iResult = myfunction();
}
```

Энд `iResult=myfunction();` командаар `myfunction()` функцийг дуудахдаа ямар ч аргумент дамжуулж өгөөгүй байхад уг командыг `iResult=yfunction(10, 20, 30, 40);` болгож хөрвүүлдэг байна. Энэ функцийн авах утыг дараах байдлаар дахин тодорхойлж болно.

```
iResult = myfunction(100, 200);
```

Энэ тохиолдолд функцийн эхний хоёр параметрийн авах уттыг дахин тодорхойлж үлдсэн параметруүдийн гарааны уттуудыг хэвээр үлдээсэн бөгөөд үүнийг C++ компайлер доор үзүүлснээр ойлгож өөрчилнэ.

```
iResult = myfunction(100, 200, 30, 40);
```

Харин функцийн параметрийн авах уттыг алгасаж тодорхойлж болдоггүй. Жишээ нь,

```
iResult = myfunction(100,
                     //буруу
                     300,
                     400) ;
```

Гарааны утгатай параметрийг жагсаалтын нь сүүлд бичиж өгөх ёстой. Аль нэгэн параметрийн гарааны уттыг хэрэглэх бол түүний дараах бүх параметрийн гарааны уттыг бас хэрэглэх ёстой болдог.

## 2.2.5 Дахин тодорхойлсон функц, полиморф функц

Дахин тодорхойлсон функц буюу полиморфизм буюу олон хэлбэрт функцийн шинж нь нэг програм доторх олон функц нэг нэрийг дундаа хэрэглэхийг зөвшөөрдөг. C++ хэлний хувьд дахин тодорхойлох гэдэг нь нэг зүйл олон утга холбогдож болохыг илэрхийлнэ. Тэгэхлээр дахин тодорхойлсон функц гэдэг нь ижил нэргэй олон функц байна гэсэн үг юм. Полиморфизм нь нэг зүйл олон оөр төлөвт оршиж болохыг илэрхийлэх ойлголт юм. Энэ талаар “Тавдугаар бүлэг”-ээс дэлгэрүүлж үзэж болно.

### Мэдлэгээ шалгах асуулт

- 2.1. Процедур хандлагат програмчлал гэж юу болох, түүний гол шинжийг нэрлэх
- 2.2. ОХП гэж юу болох, процедур хандлагат програмчлаас ямар ялгаатай болох
- 2.3. Хэн C++ хэлийг бүтээсэн, яагаад C++ гэж нэрлэх болсон
- 2.4. C++ хэлийн сайн шинж юу болох
- 2.5. C++ хэлийн бүтвэрийг тайлбарлах
- 2.6. C++ хэлийн дотоод өгөгдлийн төрлүүд юу болох, тэдгээрт шаардагдах ойн хэмжээ
- 2.7. C++ програмын бүтцийг жишээн дээр тайлбарлах
- 2.8. Арифметик операторууд, тэдгээрийн зориулалт
- 2.9. Нэмэгдүүлэх, хорогдуулах операторууд, тэдгээрийн хэрэглээ
- 2.10. Харьцуулах операторууд, тэдгээрийн зориулалт
- 2.11. Логик операторууд, тэдгээрийн зориулалт
- 2.12. `if-else` командын дүрэм, түүнийг програмд хэрхэн хэрэглэх
- 2.13. `switch-case` командын дүрэм, түүний зориулалт
- 2.14. `for` командын дүрэм, түүнийг хэзээ хэрэглэвэл тустай болох
- 2.15. `while` командын дүрэм, түүнийг хэзээ хэрэглэж болох
- 2.16. `do-while` командын дүрэм, `while` командаас ямар ялгаатай болох
- 2.17. Давхар давталт гэж юу болох, түүнийг хэрэглэх дүрэм
- 2.18. Хүснэгт гэж юу болох, түүний давуу тал юу болох
- 2.19. Нэг ба хоёр хэмжээст хүснэгт хоорондын ялгаатай байдал
- 2.20. Программ функц хэрэглэхийн ашиг тус юу болох
- 2.21. Параметр ба аргумент гэж юу болох
- 2.22. Аргументийг утгаар нь, заалтаар/хаягаар нь дамжуулахын ялгаа юу болох
- 2.23. Хувьсагчийн үйлчлэх хүрээ гэж юу болох
- 2.24. `auto`, `register`, `external` ба `static auto` хувьсагчуудын үйлчлэх хүрээ
- 2.25. Бүтэц гэж юу болох, бүтэц хүснэгтээс юугаараа ялгаатай болох
- 2.26. Гишүүн бүтвэр гэж юу болох, бүтэц ба гишүүн хоорондын хамаарал
- 2.27. Бүтцийн тодорхойлолтын загварыг бичих; бүтцийн тодорхойлолт дотор аль нэгэн гишүүн бүтвэрт гарцааны утга оноож болох эсэх
- 2.28. Бүтэн хувьсагчийг хэрхэн байгуулах
- 2.29. Бүтнээл шаардагдах ойн хэмжээг яаж тооцох
- 2.30. Бүтцийн гишүүн нь өөр бүтцийн хувьсагч байж болох эсэх
- 2.31. Хүснэгт бүтцийн гишүүн байж болох эсэх
- 2.32. Бүтэн хувьсагчийн гишүүдэд хэрхэн утга оноох
- 2.33. Бүтэн хүснэгтэнд хэрхэн гарцааны утга оноох
- 2.34. Бүтцийн гишүүд рүү яаж хандах
- 2.35. Бүтцийн гишүүд дээр яаж бодолт хийх
- 2.36. Цэг оператор гэж юу болох
- 2.37. Цэг операторыг яаж бүтэн хүснэгтэнд хэрэглэх
- 2.38. түрэдээ тусгай үгийн зориулалт юу болох, түүнийг бүтэцгэй яаж хэрэглэх

- 2.39. Бүтцийг функц рүү яаж дамжуулах
- 2.40. Нэрлэсэн тогтмол гэж юу болох, түүнийг яаж тодорхойлох
- 2.41. Нэрлэсэн тогтмолон төрөлд яаж утга оноох
- 2.42. Хоёроос цөөнгүй нэрлэсэн тогтмол ижилхэн утгатай байж болох эсэх
- 2.43. Нэрлэсэн тогтмолон хувьсагч гэж юу болох, түүнийг яаж тодорхойлох
- 2.44. Нэрлэсэн тогтмолон хувьсагч дээр хэрхэн бодолт хийх
- 2.45. Програмд нэрлэсэн тогтмолыг хэрэглэсний давуу тал
- 2.46. Нэрлэсэн тогтмолон төрлийн хэрэглээг жишээгээр үзүүлэх
- 2.47. Тэмдэгтэн тогтмол гэж юу болох
- 2.48. Тэмдэгтэн тогтмолыг яаж тодорхойлох
- 2.49. Тэмдэгтэн тогтмолын тодорхойлолт програмын аль хэсэгт байх

Жишээ бодлого

- 2.50. Өгөгдсон

```
int a = 4 ;
float b = 6.5 ;
char c = 'm' ;
```

командууд дээр үндэслэн дараах илэрхийллүүдийн хариуг олох

- (a) a++
- (b) ++a
- (c) b < 8
- (d) m < 'n'

- 2.51. Дараах командуудаас алдаатайг нь олох

- (a) int y = "b" ;
- (b) double v = "yes" ;
- (c) char s = "n" ;
- (d) long m = 12345678999 ;

- 2.52. Дараах багц командын дараагаар t-ийн утга ямар болох

```
t = 100 ;
for (m=-1; m<10; m++)
    for(n=m; n>-4; n--)
        t += m ;
```

- 2.53. Дараах программын үр дүнг бичих

```
#include <iostream.h>
int a = 1 ;
int b = 2 ;
void main()
{
    void get(int, int&) ;
    get(a,b) ;
    cout << a << b ;
}
void get(int y, int& x)
{
    b = x ;
    y += b ;
    x = y ;
    cout << x << y << b << endl ;
}
```

- 2.54. complex бүтцийг бодит тоон real, imaginary хоёр гишүүнтэй байхаар тодорхойлох
- 2.55. Бодлого 2.54-т тодорхойлох complex төрлийн x, y, z гурван хувьсагч тодорхойлох
- 2.56. Бодлого 2.54-т тодорхойлох complex төрлийн с хувьсагч тодорхойж түүний c.real, c.imaginary гишүүдэд 5.5, 2.5 гэсэн утгуудыг харгалзуулан оноох
- 2.57. Бүхэл тоон төрлийн hours, minutes, seconds гурван гишүүдтэй time бүтцийг тодорхойлох
- 2.58. Бүхэл тоон төрлийн day, month, year гэсэн гурван гишүүнтэй date бүтшийг тодорхойлох
- 2.59. Доор жагсаасан гишүүд бүхий customer бүтэц тодорхойлох
- (a) char төрлийн 20 бүтвэртэй name хүснэгт
  - (b) int төрлийн мэдээлэл хадгалах accountnumber гишүүн
  - (c) char төрлийн мэдээлэл хадгалах accounttype гишүүн
  - (d) бодит тоон мэдээлэл хадгалах balance гишүүн
  - (e) бодлого 2.58-д тодорхойлох date бүтцэн cusdate гишүүн
- 2.60. Бодлого 2.59-д тодорхойлох customer төрлийн с хувьсагч байгуулж түүний гишүүд, дэд гишүүдэд яаж хандахыг жишээн дээр үзүүлэх
- 2.61. Оюутны нэр, хувийн дугаар, нийт дун зэрэг үзүүлэлтэд нийцэх төрөл бүхий гишүүдтэй student бүтцийг тодорхойлох

Програмчлах боллого

- 2.62. Өгөгдсөн дөрвөн тооны ихийг олох
- 2.63. Өгөгдсөн жил (4 оронтой) нам жил мен эсэхийг шалгаж тогтоох
- 2.64. Хоёр бүхэл тоон хувьсагчийн уттыг хооронд нь гуравдах хувьсагч хэрэглэхгүйгээр сольж хийх
- 2.65. Гурвалжны гурван талын урт өгөгдсөн бол түүнийг зөв гурвалжин эсэхийг шалгах
- 2.66. Бүх 3 оронтой Фибоначийн тоог олох; (Эхний хоёр Фибоначийн тоо бол 1 ба 1. Бусад тоо нь емнох хоёр тооныхоо нийлбэртэй тэнцдэг. Тэгэхлээр 1,1, 2, 3, 5, 8, 13 г.м.)
- 2.67. Дараах илэрхийллийг бодох

$$1 + \frac{5}{3} - \frac{7}{5} + \frac{9}{7} - \frac{11}{9} + \dots + \frac{45}{43}$$

- 2.68. Утга нь 20-оос ихгүй байх Пифагорийн гурвалжныг олох
- 2.69. Өгөгдсөн бүхэл тооны бүх хуваагчийг олох
- 2.70. Дөрвөн оронтой бүхэл тоог хөрвөсөн тоон хэлбэрт оруулах
- 2.71. Өгөгдсөн хоёр зэрэг бүхэл тооны хамгийн их өронхий хуваагчийг олох
- 2.72. Дөрвөн бигийн хоёртын тоог аравтын тоонд хөрвүүлэх
- 2.73.  $40x^n < 2 \times 3^n$  тэнцтэгэл бишиг хангах хамгийн бага зэрэг бүхэл n-ийг олох
- 2.74. Фибоначийн 3, 6, 9, ..., 90 дүгээр тоог олох
- 2.75. 1000-аас их биш бүх зохномол тоог олох
- 2.76. 1000-аас бага бүх тогс тоог олох; Бүх хуваагчийнхаяа нийлбэртэй тэнцдэг тоог төгс тоог энэ. Жишээ нь, 6=1+2+3.

- 2.77. Хамгийн бага 4 оронтой анхны тоог олох
- 2.78. 3 оронтой Амстронгийн тооноос ихийг нь олох. Цифруудийнхээ куб зэрэгтийн нийлбэртэй тэнцэх тоог Амстронгийн тоо гэнэ. Жишээ нь,  $371 = 3^3 + 7^3 + 1^3$
- 2.79. 3 оронтой Фибоначийн тоонуудаас ихийг нь олох
- 2.80. Хоёр зэрэг бүхэл тооны хамгийн их сронхий хуваагчийг олох програмыг /, % хоёр операторыг хэрэглэхгүйгээр бичих
- 2.81. Цифрэн тэмдэгт мөр хэрэглэхгүйгээр дараах хэлбэрийн пирамидыг байгуулах

			1					
		2	3	2				
	3	4	5	4	3			
	4	5	6	7	6	5	4	
	5	6	7	8	9	8	7	6
	6	7	8	9	0	1	0	9
	7	8	9	0	1	2	3	2
	8	9	0	1	2	3	4	5
	9	0	1	2	3	4	5	6
	0	1	2	3	4	5	6	7

- 2.82. Бодлого 2.58-д тодорхойлогдох `date` төрлийн хоёр өгөгдлийг компьютерийн гараас авч тэдгээрийн ялгарыг олох
- 2.83. Бодлого 2.54+2.56-д тодорхойлох бүтцийн дагуу байгуулагдах хоёр комплекс тоог нэмэх, хасах, үржих, хуваах функциүүдийг бичих
- 2.84. Бодлого 2.57-д тодорхойлогдох `time` төрлийн хоёр өгөгдлийг компьютерийн гараас авч тэдгээрийн нийлбэрийг олох
- 2.85. Бодлого 2.59-д тодорхойлогдох `customer` төрлийн өгөгдлийг гаднаас авч дэлгэцлэх
- 2.86. Шимтгэлтэй холбоотой гишүүн өгөгдлийг `Program #2.15`-д шинээр нэмж ажилчны нийт цалинг олохдоо шимтгэлийг хасч тооцох
- 2.87. Шимтгэлтэй холбоотой гишүүн өгөгдлийг `Program #2.17`-д шинээр нэмж ажилчин бүрийн нийт цалинг олохдоо шимтгэлийг хасч тооцох

# ГУРАВДУГААР БҮЛЭГ

## КЛАСС, ОБЪЕКТ

- Объект гэж юу болох, 95
- Класс гэж юу болох, 95
- Классын тодорхойлолт, 96
- Класс зарлах, 96
- Гишүүн өгөгдөл, 97
- Гишүүн функц, 97
- private/public гишүүд, 97
- Өгөгдөл далдлал, битүүмжлэл , 101
- Объект байгуулах, 101
- Гишүүн өгөгдөл хандах, 102
- Гишүүн функц рүү хандах, 103
- Гишүүн функц тодорхойлох, 103
- Функц, объектын ялгаа, 106

Үргэлжлэл →



# ЗӨЛГӨ СААДЧИВАЧУУ

## -Үргэлжлэл-

- *private, public* гишүүн функц, 106
- Гишүүн функцийг гишүүн функцээс дуудах, 108
- Класс доторх хүснэгт, 109
- Объектон хүснэгт, 111
- Объект, функц, 113



## ГУРАВДУГААР БҮЛЭГ

### КЛАСС, ОБЪЕКТ

ООР нь огогдол хийсвэрлэл (data abstraction), удамшил (inheritance), өгөгдөл дашалт (data hiding), битүүмжлэл (encapsulation), олон хэлбэршил (polymorphism) зэрэг суурь ойлголт, тэдгээртэй хамаатай шинжүүд бүхий одоо цагийн програмчлалын дэвшилтэй технологи юм.

C++ нь ОО технологийн хэрэглээний гол хөдөлгүүр болдог нь уг хэлний өргөн боломжтой холбоотой. Ийм хөдөлгүүр болоход нь түүний олон чухал шинжийн нэг болох класс гэж ирлэдэг өгөгдөл хийсвэрлэл чухал үүрэгтэй байдаг.

Өгөгдлийн бүтэц бол яиз бурийн төрлийн өгөгдлийн нэгдэл юм. Ийм бүтцэн өгөгдлийг зүйл бурийн функцизэр боловсруулна. Класс нь ийм функци болон өгөгдлийг хоорондоо холбоотой нэгэн цул битүүмжилсэн логик зүйл гэж үздэг. Иймд класс бол өгөгдөл ба функцийн нэгдэл, хэрэглэгчийн тодорхойлох зохиомол өгогдлийн төрөл юм. Класс тодорхойлсны дараагаар түүний объектыг хялбар байгуулж болно.

#### 3.1 Объект гэж юу болох

Хүн орчноо хэрхэн таньж мэддэг вэ? Биднийг хүрээлэн буй орчин янз бурийн шинж тэмдэг, талев байдал, үйл хөдлөл бүхий объектуудаас тогтох бөгөөд бид тэдгээрээр дамжуулан орчин тойрноо танин мэддэг. Иймд эргэн тойронд байгаа юмыг сөрөнхийд нь объект гэж нэрлэвэл хүрээлэн байгаа орчин маань олон зүйлийн объектын нэгдэл болно. Ийм объект бүр өөрийн гэсэн шинж тэмдэг, үйл хөдлөлтэй байх ба объектын шинжийг өгөгдөл (data), үйл хөдлөлийг нь функци function) гэж нэрлэж болно.

Объектуудыг тэдгээрт байдаг нийтлэг шинжээр нь бүлэглэн ангилж болно. Жишээ нь, тээврийн хэрэгслийн бүлэгт багтах машиныг хоорондоо ялгаатай шинжээр нь суудлын машин, автобус, ачааны машин гэх зэргээр ангилж болно. Эдгээр нь машинд байх нийтлэг шинж тэмдгээс гадна бусаддаа байдаггүй шинж тэмдгүүдтэй байна.

Объект нь машин шиг заавал хөдөлдөг байх албагүй. Сургалт явуулдаг мэргэжлийн чиглэлээрээ МУИС-ийн салбар сургууль, факультетүүд хоорондоо ялгаатай. Гэхдээ сургуульд байх ёстой нийтлэг зүйлс сургууль, факультет бүрд байна.

Компьютерийн дэлгэц дээр зурах цэг бол бас объект юм. Цэгийн байршил, онго бол түүний шинж болно. Цэгийг зурж, арчиж өнгийг нь өөрчилж болно.

Объект нь банкны харилцагч мэтийн хийсвэр объект байж болно. Иймээс объектын тухай ойлголт бол програм бичихдээ юуг хэрхэн хийсвэрлэхээс хамаарна.

Дээр жишээ татсан объектуудаас үзэхэд объект бүр өөрийн гэсэн шинж, үйл хөдлөлийн төлөвлэтийн байна. Объект ямар нэгэн зүйл хийнэ. Тэр хөдөлж, өөрийгөө хүрээлэн байгаа орчны объектод нөлөөлж чадна. Иймд объект нь өгөгдөл ба өгөгдөл боловсруулалт хийх функцийтэй байна. Объект бүрд байх ийм өгөгдлийг объектын гишүүн өгөгдөл (data members) гэнэ. Функци нь объектын гишүүн функци (member functions) болно.

#### 3.2 Класс гэж юу болох

Класс нь объектын загвар юм. Объектыг загварын дагуу бүтээнэ. Жишээ нь, байшинг тодорхой зураг төсөл, загварын дагуу барих ба ийм загварыг класс, загварын дагуу барьсан байшинг объект гэж болно. Бас аль ч үйлдвэрийн ажилчин бүр нэртэй, ажил үйлчилгээ

явуулсны толеө авах суурь цалин, тэтгэлэгтэй байна. Сарын цалин бодоход ажилчны мэдээллийг гаднаас авч харуулах `getdata()`, `showdata()` функц хэрэгтэй байж болно. Тодорхой нэр бүхий ажилчин бол ажилчдын нэг төлөөлөл болно. Тэгэхээр, Ананд (Anand) гэдэг ажилчин суурь цалин (`500$`), тэтгэлэгээ (`200$`) сар бүр цалингийн цэсээс харж авдаг бол түүнийг `employee` (ажилчин) классын нэгэн төлөөлөл тэж үзэж болно (Зураг 3.1).

Класс	<code>Employee</code>	Объект	
DATA:	<code>name</code> <code>basicpay</code> <code>allowance</code>	DATA:	Anand 500 200
FUNCTIONS:	<code>getdata</code> <code>showdata</code>	FUNCTIONS:	<code>get the data</code> <code>show the data</code>

Зураг 3.1: Класс, объект хоорондын харьцаа

### 3.3 Классын тодорхойлолт

Тодорхойлолт нь классын тодорхойлолт, классын гишүүн функцийн тодорхойлолт гэсэн хоёр хэсэгтэй байна.

**Классын тодорхойлолт.** Класс нь гишүүн өгөгдол, гишүүн функцийн нэгдэл түз тодорхойлолт дотор нь өгөгдлийг тодорхойлж, функцийт гол төлөв зарлана.

**Классын гишүүн функцийн тодорхойлолт.** Классын гишүүн функцийн тодорхойлолтыг классын тодорхойлолтын нь гадна талд хийнэ. Гэхдээ гишүүн функцийг классын тодорхойлолт дотор тодорхойлох боломж бас байдаг.

Классын тодорхойлолт ерөнхийдөө бүтцийнхтэй тестэй. Нээх их хаалтаар эхэлж ардаа цэгтэй таслал бүхий хаах их хаалтаар төгсөнө. Эдгээр хаалт хооронд классын гишүүн өгөгдол, гишүүн функцийг жагсааж бичдэг. Классын тодорхойлолтын загварыг доор үзүүлснээр бичиж болно.

```
class class_name
{
    private/protected/public:
        data_members ;
    private/public:
        member_functions ;
}
```

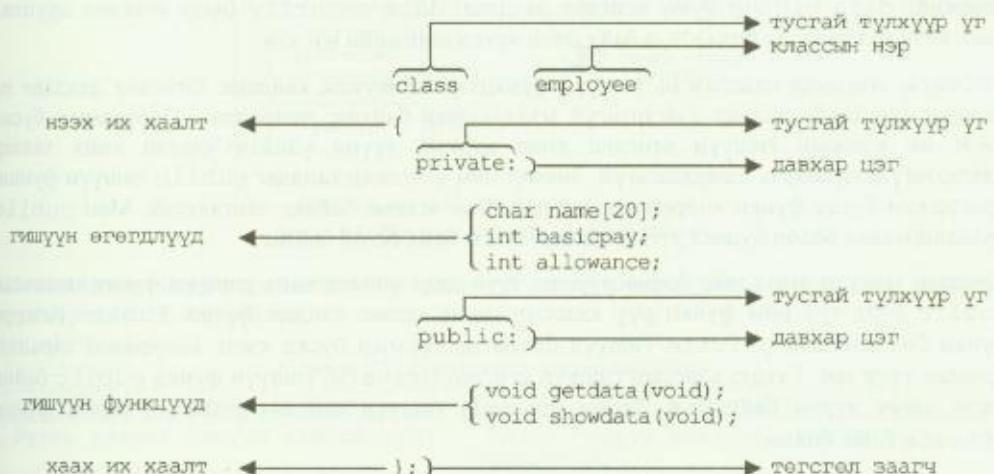
Зураг 3.1-д томъёолсон `employee` классыг дээрх загварын дагуу Зураг 3.2-т үзүүлснээр тодорхойлж болно.

### 3.4 Класс зарлах

Зураг 3.2-т үзүүлсэн тодорхойлолтод `class employee` бүхий эхний мөрөнд байгаа `class` нь түүний дараах `employee` нь классын нэр болохыг заах үүрэгтэй тусгай үг юм. Классын гишүүдийн тодорхойлолт, жагсаалтын хэсэг нь нээх их хаалтаар эхэлж ардаа цэгтэй таслал бүхий хаах их хаалтаар төгсөнө. Классын гишүүдийн тодорхойлолт нь `private`, `public`, `protected`<sup>34</sup> хэмээх түрүүлэгчийн хэсэгтэй байж болно.

<sup>34</sup> `Private` нь хувийн, хаалттай, хаагдмал; `public` нь ийнтийн, илтгэтийн; `protected` нь хувийн тус тус илтгэнэ.

Private хэсэг ардаа давхар цэг (:) бүхий private тусгай үгээр эхэлнэ. Үүнтэй адил, public хэсэг ардаа давхар цэгтэй public, харин protected хэсэг ардаа давхар цэгтэй protected тусгай үгээр тус тус эхэлнэ. Private хэсэгт байгаа гишүүд нь private гишүүн өгөгдөл эсвэл private гишүүн функц болно. Үүний адил, public хэсгийн гишүүд нь public гишүүн өгөгдөл эсвэл public гишүүн функц болно. Ихэвчлэн гишүүн өгөгдөл protected шинжтэй байна.



Зураг 3.2: employee классын тодорхойлалт (private бүхий)

Классын тодорхойлолтоор ой бэлддэггүй. Энэ бол зөвхөн классан хувьсагч буюу объект үүсгэх логик загвар юм.

### 3.5 Гишүүн өгөгдөл

Бүтэц гишүүн өгөгдөлтэй байдаг шиг класс бас гишүүн өгөгдөлтэй байна. Зураг 3.2-т үзүүлсэн жишээний name, basicpay, allowance зэрэг нь бүтцийн гишүүн өгөгдлийг тодорхойлдог тэр жаягаар тодорхойлогдононг харж болно.

### 3.6 Гишүүн функц

Гишүүн функц классынхаа гишүүн өгөгдөл дээр боловсруулалт хийж програмын бусад функцтэй мэдээлэл солилцно. Зураг 3.2-т тодорхойлсон employee класс нь getdata(), showdata() гэсэн хоёр public гишүүн функцтэй.

### 3.7 private/public гишүүд

Програм нь олон классын иэгдэл юм. Класс түүний гишүүд рүү хандах хандалтын private, public, protected гэсэн гурван түвшинтэй. Private, public тусгай үгс классын гишүүдийн үйлчлэх хүрээг заана. Private гишүүд рүү зөвхөн тухайн класс дотроос, тухайлбал гишүүн функцээс хандаж болно. Public гишүүд рүү бас классын гаднаас, тухайлбал өөр классаас, өөр функцээс хандаж болдог. Харин protected түвшин нь тухайн классынхаа хувьд private түвшинтэй адил юм. Гэхдээ тэдгээр нь уdamших байдлаараа хоорондоо ялгаатай. Энэ талаар "Удамших, уdamшуулах" бүлэгт тодорхой авч үзсэн.

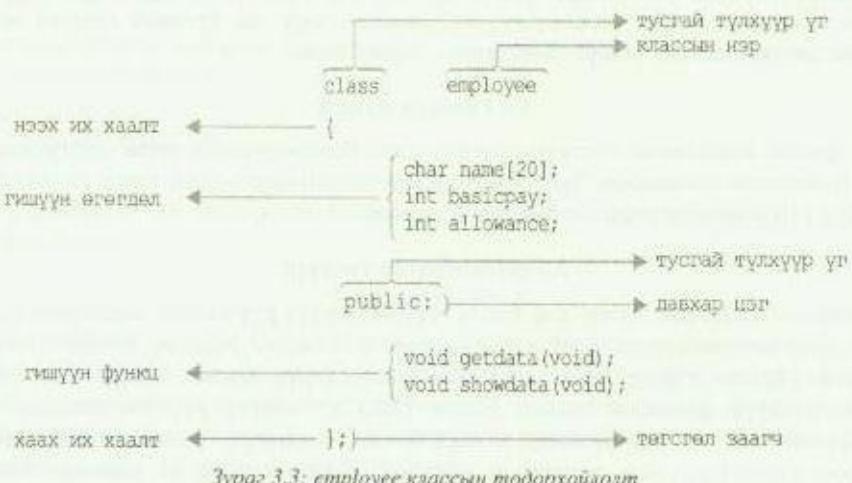
Классын тодорхойлолт доторх тусгай үгийн үйлчлэх хүрээ нь дараагийн тусгай уг эсвэл тодорхойлолтын төгсголийн хаах их хаалт гарах хүртэл үргэлжилнэ. Жишээ нь, өмнө тодорхойсон `employee` класс доторх `private` үгийн үйлчлэх хүрээ нь `public` уг гарах хүртэл үйлчилнэ.

Гол төлов, классын гишүүн өгөгдөл `private`, гишүүн функци `public` шинжтэй байна. Тэгэхлиэр `private` гишүүн өгөгдөл нь програмын бусад функцийн хувьд далд байна. Ийм шинжийг `data hiding` буюу огогдол далдлал, `data security` буюу огогдол нуушал гэнэ. Огогдол далдлал бол OOP-д байх олон чухал шинжийн нэг юм.

`Private` өгөгдлөд классын нь гишүүн функцийн дамжуулж хандана. Өгөгдөл далдлал нь гишүүн өгөгдлийн талаар дэлгэрэнгүй мэдээлэлтэй байхаас чөлөөлдөг. Программын бусад хэсэг нь классын гишүүн өгөгдөл ямар нэргэй, түүнд үйлэлт хэрхэн хийх талаар дэлгэрэнгүйгээр мэдэх шаардлагагүй. Зовхон ийм өгөгдлөд хандаж `public` гишүүн функци, программын бусад функци хоорондын интерфейсыг мэдэж байхад хангалттай. Мөн `public` функцийн авах болон буцаах утгын талаар мэлж байх ёстой байдаг.

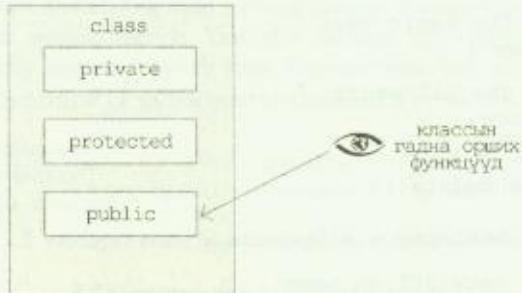
Классын гишүүн огогдлийг боловсруулах, түүн дээр үйлдэл хийх гишүүн функци ихэвчлэн `public` байх тул ийм функци рүү классын гадна талаас хандаж болно. `Public` гишүүн функци бол классын `private` гишүүд болон программын бусад хэсэг хоорондоо харьцах, ярицах гүүр юм. Гэхдээ классын гишүүн отөгдөл `private`, гишүүн функци `public` байна гэсэн хатуу дүрэм байдагтгүй. Зарим програмд гишүүн огогдол `public`, зарим функци `private` байж болно.

Хоорондоо логик уялдаатай өгөгдлийн нэгдэл тодорхойлоход бүтшиг гол төлөв хэрэглээн. Харин класс нь логик холбоотой гишүүн өгөгдөл, гишүүн функцийн нэгдэл юм. Бүтцийн гишүүд анхнаасаа `public` шинжтэй байх тул программын аль ч хэсгээс бүтцийн гишүүд рүү хандаж болдог. Гэтэл классын гишүүд `private` эсвэл `public` шинжтэй байна. Гэхдээ өөроор зааж огоогүй, тухайлбал `private`, `public` хоёр үгийн аль нэгийг хэрэглэсгүй бол классын бүх гишүүн анхнаасаа `private` байна. Иймээс `private` үгийг завал бичиж өгөх шаардлагагүй байдаг. Өмнөх `employee` классыг Зураг 3.3-т үзүүлснээр тодорхойлж болно.



Гэхдээ программын кодын уншууртай байдлыг хангах үүднээс `private` үгийг классын тодорхойлолтод хэрэглэж байх нь зөв хэрэглээ юм.

Private гишүүдэд классын гадна талаас хандаж болдогтүй. Хэрэв классын гишүүн өгөгдөл, гишүүн функции public хэсэгт байвал тэдгээрт классын гаднах функцийн хандаж болно (Зураг 3.4).



Зураг 3.4: Классын гаднаас хандах боломж

Классын тодорхойлолт ерөнхийдөө бүтцийнхтэй төстэй ч тэдгээрийн хооронд байх ялгаатай талыг Хүснэгт 3.1-д үзүүллээ.

Хүснэгт 3.1: Бүтэц, класс хоорондын ялгаа

Бүтэц	Класс
Бүтэц зөвхөн гишүүн өгөгдлөтэй.	Класс гишүүн өгөгдөл, гишүүн функцийтэй.
Бүтцийн бүх гишүүд public шинжтэй.	Классын бүх гишүүд өөрөөр зааж өгөгчүй бол анхнаасаа private шинжтэй.
Бүтцийн гишүүдэд програмын бүх функцийн хандана.	Классын гишүүн өгөгдөл түүний гишүүн функцийн дамжуулж хандана.
Бүтцийн гишүүд далд биш.	Классын гишүүн өгөгдөл далд байх тул уг классын бус функцийн хандаж чадахгүй.

Ажилчны цалинг бодох программыг employee класс түшиглэн хэрхэн бичихийг авч үзье.

```

//Program #3.1
//Finding the gross pay of an employee using class

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void getdata(void);
    void showdata(void);
} ;
  
```

```

void employee::getdata(void)
{
    cout << "\nEnter the Employee Name: ";
    gets(name) ;

    cout << "Enter the Basic Pay: ";
    cin  >> basicpay ;

    cout << "Enter the Allowance: ";
    cin  >> allowance ;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance ;

    cout << "\n" << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
    cout.width(8);
    cout << grosspay ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Gross" ;
}

void main()
{
    clrscr() ;
    employee emp ;
    emp.getdata () ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

 Enter the Employee Name: George.
Enter the Basic Pay: 9000.
Enter the Allowance: 2000.

Employee Name  Basic  Allowance  Gross
                George      9000      2000     11000

```

Дээрх програмл тодорхойлсон `employee` класс нь `name`, `basicpay`, `allowance` гэсэн турван гишүүн өгөгдэл, `getdata()` ба `showdata()` гэсэн хоёр гишүүн функцигтэй. Классын `private` гишүүн өгөгдэл зөвхөн гишүүн функцизэр нь дамжуулж хандана. Ийм өгөгдэл `main()` функцизэс шууд хандаж чадахгүй. Иймд дээр дурдсан функциүүд нь `employee` класс `main()` функци хоорондын гүүр интерфейс юм (Зураг 3.5).

Program #3.1-д дэлгэцэх зүйлийн өргөнийг тогтооход `setw(int)` функцийг хэрэглэсэн. Үг функцийг хувь эх зүйлийнхээ омне хэрэглэх ба энэ нь түүний дараах зөвхөн илгэ гаргах операторт үйлчилнэ. Программын үр дүнг хүснэгтэлж гаргахад хэрэглэдэг энэ функцийн эх загвар `iomanip.h` толгой файлд байдал. Хэвлэх объектын өргөнийг `showdata()`

функцийн тодорхойлолтод үзүүлснээр соул объектын `width(int)` функцийн бас тогтоож болно.

Гишүүн функцийг `accessor` буюу хандагч, `modifier` буюу өөрчлөгч, `modifying function` буюу өөрчлох функцийг хэж хоёр ангилж болдог. Хандагч функцийн гишүүн өгөгдөл рүү хандахдаа түүнийг өөрчилгэгүй. Хамгийн багадаа нэг гишүүн өгөгдлөө өөрчилдэг функцийг өөрчлөгч буюу өөрчлох функцийг энэ. Өөрчлөгчийг бас `mutator` буюу хувиргач гэнэ. Дээрх програмд `getdata()` нь өөрчлөгч, `showdata()` нь хандагч функцийн.



Зураг 3.5: `employee` класс ба `main()` функцийн хоорондын интерфейс

### 3.8 Өгөгдөл далдлал, битүүмжлэл

Класс нь объектынхоо загвар болдог талаар өмнө үзсэн билээ. Объектын утга далд байна. Гэхдээ энэ нь түүнийг бусдаас хамгаалахаар нуух гэсэн санаа биш. Объектын өгөгдэл зөвхөн гишүүн функцийн нь дамжуулж хандахыг хэлнэ. Объектын өгөгдэл тухайн классын биш функцийн хандаж болохгүй. Ингэснээр гишүүн өгөгдөл санаандгүй үйлдлээс хамгаалагдана.

Өгогдол, түүн дээр үйлдэл хийх функцийг хамтад нь холбоотой авч үздэг нь ОО технологийн нэгэн чухал шинж юм. Еронхийдоо, өгөгдөл ба функцийн хамтдаа класс гэгдэх нэгэн цогц ойлголтонд нэгддэг. Иймд битүүмжлэл бол функцийн өгөгдлийг тодорхой түвшинд ишлэлтэй биш объектын хүрээнд хамтад нь холбоотойгоор авч үздэг ойлголт юм.

### 3.9 Объект байгуулах

Класс тодорхойлсны дараагаар түүний хэдэн ч хувьсагч зарлаж болох ба ийм (классан) хувьсагчийг товчоор объект, зарлах үйлийг нь объект байгуулах гэнэ. Өмнөх Program #3.1-ийн `main()` функцийн дотор байгаа

```
employee emp ;
```

ий С++ хэлний `char`, `int` мэтийн тусгай үг хэрэглэн хувьсагч зарлахтай тестэй юм. Класс бол объект байгуулахад хэрэглэх өгөгдлийн зохиомол төрөл. Үг командаар байгуулагдах `emp` объектод зориулж C++ компайлэр ой бэлдэнэ (Зураг 3.6).



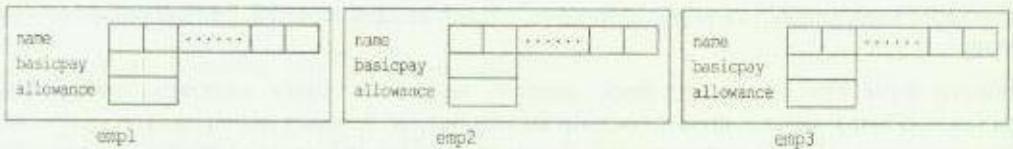
Зураг 3.6: `emp` объектын дүрслэл

`emp` бол `employee` классын хувьд байж болох олон объектын нэгэн төлөөлөлт объект юм. Байгуулагдах объект бүрийн гишүүн өгөгдлөө ой бэлдэнэ. `employee` классын `emp1`, `emp2`, `emp3` гэсэн нэр бүхий турван ялгаатай объектыг дараах командаар байгуулахад

```
employee emp1, emp2, emp3 ;
```

эдгээр объект тус бүрд ой бэлдэнэ (Зураг 3.7).

Гишүүн өгөгдөл классын объект бүрд, гишүүн функцийн классын хувьд зөвхөн нэг удаа байгуулагдана. Нэг классын объектууд классын гишүүн функцийн дундаа хэрэглэнэ.



Зураг 3.7: Объект тус бүрийн утга тусдаа ойд байрлана

### 3.10 Гишүүн огогдолд хандах

C++ хэлний `char`, `int` мэтийн огогдлийн төрөл, хувьсагч хооронд байдаг харьцаа бас класс, объект хооронд байдаг. Иймээс объектыг C++ хэлний бусад огогдлийн адилаар хэрэглэж болно. Тухайлбал, функцийн авах, буцаах утга нь объект байж болно. Мөн шинэ объектод шаардагдах ойг `new` оператороор бэлдэх боломжтой.

Бүтцийн огогдолд бүтцэн хувьсагчаар дамжуулан хандаж болдог. Гэтэл классын `private` болон `protected` гишүүн огогдол рүү объектоор нь дамжуулж хандаж болдоггүй. Иймд өмнөх программын `main()` функцийн дотор

```
cin >> emp.basicpay ;
```

командын мөрийг нэмж оруулсны дараагаар программыг шалгахад алдаа гарч систем

*"basicpay can not be accessible"*

мэдээллийг дэлгэцлэнз, `employee` классын огогдлууд `private` шинжтэй учир классынхаа биш функцийн хувьд далдлагдмал байдаг нь дээрх алдаа гарах шалтгаан болдог. Энэ нь гишүүн огогдлийн үйлчлэх хүрээ нь классын гишүүн функцийн хязгаарлагдсан байдагтай холбоотой. Энэ бол функци, класс хоорондын ялгаатай чанаруудын нэг юм.

Классын гишүүн огогдолд гишүүн функцийн нь дамжуулж хандана. Гэхдээ бүх гишүүн огогдлийг `public` гэж тодорхойлж отвэл ийм огогдолд объектоор нь дамжуулж хандах боломжтой. Энэ тохиолдолд класс гишүүн функцийтэй байх шаардлагагүй болно. Үүнийг шалгаж магадлах үүднээс өмнөх программыг дараах байлаар бөрчлөн бичиж болно.

```
//Program #3.2
//Finding the gross pay of an employee using class

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
public:
    char name[20];
    int basicpay;
    int allowance;
};

employee emp;
void getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(emp.name);
    cout << "Enter the Basic Pay: ";
    cin >> emp.basicpay;

    cout << "Enter the Allowance: ";
}
```

```

    cin >> emp.allowance ;
}

void showdata()
{
    int grosspay = emp.basicpay + emp.allowance ;
    cout << setw(20) << emp.name ;
    cout << setw(8)  << emp.basicpay ;
    cout << setw(12) << emp.allowance ;
    cout.width(8);
    cout<< grosspay ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Gross" ;
    cout << endl ;
}

void main()
{
    clrscr() ;
    getdata() ;
    heading() ;
    showdata() ;
    getch() ;
}

```

Энэ програмын үр дүн нь Program #3.1-ийнхтэй адил байна. Классын name, basicpay, allowance гишүүд рүү emp объектоор дамжуулан хандсан бөгөөд employee класс нэг ч гишүүн функцийг болохыг програмын кодоос харж болно. Иймд Program #3.2-т OOP боломжийг огт хэрэглээгүй байна.

### 3.11 Гишүүн функцийн рүү хандах

Классын гишүүн функцийн рүү тухайн классын объектоор нь дамжуулж хандана. Ингэхдээ объектын нэрийг гишүүн функцийн нэрээс шууд сонголтын цэг (.) оператороор зааглаж бичин. Program #3.1-д employee классын getdata(), showdata() функцийндийг emp объектоор дамжуулж дараах байдлаар дууддаг.

```

emp.getdata() ;
emp.showdata() ;

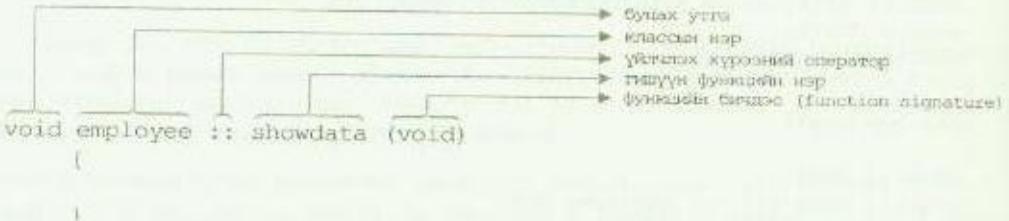
```

Гэхдээ гишүүн функцийн нэрийн өмнө классын нэрийг цэг оператороор зааглаж бичих дүрэм зөрчигдөх тохиолдол бий. Гишүүн функцийн дотроо классынхаа өөр гишүүн функцийг дуудаж хэрэглэж байвал дуудагдаж байгаа гишүүн функцийн нэрийн омно ямар ч объектын нэр бичихгүй, ингэж бичих ч боломжгүй юм.

### 3.12 Гишүүн функцийн тодорхойллох

Гишүүн функцийг тухайн классын нь тодорхойлолт дотор эсвэл классын тодорхойлолтын гадна тодорхойлж болно. Функцийн тодорхойлолт классын гадна байх тохиолдолд Зураг 3.8-д үзүүлсэн загварыг дагана.

Өмнөх Program #3.1-д employee классын `getdata()`, `showdata()` функциүүдийг Зураг 3.8-д үзүүлсэн байгаа загварын дагуу классын гадна тодорхойлсон байна. Функцийн тодорхойлолтыг классын нь тодорхойлолтын гадна талд хийх арга нь хэрэглэгчийн функцийн тодорхойлох аргатай төстэй. Функцийн тодорхойлолтын толгойн хэсэгт функцийн нэрийг классын нь ирээдээ үйлчилх хүрээний `(::)` оператороор зааглах бичнэ. Энэ оператор нь Зураг 3.8-д үзүүлсэнээр `showdata()` функцийг employee классын учир түүнчлийн харьяалагдж, түүний объектод үйлчилнэ гэдэг утсыг илэрхийлнэ.



Зураг 3.8: Классын гишүүн функцийг тодорхойлох загвар

Ер нь, үйлчилх хүрээний оператор, жишээ нь программын ерөнхий хувьсагч, функцийн дотоод хувьсагч ижил нэртэй байх тохиолдолд тухайн функции дотроос ерөнхий хувьсагчид хандах боломжийг олгодог оператор юм. Ингэхдээ ерөнхий хувьсагч руу хандахдаа түүний нэрийн омнё `(::)` операторыг бичнэ. Үйлчилх хүрээний операторыг хэрэглэх жишээ програмыг Program #3.3-т үзүүллээ.

```

//Program #3.3
#include <iostream.h>
#include <conio.h>

int sybil ;
void subfunction() ;

void main()
{
    sybil = 2 ;
    cout << "\nThis is global variable sybil= " << sybil ;
    subfunction() ;
    getch() ;
}

void subfunction()
{
    int sybil ;
    sybil = 1 ;
    ::sybil++ ;
    cout << "\nThis is local variable sybil = " << sybil ;
    cout << "\nThis is global variable sybil = " << ::sybil << "\n" ;
}
  
```



This is global variable sybil = 2  
 This is local variable sybil = 1  
 This is global variable sybil = 3

Гишүүн функцийг бусад функцийсээ хандаж болно. Функцийт классын нь тодорхойлолт дотор ньг дор зарлаж бас тодорхойлж болно. Классын дотор тодорхойлсон гишүүн функции дотоод мөр (`inline`) функцийг маягаар байтуулагдана. Program #3.1-ийн employee классын

`getdata()`, `showdata()` функцуудийг дотоод функц болгон Program #3.4-т үзүүлсэн шигээр өөрчилж болно.

```
//Program #3.4
//Finding the gross pay of an employee using class

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void getdata()
    {
        cout << "\nEnter the Employee Name: ";
        gets(name);
        cout << "Enter the Basic Pay: ";
        cin >> basicpay;
        cout << "Enter the Allowance: ";
        cin >> allowance;
    }

    void showdata()
    {
        int grosspay = basicpay + allowance;
        cout << setw(20) << name;
        cout << setw(8) << basicpay;
        cout << setw(12) << allowance;
        cout.width(8);
        cout << grosspay;
    }
};

void heading()
{
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
    cout << endl;
}

void main()
{
    clrscr();
    employee emp;
    emp.getdata();
    heading();
    emp.showdata();
    getch();
}
```

Энэ програмын үр дүн Program #3.1-ийнхтэй адил байна.

Классын дотор тодорхойлсон функцийн анхаасаа л дотоод мөр функцийн байх тул систем түүний командын хэсгийг уг функцийн дуудах газар бүрд дахин дахин хувилж бичнэ. Үйлчлэх хүрээний операторыг дотоод гишүүн функцийн хувьд хэрэглэдэгтүй.

Классын доторх гишүүн функцийн дотор `break`, `switch-case`, `do`, `for`, `continue`, `goto`, `while`, `do-while` зэрэг команд хэрэглэхийг Turbo C++ компайлер зөвшөөрдөгтүй. Тэдгээрийг гишүүн функцийн дотор заавал хэрэглэх бол ийм функцийн классын гадна тодорхойлно.

### 3.13 Функцийн ялгаа

Объект нь мэдээлэл хадгалж зүйт бүрийн мэдээллийг команд хэлбэрээр бусдад илгээж бусдаас хүлээж авна. Объект ба функцийн хоорондын үндэсн ялгаа нь функцийн зөвхөн ганц үйлдэл хийх бол объект олон үйлдэл хийх боломжтой байдагт оршино.

Объектын хийх үйлдэл бүр оөр оөр функцизэр идэвхжинэ.

### 3.14 private, public гишүүн функцийн ялгаа

Өмнө үзсэн бүх програмын гишүүн функциүүд `public` тул тэдгээр рүү програмын аль ч функцизэс хандаж байсан. Зарим уял `private` гишүүн функцийн хэрэгтэй гардаг. Ийм функцийн язж тодорхойдох, түүнийг хэрхэн дуудахыг Program #3.1-д оөрчлөлт оруулах замаар авч үзье.

```
//Program #3.5
//Finding the gross pay of an employee using class
#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
    void getdata()
    {
        cout << "\nEnter the Employee Name: " ;
        gets(name) ;
        cout << "Enter the Basic Pay: " ;
        cin >> basicpay ;
        cout << "Enter the Allowance: " ;
        cin >> allowance ;
    }
public:
    void showdata()
    {
        int grosspay = basicpay + allowance ;
        cout << setw(20) << name ;
        cout << setw(8) << basicpay ;
        cout << setw(12) << allowance ;
        cout.width(8);
        cout << grosspay ;
    }
}
```

```

    void readdata() ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Gross" ;
    cout << endl ;
}

void employee :: readdata()
{
    getdata() ;
}

void main()
{
    clrscr() ;
    employee emp ;
    //emp.getdata() ;
    emp.readdata() ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

```

Энэ програмын үр дүн Program #3.1-ийнхтэй адил гарна. Уг програмын `getdata()` нь `private` функц тул түүнд тухайн классын объектоос доор үзүүлсэнэр

`emp.getdata()`

ижэх хандаж болохгүй. Харин уг функцийг классын нь бусад `public` функцээс дуудаж болно. Иймээс Program #3.5-д `public readdata()` функц тодорхойлж `main()` функц дотроос дуудах замаар `getdata()` функц рүү хандсан.

Гишүүн функцийг классын нь гадна талд тодорхойлох үед үйлчлэх хүрээний операторын туслаамжтайгаар класс, функцийг хооронд нь холбож өгөх шаардлага яагаад гардаг талаар авч үзье. Програм бол олон классын нэгдэл юм. Тэдгээр классын зарим гишүүн функц ижил нэргэй байж болно. Гэхдээ ийм функцууд нэр ижилхэн хэдий ч өөр өөр үйлдэл хийж болно. Энэ тохиолдолд классын гадна талд тодорхойлсон гишүүн функцийг харгалзах класстай нь зөв уяж өгөхийн тулд доор үзүүлсэн загварын дагуу бичиж өгно.

`data-type class-name :: function-name()`

Энэ дүрмийн дагуу Program #3.1-д `employee` классын `getdata()` болон `showdata()` функцуудийг

```

void employee::getdata()
{
    //-----
}

void employee::showdata()
{
    //-----
}

```

гэж тодорхойлсон байна.

### 3.15 Гишүүн функцийг гишүүн функцийн дуудах

Классын public функцийг классын нь объектоос дуудаж болдог бол private функцийг классын нь объектоос биш, харин public функцийн нь дуудаж болно. Классын гишүүн функцийг классын нь еөр гишүүн функцийн дуудахыг гишүүн функцийн багтаалал, багасан гишүүн функцийн гээн. Үүнийг хэрхэн програмчилж болохыг Program #3.6-д үзүүллээ.

```
//Program #3.6
//Finding the gross pay of an employee using class

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    void getdata() ;
    void showdata() ;
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: " ;
    gets(name);
    cout << "Enter the Basic Pay: " ;
    cin >> basicpay ;
    cout << "Enter the Allowance: " ;
    cin >> allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8) << "Gross" ;
    cout << endl ;
}

void employee::showdata()
{
    getdata() ;
    heading() ;
    int grosspay = basicpay + allowance ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
    cout.width(8) ;
    cout << grosspay ;
}
```

```

void main()
{
    clrscr() ;
    employee emp ;
    emp.showdata() ;
    getch() ;
}

 Enter the Employee Name: George
Enter the Basic Pay: 9000
Enter the Allowance: 2000

Employee Name    Basic      Allowance     Gross
        George       9000          2000      11000

```

Энэ програмд `getdata()`, `showdata()` гэсэн функциүүдийг `employee` класс дотор зарлаж гадна талд нь тодорхойлсон байна. Программын `main()` функц дотор `employee` классын `emp` объектыг `employee emp;` гэж байгуулаал эл объектоос `showdata()` функцийг дуудсан. Харин `showdata()` функц нь дотроо `getdata()` функцийг дуудах тул `getdata()` нь `showdata()` функц дотор багтсан функц болно.

### 3.16 Класс доторх хүснэгт

Program #3.1-д зөвхөн нэг ажилчны цалингийн мэдээллийг хадгалдаг байхаар `employee` классыг тодорхойлсон. Ийм `employee` классыг 3 ажилчны цалингийн мэдээллийг хадгалах, боловсруулах боломжтой байхаар өөрчилж болно. Ингэхдээ `employee` класс дотор хүснэгт хэрэглэж түүнийг яж програмчилж болохыг Program #3.7-д харууллаа.

```

//Program #3.7
//Finding the gross pay of three employees

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[3][20] ;
    int basicpay[3] ;
    int allowance [3];
public:
    void getdata() ;
    void showdata() ;
};

void employee::getdata()
{
    for (int k=0; k<3; k++)
    {
        cout << "\nEnter the Employee Name " << k+1 << ": " ;
        gets(name[k]);
        cout << "Enter the Basic Pay: " ;
        cin  >> basicpay [k] ;

        cout << "Enter the Allowance: " ;
    }
}

```

```

        cin >> allowance [k];
    }
}

void heading()
{
    cout << endl ;
    for (int k = 0 ; k < 49; k++)
        cout << "-" ;
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8) << "Gross" ;
    cout << endl ;

    for (k = 0 ; k < 49; k++)
        cout << "-" ;
    cout << endl ;
}

void employee::showdata()
{
    int grosspay [3] ;
    for (int k=0; k < 3; k++)
    {
        grosspay[k] = basicpay[k] + allowance [k];
        cout << setw(20) << name [k] ;
        cout << setw(8) << basicpay [k];
        cout << setw(12) << allowance [k];
        cout.width(8);
        cout<< grosspay [k];
        cout << endl ;
    }
}

void main()
{
    clrscr() ;
    employee emp ;
    emp.getdata() ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

```

 Enter the Employee Name 1: George.  
Enter the Basic Pay: 9000.  
Enter the Allowance: 2000.  
  
Enter the Employee Name 2: Bill.  
Enter the Basic Pay: 9000.  
Enter the Allowance: 1500.  
  
Enter the Employee Name 3: Validimar.  
Enter the Basic Pay: 9000.  
Enter the Allowance: 1000.

Employee Name	Basic	Allowance	Gross
George	9000	2000	11000
Bill	9000	1500	10500
Validimar	9000	1000	10000

Энэ програмыг Program #3.1-тай харьцуулж үзье. Program #3.1-д employee классын егөгдлүүдийг энгийн хувьсагч хэлбэрээр, Program #3.7-д турван бүтвэртэй хүснэгт хэлбэрээр тус тус тодорхойлсон. Класс дотор тодорхойлсон хүснэгтэд турван ажилчны мэдээллийг хадгалах боломжтой. Ийм классын хувьд дотоод гишүүн функцийн командыг хэрэглэх боломжгүй тул гишүүн функцийн классын тадаа талд тодорхойлжээ.

### 3.17 Объектон хүснэгт

Объектон хүснэгтийг C++ хэлний дотоод төрлийн хүснэгтийг байгуулдаг жишгээр үүсгэх тул өмнө үзсэн employee классын 5 объектон хүснэгтийг доор үзүүлснээр байгуулж болно.

```
employee e[5] ;
```

Энэ хүснэгтийн бүтвэр рүү хандаадаа түүний дугаарыг нь хэрэглэн e[0], e[1], e[2], e[3], e[4] гэж хандана. Объектон хүснэгтийн бүтвэр нь классынхаа гишүүн функцийн зараах байдлаар хэрэглээнэ.

```
e[0].getdata() ;
e[0].showdata() ;

e[4].getdata() ;
e[4].showdata() ;
```

Объектон хүснэгтийн тухайлсан бүтвэрт утга оноохдоо байгуулагч функцийг хэрэглэн. Энэ талаар "Байгуулагч, устгач функцийн" бүлэгт дэлгэрэнгүй авч үзнэ.

Өмнөх Program #3.7-гийн үр дүнтэй ижил үр дүнг өгөх програмыг объектон хүснэгт хэрэглэн бичиж болохыг Program #3.8-д үзүүллээ.

```
//Program #3.8
//Finding the gross pay of three employees

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

const int max = 3 ;
class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    void getdata() ;
    void showdata() ;
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: " ;
}
```

```

gets(name);

cout << "Enter the Basic Pay: ";
cin >> basicpay;

cout << "Enter the Allowance: ";
cin >> allowance;
}

void heading()
{
    cout << endl;
    for (int k = 0 ; k < 49; k++)
        cout << "-";
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
    cout << endl;

    for (k = 0 ; k < 49; k++)
        cout << "-";
    cout << endl;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance;
    cout << setw(20) << name;
    cout << setw(8) << basicpay;
    cout << setw(12) << allowance;
    cout.width(8);
    cout << grosspay;
    cout << endl;
}

void main()
{
    clrscr();
    employee emp [max];
    for (int k = 0; k < max ; k++)
        emp[k].getdata();

    heading();

    for (k = 0; k < max ; k++)
        emp[k].showdata();
    getch();
}

```

Энэ програмын үр дүн нь өмнөх Program #3.7-гийнхтай ижил байна. Классын наамж иргэдлээс бусад огогдол нь хүснэгт биш. Харин main() функцийн дотор employee классын объектон хүснэгтийг employee emp[max]; командаар байгуулна. Энд тах тогтолцоогоо утга З байхаар програмын эхэнд const int max=3; гэж зааж өгөх тул employee классын emp объектон хүснэгт нь emp[0], emp[1] ба emp[2] гэсэн бүтвэрүүдтэй байхаар байгуулагдана. Харин

```

for (k=0; k < max ; k++)
    emp[k].getdata();

```

дэвталтаар З ажилчны мэдээллийг гараас авч харгалзах emp[0], emp[1] ба emp[2] бүтвэрт хадгална. Дараагаар нь

```
for (k = 0; k < max ; k++)
    emp[k].showdata();
```

дэвталтаар ажилчин тус бүрийн нийт цалинг бодоод хүснэгтлэн жагсааж дэлгэцлэнэ.

### 3.18 Объект, функци

Функцийн параметр нь бутиэн хувьсагч байж болдгийн адилгаар объект мөн функцийн параметр байж болохыг хоёр ажилчны цалингийн өгөгдлийг авч нийлбэрийг олох программын жишээн дээр Program #3.9-д үзүүллээ.

```
//Program #3.9
//Finding the gross pay of three employees

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
    int grosspay;
public:
    void getdata();
    void showdata();
    void addpay(employee, employee);
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);
    cout << "Enter the Basic Pay: ";
    cin >> basicpay;

    cout << "Enter the Allowance: ";
    cin >> allowance;
}

void heading()
{
    cout << endl;
    for (int k = 0; k < 49; k++)
        cout << "-";
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
    cout << endl;
    for (k = 0; k < 49; k++)
        cout << "-";
}
```

```

        cout << endl ;
    }

void employee::showdata()
{
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
    cout.width(8);
    cout << grosspay ;
    cout << endl ;
}

void employee::addpay(employee e1, employee e2)
{
    basicpay = e1.basicpay + e2.basicpay ;
    allowance = e1.allowance + e2.allowance ;
    grosspay = e1.grosspay + e2.grosspay ;
    for (int k = 0 ; k < 49; k++)
        cout << "-" ;
    cout << endl ;
    cout << setw(20) << "Total:" ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
    cout << setw(8)  << grosspay ;
    cout << endl ;
    for (k = 0 ; k < 49; k++)
        cout << "-" ;
    cout << endl ;
}

void main()
{
    clrscr() ;
    employee e1, e2, e3 ;
    e1.getdata() ;
    e2.getdata() ;
    heading() ;
    e1.showdata() ;
    e2.showdata() ;
    e3.addpay(e1,e2) ;
    getch() ;
}

```

 Enter the Employee Name: George.  
 Enter the Basic Pay: 9000.  
 Enter the Allowance: 2000.  
  
 Enter the Employee Name: Bill.  
 Enter the Basic Pay: 9000.  
 Enter the Allowance: 2000.

Employee Name	Basic	Allowance	Gross
George	9000	2000	11000
Bill	9000	2000	11000
Total:	18000	4000	22000

Дээрх employee класс нь `getdata()`, `showdata()`, `addpay()` гэсэн гишүүн функцийтэй. Энэ класст нэмж тодорхойлсон `grosspay` өгөгдлийг `showdata()`, `addpay()` функциүүд хөрөглэн. `addpay()` нь классынхаа хоёр объектыг параметр хэлбэрээр авч бодолт хийх үүрэгтэй функций юм. Гишүүн функций ихэвчлэн объектоос дуудагдах учир ийм функцийг объектын гишүүн гэж болно. employee классын `e1`, `e2` ба `e3` объектуудын аль нь ч классынхаа 3 гишүүн функцийг дуудаж болно. Объектоос дуудагдах функци зөвхөн тухайн объектынхоо өгөгдолд, утгад хандана. Тухайлбал,

```
e1.getdata() ;  
команд нь getdata() функци зөвхөн e1 объектын өгөгдөл хандахыг заана. Гэтэл  
e3.addpay(e1,e2) ;
```

командын хувьд `addpay()` функци нь `e1`, `e2` ба `e3` объектод хандах эрхтэй. Уг функци нь `e1`, `e2` гэсэн хоёр объектын уттыг параметрээр авах тул тэдгээр объектын өгөгдөл хандаж чадна. Энэ функци мөн `e3` объектын өгөгдөл хандах эрхтэй, учир нь уг функци `e3` объектын гишүүн, энэхүү объектоос `addpay()` функци дуудагдсан байна. Ямар нэгэн утга `void` төрлийн `addpay()` функцийн буцахгүй. Харин функцийн бодолтын үр дүн болох нийт `basicpay`, `allowance` ба `grosspay` нь `e3` объектын утга болж хадгалагдана. Энэ нь `e3` объектоос `addpay()` функцийг дуудсантай холбоотой юм. Тэгэхлээр,

```
basicpay = e1.baiscpay + e2.basicpay ;  
allowance = e1.allowance + e2.allowance ;  
grosspay = e1.grosspay + e2.grosspay ;
```

бүлэг команд нь `e1`, `e2` объектуудын уттын нийлбэрийг олж `e3` объект руу хадгална. Дараа нь `e3` объектын мэдээллийг

```
cout << setw(20) << "Total:" ;  
cout << setw(8) << basicpay ;  
cout << setw(12) << allowance ;  
cout << setw(8) << grosspay ;  
cout << endl ;
```

бүлэг командаар хүснэгт хэлбэрээр дэлгэцлийз. Өмнөх `addpay()` функци рүү `e1`, `e2` хоёр объектыг утгаар нь дамжуулж байгааг `addpay(employee e1, employee e2);` командаас харж болно. Иймд `e1`, `e2` хоёр объектын өгөгдлийг `addpay()` функци өөрчилж чадахгүй.

#### Мэдлэгээ шалгах асуулт

- 3.1. ОО програмын огогдол, функцийг хэрхэн зохион байгуулдаг
- 3.2. Бүтэц, класс хоорондын ялгаа
- 3.3. ОО програмын давуу тал
- 3.4. Дараах ойлголт хоорондын ялгаа
  - a) Объект ба класс
  - b) Өгогдол хийсвэрлэл ба битүүмжлэл
- 3.5. Өгөгдлийн битүүмжлэл, өгогдол далдлал гэж юу болох, эдгээр ойлголт C++ хэлнээ хэрхэн буудаг
- 3.6. Объект гэж юу болохыг жишээн дээр тайлбарлах
- 3.7. Объектууд гэж юу болохыг жишээн дээр тайлбарлах

- 3.8. Класс гэж юу болох, энэ нь өгөгдлийн далдлалыг яаж хангадаг
- 3.9. Классыг C++ хэлний хувьд гэж юу ойлгож болох
- 3.10. C++ бүтэц, класс хоорондын ялгаа
- 3.11. Классын гишүүн функцийг яаж тодорхойлох
- 3.12. `private`, `public` гишүүн функцийн ялгаа
- 3.13. Оороор зааж өгөгүй тохиолдолд классын гишүүдийн шууд авах шинж юу болохыг жишсэн дээр тайлбарлах
- 3.14. Объект, классын ялгаа
- 3.15. Классын гишүүн өгөгдлөд хандах арга
- 3.16. Классын гишүүн функции рүү хандах арга
- 3.17. Гишүүн функци, срийн функци хоорондын ялгаа
- 3.18. Ямар тохиолдолд дотоод функцийг хэрэглэх, Яагаад?
- 3.19. `const` функцийн хэрэгзэж юу болох
- 3.20. C++ хэлний хувьд ( :: ) үйлчилх хүрээний операторын хэрэглээг жишээгээр тайлбарлах
- 3.21. Дараах тохиолдолд классын гишүүд рүү хандах механизм ямар байх
- а) Их програм дотроос
  - б) Классын ик гишүүн функци дотроос
  - в) Оөр классын гишүүн функци дотроос
- 3.22. Классын гишүүн функци, програмын бусад функци ижил нэртэй байж болох эсэх, Ижил нэртэй байж болох бол тэдгээрийг яаж ялгах; Ижил нэртэй байж болохгүй бол шалтгаан нь юу болох
- 3.23. Классын тодорхойлолтын гадна талаас гишүүн огогдол рүү, гишүүн функци рүү хандах механизм ямар байх
- 3.24. Багтсан гишүүн функци гэж юу болохыг жишсэн дээр тайлбарлах
- 3.25. Класс дотор хүснэгт хэрхэн тодорхойлж болохыг жишээгээр үзүүлэх
- 3.26. Объектон хүснэгт хэрхэн тодорхойлж болохыг жишээгээр үзүүлэх
- 3.27. Объектыг функци рүү хэрхэн дамжуулж болохыг тохирох жишсэн дээр тайлбарлах

#### Класс тодорхойлох болдого

- 3.28. Бүхэл тоон `x`, у гишүүн өгөгдлөтэй, өгогдсон хоёр цэгийн хоорондох зайлг олох `distance()` функций `point` класс тодорхойлох
- 3.29. Бодит тоон `real`, `imaginary` гишүүн өгөгдлөтэй, өгогдсон комплекс тооны абсолют утга олох гишүүн функций `complex` класс тодорхойлох
- 3.30. Бүхэл тоон `hours`, `minutes` ба `seconds` гишүүн өгөгдлөтэй, өгөгдсан хоёр цагийн хоорондох нийт цагийг олох гишүүн функций `time` класс тодорхойлох
- 3.31. Бүхэл тоон `day`, `month` ба `year` өгөгдлөтэй, өгөгдсон хоёр огноо хоорондох нийт одрийг олох функций `date` класс тодорхойлох
- 3.32. Банкны харилцагчийн нэр, дугаар ба дансны үлдэгдэл зэрэг өгөгдлөтэй, данс нэх, дансаанд мөнгө хийх, дансаас мөнгө авах, данс хаах ба дансны үлдэгдэл тооцож харах зэрэг үйлдэл хийх гишүүн функций `bankaccount` класс тодорхойлох

- 3.33. Номын бүртгэлийн дугаар, номын нэр, зохиогч, хэвлэлийн газар ба номын үнэ зэрэг өгөгдлөй, ном олгох ба ном буцааж авах гишүүн функцтэй `library` класс тодорхойлох
- 3.34. `Inventory` классыг TV, VCR, PC зэрэг бараатай, бараа хүлээн авах, зарагдсан барааг бүртгэлээс хасах, барааны үлдэгдэл тооцохтой холбоотой З гишүүн функцтэй байхаар тодорхойлох

Програмчлах болдого

- 3.35. Өгөгдсон хоёр цэгийн хоорондох зайл олох
- 3.36. Өгөгдсон комплекс тооны абсолют уттыг олох
- 3.37. Өгөгдсон хоёр огнооны хоорондох одрийг олох
- 3.38. Хариуцах данс нээх, дансанд мөнгө хийх, дансаас мөнгө авах, данс хаах зэрэг командыг цэсээс сонгож боломжтой байх
- 3.39. Номын сангаас ном авах, буцааж вгөх зэргийг цэсээс сонгодог байх
- 3.40. Бараа хүлээн авч зарж барааны үлдэгдэл гаргах үйлдлийг цэсээс сонгож хийх
- 3.41. Оюутны нэр, үнэмлэхийн дугаар, таван өөр хичээлийн дунг хадгалах өгөгдлөй, огөгдлийг хүснэгтлэн гаргахад хэрэглэх функцууд бүхий `student` класс тодорхойлж хэрэглэх
- 3.42. Өвчтөний нэр (`name`), нас (`age`), хүйс (`sex`), үзлэгийн огноо (`date`) ба онош (`diagnosis`) зэрэг өгөгдлөй, өвчтөн хүлээн авах, өвчтөнг бүртгэлээс хасах гишүүн функцтэй `patient` класс хэрэглэн өвчтөний бүртгэлийг хүснэгтлэн хадгалах

# ДӨРӨВДҮГЭЭР БҮЛЭГ

## БАЙГУУЛАГЧ, УСТГАГЧ ФУНКЦ

- Байгуулагч функцийн загвар, тодорхойлолт, 121
- Анхдагч байгуулагч, 125
- Байгуулагч функцийн шинж, 127
- Параметртэй байгуулагч, 128
- Гарааны утгыг нь аргумент хэлбэрээр объект руу дамжуулах, 128
- Хуулагч байгуулагч, 130
- Устгагч функци, 132
- Устгагч функцийн тодорхойлолт, 137
- Устгагч функцийн шинж, 138
- Устгагч функцийн хэрэглээ, 138
- Объектон хүснэгт ба байгуулагч функци, 140



## ДӨРӨВДҮГЭЭР БҮЛЭГ

### БАЙГУУЛАГЧ, УСТГАГЧ ФУНКЦ

Класс тодорхойлж объект байгуулах талаар өмнөх бүлэгт үзсэн билээ. Гэхдээ объект байгуулах үед чухам юу хийгддэг талаар бид судалж үзээгүй байгаа. Объект байгуулахын тулд C++ компайлер тодорхой ажлууд хийдэг, тухайлбал гишүүн өгөгдэл ой бэлдэж түүнд гарааны утга оноодог байна.

Нэр нь классын нэртэй ижил байх гишүүн функцийг класс дотор нэмж тодорхойлон объектод ой бэлдэх, гарааны утга оноох ажлыг түүнд хариуцуулж болдог. Ийм функцийг *constructor* буюу байгуулагч функц, байгуулах функц, товчоор байгуулагч гэвээ.

Объектод ой бэлдэж түүний гишүүн өгөгдэл гарааны утга оноож өгөхийн тулд систем байгуулагчийг дууддаг. Эл функц ямар ч утга буцаадаггүй. Гэхдээ түүнийг *void* гэж зарлах шаардлага байдаггүй, учир нь уг функц анхнаасаа *void* төрлийнх. Байгуулагчийг дахин тодорхойлж болдог. Ингэснээр объектыг олон янзаар байгуулах боломжтой болно. Гэхдээ ийм байгуулагчуудын параметрийн төрөл, тоо хэмжээ, дараалал өөр өөр байх ёстой.

#### 4.1 Байгуулагч функцийн загвар, тодорхойлолт

Байгуулагч функц нь класстайгаа ижил нэртэй байна. Гишүүн өгөгдэл гарааны утга оноох зорилгоор энэ функцийг ихэвчлэн тодорхойлно. Өмнөх бүлэгт тодорхойлсон *employee* классын байгуулагчийг дараах байдлаар нэмж тодорхойлж болно.

```
employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}
```

Эхний *employee()* мөр нь *employee()* нь байгуулагч функц болохыг заана. Харин нээх ба хаах их хаалт дотор бичигдэх

```
strcpy(name, "") ;
basicpay = 0 ;
allowance = 0 ;
```

командууд бол байгуулагч функцийн хийх нэмэлт үйлдлүүд юм. Байгуулагч функц чухам юу хийдэг, объект байгуулахад ямар үүрэгтэй оролцдогийг дараах жишээ програмаар үзүүлж болно.

```
//Program #4.1
//Displaying assigned values in the constructor

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
```

```

public:
    employee()
    {
        strcpy(name, " ");
        basicpay = 0 ;
        allowance = 0 ;
    }

    void showdata() ;
};

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee emp ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

```

	Employee Name	Basic	Allowance
		0	0

Дээрх програмын байгуулагч функцийг мөн Program #3.1-д хэрэглэж болохыг дараах програмаар үзүүллээ.

```

//Program #4.2
//Finding the gross pay of an employee using class
//(constructor included)

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:

```

```

employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}
void getdata(void) ;
void showdata(void) ;
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: " ;
    gets(name);
    cout << "Enter the Basic Pay: " ;
    cin  >> basicpay ;
    cout << "Enter the Allowance: " ;
    cin  >> allowance ;
}
void employee::showdata()
{
    int grosspay = basicpay + allowance ;
    cout << setw(20) << name ;
    cout << setw(8)   << basicpay ;
    cout << setw(12) << allowance ;
    cout.width(8);
    cout << grosspay ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)   << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)   << "Gross" ;
}

void main()
{
    clrscr() ;
    employee emp ;
    heading() ;
    emp.showdata() ;
    emp.getdata() ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

Employee Name      Basic      Allowance      Gross
          0           0            0
 Enter the Employee Name : George
Enter the Basic Pay: 9000
Enter the Allowance: 2000
Employee Name      Basic      Allowance      Gross
          George       9000        2000       11000

```

Program #4.2-ын main() функци найман командтайгаас эхний clrscr() нь дэлзэ арчих, сүүлийн getch() нь хэрэглэгчийн ажлын цонхыг ямар нэгэн товч төвшигдох хүртэл харуулж байх үүрэгтэй функцийн төрөл юм. Харин хоёрдугаар мөрөнд буй employee emp; нь emp объект байгуулах үүрэгтэй. Системийн объект байгуулахаар харгалзах байгуулагч функцийт нь автоматаар дуудна. Үнэхээр байгуулагч функцийг дуудагдаж ажилласан эсэхийг emp.showdata(); командын үр дүнгээс харж болно. Бусад команд нь Program #3.1-д үзүүлсэн шигээр ажиллана. Хоёр объект байгуулахаар Program #4.1-ийг оорчилж бичье.

```
//Program #4.3
//Displaying assigned values in two objects

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee()
    {
        strcpy(name, "") ;
        basicpay = 0 ;
        allowance = 0 ;
    }
    void showdata() ;
};

void employee::showdata()
{
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
    cout << endl ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" << endl ;
}

main()
{
    clrscr() ;
    employee emp1, emp2 ;
    heading() ;
    emp1.showdata() ;
    emp2.showdata() ;
    getch() ;
}
```

Employee Name	Basic	Allowance
	0	0
	0	0

Програмын `main()` функц доторх `employee emp1, emp2;` командаар `emp1, emp2` гэсэн хоёр объект байгуулагдана. Ингэхдээ байгуулагч функц нь байгуулагдах объектын тоогоор, тухайлбал 2 удаа дуудагсаныг програмын үр дүнгээс харж болно. Энд

0 0

мерийг хоёр удаа хэвлэсэн байна.

#### 4.2 Ахдагч байгуулагч

Өмнөх “Класс, объект” бүлэгт класс хэрэглэсэн жишээ програмд ямар нэгэн байгуулагч функц тодорхойлж өгөгүй байхад хэвийн ажиллаж байсныг бид мэднэ. Яагаад? Хэрэв програм хөгжүүлэгч ямар нэгэн байгуулагч тодорхойлж өгөгүй байвал C++ компайлер өөрөө ийм функцийг програмд нэмж тодорхойлдог. Ийм байгуулагчийг `default constructor` буюу ахдагч байгуулагч гэнэ. Програм хөгжүүлэгч нь байгуулагч функцийг класс тодорхойлж өгвэл C++ компайлер нь ахдагч байгуулагчийг үүсгэдэгтүй. Гэхдээ ахдагч байгуулагч функцийг бас илзэр тодорхойлж болно. Ийм байгуулагч нь цөмөөрөө гарааны утга бүхий параметртэй эсвэл параметргүй байгуулагч функц байна. Тэгэхлээр, Program #4.1+#4.3-т тодорхойлсон бүх байгуулагч функц параметргүй учраас ахдагч байгуулагч функцууд юм. Эдгээр байгуулагч функц нь `name`, `basicpay` ба `allowance` өгөгдлүүдийн талбарыг доорх шигээр хоосолно.

```
strcpy(name, "") ;
basicpay = 0 ;
allowance = 0 ;
```

Гэхдээ байгуулагч функц бүр ийм цэвэрлэх үйлийг заавал хийх албагүй. Байгуулагч нь бас тодорхой гарааны уттыг гишүүн өгөгдлүүдэд оноож болно. Үүнийг Program #4.1-д өөрчлөлт хийх замаар гаргаж авах Program #4.4-т үзүүлээ.

```
//Program #4.4
//Displaying assigned values in the constructor

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee()
    {
        strcpy(name, "George") ;
        basicpay = 9000 ;
        allowance = 2000 ;
    }
    void showdata() ;
} ;
```

```

void employee::showdata()
{
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << endl ;
}

main()
{
    clrscr() ;
    employee emp ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

```

 Employee Name Basic Allowance  
George 9000 2000

Өмнөх бүх програмда тодорхойлсон байгуулагчүүд нь цөм параметргүй функциүүд юм. Иймээс параметртэй анхлагч байгуулагч хэрхэн тодорхойлж болохыг дараах програмаар үзүүлс.

```

//Program #4.5
//Creating default constructor with arguments

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[]="George", int b=9000, int a=2000)
    {
        strcpy(name, n) ;
        basicpay = b ;
        allowance = a ;
    }
    void showdata() ;
};

void employee::showdata()
{
    cout << setw(20) << name ;

```

```

cout << setw(8) << basicpay ;
cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << endl ;
}

main()
{
    clrscr() ;
    employee emp ;
    heading() ;
    emp.showdata() ;
    getch() ;
}

```

	Employee Name	Basic	Allowance
	George	9000	2000

Анхдагч байгуулагчийг зарлаж байхдаа түүний бүх параметрт гарааны утга оноож байгааг дээрх програмын

```
employee(char n[]="George", int b=9000, int a=2000);
```

командын мөрөөс харж болно. Ийм объектын гарааны утгыг showdata() функцийн дэлгэцэлж болдог.

#### 4.3 Байгуулагч функцийн шинж

Байгуулагч бол тусгай зориулгатай онцгой функция. Түүний онцлог шинжийг дараах байдлаар жагсааж болно. Тухайлбал,

- Байгуулагч бол **public** функция.
- Байгуулагч нь дотоод мөр функцийн байж болно.
- Байгуулагч функцийн нь **const**, **static**, **volatile** болон **virtual** байж болохгүй.
- Байгуулагч функцийн нэр нь классынхаа нэргээй ижил байна.
- C++ компайлдер нь объект байгуулахдаа байгуулагчийг автоматаар дуудна.
- Байгуулагч нь утга буцаахгүй.
- Байгуулагч функция гарааны утгатай параметрүүдтэй байж болно.
- Объект хэрхэн байгуулагдахыг байгуулагч функцийн тодорхойлно.
- Байгуулагч функция оороо төрөл хувиргалт хийх аргатай.
- Байгуулагч функцийн аргумент нь объект байвал түүнийг хувилна.
- Хаягаар нь байгуулагч функция рүү хандаж болдогтүй.
- Байгуулагч функция нь **new** ба **delete** операторыг ил бишээр дуудаж хэрэглэнэ.
- Байгуулагч функция нь бусад гишүүн функцийн адил удамшидагтүй.

#### 4.4 Параметртэй байгуулагч

Тодорхойлолтдоо ямар илгэн байгуулагч функцийүй классын хувьд систем тухайн классын объектыг байгуулахдаа анхдагч байгуулагч функцийг далдуур програмд нэмж тодорхойлоод дуудаж ажиллуулдаг. Эл байгуулагч нь байгуулах объектод ой бэлдэнэ. Анхдагч байгуулагчийг илзэр тодорхойлж болох ба гэхдээ ийм байгуулагчийн хийх ажлын хүрээ нийлээд хязгаарлагдмал байдаг. Тухайлбал,

- Гишүүн огогдэд тогтмол ирг утга онооно. Гэтэл программын хувьд байгуулах объект бүр ашиг утгатай байх шаардлага тэр бүр байдалгүй.
- Анхдагч байгуулагч нь объектын гарцааны уттыг гаднаас авч чадаигүй.

Анхдагч байгуулагчийн дээрх сүл талыг параметртэй байгуулагч хэрэглэх замаар арилгаж болдог. Параметртэй байгуулагчийн динамикаар аргумент авдаг байхаар тодорхойлио. Ийм байгуулагч нь доор жагсаасан давуу чанартай.

- Байгуулагч функцийн тодорхойлолт дотор гишүүн егөгдэлд утга онохыг шаарддагтүй.
- Объект байгуулахдаа дамжуулан огох уттыг гаднаас авна.
- Объектоор дамжуулж авсан огогдлийг автоматаар гишүүн огогдэлд хадгална.
- Бусдаас өөр утгатай объект байгуулахыг зохионрон.
- Гишүүн егөгдэлд гарцааны утга оноож болно.
- Анхдагч байгуулагчийн иргэн адилдар байгуулагдах объектод ой бэлдэнэ.
- Параметртэй байгуулагчийг илзэр юм уу ил бишээр дуудаж болно.

#### 4.5 Гарааны уттыг нь аргумент хэлбэрээр объект руу дамжуулах

Объектын гарцааны уттыг аргумент хэлбэрээр параметртэй байгуулагч руу дамжуулж болдог тухай омниө үзсэн билээ. Ийм гарцааны уттыг байгуулагдах объектоос дамжуулж өгнө. Program #4, 5-ыг параметртэй байгуулагчтай болгон дараах байдлаар бөрчилж бичье.

```
//Program #4,6
//Parameterized constructor with implicit call and explicit call

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a)
    {
        strcpy(name, n) ;
        basicpay = b ;
        allowance = a ;
    }
    void showdata() ;
};

void employee::showdata()
{
    cout << endl ;
}
```

```

cout << setw(20) << name ;
cout << setw(8)  << basicpay ;
cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
//implicit call
    employee empl("George", 9000, 2000) ;
    heading() ;
    empl.showdata() ;
//explicit call
    employee emp2 = employee("Natsag", 5000, 1000) ;
    emp2.showdata() ;
    getch() ;
}

    Employee Name      Basic      Allowance
        George       9000        2000
        Natsag       5000        1000

```

Дээрх програмд параметртэй байгуулагчийг түүний гишүүн өгөгдлүүдийнх шиг дараалалтай 3 параметр гаднаас авахаар

```

employee(char n[], int b, int a);

```

тэж тодорхойлсон байна. Гол функциид, тухайлбал main() функци дотор empl объектыг

```

employee empl("George", 9000, 2000);

```

командаар байгуулахдаа employee(char, int, int); байгуулагч функцийг далдуур дуудаж empl объект руу дамжуулж өгөх

"George", 9000 ба 2000

тэсэн турван утгыг уг объектын гарцааны утга болгож түүний гишүүн өгөгдлүүлэд харгалзуулан хадгална (Зураг 4.1).

name	George	..		
basicpay	9000			
allowance	2000			

Зураг 4.1: Параметртэй байгуулаччаар байгуулсан объект

Бас emp2 объектыг байгуулахдаа доор үзүүлсэн шигээр параметртэй байгуулагчийг илэр, шууд дуудаж хэрэглэсэн.

```

employee emp2 = employee("Natsag", 5000, 1000) ;

```

Энд employee("Natsag", 5000, 1000) командын аргументийн хэсгийн бичдэс нь утга оноох операторын баруун гар талд байгаа employee байгуулагчийн тодорхойлолтын толгойн хэсгийн бичдэстэй харгалзан тохирч байна. Тэгэхлээр,

```
"Natsag", 5000, 1000
```

гэсэн өгөгдлүүд нь emp2 объектын гарааны утга болж хадгалагдана. Ер нь, параметртэй байгуулагчийн тодорхойлолтын толгойн хэсгт зааж өгдөг гаднаас авах аргументийн тоо нь объектоор дамжуулан өгч байгаа аргументийн тоотой тохирч байх ёстой. Program #4.6-гийн хувьд ийм тоо нь 3. Мөн параметртэй байгуулагчийн тодорхойлолтын толгойн хэсг дэх гаднаас авах параметрийн дэс дараа нь объектоор дамжуулан өгч байгаа утгын дэс дараатай харгалзан тохирч байх шаардлагатай байдаг.

#### 4.6 Гарааны утгатай байгуулагч

Параметртэй байгуулагч олон сайн чанартай болохыг өмнөх будэгт үзсэн билээ. Гэвч энэ функци бас сүл талтай. Параметртэй байгуулагчаар объект байгуулах бүрдээ гаднаас бүх аргументээр нь байнга хангаж байх ёстой. Гэтэл ийм байгуулагчийн зарим аргументийг гаднаас огох, заримыг нь анхнаасаа зааж өгөх тогтмол утгаар нь хэрэглэхийг зарим програм шаардлаг. Энэ нь анхдагч байгуулагч ба параметртэй байгуулагч хоёрны аль алиинд нь байх давуу чанарыг сөртөө шингэсэн байгуулагч хэрэгтэй гэсэн үг юм. Ийм байгуулагч бол гарааны утгатай байх юм. Үүнийг дараах програмын жишээн дээр авч үзье.

```
//Program #4.7
//Constructor with default arguments

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    employee(char n[], int b, int a = 2000)
    {
        strcpy(name, n);
        basicpay = b;
        allowance = a;
    }

    void showdata();
};

void employee::showdata()
{
    cout << endl;
    cout << setw(20) << name;
    cout << setw(8) << basicpay;
    cout << setw(12) << allowance;
}

void heading()
{
    cout << endl;
```

```

cout << setw(20) << "Employee Name" ;
cout << setw(8)  << "Basic" ;
cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee emp1("George", 9000) ;
    heading() ;
    emp1.showdata() ;
    employee emp2("Natsag", 5000, 7000) ;
    emp2.showdata() ;
    getch() ;
}

```

 Employee Name      Basic      Allowance  
 George                9000        2000  
 Natsag                5000        7000

Дээрх програмд 3 параметр авахаар `employee` байгуулагчийг тодорхойлохдоо түүний а параметрийн шууд авах гарааны утгыг 2000 байхаар

```

employee(char n[20], int b, int a=2000)
тэж зааж өгсөн. Гол функци, тухайлбал main() функци дотор emp1 объектыг
employee emp1 ("George", 9000) ;
командаар байгуулахдаа
employee(char n[], int b, int a=2000)

```

байгуулагч функцийг ил биш далдуур дуудаж эл объектын утга болох "George", 9000 гэсэн хоёр өгөгдлийг 3 параметр авах ёстой байгуулагч функци рүү дамжуулах ба тэдгээр утга нь дарааллаараа уг байгуулагчийн авах эхний хоёр аргумент болно. Харин гарааны утгатай гуравдахь параметрийн хувьд объектоос харгалзах ямар нэгэн утга дамжуулж өгөөгүй тул гарааны утгаа (2000) шууд авна (Зураг 4.2).

name	George	..	
basicpay	9000		
allowance	2000		

Зураг 4.2: Гарааны утга бүхий параметртэй байгуулагчаар байгуулах объект

Харин emp2 объектыг байгуулахдаа

```

employee emp2 ("Natsag", 5000, 1000) ;
командаар түүнд 3 утга дамжуулах тул гуравдахь параметр нь гарааны утгаа шууд авахгүй.
Тэгэхлээр, энэ объектын хувьд

```

allowance = 1000

болно.

Параметртэй байгуулагчийн гаднаас авах хэдэн ч параметр гарааны утгатай байж болно. Гэхдээ тэдгээрийн дэс дараа их чухал байдаг. Дарааллын хувьд гарааны утгагүй нь гарааны

утгатай параметруүдийнхээ дараагаар байрлахыг C++ компайлер зөвшөөрдөгтүй. Иймд гарааны утгатай параметрууд нь параметрийн жагсаалтын төгсгөлд байхаар лараалуулдаг.

Өмнөх програмын хувьд

```
employee(char n[], int b, int a=2000)
ба
employee empl("George", 9000) ;
командуудыг C++ компайлер хүлээн зөвшөөрдөг. Гэтэл
employee(char n[], int b = 9000, int a)
ба
employee empl("George", 9000) ;
моруудийн хувьд C++ компайлер
Can't find a match
гэсэн алдааны мэдээлэл өгнө, учир нь байгуулагчийн тодорхойлолтод (гарааны утгатуй а нь
гарааны утгатай b-гийн дараа) байгаа доорх дэс дараа нь
employee(char n[], int b = 9000, int a)
гаднаас огч байгаа утгын дараалалтай
employee empl("George", 9000) ;
харгалзан таарахгүй байна. Тэгвэл
employee(char n[], int b = 9000, int a)
ба
employee empl("George", 8000, 2000) ;
гэсэн командууд байж болох уу? Ийм тохиолдлыг систем зөвшөөрно, учир нь байгуулагчийн
тодорхойлолтод зааж өгох, авах параметрийн тоо объектоос өгч байгаа аргументийн тоотой
таарч байна. Энэ тохиолдолд (9000) гарааны утгын оронд гаднаас авах (8000) утгыг
хадгалах тул гарах үр дүн нь
George     8000    2000
болов. Эндээс үзвэл байгуулагчийн тодорхойлолт оорчлөгдсөн хэдий ч гарах үр дүн нь адил
байна.
```

#### 4.7 Хуулагч байгуулагч

Бүх параметр нь гарааны утгатай байх байгуулагч функцийн бүхий employee класс хэрэглэсэн Program #4.8-ыг авч үзье. Уг классын бүх функцийн public шинжтэй. Ийм классын emp1 объект байгуулагдах үед байгуулагч нь уг объектод ой бэлдэж огогдуулдэл нь гарааны утга онооно. Дараа нь програмын явцад хийгдэх бодолтоор объектын утга оорчлогдено. Тэгхэлэр, шинээр байгуулах emp2 объект нь emp1 объектын тухайн үеийн утгыг, оореөр хэлбэл бодолтын явцад оорчлөгдсөний дараах утгыг авч болох уу? Хэрэв emp2 объектыг өмнө үзсэн аргыг хэрэглэн employee emp2; гэж байгуулах тохиолдолд байгуулагч функцизэс авах гарааны утга л emp2 объектын утга болохоос биш emp2 объект байгуулагдах үеийн emp1 объектын утга биш байдаг. Үүнийг дараах жишээ програмаар иягталж болно.

```

//Program #4.8
//Creating a constructor

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[]{"George", int b=9000, int a=2000)
    {
        strcpy(name, n) ;
        basicpay = b ;
        allowance = a ;
    }
    void showdata() ;
    void changebasic() ;
};

void employee::changebasic()
{
    basicpay = 10000 ;
}

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << endl ;
}

main()
{
    clrscr() ;
    employee empl ;
    heading() ;
    empl.showdata() ;
    empl.changebasic() ;
    empl.showdata() ;
    employee emp2 ;
    emp2.showdata() ;
    getch() ;
}

```



Employee Name	Basic	Allowance
George	9000	2000
George	10000	2000
George	9000	2000

Дээрх программын main() функци доторх

```
employee empl ;
empl.showdata();
```

Хоёр командаар empl объект байгуулж түүний утгыг дэлгэцлэхэд гарах үр дүн нь

```
George    9000    2000
```

Болно. Үүний дараагаар

```
empl.changebasic();
empl.showdata();
```

Командуудаар empl объектын basicpay өгөгдлийг өөрчлоол объектын утгыг дахин дэлгэцлэхэд доор үзүүлсэн шиг үр дүн гарна.

```
George    10000   2000
```

Дараа нь

```
employee emp2 ;
emp2.showdata();
```

Командаар emp2 объект байгуулаад утгыг нь хэвлэвэл үр дүн нь

```
George    9000    2000
```

байна. Программын үр дүнгээс үзүүхэд empl объектын basicpay=10000 байхад emp2 объектын basicpay=9000 байгаа ба empl, emp2 объектуудын утга ялгаатай байна. Иймд дээр дурдсан асуудлыг, тухайлбал шинээр байгуулагдах объект нь омниох объектын тухайн үеийн утгыг гарсаны утга болгож авахыг хуулагч байгуулагчаар шийдэж болно.

Хуулагч байгуулагч бол шинэ объект нь түүний омнех аль нэгэн объектын сүүлийн утгыг хуулбарлаж авах боломжийт олгох функци юм. Ийм маягаар байгуулагдах шинэ объект ба утга нь хуулбарлагдах объект хоорондоо зөвхөн нэрээрээ ялгаатай. Тэгэхлээр, хуулагч байгуулагч нь объект хуулбарлах функци юм. Ийм байгуулагч нэг параметр авах ба тэр нь класс нэгтэй хуулагдах объектын заалт байна. Ийм байгуулагчийн талаарх ойлголтыг дараах жишээ программаар загварчлан үзүүлж болно.

```
//Program #4.9
//copy constructor
#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
```

```

employee(char n[]="George", int b = 9000, int a=2000)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee(employee& emp)
{
    cout << "\nCopy constructor invoked: " << endl ;
    strcpy(name, emp.name) ;
    basicpay = emp.basicpay ;
    allowance = emp.allowance ;
}

void showdata() ;
void changebasic() ;
} ;

void employee::changebasic()
{
    basicpay = 10000 ;
}

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee empl ;
    heading() ;
    empl.showdata() ;
    empl.changebasic() ;
    empl.showdata() ;

    employee emp2 (empl);
    emp2.showdata() ;
    getch() ;
}

```



Employee Name	Basic	Allowance
George	9000	2000
George	10000	2000

Copy constructor invoked:

George	10000	2000
--------	-------	------

Ажилчны нэр бүхий сүүлийн хоёр мөр ижил байгааг програмын үр дунгээс харж болна. Иймд emp2 бол emp1 объектын хуулбар юм. Объект байгуулах

```
employee emp2 (emp1);
```

команд нь emp1 объектын заалтыг гаднаас авах хуулагч байгуулагчийг доорх шигээ дуудна.

```
employee(employee& emp)
```

Ингэснээр emp1 объектыг emp2 объект руу хуулна, оороор хэлбэл emp1 объектын утрын хувилж emp2 объектод хадгална.

Хуулагч байгуулагчийг далдуур дуудах

```
employee emp2 (emp1);
```

командыг

```
employee emp2 = emp1 ;
```

гэж өөр хэлбэрээр бичиж болно.

Хуулагч байгуулагч дуудагдах үед C++ компайлер класс дотор хуулагч байгуулагч тодорхойлогдсон эсхийг эхэлж шалгаад ийм байгуулагч байвал түүнийг дуудаж ажиллуулна. Класс хуулагч байгуулагчгүй бол компайлер нь анхдагч хуулагч байгуулагчийг дуудах ба энэ байгуулагч эх объектын мэдээллийг хэсэг хэсгээр нь шинэ объект руу бүрэн хуулна. C++ компайлер нь шинэ объект бүрийн хувьд хуулагч байгуулагчийг бэлдэж өгдөг. Тэгэхлээр хуулагч байгуулагч заавал байх албагүй юм. Program #4.9-оос хуулагч байгуулагчийг нь хасаад программыг ажиллуулж үзье.

```
//Program #4.10
//default copy constructor

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[]="George", int b = 9000, int a=2000)
    {
        strcpy(name, n) ;
        basicpay = b ;
        allowance = a ;
    }

    void showdata() ;
    void changebasic() ;
} ;

void employee::changebasic()
{
    basicpay = 10000 ;
}
```

```

void employee::showdata()
{
    cout << endl << setw(20) << name ;
    cout << setw(8)   << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)   << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee empl ;
    heading() ;
    empl.showdata() ;
    empl.changebasic() ;
    empl.showdata() ;

    employee emp2 (empl);
    emp2.showdata() ;
    getch() ;
}

```

 Employee Name      Basic      Allowance  
 George            9000        2000  
 George            10000      2000  
 George            10000      2000

Анхдагч байгуулагч дуудагдаж `empl` объектын утгыг `emp2` объект руу хуулсныг програмын үр дүн харуулж байна. Program #4.9-д хуулагч байгуулагчийг нэг параметр авахаар `employee& employee emp`) гэж тодорхойлсон. Уг параметр нь класс нэгтэй `empl` объектын заалт байгаа нь хуулагч байгуулагчийн параметр заавал заалт байхыг шаарддагтай холбоотой юм. Хуулагч байгуулагч нь аргумент болох объект руу заалтаар нь хандаж түүний хуулбар объектыг зөвхөн нэг удаа үүсгэнэ.

#### 4.8 Устгагч функци

Омнөх бүлгүүдэд байгуулагч гэж ямар функци болох, түүнийг програмд хэрхэн хэрэглэх талаар үзсэн билээ. Объект байгуулагдахад түүнд зориулан ой бэлдэхээр анхдагч эсвэл хэрэглэгчийн тодорхойлсон байгуулагч функци үргэлж дуудагдана. Тэгэхлээр, ямар нэгэн байгуулагч функци дуудагдахгүйгээр объект байгуулагдахгүй. Үүнтэй адиллаар, объект нь устгагч (*destructor*) функцигүйгээр устаж чадахгүй. Устгагч функци нь цаашдаа хэрэглэгдэхгүй объект устахад түүний эзэмшиж байсан ойг чөлөөлнө. Хэрэв класс дотор устгагч функцийг илээр тодорхойлж өгөөгүй тохиолдолд объект устахад анхдагч (*default*) устгагч функцийг C++ компайлэр нэмж тодорхойлж хэрэглэнэ.

#### 4.9 Устгагч функцийн тодорхойлолт

Устгагч функци нь классынхаа гишүүн функци болно. Функцийн нэр нь классынхаа нэртэй ижил байна. Зөвхөн өмнөө – тэмдэгтэй бичигдэгээрээ ялгаатай юм. Энэ дүрмийн дагуу

`employee` классын устгагч функцийг `-employee()` гэж зарлахдаа классын иерийн омнэ ~, иерийн дараа функцийн оператор (нэх ба хаах бага хаалт) тус тус бичнэ.

#### 4.10 Устгагч функцийн шинж

Устгагч функци олон чухал шинжтэйг доор жагсаалаа. Тухайлбал,

- Устгагч функци гол төлөв `public` шинжтэй.
- Устгагч функци нь дотоод мөр функци байж болно.
- Класс ганц устгагч функцийтэй байна.
- Устгагч функцийн иэр нь классынхаа иortтэй ижил байх ба түүний иерийн омно – тэмдэг заавал бичнэ.
- Устгагч функци нь `const`, `static` ба `volatile` төрлийнх байж болохгүй.
- Устгагч функци нь `virtual` байж болно.
- Объект устахад үед устгагч функцийг зөвхөн C++ компайлдер дуудна.
- Устгагч функци утта бушаахгүй.
- Устгагч функци параметр авахгүй.
- Объект устахад түүний эзэмшиж байсан ойг устгагч функци чөлөөлнө.
- Устгагч функци нь байгуулагч функцийн адилаар нэмэлт үйлэл хийж болно.
- Устгагч функци рүү хаягаар нь хандаж болдогтүй.
- Классын бусад функцийн адилаар удамшдагтүй.

#### 4.11 Устгагч функцийн хэрэглээ

Объектод зориулан байгуулагч функцийн бэлдэх ойг хэрэглэж дууссаны дараагаар чөлөөлөхөд устгагч функцийг хэрэглэнэ.

Класс олон байгуулагчтай байж болох ч зөвхөн ганц устгагч функцийтэй байна. Устгагч функцийг класс дотор зарлаж классын гадна талд тодорхойлж болно. Устгагч функци хэрхэн ажилладгийг дараах програмаар харууллаа.

```
//Program #4.11
//destructor

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

int k = 0 ;
class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a)
    {
        strcpy(name, n) ;
        basicpay = b ;
        allowance = a ;
        k++ ;
        cout << "\nObject " << "emp" << k << " is created: " ;
    }
}
```

```

~employee( )
{
    cout << "\nObject " << "emp" << k << " is destroyed" ;
    k-- ;
    getch() ;
}

void showdata() ;
};

void employee::showdata()
{
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(44) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    heading() ;
    employee empl ("George", 9000, 8000) ;
    empl.showdata() ;
    employee emp2 ("Vladimer", 5000, 4000) ;
    emp2.showdata() ;

    employee emp3 ("Natsag", 3000, 2000) ;
    emp3.showdata() ;
    cout << "\nEnd of the program\n" ;
}

```



	Employee Name	Basic	Allowance
Object emp1 is created:	George	9000	8000
Object emp2 is created:	Vladimar	5000	4000
Object emp3 is created:	Natsag	3000	2000

End of the program

Object emp3 is destroyed  
 Object emp2 is destroyed  
 Object emp1 is destroyed

Байгуулагдах объектыг тоолоход к өрөнхий хувьсагчийг хэрэглэнэ. Объект байгуулагдах бүрд байгуулагч функцийн уг тоолуурлын уттыг нэгээр нэмнэ. Бас устах объект бүрд устгагч функцийн нь байгуулагдсан объектын тоог нэгээр багасгана.

Утгаа гаднаас авах З объект байгуулагдаж улмаар програмын төгстөлийн команд болох cout<<"End of the program"<<endl; хийгдсэнийг програмын үр дүнд байгаа "End of the program" мөр харуулна. Гэхдээ энэ цэгт програм дуусгавар болоогүй нь түүний үр дүнгээс илрхий байна. Ямар нэгэн товч дээр товших хүртэл хэрэглэгчийн ажлын цонхыг

харуулж байх үүрэгтэй getch() функцийг main() функц дотор биш, харин устгагч функц дотор хэрэглэсэн байгаа. Програм нь:

```
End of the program  
Object emp3 is destroyed
```

хоёр мөрийг дараалуулан дэлгэн рүү бичих ба ингэхэд устгагч функц дуудагдсаныг сүүжийн мөр харуулж байна. Устгагч функцийн доторх getch() функц програмыг түр тасалдуулж хэрэглэгчийн цонхыг харуулсан хэвээр байна. Хэрэв аль нэгэн товч дээр товшиход эзлжит

```
Object emp2 is destroyed
```

мөрийг дэлгэшилээд програм түр зогсолт хийнэ. Энэ нь устгагч функц хоёрдахь удаагаа дуудагдсаныг илтгэнэ. Дахин аль нэгэн товч дээр товшиход устгагч функц гуравдахь удаагаа дуудагдсаныг дэлгэцэд бичигдэх

```
Object emp1 is destroyed
```

мер заана. Устгагч функц доторх getch() функц програмыг дахин түр тасалдуулна. Дахины дурын товч дээр товшисноор, жишээ нь Turbo C++ ажлын цонхопд буцаж орно. Програмын болит төгсгэл болохын омно устгагч функц 3 удаа дуудагдсаныг үр дүнгийн сүүлийн 3 мер заана.

#### 4.12 Объектон хүснэгт ба байгуулагч функц

Объектон хүснэгтийг доор үзүүлсэн шигээр байгуулдаг талаар "Класс, объект" бүлэгт үзсэн билээ.

```
employee e[5] ;
```

Энэ командаар employee төрлийн 5 объекттой е хүснэгт байгуулагдана. Ямар нэгэн байгуулагч функц классын тодорхойлолтод байгаа бол түүнийг програм үргэлж дуудаж хэрэглэнэ. Иймээс классын тодорхойлолт нь ямар ч байгуулагчгүй эсвэл ядаж анхдагч байгуулагчтай байхыг програм шаардлаг.

Дээрх хүснэгтийн тухайлсан бүтвэрт утга олгоходо байгуулагч функцийг доор үзүүлснээр дуудаж болно.

```
e[3] = employee ("Bill", 5000, 2000) ;
```

Харин параметртэй байгуулагч функцийг хэрэглэн объектон хүснэгтийн бүтвэрт гарсаны утга оноож болох ба ингэхдээ байгуулагч функцийг бүтвэр бүрл хэрэглэнэ.

```
employee e[5] = {employee ("1", 1, 1),  
                  employee ("2", 2, 2),  
                  employee ("3", 3, 3),  
                  employee ("4", 4, 4),  
                  employee ("5", 5, 5) };
```

Объектон хүснэгт хэрэглэх жишээг Program #4.12-т үзүүллээ.

```
//Program #4.12  
//Array of objects  
  
#include <iostream.h>  
#include <conio.h>  
#include <iomanip.h>  
#include <string.h>
```

```

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a) ;
    employee() ;
    void showdata() ;
} ;

employee :: employee(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee :: employee()
{
}

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee e[5] = {employee ("1", 1, 1),
                    employee ("2", 2, 2),
                    employee ("3", 3, 3),
                    employee ("4", 4, 4),
                    employee ("5", 5, 5)
                   } ;

    heading() ;
    for(int i=0; i<5; i++)
        e[i].showdata() ;
    getch() ;
}

```



Employee Name	Basic	Allowance
1	1	1
2	2	2
3	3	3

4	4	4
5	5	5

Давталтын `for` команд хэрэглэн хүснэгтийн бүх бүтвэр рүү хандах дурмийг бас объектон хүснэгтийн хувьд доор үзүүлсэн шигээр

```
for(int i=0; i<5; i++)
    e[i].showdata();
```

хэрэглэж болно.

#### Мэдлэгээ шалгах асуулт

- 4.1. Байгуулагч функци гэж юу болох, ийм функцийг класс дотор заавал хэрэглэх ёстой эсэх
- 4.2. Байгуулагч функцийг яаж дуудах
- 4.3. Байгуулагчийг яаж тодорхойлох
- 4.4. Аихдагч байгуулагч функци гэж юу болох, ийм функцийг яаж үүсгэх
- 4.5. Байгуулагч функцийн онцлог шинжийг жагсааж бичих
- 4.6. Аихдагч байгуулагч функцийн сүл тал юу болох
- 4.7. Параметртэй байгуулагч гэж юу болох
- 4.8. Параметртэй байгуулагчийн давуу тал юу болох
- 4.9. Класс олон байгуулагчтай байж болох эсэх; Хэрэв байж болох бол үүнийг баталгаажуулах нөхцөлийг тайлбарлах
- 4.10. Объектын динамик утга оноолт гэж юу болох, түүнийг яагаад хэрэглэх ёстой болдог
- 4.11. Яаж динамикаар объектод утга оноож болох
- 4.12. Байгуулагч руу хэрхэн илээр аргумент дамжуулах
- 4.13. Байгуулагч руу хэрхэн ил бишээр аргумент дамжуулах
- 4.14. Параметржсэн байгуулагчийн хязгаарлагдмал шинж юу болох
- 4.15. Гарааны утгатай байгуулагчийн давуу тал юу болох
- 4.16. Хуулагч байгуулагч гэж юу болох, түүний сайн чанар юунд орших
- 4.17. Хуулагч байгуулагч иэгээс олон параметр авч болох эсэх
- 4.18. Хуулагч байгуулагчийн функци рүү бэлэн объектыг утгаар нь дамжуулж болох эсэх
- 4.19. Байгуулагч функци мэдээллийн далд байдлыг хэрхэн нэмэгдүүлэх
- 4.20. Байгуулагч дотор гишүүдэд утга оноохыг хэзээ хэрэглэх
- 4.21. Устгач функци гэж юу болох, хэзээ хэрхэн түүнийг луудах
- 4.22. Устгач функцийг хэрхэн тодорхойлох
- 4.23. Устгач функцийн онцлог шинжүүд юу болох
- 4.24. Устгач функци чухал болохыг тайлбарлах
- 4.25. Устгач функцийн авч холбогдол юунд орших
- 4.26. Устгач функци мэдээллийн далд байдлыг хэрхэн нэмэгдүүлэх
- 4.27. Объект устах үед устгач функцийг дууддагтуй учир юу болох
- 4.28. Яагаад класс дотор гишүүн өгөгдлөд утга оноож болдоггүй
- 4.29. Гишүүн функцийн ямар ямар төрөл байж болох

### Жишээ бодлого

4.30. Дараах хоёр командын ялгаа юу болох

```
time t2(t1);  
time t2= t1;
```

4.31. Дараах турван үзүүллийн аль алийг нь хийх боломжтой байхаар string байгуулагчийг тодорхойлох

```
string s1("See Naples and die", 13) ;  
string s2(169) ;  
string s3 ;
```

4.32. Доор үзүүлсэн тодорхойлолт бүхий p/q ( $q \neq 0$ ) рациональ тооны rationalnumber класст add(), multiply() болон байгуулагч функциүүдийг нэмж тодорхойлох

```
class rationalnumber  
{  
    public:  
        rationalnumber(int p=1, int q=1) ;  
    private:  
        int p, q ;  
};
```

4.33. Комплекс тооны complex классыг хуулагч болон анхдагч байгуулагч функциүүдээс гадна add(), multiply(), show() функциүүдтэй байхаар тодорхойлох

4.34. Банкны харилцагчийн нэр, хувийн дугаар, хадгаламжийн хэмжээ гэсэн өгөгдлөө бүхий bankaccount классыг хадгаламж шинээр нээх (байгуулагч), мөнгө нэмж хийх, мөнгө авах, хадгаламжийг хаах, мөнгөний хэмжээг дэлгэц рүү гаргах гэсэн бүйлдэл хийх гишүүн функциүүдтэй байхаар тодорхойлох

4.35. Хоёр хэмжээст огторгуйн дурын цэгийн байршилыг заах бүхэл тоон x, у өгөгдлүүд бүхий point классыг шаардлагатай байгуулагч функциүүдээс гадна хоёр цэгийн хоорондын зайн олох функцийтэй байхаар тодорхойлох

4.36. Хувийн дугаар, нэр, зохиогч, хэвлэлийн газар, үнэ гэсэн үзүүлэлттэй library классыг ном олгох, бушаах гэсэн хоёр функцийн гадна шаардлагатай байгуулагч функциүүдтэй байхаар тодорхойлох

4.37. Цаг, минут, секундыг тус тусад нь хадгалахад тохирох бүхэл тоон өгөгдлүүд бүхий time классыг шаардлагатай байгуулагч, устгагч функциүүдээс гадна өгөгдсон цагуудын хоорондох нийт цагийг олох функцийтэй байхаар тодорхойлох

4.38. Огноо (жил, сар, өдөр) хадгалах өгөгдлүүд бүхий date классыг хоёр огнооны хоорондох нийт өдрийг олох функци, зохих байгуулагч болон устгагч функцийтэй байхаар тодорхойлох

4.39. Оюутны нэр, хувийн дугаар, голч дүн гэсэн мэдээлэл бүхий student классыг гаднаас мэдээлэл авч дэлгэц рүү гаргах хоёр функцийтэй байхаар тодорхойлох; Бас зохих байгуулагч болон устгагч функцийг нэмж тодорхойлох

### Програмчлах бодлого

4.40. Хоёр рациональ тооны нийлбэр болон үржвэрийг олох програмыг 4.32 дугаар бодлогод тодорхойлох классыг ашиглан бичих

4.41. Хоёр комплекс тооны нийлбэр болон үржвэрийг олох програмыг 4.33-т тодорхойлох классыг ашиглан бичих

4.42. Хадгаламж нээж, хаах, мөнгө нэмж хийх, авах командын цэс бүхий програмыг 4.34 дүгээр бодлогод тодорхойлох классыг ашиглан бичих

- 4.43. Өгөгдсон хоёр цэгийн хоорондын зайл олох програмыг 4.35 дугаар боллогод тодорхойлох классыг ашиглан бичих
- 4.44. Ном олгох, ном буцаах зэрэг командын цэстэй програмыг 4.36 дугаар боллогод тодорхойлох классыг ашиглан бичих
- 4.45. Өгөгдсон цагуудын хоорондох нийт цагийг олох програмыг 4.37 дугаар боллогод тодорхойлох классыг ашиглан бичих
- 4.46. Өгөгдсон хоёр огноо хооронд хэдэн одрийн зөрөө байгааг тооцож олох програмыг 4.38 дугаар боллогод тодорхойлох классыг ашиглан бичих
- 4.47. Оюутны мэдээллийг гараас авч дэлгэн рүү гаргах програмыг 4.39 дүгээр боллогод тодорхойлох классыг ашиглан бичих
- 4.48. Өвчтөний мэдээллийн *patient* классыг name, age, sex, date, diagnosis гэсэн өгөгдлийтэй, өвчтөнийг хүлээн авах, гаргах байгуулагч гишүүн функцуудтэй байхаар тодорхойлж програм бичих; Өвчтиүүдийн мэдээллийг хүснэгтэд хадгалдаг байх
- 4.49. Гол огогдлийн төрөл болох int-ыг орлох newint классыг int төрлийн танс хувьсагчтай байхаар тодорхойлох; Энэ класс дараах гишүүн функцуудтэй байна.
- (i) хувьсагч руу 0 хийх
  - (ii) хувьсагч руу int төрлийн утга хийх
  - (iii) хувьсагчийн утгыг int төрлийн тоо шиг харагдахаар дэлгэцлэх
  - (iv) хоёр гишүүн өгөгдлийн нийлбэрийг олох

Бас гарааны утгатай хоёр, гарааны утгагүй нэг объект тус тус байгуулж утга бүхий объектуудын нийлбэрийг олж гуравдахь объект руу хадгалаад хэвлэх програм бичих

- 4.50. Телбертэй гүүрээр гарах машин бүр 10 доллар толох ёстой ч заримдаа телбергүйгээр гарах тохиолдол байх ба гүүрээр гарсан машины тоо болон толборийн хэмжээг тоолж байх ёстой; Дээрх шаардлагыг хангах tollbridge класс тодорхойлох; Ингэхдээ хоёр гишүүн огогдлийн нэг нь unsigned int төрлийнх байх ба гүүрээр гарсан машины тоог, нэгөө нь double төрлийнх байх ба толборийн нийт хэмжээг тус тус хадгалах үүрэгтэй байна; Элгээр өгөгдлийг 0 болгох байгуулагчийг тодорхойлно;

Бас payingcar() функци машины тоог нэгээр, толборийг арваар нэмэгдүүлэх бол пораусcar() функци зөвхөн машины тоог нэгээр нэмэгдүүлэх ёстой; Эцэст нь машины тоо, толборийн нийт хэмжээг дэлгэнээд харуулах display() функци байх ёстой.

tollbridge классыг шалгах програмын алгоритм: Хэрэглэгч телбертэй, толборгүй машин тус бүрд харгалзуулан сонгосон нэг товчийг товших боломжтой байна; Харин ESC товчоор гүүрээр гарсан машины тоо, толборийн хэмжээг дэлгэнээд програм дуусгавар болох

- 4.51. Цаг, минут, секунд хадгалах гурван гишүүн өгөгдлөтэй time классыг тодорхойлох; Эл классын нэг байгуулагч нь бүх өгөгдэл 0, нэгөө байгуулагч нь гаднаас огех утгуудыг тус тус хийдэг байх ёстой; Мөн эл класс нь объектын утгыг xx:xx:xx хэлбэрээр хэвлэх, гаднаас авах time төрлийн хоёр объектын нийлбэрийг олох зориулалтын хоёр функцийтэй байна.

Өмнөх time классыг шалгах програмын алгоритм: Гурван объект байгуулахаас эхийн хоёр объект нь тодорхой утгатай, сүүлийнх нь хоосон объект байна. Утга бүхий хоёр объектын нийлбэрийг хоосон объект руу хийгээд дэлгэн рүү гаргах ёстой.

## ТАВДУГААР БҮЛЭГ

### ФУНКЦ ДАХИН ТОДОРХОЙЛОХ

- Параметрийн тооны ялгаа ба дахин тодорхойлох функц, 147
- Функц зарлах, тодорхойлох, 148
- Параметрийн төрөл ба дахин тодорхойлох функц, 148
- Функцийн төрөл ба дахин тодорхойлох функц, 151
- Дахин тодорхойлох функц хэрэглэхэд хийгдэх үйлдэл, 152
- Загвар функц, загвар класс, 153
  - Загвар функц, 153
  - Загвар класс, 155



## ТАВДУГААР БҮЛЭГ

### ФУНКЦ ДАХИН ТОДОРХОЙЛОХ

Ихэнх програмчлалын хэлэнд функц бүр бусдаас ялгагдах нэртэй байдаг. Иймд, жишээ нь хоёр, гурван тооны нийлбэр олох хоёр өөр функцийг Си хэлээр доор үзүүлсэн шигээр бичих шаардлагатай болно.

```
int sum2(int a, int b) ;
int sum3(int a, int b, int c) ;
```

Энэ sum2(), sum3() хоёр функц хобуулаа өгөгдсөн тоонуудын нийлбэр олох үүрэгтэй боловч өөр өөр нэртэй байна. Харин C++ хэлэнд ижил нэртэй, нэгэн төрлийн<sup>35</sup> боловч авах аргументийн тоо, торол, дараалал нь ялгаатай байх олон янзын функцийг тодорхойлох боломж бий. Ингэж нэр, торол нь ижил олон функц тодорхойлох энэхүү үйлийг функц дахин тодорхойлох гэх ба харин ийм функцийг дахин тодорхойло(гдо)х функц гээд. Дахин тодорхойлох бол тухайн байдалд бичдээ нь ялгаатай байхаар тодорхойлогдох ижил нэртэй функцуудийн чадвар юм.

#### 5.1 Параметрийн тооны ялгаа ба дахин тодорхойлох функц

Хоёр эсвэл гурван тооны нийлбэр олох програм бичих хэрэгцээ гарвал гаднаас авах параметрийн тоо нь ялгаатай байх хоёр өөр функцийг бичих шаардлагатай болно. Параметрийн тоо нь эхний функцийн хувьд 2 байхад удаах функцийн хувьд 3 байх жишээтэй. Эдгээр функцийг доор үзүүлснээр тодорхойлж болно.

```
//Program #5.1
//finding the sum

#include <iostream.h>
#include <conio.h>

int sum (int, int) ;
int sum (int, int, int) ;
void main()
{
    clrscr() ;
    cout << sum(7, 3, 4) << endl ;
    cout << sum(2, 9) << endl ;
    getch() ;
}

int sum(int a, int b)
{
    return (a + b) ;
}
int sum(int a, int b, int c)
{
    return (a + b + c) ;
}
```



14  
11

<sup>35</sup> Дахин тодорхойлох функцийн төрөл ижил байх эсэх нь програмчлалын холцс хамаардаг. C++ нь дахин тодорхойлох функц өөр өөр төрлийнх байхыг зөвшөөрдог хэл юм.

## 5.2 Функции зарлах, тодорхойлох

Өмнөх програмын эхэн хэсэгт `int` төрлийн хоёр функцийг адилхан `sum` нэртэйгээр зарлаж

```
int sum (int, int) ;
int sum (int, int, int) ;
```

хэрэглэгчийн бусад функцийн нэгэн адилгаар `main()` функцийн дараагаар тодорхойлсон `main()` функци дотроос тэдгээр функцийг иэрээр нь дуудаж хэрэглэхэд C++ компайлер хэрхэн зөв функцийт нь сонгож авдаг вэ? Компайлер нь дуудагдаж буй функцийн аргументийн тоо, төрлийг функцийн тодорхойлолтын толгойн хэсгийн бичээстэй дүйлгэн үзэх замаар аль функцийг дуудахаа шийднэ. Иймд турван аргумент авах ба тэдгээр нь цом `int` төрлийнх байх `sum` функцийг `main()` функцийн эхний

```
cout << sum(7, 3, 4) << endl;
```

командаар дуудах тул

```
int sum(int a, int b, int c)
{
    return (a + b + c) ;
```

функци ажиллаж 14 гэсэн хариуг буцааж өгнө. Үүнтэй нэгэн адилгаар, `main()` функцийн удаах

```
cout << sum(2, 9) << endl ;
```

командын хувьд

```
int sum(int a, int b)
{
    return (a + b) ;
```

функци дуудагдах болгоод түүнээс 11 гэсэн утга бушина.

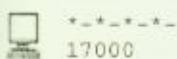
## 5.3 Параметрийн төрөл ба дахин тодорхойлох функци

Дахин тодорхойлогоо функцийн өөр нэг шинж бол төрөл ба нэр нь ижил, тоо нь адил боловч өөр өөр төрлийн параметртэй олон функци тодорхойлох юм. Энэ шинжийг жишээ програмаар үзүүльс.

```
//Program #5.2
//Displaying characters and finding the sum
#include <iostream.h>
#include <conio.h>
void show(char, char) ;
void show(int, int) ;
void main()
{
    clrscr() ;
    show('*', '-') ;
    show(9000, 8000) ;
    getch() ;
}
void show (char a, char b)
{
    for (int k=1; k<5; k++)
        cout << a << b ;
```

```
    cout << endl ;
}

void show (int m, int n)
{
    cout << m + n << endl ;
}
```



Program #5.2-т хоёулаа show() нэртэй хоёр функцийг main() функцийн өмнө доорх шигээр зарласан байна.

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

Эдгээр функцийн тус бүрдээ хоёр параметртэй. Гэхдээ параметрүүд нь өөр өөр төрлийнх байна. Эхний функцийн хоёр параметр нь char төрлийнх байх бөгөөд дараалсан од (\*), зураасыг (-) 4 удаа дэлгэцэд гаргана. Харин ногоге show() функцийн int хоёр параметр авч нийлбэрийг нь олж дэлгэших үүрэгтэй. Эдгээр функцийн тодорхойлолтыг main() функцийн дараагаар бичиж өгснийг програмын кодоос харж болно.

Program #5.2-ын main() функцийн доторх show('\*', '-'); командаар дараалсан од, зураасыг 4 удаа дэлгэцэд бичих ба ингэхдээ

```
void show (char a, char b)
{
    for (int k=1; k<5; k++)
        cout << a << b ;
    cout << endl ;
}
```

функцийг дуудна. Дараагаар нь show(9000, 8000); командаар int төрлийн хоёр параметр авах

```
void show (int m, int n)
{
    cout << m + n << endl ;
}
```

функцийг дуудснаар энэ функцийн 17000 гэсэн тоог дэлгэцэд бичин.

Нэр ижилтэй боловч авах параметрийн тоо, төрлөөрөө өөр олон функцийг байж болох уу? Ийм олон функцийн тодорхойлж 'болно. Энэ санааг програмд хэрхэн буулгахыг, дахин тодорхойлогдсон функцийг класс дотор хэрхэн хэрэглэхийг дараах жишээ программаар язувье.

```
//Program #5.336
//Finding the gross pay of an employee using function overloading

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>
```

<sup>36</sup>Энэ програм нь Program #3.1-той тостой.

```

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void getdata(void);
    void showdata(void);
    void showdata(char);
    void showdata(employee);
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);

    cout << "Enter the Basic Pay: ";
    cin >> basicpay;
    cout << "Enter the Allowance: ";
    cin >> allowance;
}

void employee::showdata()
{
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
}

void employee::showdata(char ch)
{
    cout << endl;
    for(int k = 1; k < 49; k++)
        cout << ch;
}

void employee::showdata(employee e)
{
    int grosspay = e.basicpay + e.allowance;

    cout << endl;
    cout << setw(20) << e.name;
    cout << setw(8) << e.basicpay;
    cout << setw(12) << e.allowance;
    cout.width(8);
    cout << grosspay;
}

void main()
{
    clrscr();
    employee emp;
    emp.getdata();
    emp.showdata('-');
    emp.showdata();
    emp.showdata('-');
}

```

```

    emp.showdata(emp) ;
    getch() ;
}

[ ] Enter the Employee Name: George.
[ ] Enter the Basic Pay: 9000.
[ ] Enter the Allowance: 2000.

-----  

Employee Name   Basic     Allowance    Gross  

-----  

George        9000       2000        11000

```

Program #5.3-т showdata() нэртэй 3 функц тодорхойлсноос эхнийх нь параметргүй, удаах нь char төрлийн нэг параметртэй, сүүлийних нь employee төрлийн объектон параметртэй юм. Эдгэр функцийг employee класс дотор зарлаж, үйлчлэх хүрээний оператор хэрэглэн классын гадна нь тодорхойлсон байна.

#### 5.4 Функцийн төрөл ба дахин тодорхойлох функци

Дахин тодорхойлох нь тухайн байдалд өөр өөр бичдэстэй байх ижил нэртэй функциүүдийн чадвар болох, гэхдээ функцийн төрлийг бичдэсийн нэг хэсэг гэж үзэх эсэх нь програмчлалын хэлээс хамаардаг тухай өмнө үзсэн билээ. Функцийн төрөл нь өөр өөр байхаар функцийг дахин тодорхойлох аргыг доорх жишээнд үзүүллээ.

```

//Program #5.4
//Demonstration of overloaded function with different types
//
#include <stdio.h>
#include <iostream.h>

class student
{
    int i ;
    float f ;
public:
    student(): i(2), f(5.5) {}
    float square(float f)
    {
        this->f=f*f ;
        return this->f;
    }
    int square(int i)
    {
        this->i=i*i;
        return this->i ;
    }
    void show()
    {
        cout << "f= " << f << "\ni= " << i;
    }
}
main()
{
    student aa;
    aa.show();
}

```

```

float ff= aa.square ((float) 3.5) ;
int ii = aa.square (3);
cout << endl << endl << "ff= " << ff << "\nii= " << ii;
}

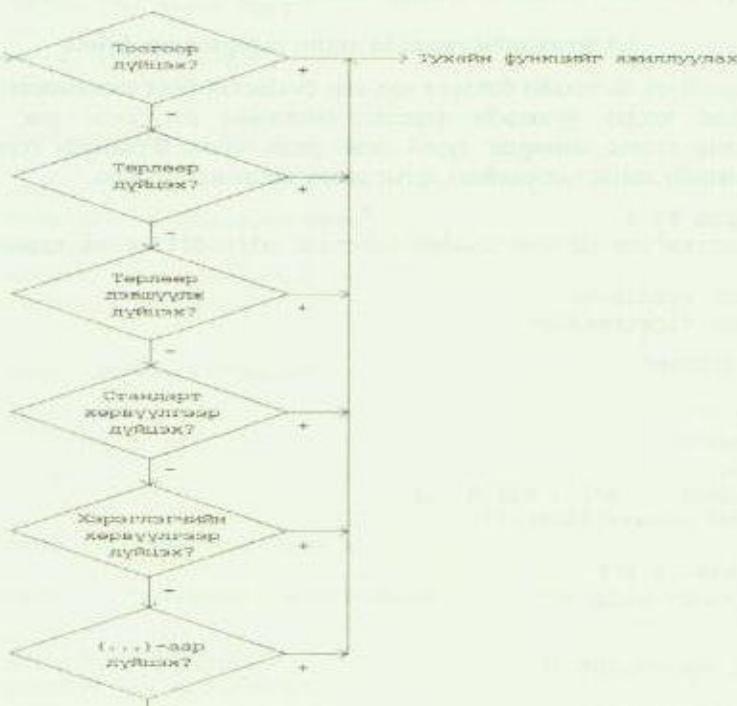
f= 5.5
i= 2

ff=12.25
ii=9

```

### 5.5 Дахин тодорхойлох функцийн хэрэглэхэд хийгдэх үйлдэл

Дахин тодорхойлогдох функцийг C++ програмд хэрэглэх тохиолдолд нийм функцийг дуудсан командын мөр бүрийн хувьд хамгийн сайн тохирох функцийг C++ компайлер олж ажилтуулах үүрэгтэй. Үүний тулд компайлер дараах үйлдлийг хийнэ (Зураг 5.1).



Зураг 5.1: Дахин тодорхойлсон функцийг олж дуудах дараалал

- Параметрийн тоо.** Дуудагдах функцийн аргументийн тоо нь түүний тодорхойлолт дахь параметрийн тоотой тохирч байвал дараагийн үйлдэлл шилжинэ.
- Параметрийн төрөл.** Дахин тодорхойлогдсон функциүүл дотроос хамгийн сайн тохирохыг нь сонгохлоо 5 шаталгатай шалгалт хийх ба тухайн шалгальт бүтэлгүйтвэл дараагийн шатныг руу шилжинэ. Тухайлбал,
  - Харгалзан яг цав дүйх:** C++ компайлер нь функцийн аргумент ба параметр хоорондос дүйж байгаа эсэхийг шалгана. Хэрэв шалгальт зөв, тэдгээр нь хоорондоо яв цав дүйж байвал тухайн функцийг дуудаж ажиллуулна.

- (ii) *Дэвшүүлж дүйцүүлэх*: Харгалзуулан дүйлгэх (i) дүгээр шалгалт бүтэлгүйтвэл энэ шалгалт хийгдээ. Ингэхдээ
- `char, short, int` ба `enum` төрлийн объектыг `int` төрөлд дэвшүүлж дүйлгэнэ.
  - `float` төрлийг `double` төрөлд дэвшүүлж дүйлгэнэ.
- Энэ шалгалтыг олон функц хангаж байвал тэдгээрээс хамгийн бага дэвшүүлэлтийг нь сонгож ажиллуулна.
- (iii) *Стандарт хөрвүүлэгээр дүйцүүлэх*: Стандарт төрлийг стандарт төрөлд нь хөрвүүлэх дүрмийн дагуу дүйцүүлэх ажлыг хийнэ.
- `int, long, float, double` мэтийн тоон төрөл хооронд хөрвүүлэг хийнэ. Жишээ нь, `int` төрлийг `float` төрөлд хөрвүүлэх гэх мэт.
  - Нээрэсэн тогтмолон төрлийг тоон төрлийн параметр рүү хөрвүүлэх
  - Тэгийг тоон эсвэл хаяган төрлийн параметр рүү хөрвүүлэх
  - Дурын хаяган төрлийг `void*` төрлийн параметр рүү хөрвүүлэх
  - Хэрэв хоорондоо зөрчилдөхгүй бол удамших классын хаягийг эх классын нь хаяг рүү хөрвүүлэх
- (iv) *Хэрэглэгчийн хөрвүүлгийн функцээр дүйцүүлэх*: Хэрэглэгчийн хөрвүүлгийн функц нь классын гишүүн функц байна. Ийм функц нь дараах хоёр төрлийнх байж болно.
- Зөвхөн нэг параметртэй байгуулагч функц
  - Хөрвүүлэг хийх оператор функц
- C++ компайлер нь эдгээр функцээр дүйцүүлэх оролдлого хийнэ.
- (v) *Тодорхойлох үргэлжлэлээр дүйцүүлэх*: (...) тэмдэглээ нь хувьсах тооны параметртэй эсвэл төрөл өөртэй параметртэй функц гэдгийг заана. Энэ дүйцүүлэх шалгалтыг өмнөх бүх шалгалт бүтэлгүйтэх үсд хийнэ.

Нэр ижилтэй функцуудийн үйлчлэх хүрээ ижил байх тохиолдолд тэдгээрийг дахин тодорхойлж болдог. Жишээ нь, ижилхэн F нэртэй хоёр функцийн нэг нь еронхий юм уу гадаад функц, ногдо нь аль нэгэн классын гишүүн функц байх тохиолдолд энэ хоёр функцийг дахин тодорхойлогдсон функц шигээр хэрэглэж болохгүй. Ийм боломжтой болгохын тулд тэдгээр функцийг хоёуланг нь еронхий эсвэл гадаад функц эсвэл классын гишүүн функц болгох шаардлагатай. Ер нь, хоорондоо төстэй ажилд дахин тодорхойлогдсон функцийг хэрэглэх нь хамгийн зөв шийдэл юм.

## 5.6 Загвар функц, загвар класс

Template буюу загвар бол ANSI C++ хэлний шинэ боломж бөгөөд сренхий (*generic*) буюу параметржсэн (*parameterized*) гэсэн уттыг илэрхийлнэ. C++ компайлериийн хувьд загвар функц, загвар класс нь ерийн функц, ерийн класс биш зөвхөн хийсвэр загвар тул тэдгээрийг бодитойгоор хэрэглэх хүртэл ийм загвар функцийн, ийм загвар классын эх кодыг систем үүсгэдэггүй байна. Загвар функцийн болон классын тодорхойлолт, тэдгээрийн хэрэглээ цутгаа нэг файлд байх ёстой.

### 5.6.1 Загвар функц

Функцийг дахин тодорхойлохын оронд template тусгай үгийг хэрэглэн аль ч төрлийн огогдол авах, буцаах сренхий функц, сренхий класс тодорхойлох боломжтой юм. Загвар функц нь макро функцтэй төстэй юм. Загвар функц тодорхойлоходоо доор үзүүлсэн дурмийг мөрдөнө.

```
template <class identifier> function_declaraction ;
```

Хоёр тооны ихийг нь олох GetMax функцийг өмнөх загварын дагуу

```
template <class GenericType>
GenericType GetMax(GenericType a, GenericType b)
{
    return (a>b?a:b);
}
```

гэж тодорхойлно. Энэ загвар функцийн тодорхойлолт дотор хэрэглэж байгаа GenericType бол тодорхой торлийг заахгүй тул функцийг дуудахдаа өгөгдлийн торлийт нь дараах загварын дагуу зааж өгнө.

```
function <pattern> (parameters) ;
```

Функцияг гаднаас дамжуулах өгөгдлийн торлийт pattern заах ба функцийн тодорхойлолтод байх GenericType үгийн ороонд бичигдэнэ. Тэгэхлээр int тооны хувьд GetMax() функцийг

```
int x, y ;
GetMax <int> (x,y) ;
```

гэсэн загварын дагуу дуудаж хэрэглэж болохыг дараах програмаас харж болно.

```
//Program #5.5
//function template

#include <iostream.h>

template <class T>
T GetMax(T a, T b) {
    return (a>b ? a: b);
}

main()
{
    int i=5, j=6, k ;
    long l=10, m=5, n ;
    k = GetMax (i, j) ;
    n = GetMax (l, m) ;
    cout << endl ;
    cout << k << endl ;
    cout << n << endl ;
    return 0 ;
}
```

□ 6  
10

Загвар функцийг main() дотроос дуудахдаа дамжуулах өгөгдлийн төрлийт өндөгт хаалт дотор бичиж өгөхгүй. Систем түүнийг гаднаас авах аргумент дээр нь тулгуурлан тодорхойлж измж бичнэ. Дээр тодорхойлсон GetMax() функцийн хоёр параметр ижил торлийнх байх ёстой. Оор оор торлийн хоёр параметр авах функцийн загварыг доор үзүүлсэн шигээр тодорхойлно.

```
template <class T, class U>
T GetMin(T a, U b)
{
    return (a<b?a:b);
}
```

Дээрх тодорхойлолтод GetMin() функц хоёр өөр параметр гаднаас авах ба функцээс буцах объектын төрөл нь эхний параметрийнхтэй адил гэж тодорхойлжээ. Энэ функцийг дараах програмд үзүүлсэн шигээр хэрэглэнэ.

```
//Program #5.6
//function template 2

#include <iostream.h>

template <class T, class U>
T GetMin(T a, U b) {
    return (a<b ? a: b) ;
}

main()
{
    int i=5, k ;
    long l=10, m ;
    k = GetMin (i, l) ;
    m = GetMin (l, i) ;
    cout << endl ;
    cout << k << endl ;
    cout << m << endl ;
    return 0 ;
}
```



5

5

### 5.6.2 Загвар класс

Зарим өгогдлийн төрөл нь классын тодорхойлолтын түвшинд тодорхой биш байх классыг доор үзүүлсэн шигээр загвар классаар тодорхойлж болно.

```
//Program #5.7
//class template

#include <iostream.h>

template <class T>
class pair {
private:
    T value1[2] ;
public:
    pair(T first, T second)
    {
        value1[0] = first ;
        value1[1] = second;
    }
    T getMax() ;
};

template <class T>
T pair<T>::getMax()
{
    return (value1[0] > value1[1] ? value1[0]:value1[1]);
}

main()
{
    pair <int> myObj (115, 36) ;
```

```
pair <float> myObj2 (3.3, 2.18) ;
cout << endl ;
cout << myObj.getMax() ;
cout << endl ;
cout << myObj2.getMax() ;
return 0 ;
}
```

 115  
3.3

Гишүүн функцийг классын гадна тодорхойлохдоо `template <...>` утварыг өмнөх жишээ програмд үзүүлсний адил аар заавал хэрэглэнэ. Загвар классын хувьд `class` тусгай үгийг C++ хэлний дотоод суурь төрлийн хамт хэрэглэж болох ба үүнийг дараах програмаар харууллаа.

```
//Program #5.8
//class template

#include <iostream.h>
template <class T, int N>
class array {
private:
    T memblock[N] ;
public:
    setmember(int x, T value) ;
    T getmember(int x) ;
};

template <class T, int N>
array<T,N>::setmember(int x, T value)
{
    memblock[x] = value ;
    return 0 ;
}

template <class T, int N>
T array<T,N>::getmember(int x)
{
    return memblock[x] ;
}

main()
{
    array <int,5> myints ;
    array <float, 5> myfloats ;
    myints.setmember(0, 100) ;
    myfloats.setmember(3.0, 3.14) ;
    cout << endl ;
    cout << myints.getmember(0) ;
    cout << endl ;
    cout << myfloats.getmember(3) ;
    return 0 ;
}
```

 100  
3.14

**Мэдлэгээ шалгах асуулт**

- 5.1. Функцийг дахин тодорхойлох гэж юу болох, энэ ойлголтыг хэзээ хэрэглэх
- 5.2. Параметрийн тоо, төрөл нь өөр байх функциүүдийг тодорхойлохын давуу тал юу болох
- 5.3. Параметрийн тоо нь өөр байх функциүүдийг хэрхэн тодорхойлох
- 5.4. Параметрийн төрөл нь янз янз байх функциүүдийг хэрхэн тодорхойлох
- 5.5. Функцийн аргументэд хамгийн сайн дүйх функцийг олох алгоритмд ямар ямар алхам байдаг
- 5.6. Тогтолцон гишүүн огогдлийн хүснэгтийг класс дотор байгуулж болдогтүйн учир юу болох
- 5.7. Функцийг дахин тодорхойлох шаардлага юу болох

**Жишээ бодлого**

- 5.8. Дөрвөлжин, гурвалжны талбай олох хоёр функцийг адилхан нэртэй байхаар тодорхойлох
- 5.9. Гаднаас авах нэг тооны куб зэрэгтийг бодож дэлгэшлэх гурван функцийг ижил нэртэй байхаар тодорхойлох
- 5.10. Дараах хоёр функцизэс аль нь f(0) командаар идэвхжих

```
void f(const char *);  
void f(char *);
```
- 5.11. Дараах 3 тохиолдлын аль алийг нь зөвшөөрөх message() функцийг тодорхойлох

```
message("File not found", pfile);  
message("No free store");  
message("Array out of bound", Index);
```

**Програмлах бодлого**

- 5.12. Бодлого 5.8-д тодорхойлсны дагуу дөрвөлжин, гурвалжны талбайг олох програм бичих
- 5.13. Бодлого 5.9-д тодорхойлсны дагуу тооны куб зэрэгтийг олох програм бичих
- 5.14. Талын урт нь өгөгдхөн квадратын, 3 талын урт нь өгөгдхөн гурвалжны, радиус нь өгөгдхөн тойргийн талбайг олох програмыг дахин тодорхойлох функцийг хэрэглэн бичих
- 5.15. Өгөгдхөн зерэг тоог шалгах програмыг функцийг дахин тодорхойлох ойлголтыг ашиглан бичих
  - a) анхны тоо
  - b) Фибоначийн тоо
  - c) анхны ба Фибоначийн тоонууд
- 5.16. Гурвалжин, квадрат, тэгш өнцөгтийг зурах програмыг функцийг дахин тодорхойлох ойлголтыг ашиглан бичих

# ЗУРГААДУГААР БҮЛЭГ

## УДАМШИХ, УДАМШУУПАХ

- Удамшлын тухай ойлголт, 161
- Эх класс, удамших класс, 162
- Удамшлын төрөл, 162
- Удамших классын тодорхойлолт, 165
- Private, public горимоор удамших класс, 166
- Дан удамшил, 170
- Харагдах байдал: private, public ба protected, 172
- Олон тувшинт удамшил, 174
- Олон-нэг буюу нийлмэл удамшил, 178
- Удамших, удамшуулах үйлдлийн давуу тал, 179
- Хийсвэр функц, 179
- Удамшлын давхардлыг арилгах, 182
- Давхар класс, 183

-Үргэлжлэл-



-Үргэлжлэл-

- Удамшил ба байгуулагч функц, 186
- Удамшил ба устгагч функц, 193
- *derived\_class (const base\_class &)* байгуулагч, 193
- Функт дахин програмчлах, 197
  - Дахин тодорхойлсон функцийг дахин програмчлах, 200
  - Жинхэнэ хийсвэр функц ба хийсвэр класс, 200
- C++ класс тодорхойлоход анхаарах зүйл, 201



## ЗУРГААДУГААР БҮЛЭГ

### УДАМШИХ, УДАМШУУЛАХ

Класс, түүний объектын талаар гуравдугаар бүлэгт судалж объект байгуулах, классын гишүүд рүү хандах аргуудыг програмд хэрэглэж үзсэн билээ. Үүнд тулгуурлан классын объект, түүний гишүүн функцийн хандалтын мужийг Хүснэгт 6.1-д үзүүлсэн байдлаар тодорхойлж болно.

Хүснэгт 6.1: Класс доторх хандалтын муж

	Хандалт	Гишүүн өгөгдөл рүү	Гишүүн функци рүү
Объект	чадахгүй	private	private
	чадахгүй	protected	protected
	чадна	public	public
Гишүүн функци	чадна	private	private
	чадна	protected	protected
	чадна	public	public

Объект зөвхөн классынхаа public гишүүд рүү, гишүүн функци бүх гишүүд рүүгээ тус тус хандаж чадна.

#### 6.1 Удамшлын тухай ойлголт

ОХП-д удамшил нь нэгэнт бий болсон зүйлийг дахин хэрэглэхтэй холбогдох ойлголт юм. Туршиж бэлэн болгосон классын кодыг өөр системд дахин хэрэглэж болох ба үүнийг эргэн хэрэглэгдэх чадвар, чадамж гэнэ. Зүгширсэн классыг өөр програмд аль болох өргөн хэрэглэснээр хийх ажлын хэмжээ багасахаас гадна програмд гарч болох алдааг багасгах хирээр програм боловсруулах хугацаа богиносож өртөг нь буурна. Иймд програм бичихдээ C++ хэлний бэлэн функцийн санг аль болох хэрэглэх нь зөв сонголт юм. Гэхдээ стандарт функциүүдийн эх код нийтэд нээлттэй байдаггүй учраас тэдгээрийг зөвхөн зааврын дагуу нь дудаж хэрэглэж болохоос өөрчилж өргөтгөх боломжгүй. Тэгэхлээр, C++ функцийн сан нь нэг түвшинд дахин хэрэглэгдэх зүйл юм.

C++ хэл мөн классын сантай бөгөөд түүнд классын тодорхойлолт нь агуулагдаж байх тул хэрэв ийм сангийн эх нь байвал түүнийг өөрчилж өргөтгэж хэрэглэх боломж бий. Энэ ойлголтыг ОХП-д удамшил гэдэг. Энэ нь бэлэн байгаа классаас шинэ класс үүсгэх арга технологи юм. Шинээр үүсэх класс нь өмнөх классынхаа шинжийг өвлөн авна.

C++ хэлний хувьд эргэн хэрэглэгдэх чадвар, чадамжийн тухай ойлголтыг удамшлын ойлголтоор өргөжүүлэн хэрэглэх болжээ. Үнэндээ, удамшил, удамших нь ОО технологид байдаг чухал шинжүүдийн нэг, байгаа классаас шинэ класс үүсгэх процесс юм. Эх класс эх кодын эсвэл хоёртын кодын хэлбэрт байгаагаас хамаарахгүйгээр түүнээс шинэ класс гарган авч болох тул програм үйлдвэрлэгчид<sup>37</sup> энэ чиглэлээр дагнан ажилладаг байна. Visual C++ MFC (Microsoft Foundation Class) бол үүний сонгодог жишээ юм.

Удамшилаар үүсэх класс өмнөх классынхаа бүх эсвэл зарим шинжийг өвлөн авах ба үүнийг байгаа классаас шинээр класс үүсгэх удамшлын механизм гэнэ.

Боловсруулах объект нь тээврийн хэрэгсэл байх програмд тодорхойлж зүгшрүүлсэн automobiles класс байвал түүнээс бүх эсвэл зарим шинжийг нь өвлөх суудлын машины cars, автобусны buses гэх зэрэг шинэ классыг удамшуулан үүсгэж болно.

<sup>37</sup> Системийн түвшинд програм хөгжүүлэгчдийг програм үйлдвэрлэгчид гэж үзлээ.

Програмчлалд удамших шинжийг хэрэглэснээр код болон програмыг дахин хэрэглэх, кодыг дундаа хэрэглэх, тоотмол интерфейс хэрэглэх боломж бүрдэнэ.

### 6.2 Эх класс, удамших класс

Өөрөөс нь шинэ класс үүсч байгаа классыг үндсэн класс, шинээр үүсч байгааг нь удамших, удамшсан класс гэнэ. Үндсэн классыг бас дэд класс, эх класс гэдэг. Үүнтэй адил, удамших классыг дэд класс, охин класс гээц. А нь В классын дэд класс бол А нь В классын оргетгэл гэнэ. Удамших класс түүнээс шинэ класс удамшиж үүсэх үед эх класс болох тул эх, удамших класс гэдэг нь харьцангуй шинжтэй ойлголт юм. Удамшил үе дамжих шинжтэй.

Удамшиж үүсэх класс нь эх классынхаа шинж, аргаас өвлөж авахаас гадна өөрийн гэсэн шинжтэй, өөрөөр хэлбэл эх классст байхгүй гишүүдтэй байж болно. Эх классын гишүүн функцийг удамших класс дотор дахин тодорхойлж болох ба ингэхэд эх классын эх код заавал байх албагүй.

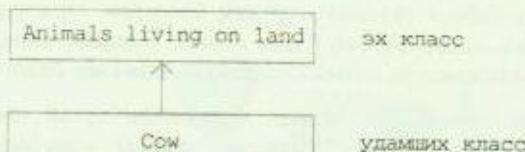
Ямар ч албан байгууллага, үйлдвэрийн газарт менежер, инженер, техник ажилтан зэрэг олон ажил мэргэжлийн хүмүүс байх тул employee гэх эх класс тодорхойлоод түүнээс, жишээ нь engineer классыг удамшуулан шинээр үүсгэж болно.

### 6.3 Удамшлын төрөл

Удамшлын 5 хэлбэр байдаг. Тухайлбал,

- Нэг-нэг буюу дан, энгийн удамшил,
- Олон-нэг буюу нийлмэл удамшил,
- Олон түвшинг удамшил,
- Шаталсан удамшил,
- Холимог удамшил.

*Инг-нэг буюу энгийн удамшил.* Удамших класс зөвхөн инг эх класстай байна. Жишээ нь, (cow) үхэр нь (Animals living on land) амьтны аймагт багтах тул cow класс нь "Animals living on land" классаас удамшиж үүссэн гэж үзэж болно. Эх класс, удамших класс хоорондох ийм хамаарлыг доор үзүүлсэн шигээр дүрслэхдээ сумыг эх класс руу нь харуулж зурна (Зураг 6.1).



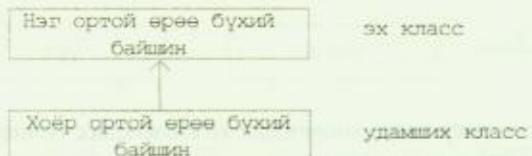
Зураг 6.1: Инг-нэг буюу энгийн удамшил

```
class animals
{
}

class cow : public animals
{
```

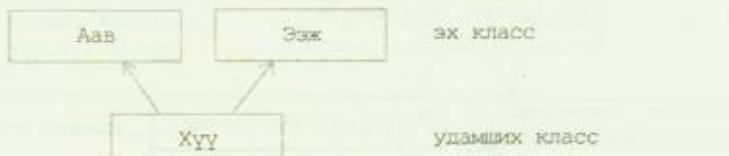
Зураг 6.1-д үзүүлсэн харьцаа нь үхэр амьтны аймгийн дэд бүлэг болохыг заана. Гэхдээ удамших дэд класс бүр заавал эх классынхаа дэд бүлэг байх албагүй. Жишээ нь, нэг ортой өрөө бүхий байшинг эх класс гэвэл түүнээс удамших классыг хоёр ортой өрөө бүхий байшин

тэж болно (Зураг 6.2). Гэхдээ бүх хоёр ортой өрөө бүхий байшин нэг ортой өрөө бүхий байшингийн дэд бүлэг байх албагүй.



Зураг 6.2: Нэг-нэг удамшил

**Олон-нэг буюу нийлмэл удамшил.** Зарим тохиолдолд удамших класс нь хоёрроос цөөнгүй эх класстай байж болно. Жишээ нь, аав, эзжээс хүү төрөх тул аав, эзж нь "эх классууд", хүү нь "удамших класс" болно (Зураг 6.3).



Зураг 6.3: Олон-нэг буюу нийлмэл удамшил

```

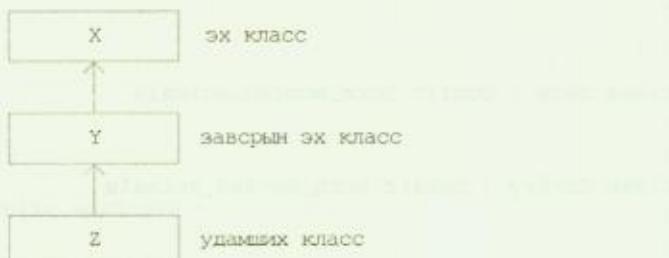
class father
{
};

class mother
{
};

class son : public father, public mother
{
};
  
```

Ийм удамшил нь удамших класс хэд хэдэн эх классын шинжийг овлож авах үед илэрдэг.

**Олон түвшинт удамшил.** Удамших класс өөрөө удамших классаас үүснэ. Жишээ нь, Z класс Y классаас, Y класс өөрийнхөө эзлжинд X классаас үүсэх бол энэ шинжийг олон түвшинт удамшил гэх ба Зураг 6.4-т үзүүлсэн шигээр дүрсэлж болно.



Зураг 6.4: Олон түвшинт удамшил

```

class X
{
};
  
```

```

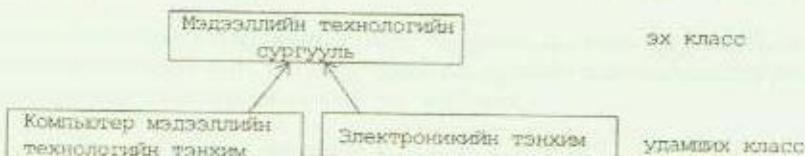
class Y : public X
{
};

class Z : public Y
{
};

```

Дээрх Y класс нь Z, X гэсэн хоёр класс хоорондын холбоос класс болно. XYZ нь Z классын (*inheritance path*) удамшлын зам юм.

**Шаталсан удамшил.** Нэг эх классаас хоёрөс иеөнгүй удамших класс үүсэх бол ийм удамшлыг шаталсан удамшил гэнэ. Жишээ нь, Мэдээллийн технологийн сургууль нь Компьютер, мэдээллийн технологийн ба Электроникийн гэсэн хоёр тэнхимтэй бол үүнийг Зураг 6.5-д үзүүлсэн шигээр дүрсэлж болно.



Зураг 6.5: Шаталсан удамшил

```

class sit
{
};

class sitd : public sit
{
};

class ed : public sit
{
};

```

**Холимог удамшил.** Холимог удамшил бол олон-нэг буюу нийлмэл удамшил, шаталсан удамшил хоёрын ийллэмж удамшил юм. Эвэр туурайгтын ваймагт багтах гүү, илжиг хоёрын лундаас гарах луус бол ийм удамшлын жишээ болно (Зураг 6.6).

```

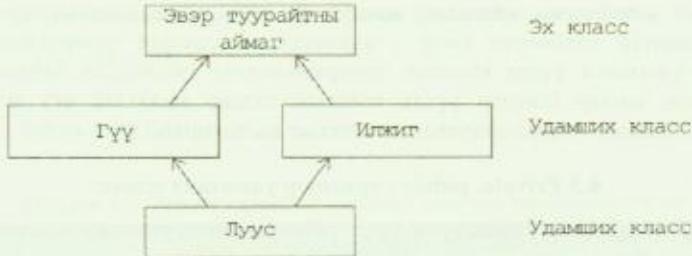
class horn_hoofed_animals
{
};

class mare : public horn_hoofed_animals
{
};

class donkey : public horn_hoofed_animals
{
};

class mul : public mare, public donkey
{
};

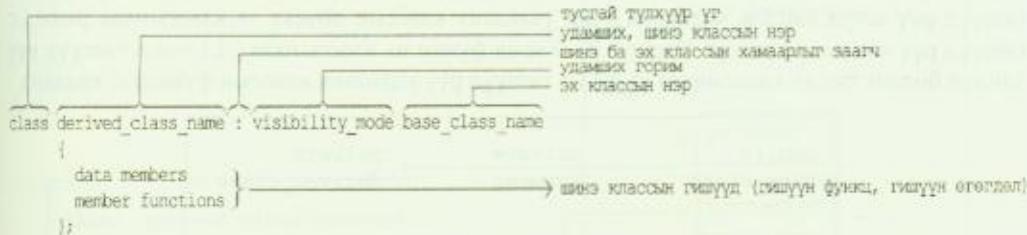
```



Зураг 6.6: Холимог удамшил

#### 6.4 Удамших классын тодорхойлолт

Удамших класс нэг эсвэл хэд хэдэн классаас удамшиж үүсч болох талаар өмнөх бүлэгт үзсэн билээ. Удамших классыг тодорхойлохдоо тухайн класс ба түүний эх класс хоорондын хамаарлыг зааж өгдөг. Удамших классын тодорхойлолтын срөнхий загварыг Зураг 6.7-д үзүүлсний дагуу дүрсэлж болно.



Зураг 6.7: Удамших классын тодорхойлолт

Давхар цэг `:` нь шинээр удамших класс ба эх класс хоорондын хамаарлыг, тухайлбал шинэ класс аль классаас удамшиж үүсэхийг заах үүрэгтэй юм. Удамших горим нь эх класс юрхэн удамшигыг заана. Эх класс `public` эсвэл `private` горимын аль нэгээр удамшдаг. Гэхдээ удамших горимыг зааж өгөхгүй байж болох ба энэ тохиолдолд удамших горим аяннаасаа `private` байна. Шинэ класс бас өөрийн гэсэн гишүүдтэй байж болно.

Өмнөх бүлгүүдэд тодорхойлж хэрэглэсэн `employee` классаас удамших `engineer` классыг доор үзүүлснээр тодорхойлж болно.

```

class employee
{
private :
    char name [20];
    int basicpay;
    int allowances;
public :
    void getdata ();
    void showdata ();
};

class engineer : public employee
{
private :
    int bonus;
public :
    void getdata() ;
    void showdata() ;
};
  
```

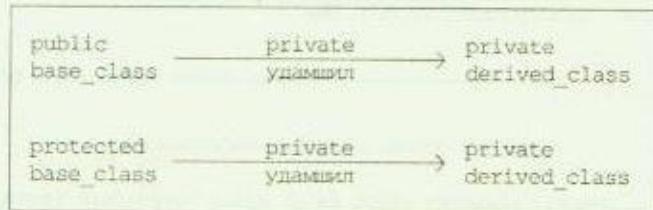
Дээрх жишээ нь `employee` классаас, жишээ нь `engineer` классыг хэрхэн үүсгэхийг үзүүлнэ. Удамшилтай холбоотой ямар ч мэдээлэл эх классын тодорхойлолтод байхгүй, зөвхөн шинээр удамшиж үүсэх классын тодорхойлолтод тусгагдсан байдаг. Эх класс нь түүнээс удамших замаар шинээр үүсэх классын талаар мэдээлэл огт авдаггүй. Харин удамшиж үүсэх классын тодорхойлолтод эх классыг нь тодорхой зааж өгдог.

### 6.5 Private, public горимоор удамших класse

Классыг `private` горимоор удамшуулж үүсгэх бол удамших классын тодорхойлолтыг доор үзүүлсэн загварын дагуу хийнэ.

```
class derived_class : private base_class
{
    data_members ;
    member_functions ;
} ;
```

Эх класс `private` горимоор удамшихад түүний `public` гишүүд удамшиж шинээр үүсэх классын `private` гишүүд болно (Зураг 6.8). Хүснэгт 6.1 ёсоор объект классынхаа `private` гишүүл рүү шууд хандаж чадахгүй учир удамших классын объект эх классынхаа `public` гишүүл рүү хандаж чадахгүй (Зураг 6.9). Харин функции нь классынхаа `private` гишүүд рүү хандаж болдог тул эх классынхаа `public` гишүүд рүү удамших классын функцизэс хандана.



Зураг 6.8: *Private удамших*

Эх класс `private` горимоор удамших тохиолдолд түүний `private` гишүүд огт удамшихгүй. Ингэхлээр, удамших классын гишүүд эх классынхаа `private` гишүүдэд хандаж чадахгүй.

Эх класс `private` горимоор удамших үед түүний `protected` гишүүл удамших классын `private` гишүүд болох ба ийм гишүүл рүү удамших классын гишүүн функцизэс хандаж чадна.

`Private` удамшлын хандалтын горим болон харагдах байдлыг Хүснэгт 6.2 -3-т үзүүлсэн шигээр нэгтгэн бичиж болно.

Хүснэгт 6.3-т дараах тайлбарыг хийж болно. Доор үзүүлсэн шигээр тодорхойлогдох

```
class BB
{
private:
    int aa ;
protected:
    int bb ;
public:
    int cc ;

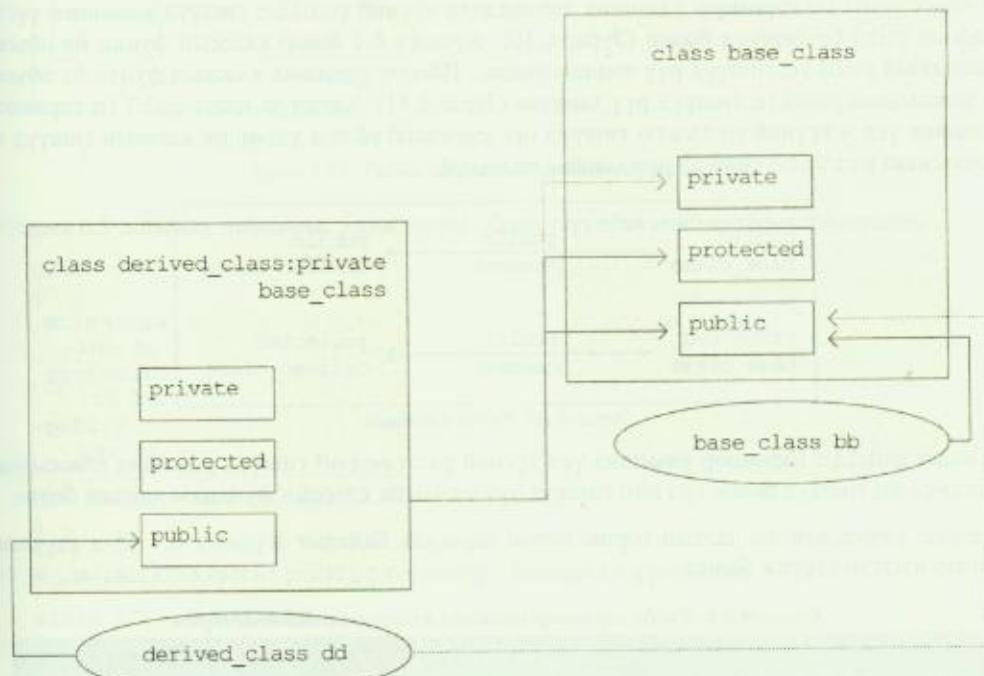
} ;
```

BB эх классаас доорх DD классыг private горимоор удамшуулж үүсгэн.

```
class DD : private BB
{
    ...
}
```

*Хүснэгт 6.2: Private горимоор удамших классын хандалтын горим*

	Хандалт	Эх классын өгөгдөл	Функц.
Private горимоор удамших классын объект	чадахгүй чадахгүй чадахгүй	private protected public	private protected public
private горимоор удамших классын private, public, protected тишигүүн функци	чадахгүй чадна чадна	private protected public	private protected public



*Зураг 6.9: Private удамшилын хандалтын хамаарал*

*Хүснэгт 6.3: Private горимоор удамших классын харагдах байдал*

Эх классын харагдах байдал	Private горимоор удамших классын хувьд харагдах байдал
private protected Public	удамшихгүй private private

Шинээр үүсэх DD класс нь доор үзүүлсэн шиг бүтэктэй болох ба улмаар өөр классын эх класс болж чадна.

```

class DD
{
    private:
        int bb ;
        int cc ;
}

```

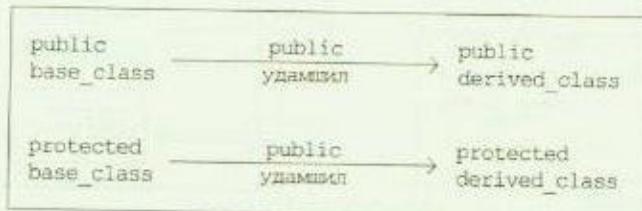
Классыг **public** горимоор удамшуулж үүсгэх бол удамших классын тодорхойлолтыг доо үзүүлсний дагуу хийнэ.

```

class derived_class : public base_class
{
    data_members ;
    member_functions ;
}

```

Эх класс **public** горимоор удамших тохиолдолд түүний **public** гишүүд удамшиж үүсж классын **public** гишүүд болно (Зураг 6.10). Хүснэгт 6.1 ёсоор классын функци ба объект классынхаа **public** гишүүд рүү хандаж чаддаг. Иймээс удамших классын функци ба объект эх классынхаа **public** гишүүд рүү хандана (Зураг 6.11). Харин эх класс **public** горимоор удамших үед түүний **private** гишүүд огт удамшихгүй тул удамших классын гишүүд эх классынхаа **private** гишүүд рүү хандаж чадахгүй.



Зураг 6.10: Public удамшил

Эх класс **public** горимоор удамших үед түүний **protected** гишүүд удамших классынхаа **protected** гишүүд болох тул ийм гишүүд рүү удамших классын функцийн эх хандаж болно.

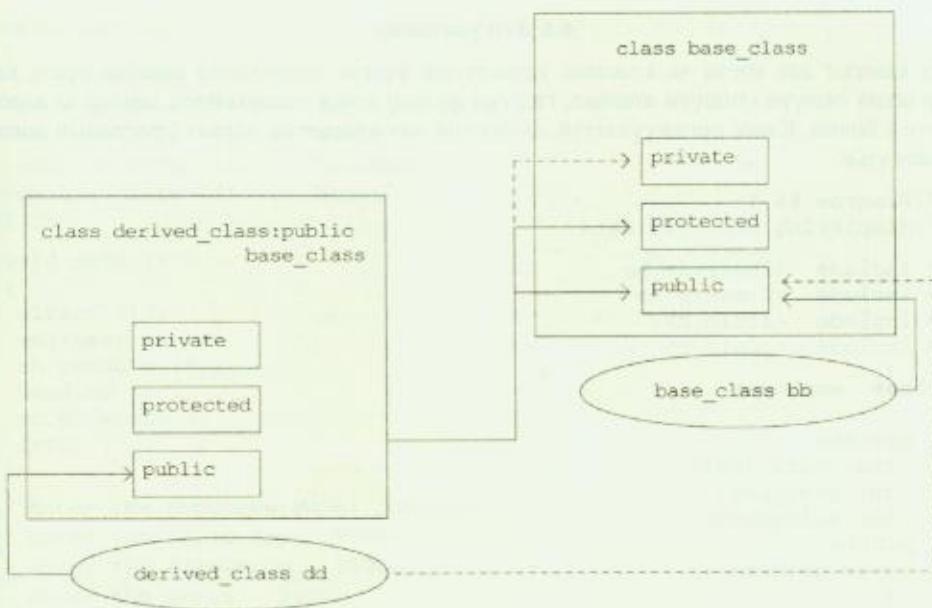
**Public** удамшлын хандалтын горим болон харагдах байдалыг Хүснэгт 6.4 -5-д үзүүлсэн шигээр нэгтгэн үзүүлж болно.

Хүснэгт 6.4: Public горимоор удамших классын хандалтын горим

	Хандалт	Эх классын	
		өгогдэл	функци
<b>public</b> горимоор удамших классын объект	чадахгүй чадахгүй чадна	<b>private</b> <b>protected</b> <b>public</b>	<b>private</b> <b>protected</b> <b>public</b>
<b>public</b> горимоор удамших классын <b>private</b> , <b>public</b> , <b>protected</b> гишүүн функци	чадахгүй чадна чадна	<b>private</b> <b>protected</b> <b>public</b>	<b>private</b> <b>protected</b> <b>public</b>

Хүснэгт 6.5: Public горимоор удамших классын харагдах байдал

Эх классын харагдах байдал	<b>public</b> горимоор удамших классын харагдах байдал
<b>private</b> <b>protected</b> <b>public</b>	удамшихгүй <b>protected</b> <b>public</b>



Зураг 6.11: Public удамшилын хандалтын хамаарал

Хүснэгт 6.5-д дараах тайлбарыг хийж болно. Доор үзүүлсэн шигээр тодорхойлогдсон

```

class BB
{
    private:
        int aa ;
    protected :
        int bb ;
    public :
        int cc ;
}
;
```

BB эх классаас DD классыг public горимоор удамшуулж үүсгэх үед

```

class DD : public BB
{
}

};
```

DD класс нь доор үзүүлсэн шиг бүтэцтэй болох бөгөөд цаашдаа өөр классын эх класс болж чадна.

```

class DD
{
    protected:
        int bb ;
    public:
        int cc ;
}

};
```

## 6.6 Дан удамшил

Шинэ классыг аль нэгэн эх классаас удамшуулж үүсгэх тохиолдолд шинээр үүсэх класс дотор шинэ гишүүн (гишүүн ёгөдөл, гишүүн функци) нэмж тодорхойлох замаар эх классыг өргөтгож болно. Класс өргөжүүлэхтэй холбоотой энэ ойлголтыг дараах программын жишээн дээр авч үзье.

```
//Program #6.1
//displaying employee data

# include <iostream.h>
# include <iomanip.h>
# include <stdio.h>
# include <conio.h>

class employee
{
private :
    char name [20];
    int basicpay;
    int allowance;
public:
    void getdata ()
    {
        cout << "\nEnter the Employee Name: ";
        gets (name);
        cout << "Enter the Basic Pay: ";
        cin >> basicpay ;
        cout << "Enter the Allowance: ";
        cin >> allowance ;
    }

    void showdata ()
    {
        cout << endl;
        cout << setw (20) << name ;
        cout << setw (8)  << basicpay ;
        cout << setw (12) << allowance ;
    }
};

void heading () ;
class engineer : public employee
{
private :
    int bonus ;
public :
    void getdata ()
    {
        employee :: getdata () ;
        cout << "Enter the Bonus: ";
        cin >> bonus ;
    }

    void showdata ()
    {
        employee :: showdata () ;
        cout << setw (8) << bonus ;
    }
};
```

```

void heading ()
{
    cout << endl ;
    cout << setw (20) << "Employee Name" ;
    cout << setw (8)  << "Basic" ;
    cout << setw (12) << "Allowance" ;
    cout << setw (8)  << "Bonus" ;
}

void main ()
{
    clrscr () ;
    engineer en ;
    en.getdata () ;
    heading () ;
    en.showdata () ;
    getch () ;
}

Enter the Employee Name: George
Enter the Basic Pay: 9000
Enter the Allowance: 8000
Enter the Bonus: 3000

Employee Name      Basic      Allowance      Bonus
                George       9000        8000        3000

```

Program #6.1-д main() функцийн эхний clrscr(); функц нь дэлгэц арчаад заагчийг нь дэлгээшний эх рүү шилжүүлнэ. Харин engineer en; командаар удамших engineer классын en объект үүснэ. Дараа нь уг объектын утгыг гараас оруулж өгөхдөө getdata() функцийг en.getdata(); гэж хэрэглэнэ. Энэ функц дотроос эх классын getdata() функцийг дуудахдаа үйлчлэх хүрээний операторыг

employee :: getdata() ;  
гэж хэрэглэнэ. Функц дор жагсаасан 3 мөр мэдээллийг зэлжээр дэлгэц рүү бичихэд харгалзах утгыг гараас оруулна.

```

Enter the Employee Name: George
Enter the Basic Pay: 9000
Enter the Allowance: 8000

```

Үүнээс гадна engineer классын getdata() функцийн бусад хоёр командаар ажилчны авах шагналтай холбоотой мэдээллийг дэлгэцдэд гаргаж гараас оруулна.

```
Enter the Bonus: 3000
```

Программын main() функцийн дуудах heading(); функц аль ч класст харьялагдахгүй бөгөөд гарах хүснэгтийн толгойг доор үзүүлсэн шигээр

Employee Name	Basic	Allowance	Bonus
George	9000	8000	3000

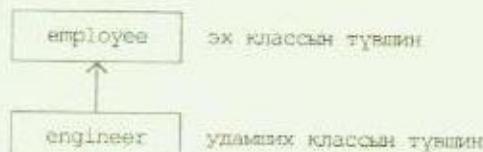
дэлгэц рүү бичнэ. Харин en.showdata(); командаар engineer классын showdata() функц дуудагдана. Түүний эхнд байгаа employee::showdata(); командаар employee классын showdata() функцийг дуудахдаа ( :: ) үйлчлэх хүрээний операторыг хэрэглэх бөгөөд уг функц нь "George 9000 8000" гэсэн мөр мэдээллийг дэлгэц рүү бичнэ. Үүний дараа программын удирдлага engineer классын showdata() функцийн хоёрдахь мөрөнд шилжиж 3000 гэсэн тоог дэлгэцлэнэ. Гэхдээ энэ командын омнох cout объектод endl

командыг хэрэглээгүй тул ажилчны шагналын хэмжээг өмнө хэвлэсэн морийн араас залгаж хувьцэс. Тэгхээр, дэлгэцдэл

George 9000 8000 3000

гэсэн мөр мэдээлэл бичигдэнэ.

Дан буюу энгийн удамшил бол эх классын түвшин, удамших классын түвшин гэсэн 2 түвшинт удамшил юм. Иймд Program #6.1-ийн хувьд Зураг 6.12-т үзүүлсэн шиг (employee) эх классын ба (engineer) удамших классын гэсэн 2 түвшин байна,



Зураг 6.12: Дан удамшил

### 6.7 Харагдах байдал: private, public ба protected

Ажилчдын ийт налинг бодох командин хэсэг Program #6.1-д байхгүй. Иймд ийт цалинг хадгалах private grosspay өгөгдлийг employee классын тодорхойлолтод нэмж болно. Гэвч энэ хувьсагч руу employee классаас удамшиж үүсэх engineer классаас шууд хандаж болохгүй нь эх классын private гишүүд удамшлагтуйгэй холбоотой юм. Тэгэхээр employee классын private grosspay өгөгдөл руу удамших engineer классаас хандлаг байхын тулд яах ёстой вэ? Үүний тулд grosspay гишүүн огогдлийн үйлчлэх хүрээний<sup>38</sup> горимыг public болгож болно. Энэ тохиолдолд програмын бүх функц энэ хувьсагч руу хандах боломжтой болох ч ингэснээр өгөгдөл дадлалтай холбоотой чанар нь алдагдана. Хамгийн зөв шийдэл бол уг гишүүнийг protected шинжтэй болгох юм.

Эх класс private горимоор удамших үед түүний protected гишүүд удамших классынхаа private гишүүд болох тул ийм гишүүд руу удамших классын гишүүн функцийс хандаж болно. Харин удамших классаас удамших классын гишүүн функцийн ийм гишүүд руу хандаж чадлагтуй, учир нь private гишүүд огт удамшлагтуй.

Эх класс public горимоор удамших үед түүний protected гишүүд удамших классын protected гишүүд болох тул ийм гишүүд руу удамших классын функцийс хандаж болно. Бас ийм гишүүд руу удамших классаас удамших классын функци хандаж чадна, учир нь protected гишүүд цааш удамшина.

Өмнөх Program #6.1-д grosspay өгөгдлийг шинээр нэмж бүх гишүүн өгөгдлийн горимыг нь protected болгож бөрчилье. Энэ шинэ програмыг доор үзүүллээ.

```
//Program #6.2
//Finding gross pay

# include <iostream.h>
# include <iomanip.h>
# include <stdio.h>
# include <conio.h>

class employee
{
```

<sup>38</sup> Харагдах байдал нь үйлчлэх хүрээгэй холбоотой ойтголт юм.

```

protected:
    char name [20];
    int basicpay;
    int allowance;
    int grosspay;
public:
    void getdata( )
    {
        cout << "\nEnter the Employee Name: " ;
        gets (name) ;
        cout << "Enter the Basic Pay: " ;
        cin  >> basicpay ;
        cout << "Enter the Allowance: " ;
        cin  >> allowance ;
    }
    void showdata()
    {
        cout << endl ;
        cout << setw (20) << name ;
        cout << setw (8)  << basicpay ;
        cout << setw (12) << allowance ;
    }
};

class engineer : public employee
{
private:
    int bonus;
public:
    void getdata()
    {
        employee :: getdata();
        cout << "Enter the Bonus: " ;
        cin  >> bonus ;
    }
    void showdata()
    {
        grosspay = basicpay + allowance + bonus ;
        employee :: showdata() ;
        cout << setw(8) << bonus;
        cout << setw(8) << grosspay;
    }
};

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Bonus" ;
    cout << setw(8)  << "Gross" ;
}

void main()
{
    clrscr () ;

```

```

engineer en ;
en.getdata () ;

heading () ;
en.showdata () ;
getch () ;
}

Enter the Employee Name: George.
Enter the Basic Pay: 9000.
Enter the Allowance: 8000.
Enter the Bonus: 3000.

Employee Name  Basic  Allowance  Bonus  Gross
George        9000      8000      3000    20000

```

Классын **protected** гишүүд рүү тухайн классын гишүүн функциүүдээс гадна түүнээс удамших дэд классын нь гишүүн функцийн хандаж болдгийг Program #6.2-оос харж болно. Гэтэл ийм **protected** гишүүд рүү классын гадна талаас, тухайлбал main() функцийн хандаж болдогтүй.

Хандалтын горим ба үйлчилх хүрээний харагдах байдалыг Хүснэгт 6.6 -7-д нэгтгэж харууллаа.

*Хүснэгт 6.6: Удамших классын хандалтын горим*

	Хандалт	Эх классын өгөгдөл функци	
<b>private</b> горимоор удамших классын объект	чадахгүй чадахгүй чадахгүй	<b>private</b> <b>protected</b> <b>public</b>	<b>private</b> <b>protected</b> <b>public</b>
<b>public</b> горимоор удамших классын объект	чадахгүй чадахгүй чадна	<b>private</b> <b>protected</b> <b>public</b>	<b>private</b> <b>protected</b> <b>public</b>
<b>private</b> эсвэл <b>public</b> горимоор удамших классын гышүүн функци ( <b>private</b> , <b>public</b> , <b>protected</b> )	чадахгүй чадна чадна	<b>private</b> <b>protected</b> <b>public</b>	<b>private</b> <b>protected</b> <b>public</b>

*Хүснэгт 6.7: Удамших классын харагдах байдал*

Эх классын харагдах байдал	Удамших классын харагдах байдал	
	<b>private</b> горимоор удамших	<b>public</b> горимоор удамших
<b>private</b>	удамшихгүй	удамшихгүй
<b>protected</b>	<b>private</b>	<b>protected</b>
<b>public</b>	<b>private</b>	<b>public</b>

Олон классын эх класс болох класс тодорхойлох тохиолдолд удамших класс нь заавал хандах эрхтэй байх гишүүд **private** биш харин **protected** шинжтэй байвал тохиромжтой.

## 6.8 Олон түвшнит удамшилт

Олон түвшнит удамшилд удамших классын эх класс нь удамших класс байдаг. Жишээ нь, ийм 3 түвшнит удамшлыг доор үзүүлсэн шигээр загварчилж болно.

```

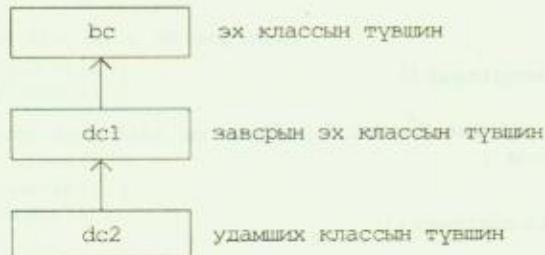
class bc
{
protected:
    data_members ;
public:
    member_functions() ;
};

class dc1 : public bc
{
protected:
    data_members ;
public:
    member_functions() ;
};

class dc2 : public dc1
{
private:
    data_members ;
public:
    member_functions() ;
};

```

Өмнөх жишээнд dc1 нь bc классаас удамших класс юм. Харин dc1 классаас dc2 класс удамшиж үүссэн. Ингэхлээр, dc1 бол dc2 классын хувьд завсрлын эх класс (Зураг 6.13) юм.



Зураг 6.13: Олон түвшинт удамшил

Энэ dc1 нь bc классаас public горимоор удамших класс. Иймд bc классын protected пишүүд dc1 классын protected гишүүд болох тул dc1 классын гишүүд эх классынхаа гишүүд рүү хандаж чадна. Үүний адил dc2 классын гишүүд dc1 классын гишүүд рүү хандаж чадна. Мөн dc2 классын гишүүд bc классын гишүүд рүү хандаж болно, учир нь bc классын protected гишүүд dc1 классын protected гишүүд болж удамшдаг. Ийм турван түвшинт удамшлыг н түвшинт удамшил болгон өргөтгөх боломжтой. Өмнө бичсэн Program #6.2-ыг өөрчилж 3 түвшинт удамшлыг хэрэглэж үзье.

```

//Program #6.3
//finding the gross pay by multi-level inheritance

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>
class employee
{
protected:
    char name[20];

```

```

int basicpay;
int allowance;
int grosspay;
public:
    void getdataemployee()
    {
        cout << "\nEnter the Employee Name: ";
        gets (name) ;
        cout << "Enter the Basic Pay: ";
        cin  >> basicpay ;
        cout << "Enter the Allowance: ";
        cin  >> allowance ;
    }
    void showdataemployee()
    {
        grosspay = basicpay + allowance ;
        cout << endl ;
        cout << setw(20) << name ;
        cout << setw(7)   << basicpay ;
        cout << setw(11) << allowance ;
    }
};

class engineer : public employee
{
protected:
    int bonus ;
public:
    void getdataengineer()
    {
        cout << "Enter the Bonus: ";
        cin  >> bonus ;
    }
    void showdataengineer()
    {
        grosspay += bonus ;
        cout << setw(7)   << bonus ;
        cout << setw(15) << grosspay ;
    }
};

class manager : public engineer
{
private:
    int travelexpense ;
public:
    void getdatamanager()
    {
        cout << "Enter the Travel expense: ";
        cin  >> travelexpense ;
    }
    void showdatamanager()
    {
        grosspay += bonus + travelexpense ;
        cout << setw(7) << bonus ;
        cout << setw(8) << travelexpense ;
    }
};

```

```

        cout << setw(7) << grosspay ;
    }
};

void heading()
{
    cout << endl ;
    for (int k=1; k<61; k++)
        cout << "-" ;
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(7) << "Basic" ;
    cout << setw(11) << "Allowance" ;
    cout << setw(7) << "Bonus" ;
    cout << setw(8) << "Travel" ;
    cout << setw(7) << "Gross" ;
    cout << endl ;
    for (k=1; k<61; k++)
        cout << "-" ;
    cout << endl ;
}

void main()
{
    clrscr() ;
    engineer en ;
    manager mn ;
    cout << "Enter the data on engineer: \n" ;
    en.getdataemployee() ;
    en.getdataengineer() ;

    cout << "\nEnter the data on manager: \n" ;
    mn.getdataemployee() ;
    mn.getdataengineer() ;
    mn.getdatamanager() ;
    heading() ;
    en.showdataemployee() ;
    en.showdataengineer() ;
    mn.showdataemployee() ;
    mn.showdatamanager() ;
    getch();
}

```

 Enter the data on engineer:

Enter the Employee Name: George..

Enter the Basic Pay: 9000..

Enter the Allowances: 8000..

Enter the Bonus: 3000..

Enter the data on manager:

Enter the Employee Name: Madan Mohan..

Enter the Basic Pay: 7000..

Enter the Allowances: 6000..

Enter the Bonus: 2000..

Enter the Travel expense: 4000..

Employee Name	Basic	Allowance	Bonus	Travel	Gross
George	9000	8000	3000		20000
Madan Mohan	7000	6000	2000	4000	19000

Program #6.3-т engineer классыг employee классаас удамшуулж үүсгэдэг. Мөн engineer классаас manager классыг үүсгэнэ. employee классын name, basicpay, allowance, grossрау зэрэг өгөгдөл protected шинжтэй. Иймээс эдгээр гишүүд рүү employee, engineer ба manager классуудын функцийн хандаж болно. Бас engineer классын bonus гишүүн protected шинжтэй учраас түүнд engineer, manager хоёр классын гишүүн функцийн хандаж чадна. Харин manager классын travalexpense гишүүн өгөгдөл private шинжтэй тул түүн рүү зөвхөн manager классын гишүүн функцийн хандана. Харин шууд ба шууд бусаар удамших классуудын нь гишүүд болох bonus ба travalexpense рүү employee классаас хандаж чадахгүй.

Омнөх Program #6.3-ын main() функцийн engineer ен; командаар engineer классын ен объектыг, manager классын таан объектыг manager ин; гэсэн командаар тус тус байгуулна.

Функцийн тавдахь моронд байгаа en.getDataemployee(); командин үр дүнг доор үзүүлснээр дүрсэлж болно.



```
Enter the Employee Name: George.
Enter the Basic Pay: 9000.
Enter the Allowances: 8000.
```

Ажилчны авах шагналыг гарыас оруулахыг сануулсан "Enter the Bonus:" мэдээллийн зургаалахь морийн en.getDataengineer(); командаар дэлгэцэд бичиж улмаар гарыас оруулах зүйлийг ушиж авна.

main() функцийн 8, 9 ба 10 дахь мөрүүдийн командин үр дүнг доор үзүүлсэн шигэр дүрсэлж болно.



```
Enter the Employee Name: Madan Mohan
Enter the Basic Pay: 7000.
Enter the Allowances: 6000.
Enter the Bonus: 2000.
Enter the Travel expense: 4000.
```

Дээр үзүүлсэн үр дүнгийн хүснэгтийг main() функцийн бусад команд хэвлэдэг.

manager классын объектоос engineer болон employee классуудын функцийн рүү хандана. Мөн manager классаас engineer болон employee хоёр классын protected өгөгдэл хандаж болно.

## 6.9 Олон-нэг буюу нийлмэл удамшил

Нийлмэл удамшлын үед олон классаас нэг класс үүснэ. Удамших класс бүх эх классынхаа боломжит гишүүдийг овален авахаас гадна оорийн гэсэн гишүүдтэй байж болно. Нийлмэлээр удамших классын тодорхойлолтод түүний изрийн араас залгаж бичих (: ) давхар цэгийн дараагаар бүх эх классыг нь удамших горимын нь хамт таслалаар зааглаж бичнэ. Удамших классын объект эх классынхаа функцийг шууд дуудаж хэрэглэх боломж бий. Нийлмэл удамшил хэрхэн тодорхойлж болохыг дараах жишээнд үзүүлэх.

```

class A
{
public:
    void show(void) ;
};

class B
{
public:
    void show(void) ;
};

class C : public A, public B
{
};

```

Нийлмэл удамшлын хувьд олон эх класс ижил нэртэй функтгэй байвал үүсч болох хоёрдмол байдлыг дээр тодорхойлсон A, B, C гэсэн турван классыг хэрэглэсэн жишээ дээр авч үзье.

```

void main()
{
    C objC ;
    //objC.show(); //ед аль эх классын show() функц дуудагдах
    //нь тодорхойгүй тул хоёрдмол байдал үүсч алдаа гарна.
    objC.A :: show(); //A классын show() функц дуудагдана.
    objC.B :: show(); //B классын show() функц дуудагдана.
    getch();
}

```

Бүх эх класс нь ижил нэртэй функтгэй байвал ийм функцийг удамших объектоор нь дамжуулж objC.show(); шигээр шууд дуудах боломжгүй, харин тухайн функцийн үндсэн классаар нь дамжуулж дуудаж болохыг доорх жишээнээс харж болно.

```

objC.A :: show() ;
objC.B :: show() ;

```

### 6.10 Удамших, удамшуулах үйлдлийн давуу тал

Эх классыг өөрчлөхийн оронд түүнээс удамших шинэ класс үүсгэх замаар эх классет хамаатай гишүүн өгөгдөл нэмэх боломжийн талаар өмнө үзсэн билээ. Ингэж эх классыг өөрчлөхгүйгээр түүнээс удамших шинэ класс үүсгэж хэрэглэх нь доор жагсаасан шиг сайн талтай.

- Эх классаас шинэ класс удамшуулах замаар эх классын бүх шинжийг овлох огогдлийн шигээ торол үүсгэж болох ба ингэснээр эх классын гишүүн функцийг өөрчлөх шаардлагагүй болно.
- Ихэнх програмын хувьд эх классынхаа шинжийг өвлюх олон класс хэрэгтэй болох ба гэлээр классыг нэг эх классаас үүсгэж болно.
- Эх классыг огогдлийн срэхний бүтэц хэлбэрээр тодорхойлоод түүнийг бодитойгоор хэрэглэх олон дэд классыг үүсгэж болно.

### 6.11 Хийсвэр функи

Нэг эх классаас олон класс удамшуулан үүсгэж тэдгээрийн b, d1, d2, ба d3 гэсэн дөрвөн объектыг доор үзүүлсэн шигээр харгалзуулан байгуулж болно.

```

class base_class
{
};

```

```

class derived_classOne : public base_class
{
};

class derived_classTwo : public base_class
{
};

class derived_classThree : public base_class
{
};

base_class b ;
derived_classOne d1 ;
derived_classTwo d2 ;
derived_classThree d3 ;

```

Эх классын объектын хаяг хадгалах pptr хаяган хувьсагч тодорхойлж түүнд омно байгуулсан b объектын хаягийг

```
base_class *pptr = &b;
```

Гэж хадгална. Ийм хаяган хувьсагч нь эх классан торлийнх ч түүнд удамших классын объектын хаягийг бас хадгалж болно.

```

pptr = &d1 ;
pptr = &d2 ;
pptr = &d3 ;

```

Эх болон удамших классуудын зарим функцийг ижил нэртэй, оөрөөр хэлбэл эх классын зарим функцийг удамших класс дотор нь дахин тодорхойлсон гэж үзье. Энэ тохиолдолд эх классын объектын хувьд классын нь харгалзах функцийг, удамших классын объектын хувьд классын нь харгалзах функцийг дуудагдах ёстай.

```

b.show();           // base_class классын show() функцийг
d1.show();          // derived_classOne классын show() функцийг
d2.show();          // derived_classTwo классын show() функцийг
d3.show();          // derived_classThree классын show() функцийг

```

Гишүүн функцийг объектон хаяган хувьсагчаар дамжуулан дуудаж болох тул ийм командыг доор үзүүлсэн шигээр бичин.

```

base_class b ;
base_class *pptr ;
pptr = &b ;
pptr->show();      // base_class классын show() функцийг b
                    // объектоор дамжуулж дуудах

derived_classOne d1, *ptr1 ;
ptr1 = &d1 ;
ptr1->show();      // derived_classOne классын
                    // show() функцийг d1 объектоор нь дамжуулж дуудах

derived_classTwo d2, *ptr2 ;
ptr2 = &d2 ;
ptr2->show();      // derived_classTwo классын
                    // show() функцийг d2 объектоор нь дамжуулж дуудах

derived_classThree d3, *ptr3 ;
ptr3 = &d3 ;
ptr3->show();      // derived_classThree классын
                    // show() функцийг d3 объектоор нь дамжуулж дуудах

```

Эх классын объектон хаяган хувьсагч руу удамших классын объектын хаягийг хадгалж болох талаар дээр үзсэн билээ. Тэгэхлээр, дараах үйлдлийг хийж болно.

```
base_class b ;
base_class *pptr ;
derived_classOne d1 ;
derived_classTwo d2 ;
derived_classThree d3 ;
pptr = &b ;
pptr->show();           // base_class классын show() функц
                         // b объектын хувьд дуудагдана.
pptr = &d1 ;
pptr->show();           // base_class классын show() функц
                         // d1 объектын хувьд дуудагдана.
pptr=&d2;
pptr->show();           // base_class классын show() функц
                         // d2 объектын хувьд дуудагдана.
pptr = &d3 ;
pptr->show();           // base_class классын show() функц
                         // d3 объектын хувьд дуудагдана.
```

Дээрх жишээнээс үзэхэд pptr хаяган хувьсагч d1, d2 эсвэл d3 гэсэн объектын хаягийг хадгалж байгаа ч pptr->show(); командаар эх классын show() функц дуудагдаг байна. Иймд pptr объектон хаяган хувьсагчид хаяг нь байгаа удамших объектын хувьд оорийн нь show() функцийг яаж дуудах вэ? Үүний тулд удамших классын объектон хаяган хувьсагч зарлаж түүгээр дамжуулан show() функцийг дуудаж үзье.

```
derived_classOne d1, *pptr ;
pptr->show();           //d1 объектоор дамжин удамших
                         //классын show() функц дуудах
```

Бас эх классын объектын хаягийг параметр хэлбэрээр авах C++ функц хэрэглэж болно.

```
void func(base_class *p) ;
void func(base_class *p)
{
    return p->show();
}
```

Ийм функцийг дараах байдлаар хэрэглэхэд эх классын show() бас функц дуудагдлаг.

```
pptr = &b ;
func(pptr) ;

pptr=&d1 ;
func(pptr) ;

pptr=&d2 ;
func(pptr) ;
```

Өмнөх func() функц нь объектын хаяг биш харин заалт авдаг байхаар

```
void func(base_class &b);
void func(base_class &b)
{
    p.show();
}
```

гэж оорчлоод дараагаар нь

```
func(b);
```

```
func(d1);  
func(d2);
```

гэж дуудаж хэрэглэхэд эх классын `show()` функц мөн дуудагдана. Хаяг эсвэл заалт бус харин объект авдаг байхаар `func()` функцийг доор үзүүлсэн шигээр

```
void func(base_class p);  
void func(base_class p)  
{  
    p.show();  
}
```

гэж өөрчилсөн ч гарах үр дүн нь омнох бүх тохиолдлынхой адил байдаг. Ер нь, омнох бүх шийдэл нь удамших классын нь объектыг эх классын объектон хаяган хувьсагчаар хаяглаж болдог ООР шинжийг ашиглаагүй байна. Үүнийг ашиглах замаар асуудлыг шийдэхийн тулд эх классын `show()` функцийг зарлахдаа `virtual` тусгай үгийг

```
virtual void show(void) ;  
гэж хэрэглээн.
```

Энэ бол функцийг хийсвэр (`virtual`) болгох арга юм. Хийсвэр функцийг зөвхөн эх класс дотор зарлана. Эх ба удамших классууд ижил нэртэй функцийтой байвал эх классынхыг нь `virtual` гэж зарласнаар эх ба удамших классын аль нь ч зөв функцээ дуудах боломж бий болдог.

Хийсвэр функц нь эх классын кодыг сольж програмчлахыг удамших класст зөвшөөрдөг. Компайлер авч үзж буй объект нь чухамдаа удамших классынх бол уг объект руу удамших хаяган объектоос биш харин эх хаяган объектоос хандаж байсан ч солих үйлийг зргэлзээгүйгээр хийдэг. Энэ нь хэрэглэгч эх классын талаар мэдэхгүй байсан ч кодыг нь удамших класст солихыг зөвшөөрдөг гэсэн үг юм.

Эх классын байгуулагч функцээс бусад нь хийсвэр байж болно. Энэ тусгай үгийг хэрэглэснээр C++ компайлер олон удамших классын аль нь дуудагдаж байгааг “танъж” зөв функцийг дуудаж чадна. Ингэснээр, өөр оор объектын хувьд ижил нэртэй функцээр оор оор үр дүнд хүрэх боломжтой болно. Үүнийг `polymorphism` буюу олон хэлбэршил гэнэ.

Функцийн параметр ямар иргэн классын объект байх тохиолдолд объектыг утгаар нь биш харин хаягаар эсвэл заалтаар нь дамжуулна. Ийм тохиолдолд объект зөв функцээ дуудаг байхын тулд эх классын харгалзах функц нь `virtual` байх ёстой.

## 6.12 Удамшлын давхардлыг арилгах

Нэг клаасаас удамших 2 классыг доор үзүүлсэн шигээр тодорхойлж болно.

```
class people  
{  
};  
  
class doctor : public people  
{  
};  
  
class rector : public people  
{  
};
```

Харин эдгээр удамших классаас удамших классыг

```

class teacher : public doctor, public rector
{
};


```

тэж тодорхойлно.

`people` классын шинжийг түүнээс удамших `doctor`, `rector` классууд овлож авна. Үүнтэй адил аар, `teacher` нь `doctor` болон `rector` классын шинжийг өвлөнө. Гэтэл анхдагч `people` классын хуулбар нь удамших `teacher` класст хоёр биш нэгт байх ёстой. Үүнийг зохицуулахын тулд `people` классын удамших горимын өмнө `virtual` үгийг хэрэглэнэ.

```

class doctor : virtual public people
{
};

class rector : virtual public people
{
};

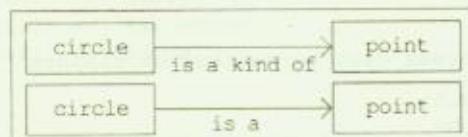

```

`teacher` классын байгуулагч нь объектоо байгуулахдаа оорийн эх класс болох `doctor` ба `rector`, өвөг класс болох `people` гурвыг үүсгэнэ. Харин `doctor` ба `rector` классууд эзлжиндээ өөрсдийн эх классаа дахин дахин үүсгэдэгтүй. Дахиж үүсгэх шаардлагагүй гэдгийг `virtual` үзэр зааж огно.

### 6.13. Давхар класс

Удамшил нь класс хооронд байж болох олон харьцааны нэг хэлбэр юм. Удамшилын үед нэг классаас өөр класс үүснэ. Жишээ нь, `bc` классаас `dc` класс удамшиж үүсэх тохиолдолд шинжээр үүсэх класс нь эх классын зарим шинжийг өвлеж авч болохоос гадна оорийн гэсэн шинжктэй байж болдог. Тэгэхлээр, тэлгээр классын хоёуланд нь байх зарим нэгэн еронхий шинж байж болно. Үндсэн ба удамших класс хооронд байх энэ харьцаа бол "is a kind of" буюу (... -ны) төрлийн юм. Ийм харьцаа, жишээ нь тас шувууны хувьд жигүүртний аймаг болон амьтны аймаг хооронд үүснэ. Тас бол шувуу учраас жигүүртний (класс) аймагт, амьтан учраас амьтны аймагт тус тус багтана. Иймээс тас бол (A condor is a kind of a bird) жигүүртний төрөл, тас бол (A circle is a kind of an animal) амьтны төрөл. Энэ харьцаа нь класс хооронд, объект хооронд байх "еронхийллөөс нарийстал" холбоо юм. Иймд "is a kind of" харьцаа нь "is a" харьцаа юм.

Тойрог нь цэгийн олонлог учир түүнийг "is a" харьцаагаар илэрхийлж (Зураг 6.14) тойрог бол (a circle is a kind of a point) цэгийн төрөл эсвэл тойрог бол (a circle is a point) цэг мөн гэж тодорхойлж болно.



Зураг 6.14: Объект хоорондын "is a" харьцаа

Класс хооронд, объект хооронд байж болох өөр харьцаа бол "has a" буюу "-тай" харьцаа юм. Ийм харьцаа нь нэг классын объект өөр классын объектыг дотроо агуулж байвал, өөрөөр хэлбэл нэг классын объект өөр классын гишүүн өгөгдөл болж байвал үүснэ. Машин нь тээврийн хэрэгслийн нэгэн төрөл бөгөөд машин бүр хөдөлгүүртэй бол машины хийсвэр

загвар болох `car` классыг тодорхойлохдоо, жишээ нь тээврийн хэрэгслийн `vehicle`, хөдөлгүүрийн `engine` классыг тус тус хэрэглэж болно. Ингэхдээ, `car` классыг эдгэр классаас доор үзүүлсэнээр удамшуулж болохгүй.

```
class car : public vehicle, public engine
{
}
```

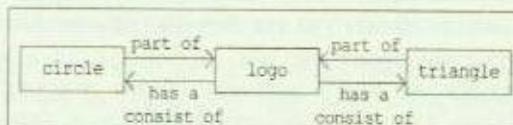
Машин бол хөдөлгүүрийн төрлийн биш, зөвхөн тээврийн хэрэгслийн нэгэн төрөл юм. Ийм машинд хөдөлгүүр байна. Машин дотроо хөдөлгүүртэй байх тул машиныг хөдөлгүүрээс удамшуулах бус харин хөдөлгүүрийг машины доторх объектон өгөгдөл хэлбэрээр зааж өгнө. Ингэхдээр, `car` классыг

```
class car : public vehicle
{
    engine myengine ;
};
```

тэж тодорхойлно. Нэг классын объект өөр классын гишүүн өгөгдөл болох энэхүү "has a" харьцааг класс зөөвөрлөлт (containership) гэх ба ийм классуудыг зөөгүүр класс, давхар класс (nesting of classes) гэнэ.

Объект олон хэсэгтэй байж болно. C++ хэлний зохиомол төрөл болох бүтэц бол үүний сонгодог жишээ юм. Үүнтэй адилaa, тойрог дотор гурвалжинтай лого байх бол "consist of, has a, part of" харьцаа үүсч болно (Зураг 6.15).

Лого нь (the logo consists of two parts: a circle and a triangle) тойрог, гурвалжин гэсэн хоёр хэсгээс тогтолцог. Тойрог, гурвалжин нь (a circle, a triangle are part of the logo) логоны бүрэлдүүлбэрүүд юм. "part-of", "consist of" гэсэн хоёр харьцаа нь класс хооронд, объект хооронд үүсэх бүрдмэл холбою юм. Харин лого нь (the logo has a circle and a triangle) тойрог болон гурвалжинтай гэвэл "has-a" харьцаа нь "part-of" харьцааны урвуу харьцаа юм. Ер нь, объект хоорондын уялдаа холбооны талаар сурал бичгийн нэгдүгээр бүлгээс үзж болно,



Зураг 6.15: Объект хоорондын "has a" харьцаа

Удамшил болон класс зөөгүүр нь зарим тохиолдолд ижил зорилгоор хэрэглэгдэж болохыг Program #6.4-т үзүүлээ.

```
//Program #6.4
//displaying employees data by nesting of classes

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
protected:
    char name[20];
    int basicpay;
```

```

        int allowance;
public:
    void getdata()
    {
        cout << "\nEnter the Employee Name: ";
        gets (name);
        cout << "Enter the Basic Pay: ";
        cin >> basicpay;
        cout << "Enter the Allowances: ";
        cin >> allowance;
    }

    void showdata()
    {
        cout << endl;
        cout << setw(20) << name ;
        cout << setw(8)  << basicpay ;
        cout << setw(12) << allowance ;
    }
};

class engineer
{
private:
    int bonus;
    employee emp;
public:
    void getdata()
    {
        emp.getdata();
        cout << "Enter the Bonus: ";
        cin >> bonus ;
    }

    void showdata()
    {
        emp.showdata() ;
        cout << setw(8) << bonus ;
    }
};

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
    cout << setw(8)  << "Bonus" ;
    cout << endl ;
}

void main()
{
    clrscr() ;
    engineer en;
    cout << "\nEnter the data on engineer:\n";
    en.getdata() ;
    heading() ;
    en.showdata() ;
}

```

```

        getch();
    }

    Enter the data on engineer:
    Enter the Employee Name: George
    Enter the Basic Pay: 9000
    Enter the Allowance: 8000
    Enter the Bonus: 3000

    Employee Name      Basic      Allowance      Bonus
                  George       9000        8000        3000

```

Program #6.4-ийн үр дүн нь Program #6.1-ийнхтэй ижил байна. Энд удамшиг хэрэглэхгүйгээр зөвхөн emp объектыг engineer классын өгөгдэл хэлбэрээр үүсгэсэн ба ут объектоор дамжуулан employee классын гишүүл рүү хандана.

Класс хоорондын, объект хоорондын харыцааг классыг өөр классаас удамшуулж үүсгэв эзэл объектон өгогдооэр хэрэглэх зэрэг асуудлыг шийлэхэд хэрэглэхэд.

#### 6.14. Удамшил ба байгуулагч функци

Удамших классын объект байгуулах үед байгуулагч функци хэрхэн ажилладгийг авч үзэхийн тулд Program #6.1-ийн доор үзүүлснээр өөрчлөн бичье.

```

//Program #6.5
//displaying employee data

# include <iostream.h>
# include <iomanip.h>
# include <stdio.h>
# include <conio.h>
# include <string.h>

class employee
{
protected:
    char name [20];
    int basicpay;
    int allowance;
public:
    void showdata ()
    {
        cout << endl;
        cout << setw (20) << name ;
        cout << setw (8)   << basicpay ;
        cout << setw (12) << allowance ;
    }
    employee() ;
    employee(char n[], int b, int a) ;
};

employee::employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}

```

```

employee::employee(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

void heading () ;

class engineer : public employee
{
public :
    void showdata ( )
    {
        employee :: showdata () ;
    }
    engineer() ;
    engineer(char n[], int b, int a) ;
};

engineer::engineer()
{
    strcpy(name, "I") ;
    basicpay = 1 ;
    allowance = 1 ;
}

engineer::engineer(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

void heading ( )
{
    cout << endl ;
    cout << setw (20) << "Employee Name" ;
    cout << setw (8)  << "Basic" ;
    cout << setw (12) << "Allowance" ;
}

void main ()
{
    clrscr () ;
    engineer en ;
    heading () ;
    en.showdata () ;

    engineer en1("George", 100, 100) ;
    en1.showdata () ;
    getch () ;
}

```

 Employee Name	Basic	Allowance
1	1	1
George	100	100

Удамших классын объектыг engineer en; командаар байгуулах тохиолдолд уг классын анхдагч байгуулагч функцийн en.engineer(); хэлбэрээр дуудагдаж ингэснээр en объектын

name, basicpay, ба allowance зэрэг гишүүдийн гарааны утга бүгд 1 болсныг en.showdata(); командаар мэдэж болно. Харин доорх

```
engineer en1("George",100, 100);
```

ЭСЭН командаар engineer классын en1 объектыг байгуулахад engineer(char n[], int b, int a) байгуулагч доор үзүүлсэн шигээр дуудагдаж

```
en1.engineer("George", 100, 100)
```

иингэснээр en1.name, en1.basicpay ба en1.allowance гишүүдийн гарааны утга харгалзан "George", 100, 100 болсныг en1.showdata(); командын үр дүн харуулна. Гэвч бодит байдал өөр байдаг. Үүнийг тодруулахын тулд дээрх програмд зарим ногхи өөрчлөлт хийх шаардлагатай болно.

```
//Program #6.6
//displaying employee data

# include <iostream.h>
# include <iomanip.h>
# include <stdio.h>
# include <conio.h>
# include <string.h>
class employee
{
protected:
    char name [20];
    int basicpay;
    int allowance;
public:
    void showdata ()
    {
        cout << endl;
        cout << setw (20) << name ;
        cout << setw (8)  << basicpay ;
        cout << setw (12) << allowance ;
    }
    employee() ;
    employee(char n[], int b, int a) ;
};

employee::employee()
{
    cout << "\nCalling base default constructor" ;
    strcpy(name, "0") ;
    basicpay = 0 ;
    allowance = 0 ;
    cout << "\nExiting base default constructor" ;
}

employee::employee(char n[], int b, int a)
{
    cout << "\nCalling base constructor(char n[], int, int)" ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
    cout << "\nExiting base constructor (char n[], int, int)" ;
}

void heading () ;
```

```

class engineer : public employee
{
public :
    void showdata ()
    {
        employee :: showdata () ;
    }
    engineer() ;
    engineer(char n[], int b, int a) ;
};

engineer::engineer()
{
    cout << "\nCalling derived default constructor" ;
    strcpy(name, "I" ) ;
    basicpay = 1 ;
    allowance = 1 ;
    cout << "\nExiting derived default constructor" ;
}

engineer::engineer(char n[], int b, int a)
{
    cout << "\nCalling derived constructor (char n[], int, int)" ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
    cout << "\nExiting derived constructor (char n[], int, int)" ;
}

void heading ()
{
    cout << endl ;
    cout << setw (20) << "Employee Name" ;
    cout << setw (8) << "Basic" ;
    cout << setw (12) << "Allowance" ;
}

void main ()
{
    clrscr () ;
    engineer en ;
    heading () ;
    en.showdata () ;
    engineer en1("George", 100, 100) ;
    en1.showdata () ;
    getch () ;
}

 Calling base default constructor
Exiting base default constructor
Calling derived default constructor
Exiting derived default constructor
    Employee Name      Basic      Allowance
                1            1            1
Calling base default constructor
Exiting base default constructor
Calling derived constructor (char n[], int, int)
Exiting derived constructor (char n[], int, int)
    George           100           100

```

Удамших классын ер объектыг engineer ер; командаар байгуулахад эх юлсын анхдагч байгуулагч, удамших классын анхдагч байгуулагч эзлжлэн дуудагдсаныг дээрх үр дүн харуулж байна.

```
o en.employee() ;
o en.engineer() ;
```

Үүнтэй адилар, engineer en1("George",100,100); командын хувьд

```
o en1.employee() ;
o en1.engineer ("George", 100, 100) ;
```

функциүүд дараалан дуудагдаж, харин en1 объектын хувьд доорх байгуулагч дуудагдаагүй байна.

```
en1.employee("George", 100,100);
```

Эх классын анхдагч байгуулагч нь удамших классын байгуулагч ажиллах бүрд дуудагдах бөгөөд түүний хийсэн зарим ажлыг удамших классын байгуулагч давтан хийдэг. Иймд, жишээ нь эх классын анхдагч байгуулагч нь гишүүн егөгдөл зориулан new оператороор бэлдэх бол уг анхдагч байгуулагч дуудагдах бүрд өмнө бэлдсэн ой нь устгагч функцийн параметртэй байгуулагч болох

```
employee("George", 100, 100)
```

огт дуудагдаагүй болохыг програмын үр дүн харуулж байна.

Удамших классын байгуулагч лээрхээ сөөрөөр ажилладаг. Уг функци эх классынхаа харгалзах байгуулагчийг дуудна. Тэгэхлээр, удамших классын анхдагч байгуулагч эх классынхаа анхдагч байгуулагчийг, удамших классын параметртэй байгуулагч эх классынхаа бичдээс ижил эсвэл түүнээс бага бичдэстэй<sup>39</sup> байгуулагчийг дуудлаг.

Удамших классын байгуулагч эх классынхаа байгуулагчийн хийснийг давтан хийхгүйгээр зөвхөн түүний зогсох газраас үргэлжлүүлэн ажиллах ёстой. Иймийн учир, удамшил хэрэглэх тохиолдолд байгуулагч функцийн тодорхойлолт нийлээд түвэгтэй асуудал болдог байна.

Дээрх шаардлагыг хангахуйц удамших классын байгуулагчийг хэрхэн үүсгэж болохыг авч үзье. Үүний тулд өмнөх програмыг дараах байдлаар өөрчилнө.

```
//Program #6.7
//displaying employee data

# include <iostream.h>
# include <iomanip.h>
# include <stdio.h>
# include <conio.h>
# include <string.h>
class employee
{
protected:
    char name [20];
    int basicpay;
    int allowance;
public:
    void showdata ()
    {
        cout << endl;
```

<sup>39</sup> Signature буюу бичдээс (int, int) нь (int, int, int) бичдэсээс бага гэж үзнэ.

```

        cout << setw (20) << name ;
        cout << setw (8)  << basicpay ;
        cout << setw (12) << allowance ;
    }

    employee() ;
    employee(char n[20], int b, int a) ;
};

employee::employee()
{
    cout << "\nCalling base default constructor" ;
    strcpy(name, "0") ;
    basicpay = 0 ;
    allowance = 0 ;
    cout << "\nExiting base default constructor" ;
}

employee::employee(char n[], int b, int a)
{
    cout << "\nCalling base constructor(char n[], int, int)" ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
    cout << "\nExiting base constructor (char n[], int, int)" ;
}

void heading () ;

class engineer : public employee
{
public :
    void showdata ()
    {
        employee :: showdata () ;
    }
    engineer() ;
    engineer(char n[], int b, int a) ;
};

engineer::engineer() : employee ()
{
    cout << "\nCalling derived default constructor" ;
    cout << "\nExiting derived default constructor" ;
}

engineer::engineer(char n[], int b, int a) : employee(n, b, a)
{
    cout << "\nCalling derived constructor(char n[], int, int)" ;
    cout << "\nExiting derived constructor (char n[], int, int)" ;
}

void heading ()
{
    cout << endl ;
    cout << setw (20) << "Employee Name" ;
    cout << setw (8)  << "Basic" ;
    cout << setw (12) << "Allowance" ;
}

```

```

void main ( )
{
    clrscr () ;
    engineer en ;
    heading () ;
    en.showdata () ;
    engineer en1("George", 100, 100) ;
    en1.showdata () ;
    getch () ;
}


Calling base default constructor
Exiting base default constructor
Calling derived default constructor
Exiting derived default constructor
Employee Name      Basic      Allowance
                0          0
Calling base constructor (char n[], int, int)
Exiting base constructor (char n[], int, int)
Calling derived constructor (char n[], int, int)
Exiting derived constructor (char n[], int, int)
        George       100        100

```

engineer en; команд хийгдэх үсд еп объект үүсэхдээ эх классынхаа анхдагч байгуулагчийг дуудсаныг дээрх үр дүн харуулж байна. Дараагаар нь эх классын анхдагч байгуулагч еп объектыг хэрэглэн удамших классын анхдагч байгуулагчийг идэвхжүүлснээр түүн тотор зааж өгсөн командууд хийгднэ<sup>40</sup>.

Удамших классын энэ байгуулагч эх классынхаа анхдагч байгуулагчийн хийснийг дахин хийхгүй буцааж сэргээхгүй. Эзлжит

```

engineer en1("George", 100, 100);

```

командаар en1.employee("George", 100, 100); команд хийгдэг. Гэхдээ эх классын анхдагч байгуулагч дуудагдахгүй байна. Дараагаар нь en1 объектын хувьд удамших классын (char, int, int) параметртэй байгуулагчийг дуудах ба уг байгуулагч эх классынхаа employee(char, int, int) байгуулагчийн хийснийг дахин хийхгүйгээр түүний зогсох газраас үргэлжлүүлэн ажиллана. Тодруулбал, удамших классын байгуулагч нэмж юу хийх ёстойгоо эх классынхаа байгуулагчийн зогсох газраас үргэлжлүүлж хийнэ.

Удамших классын байгуулагчийг дээр дурдсан дүрмийн дагуу ажиллахаар тодорхойлж болох ба үүнийг дараах байдалаар хийнэ.

1. Удамших классын байгуулагчийн тодорхойлолтын толгойн хэсгийн бичдэсийг доор үзүүлснээр сорчилно.

```

derived_class::derived_class(type1 x1,...,typei xi,...,typen xn) :
base_class(x1,...,xi)
{
}

```

Параметр x1, ..., xi нь эх болон удамших классын хувьд хөбууланд нь ижилхэн байх учир эдгээрийг удамших классаас эх класс руу дамжуулж өгөх ба ингэнснээр эх классын

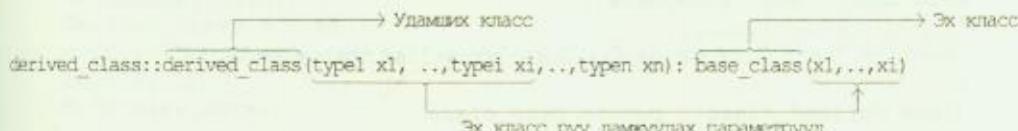
<sup>40</sup> Иймд name, basicpay ба allowance нь en.name, en.basicpay ба en.allowance болно.

байгуулагч хийх ёстой юмаа хийнэ. Дараагаар нь удамших классын байгуулагч  $x_1, \dots, x_n$  параметрын хувьд хийх үйлдээ хийнэ.

- Удамших классын байгуулагчийн нэмж хийх зүйлийг түүний тодорхойлолт дотор зааж огно. Тухайлбал, удамших классын анхдагч байгуулагчийн тодорхойлолтын толгойн хэсгийг

```
derived_class :: derived_class() : base_class()  
{  
    //  
}
```

гэж тодорхойлно. Харин бусад байгуулагчийн хувьд тэдгээрийн тодорхойлолт Зураг 6.16-д үзүүлсэн шиг бүтэцтэй байна.



Зураг 6.16: Удамших классын байгуулагчийн тодорхойлолтын хэсэг

Зураг 6.16-д үзүүлсэн шигээр тодорхойлох байгуулагч нь  $(x_1, \dots, x_i)$  параметрийг эх классын байгуулагч руу дамжуулж ингэснээр эх классын байгуулагч  $(x_1, \dots, x_i)$  аргументтэйгээр эхэлж дуудагдана. Дараагаар нь удамших классын байгуулагч нэмж хийх зүйлээ  $(x_1+1, \dots, x_n)$  параметрийн хувьд хийдэг.

### 6.15 Удамшил ба устгагч функци

Нэмэлт ажил хийлгэхгүй бол устгагч функцийг програмд тодорхойлох шаардлага байдагтүй ба ийм тохиолдолд анхдагч устгагчийг систем оороо үүсгэдэг. Удамших классын объект устахад эхлээд удамших классын устгагч, дараагаар нь эх классын устгагч тус тус дуудагдана.

### 6.16 derived\_class (const base\_class &) байгуулагч

Эх болон удамших классын объектууд адил гишүүн огогдолтэй байсан ч эх классын объектыг удамших классын объект руу утга оноож болдогтүй байна. Гэхдээ энэ асуудлыг `derived_class(const base_class &)` тусгай байгуулагч функцийг удамших классын хувьд нэмж тодорхойлох замаар шийлж болдог. Ийм байгуулагчийг яаж хэрэглэхийг доорх жишээ програм үзүүлнэ.

```
//Program #6.8  
#include <iostream.h>  
  
class base_class  
{  
protected:  
    int xx ;  
    int yy ;  
public:  
    base_class() ;  
    base_class(int x, int y) ;  
    void show_data() ;  
    base_class(const base_class &);  
};
```

```

base_class::base_class()
{
    xx = 10 ;
    yy = 10 ;
}
base_class::base_class(int x, int y)
{
    xx = x ;
    YY = Y ;
}
base_class::base_class(const base_class &b)
{
    *this = b ;
}
void base_class::show_data()
{
    cout << "xx = " << xx << " , " << "yy = " << yy << endl ;
}
class derived_class : public base_class
{
public:
    derived_class() ;
    derived_class(int x, int y) ;
    derived_class(const derived_class &d) ;
    derived_class(const base_class &b) ;
    void display_data(void) ;
};

derived_class::derived_class():base_class()
{
}

derived_class::derived_class(int x, int y) : base_class(x, y)
{
}

derived_class::derived_class(const derived_class &d) : base_class(d)
{
}

derived_class::derived_class(const base_class &b) : base_class(b)
{
}

void derived_class::display_data(void)
{
    cout << "xx = " << xx << " / " << "yy = " << yy << endl ;
}

void main(void)
{
    base_class b1 ;
    base_class b2(50, 50) ;
    derived_class d1 ;
    derived_class d2(100, 100) ;

    cout << endl ;
    cout << "b1: " ;
}

```

```

b1.show_data() ;
cout << "b2: " ;
b2.show_data() ;
cout << "d1: " ;
d1.display_data() ;
cout << "d2: " ;
d2.display_data() ;
b1 = d2 ;
cout << "After the operation b1 = d2:\n" ;
cout << "b1: " ;
b1.show_data() ;
d1 = b2 ;                                //creates no error now
cout << "After the operation d1 = b2:\n" ;
cout << "d1: " ;
d1.display_data() ;
derived_class d = b1 ;                   //creates no error now
cout << "After the initialization operation:\n" ;
cout << "d: " ;
d.display_data() ;
}

```

b1: xx = 10 , yy = 10  
b2: xx = 50 , yy = 50  
d1: xx = 10 / yy = 10  
d2: xx = 100 / yy = 100  
After the operation b1 = d2:  
b1: xx = 100 , yy = 100  
After the operation d1 = b2:  
d1: xx = 50 / yy = 50  
After the initialization operation:  
d: xx = 100 / yy = 100

Дээрх програмд тодорхойлсон эх болон удамших классууд ижил өгөгдлийтэй. Гэхдээ удамших классын байгуулагч функци болох derived\_class(const base\_class &) нь эх болон уламших классууд адилгүй өгөгдлөлтэй үел үр дүнгээ өгнө. Иймд өмнөх програмд доор үзүүлсэн шигээр өөрчлөлт хийе.

```

//Program #6.9
#include <iostream.h>

class base_class
{
protected:
    int xx ;
    int yy ;
public:
    base_class() ;
    base_class(int x, int y) ;
    void show_data() ;
    base_class(const base_class &);
} ;

base_class::base_class()
{
    xx = 10 ;
    yy = 10 ;
}

```

```

base_class::base_class(int x, int y)
{
    xx = x ;
    yy = y ;
}
base_class::base_class(const base_class &b)
{
    *this = b ;
}
void base_class::show_data()
{
    cout << "xx = " << xx << ", " << "yy = " << yy << endl ;
}
class derived_class : public base_class
{
protected:
    int zz ;
public:
    derived_class() ;
    derived_class(int x, int y, int z) ;
    derived_class(const derived_class &d);
    derived_class(const base_class &b);
    void display_data(void) ;
};
derived_class::derived_class():base_class()
{
}
derived_class::derived_class(int x, int y,int z) : base_class(x, y)
{
    zz = z ;
}
derived_class::derived_class(const derived_class &d) : base_class(d)
{
    zz = d.zz ;
}
derived_class::derived_class(const base_class &b): base_class(b)
{
}
void derived_class::display_data(void)
{
    cout << "xx = " << xx << " / " << "yy = " << yy << " / " << "zz = " << zz << endl ;
}
void main(void)
{
    base_class b1 ;
    base_class b2(50, 50) ;
    derived_class d1 ;
    derived_class d2(100, 100, 100) ;

    cout << "b1: " ;
    b1.show_data() ;
    cout << "b2: " ;
}

```

```

b2.show_data() ;
cout << "d1: " ;
d1.display_data() ;
cout << "d2: " ;
d2.display_data() ;
b1 = d2 ;
cout << "After the operation b1 = d2:\n" ;
cout << "b1: " ;
b1.show_data() ;
d1 = b2 ; //энд алдаа гаражгүй
cout << "After the operation d1 = b2:\n" ;
cout << "d1: " ;
d1.display_data() ;

derived_class d = b1 ; //энд алдаа гаражгүй
cout << "After the initialization operation:\n" ;
cout << "d: " ;
d.display_data() ;
}

b1: xx = 10 , yy = 10
b2: xx = 50 , yy = 50
d1: xx = 10 / yy = 10 / zz = 1
d2: xx = 100 / yy = 100 / zz = 100
After the operation b1 = d2:
b1: xx = 100 , yy = 100
After the operation d1 = b2:
d1: xx = 50 / yy = 50 / zz = 1200
After the initialization operation:
d: xx = 100 / yy = 100 / zz = 862

```

Удамших классын `derived_class(const base_class &)` байгуулагч үнэндээ хуулагч байгуулагч юм. Уг функция эх объектыг удамших объект руу хуулах замаар түүнд гарааны утга онооно. Хэрэв эх классын `b1` объект эхэлж байгуулагдах тохиолдолд

```
derived_class d = b1;
```

командын үед хуулагч байгуулагч `d` объектоор идэвхжиж `b1` объектыг параметр хэлбэрээр гаднаас авна. Тэгэхлээр, дээрх командыг дараах хэлбэрээр бичиж болно.

```
derived_class d ;
d.derived_class(b1) ;
```

Удамших классын объект руу эх классын объектоор `d1=b2`; гэж утга оноохдоо утга оноох операторыг дахин тодорхойлох шаардлагагүй. Энэ ажлыг хуулагч байгуулагч өөрөө хийдэг. Өмнөхтэй ижил төрлийн командын үед C++ компайлер `d1` объектын хувьд хуулагч байгуулагчийг дуудахдаа түүн рүү `b2` объектыг аргумент болгож дамжуулна. Тэгэхлээр, `d1=b2`; командыг `d1.derived_class(b2)`; гэж бичсэнтэй адил юм.

### 6.17 Функция дахин програмчлах

C++ хэлэнд арга дахин тодорхойлох, арга дахин програмчлах гэсэн суурь ойлголтууд байдаг талаар суралцсан бичгийн нэгдүгээр бүлэгт, мөн арга дахин тодорхойлох болон дахин тодорхойлох функцийн тухай суралцсан бичгийн тавдугаар бүлэгт тус тус үзсэн билээ. Иймд энэ бүлэгт зөвхөн арга дахин програмчлах тухай дэлгэрүүлж үзнэ.

Арга дахин програмчлах бол эх классын хийсвэр аргыг дахин програмчилж хэрэглэх удамших классын чадвар юм. Дахин програмчлахад удамших классын арга нь бичдэсийн хувьд эх классынхтай адил байх ёстай. Удамших класс нь эх классын зарим арги байгаагаар нь, заримыг нь дахин програмчлах замаар хэрэглэж болно.

Доорх жишээнд Pet классын speak аргыг түүнээс удамших класс тус бүрөөөрийн болгон дахин програмчилдаг. Ингэснээр эх классын speak аргыг хэрэгсэхгүйгээрөөөрийн аргы хэрэглэх боломжтой болно. Удамших классын арга нь буцах утгын төрөл, ер нь бичдэсээр хэрэгсэхгүй болгох эх классын аргатай нийцж байх ёстай.

```
//Program #6.10
//Demonstrating overriding functions

#include <iostream>
#include <string>
using namespace std; //Line #6

class Pet
{
public:
// Constructors, Destructors
Pet () : weight(1), food("Pet Chow") {}
Pet(int w) : weight(w), food("Pet Chow") {}
Pet(int w, string f) : weight(w), food(f) {}
~Pet() {}

//Accessors
void setWeight(int w)
{
    weight = w;
}

int getWeight()
{
    return weight;
}

void setFood(string f)
{
    food = f;
}

string getFood()
{
    return food;
}

//General methods
void eat();
virtual void speak();
protected:
    int weight;
    string food;
};

void Pet::eat()
{
    cout << "Eating " << food << endl;
}
```

```

void Pet::speak()
{
    cout << "Growl" << endl;
}

class Rat: public Pet
{
public:
    Rat() {}
    Rat(int w) : Pet(w) {}
    Rat(int w, string f) : Pet(w,f) {}
    ~Rat() {}

    //Other methods
    void sicken()
    {
        cout << "Spreading Plague" << endl;
    }

    void speak();
};

void Rat::speak()
{
    cout << "Rat noise" << endl;
}

class Cat: public Pet
{
public:
    Cat() : numberToes(5) {}
    Cat(int w) : Pet(w), numberToes(5) {}
    Cat(int w, string f) : Pet(w,f), numberToes(5) {}
    Cat(int w, string f, int toes) : Pet(w,f), numberToes(toes) {}
    ~Cat() {}

    //Other accessors
    void setNumberToes(int toes)
    {
        numberToes = toes;
    }

    int getNumberToes()
    {
        return numberToes;
    }

    //Other methods
    void speak();
private:
    int numberToes;
};

void Cat::speak()
{
    cout << "Meow" << endl;
}

int main()
{
    Pet aPet;

```

```

    Rat aRat;
    Cat aCat;
    aPet.speak();
    aRat.speak();
    aCat.speak();
    return 0;
}

└─ Crowl
    └─ Rat noise
        Meow

```

Өөр өөр үйлчлэх хүрээтэй зүйлсийг програмд хэрэглэх тохиолдолд нэргэй холбоотой зерчил гардаг бөгөөд үүнийг C++ хэлэнд шийдэхдээ namespace тулхүүр үгийг хэрэглэнэ. Дээрх программын зургаадугаар мөр (Line #6) нь namespace std<sup>41</sup> хэрэглэхийг, толгой файлын доторх зүйлс мөн түүний нэг хэсэг болох ёстойг компайлерт мэдээлдэг.

Тухайн хувьсагч эсвэл ялгац нэр аль классынх болохыг үйлчлэх хүрээний оператороор, жишээ нь Pet () : weight(1), food("Pet Chow") гэж заана. Энд weight, food гэсэн хоёр хувьсагч нь Pet классынх бөгөөд харгалзан 1, "Pet Chow" гэсэн утга авахыг компайлерт хэлж өгно.

#### 6.17.1 Дахин тодорхойлсон функцийг дахин програмчлах

Эх класс ямар нэг дахин тодорхойлсон аргатай бөгөөд тэдгээрийн аль нэгийг нь дэд класст хэрэгсэхгүй болгож дахин програмчилснаар бусад нь далдлагдана. Жишээ нь, Pet классын

```

void speak();
void speak(string s);
void speak(string s, int loudness);

```

зэрэг дахин тодорхойлогдсон функциүүдээс зөвхөн эхнийхийг нь Cat уdamших класст дахин тодорхойлсон гэж үзье. Ингэснээр эх классын speak() хэрэгсэгдэхгүй болж бусад хоёр нь далдлагдана. Иймд cat объект болох aCat2 нь aCat2.speak(); гэж чадах ч

```

aCat2.speak("Hello");
aCat2.speak("Hello", 10);

```

гэж хэрэглэвэл хөрвүүлгийн алдаа гарна. Ер нь, эх класст дахин тодорхойлсон функцийг дахин програмчлах бол ийм функц тус бүрийг дахин програмчлах эсэхэд болгоомжтой хандах хэрэгтэй байдал.

#### 6.17.2 Жинхэнэ хийсвэр функц ба хийсвэр класс

Доорх жишээнд үзүүлсэн шиг классын тодорхойлолт доторх эх загвартаа =0 бүхий функцийг pure abstract function буюу жинхэнэ хийсвэр функц гэнэ. Ийм функцийг класст нь (бүрэн) програмчилж өгөөгүй байдаг тул классаас нь уdamшиж үүсвх класс дотор дахин програмчлах ёстой. Гэхдээ дахин програмчилснаас нь дуудаж болохоор хэмжээнд жинхэнэ хийсвэр функцийг хөгжүүлж болно. Хэдий тийм ч жинхэнэ хийсвэр функц бүхий классын объект байгуулж болдогтуй. Ийм функцтэй классыг хийсвэр класс гэнэ. Үүнийг зарим сурх бичигт abstract base class гэх англи нэрийн товчлолоор нь abc гэж нэршилдэг.

<sup>41</sup> Ерөнхийдөө, namespace нь адил нэргэй зүйлсийн хоёрдамол байдлыг тооцож түүнд байгаа нэр, нэр томъёо, уг тэх зэрэг зүйлийн хам байдлыг хангах үүрэгтэй хийсвэр зөвгүүр муж юм. std нь тухайн мужийн ялгац нэр болохын хувьд түүн доторх зүйлсийн заалт болно.

Хийсвэр классаас удамших классыг зохиомжилж хөгжүүлж объект байгуулж болно. Объект байгуулж болдог классыг `abstract` буюу хийсвэр гэдгийн эсрэгээр `concrete` буюу бодит(ой) класс гэнэ.

```
class MyAbstractClass
{
public:
    virtual void MyVirtualMethod() = 0;
};

class MyConcreteClass : public MyAbstractClass
{
public:
    void MyVirtualMethod()
    {
        //юу хийхийг нь програмчлах
    }
};
```

`MyVirtualMethod` нь хийсвэр функц тул түүний `MyAbstractClass` классын объект байгуулж болохгүй. Харин `MyConcreteClass` классын `MyVirtualMethod` функцийг зарлаж улмаар тодорхойлсон тул уг класс нь бодитой класс юм.

### 6.18 C++ класс тодорхойлоход анхаарах зүйл

Классын аль гишүүн `private`, `public` эсвэл `protected` шинжтэй байхыг шийдэх нь програм хөгжүүлэгчээс туршлага, дадал шаардах асуудал юм. Гэхдээ доор жагсаасан дүрмийг баримталж болно. Тухайлбал,

- Огогдол огт өөрчлөгдөхгүй юм уу өгөгдлийг бусдаас далдлах шаардлагатай бол `private` үгийг хэрэглэх
- Удамших класс нь эх классынхаа огогдолд хандах давуу эрхтэй байх шаардлагатай бол ийм өгөгдлийг `protected` гэж тодотгож заах
- Удамших класст хэрэглэх функцийг `public` гэж тодорхойлох
- Өгөгдлөд хурдтай хандах бол дотоод функц хэрэглэх
- Эх классын функцийг түүнээс удамших класс дотор дахин тодорхойлох бол түүнийг эх классын тодорхойлолт дотор нь `virtual` гэж тодотгох
- Эх классын устгагчийг `virtual` гэж зарлах; Энэ функц `virtual` биш байх тохиолдода эх объектен хаяган хувьсагчар дамжуулж хэрэглэх объект устах үед эх классын устгагч функц дуудагдана. Ингэснээр объектын эх класстай холбогдох хэсэг нь зөвхөн устана. Эх классын устгагчийг `virtual` гэж тодотгож өгснээр устах объектод харгалзах устгагч функц нь бас дуудагдах боломжтой болно.

#### Мэдлэгээ шалгах асуулт

- 6.1. C++ хэлний хувьд удамшил гэж юу болох
- 6.2. Эх класс, удамших класс гэж юу болохыг жишээн дээр тайлбарлах
- 6.3. Удамших классыг хэрхэн тодорхойлох
- 6.4. Удамшлын хэлбэрүүдийг жишээн дээр тайлбарлах
- 6.5. C++ хэлний дан удамшлыг тодорхойлох дүрмийг тайлбарлах
- 6.6. Удамшлын давуу тал юу болох
- 6.7. `private, public` удамших горимуудын ялгаа

- 6.8. `public` горимоор удамших класс, `private` горимоор удамших класс хоорондын ялгаатай байдал
- 6.9. Классын гишүүдийг тодотгоч `protected` тусгай үгийг хэзээ хэрэглэх
- 6.10. Эх классын гишүүдийг `protected` болгосны давуу тал юу болох
- 6.11. Эх классаасаа бага хэмжээтэй удамших класс байж болох эсэх
- 6.12. Эх классаас нь удамших классын гишүүд рүү хандаж болох эсэх
- 6.13. Олон түвшинт удамшлын дүрэм юу болох, ийм удамшлыг хэрэглэх шаардлага хэзээ гардаг
- 6.14. Класс зөөверлөлт гэж юу болох, удамшилаас ялгагдах онцлог нь юу болох
- 6.15. Давхар класс гэдэгт юуг ойлгох

#### Жишээ бодлого

- 6.16. Хадгаламжийн хэмжээ нь 10000 доллараас хэтрэхгүй байх эх класс тодорхойлох:  
Дараа нь эх классын нь хязгаарлалт үйлчлэхгүй байхаар `currentaccount` удамших класс тодорхойлох
- 6.17. Дараах хоёр удамших классын ялгаа юу болох

```
class D1: private B { ... } ;  
class D2: public B { ... } ;
```

- 6.18. `shape` гэдэг эх класс тодорхойлох ба ут класс нь геометр дүрсийн талбайг олоход шаардлагатай давхар нарийвчлалтай хоёр тоог халгалах гишүүдтэй байна. Дараагаар нь эх классаас `triangle`, `rectangle` гэдэг хоёр класс удамшуулж үүсгэнэ. Эх класс нь утгуудаа авах `getdata()`, геометр дүрсийн талбайг олж дэлгэн рүү гаргах `displayarea()` функцийгүйдтэй байна.

- 6.19. Дараах програмыг шинжлээд асуултанд хариулах

```
#include <iostream.h>  
class bclass  
{  
    private:  
        int bprivdata;  
    protected:  
        int bprotdata;  
    public:  
        int bpubldata;  
        void bgetdata()  
        {  
            ... ...  
        }  
};  
class dclass1 : private bclass  
{  
    private:  
        int dclprivdata;  
        void dclgetdata()  
        {  
            ... ...  
        }  
};
```

```

class dclass2 : public dclass1
{
private:
    int dc2privdata;
public:
    void dc2getdata()
    {
        ...
    }
};

void main()
{
    bclass bc;
    dclass1 dc1;
    dclass2 dc2;
}

```

- a) bgetdata() функцээр хандаж болох гишүүн өгөгдлийг олох
- b) ямар гишүүн өгөгдлөр рүү dc1getdata() функцээр хандах
- c) ямар гишүүн өгөгдлөр рүү dc2getdata() функцээр хандаж болох
- d) ямар гишүүн өгөгдлөр рүү bc объектоор шууд хандаж болох
- e) ямар гишүүн өгөгдлөр рүү dc1 объектоор шууд хандаж болох
- f) ямар гишүүн өгөгдлөр рүү dc2 объектоор шууд хандаж болох
- g) ямар гишүүн функц рүү bc объектоор дамжуулж хандаж болох
- h) ямар гишүүн функц рүү dc1 объектоор дамжуулж хандаж болох
- i) ямар гишүүн функц рүү dc2 объектоор дамжуулж хандаж болох
- j) ямар гишүүн функц bpubldata руу хандаж чадах

#### 6.20. Дараах тодорхойлолт

```

class bclass
{
public:
    bclass() ;
    ...
};

class wclass : public bclass
{
public:
    wclass() ;
    ...
};

class xclass : public bclass
{
public:
    xclass() ;
    ...
};

еөгдсөн бол доорх командуудаас аль аль нь алдаатай болохыг олж тайлбарлах
bclass v = *new wclass() ;
wclass *pw = new xclass() ;
xclass *px = new bclass() ;

```

- 6.21. Дээрх 6.20 дугаар бодлогын зөвхөн `bclass` классын хувьд хуулагч байгуулагч, утсаа оноох оператор хөрүг нэмж тодорхойлсон бол дараах командаар ямар ямар байгуулагч функцийг дуудагдахыг олох

```
wclass w1 ;  
wclass w2=w1 ;  
bclass v1=w1 ;
```

- 6.22. Тойрог (`circle`), гурван ёнцөгт (`triangle`), квадрат (`square`) мэтийн геометр дурсүүд үүсгэх шаталсан классуудыг тодорхойлох; Тэдгээрийн хувьд еронхий байх гишүүн егөгдэл, гишүүн функцийг тодорхойлох

#### Програмчлах бодлого

- 6.23. Өмнөх 6.16 дугаар бодлогод тодорхойлсон `currentaccount` классыг хэрэглэх програм бичих

- 6.24. Өмнөх 6.18 дугаар бодлогод тодорхойлох гурван классыг хэрэглэн гурвалжин ба тэгш онцготийн хэмжээг гаднаас авч талбайг нь олж дэлгэшилэх програм бичих

- 6.25. Гаднаас авах хоёр утга нь тэгш онцготийн хувьд түүний хоёр талын урт, гурвалжны хувьд түүний суурь ба өндрийн урт тус тус болох ба тэдгээрийг доор үзүүлсний дагуу хэрэглэнэ.

Тэгш онцготийн талбай =  $x \times y$

Гурвалжны талбай =  $(1/2) \times x \times y$

- 6.26. Өөрийнхөө үйл ажиллагааны тухай ном, аудио хуурцаг худалдаалдаг хэвлэлийн газрын хувьд хэвлэлийн нэр (тэмдгт мор), үнэ (дан болит тоо) хадгалах `public static` гэдэг класс тодорхойлох; Дараагаар нь энэ классаас `book`, `cassette` гэдэг хоёр удамших класс үүсгээд объектуудын нь утгыг `getdata()` функцийэр гарас авч `putdata()` функцийэр дэлгэц рүү гаргах замаар тэдгээр классыг шалгах програм бичих

- 6.27. Өмнөх 6.26 дугаар бодлогод тодорхойлох эх класс дотор болит тоон гурван бүтвэртэй `sales` хүчинтэйг нэмэх; Эдгээр егөгдэл нь сүүлийн гурван сард худалдаалсан номын үнийг хадгалах юм. Худалдаалах номын үнийг гаднаас авах командыг `getdata()` функцийг дотор, дэлгэцэлж харуулах командыг `putdata()` функцийг дотор тус тус нэмж оруулна. Удамших классуудын объект бас өөрсдийн мэдээллийг оруулах, гаргах боломжтой байх ба үүнийг нь шалгах програм бичих

# ДОЛООДУГААР БҮЛЭГ

## КЛАСС БА ОЙН МЕНЕЖМЕНТ

- Объектын хаяг, 207
- Класс ба new, delete оператор, 209
- Хаяган хувьсагчийг класс дотор хэрэглэх, 211
  - Хаяган өгөгдлийн гарааны утгатай холбоотой үүсэх алдаа, 215
  - Хаяган өгөгдлийн утгатай холбоотой үүсэх алдаа, 218
  - Хаяган өгөгдлийн гарааны утгатай холбоотой үүсэх алдааг засах, 221
  - Хаяган өгөгдлийн утгатай холбоотой үүсэх алдааг засах, 224
- this хувьсагч, 226
- Статик өгөгдөл, 228
- Статик функц, 231



## ДОЛООДУГААР БҮЛЭГ

### КЛАСС БА ОЙН МЕНЕЖМЕНТ

Ойн зориулалтын new, delete хоёр операторын талаар “C++ хэлний тухай” бүлэгт судалж програмд хэрэглэж үзсэн. Классын объект динамикаар байгуулахад тэдгээр операторыг бас хэрэглэж болно.

#### 7.1 Объектын хаяг

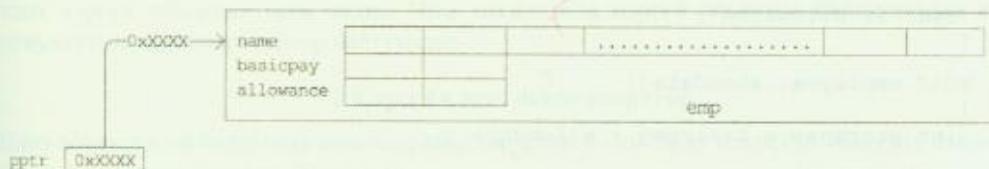
Объектын хаяг хадгалах хаяган хувьсагчийг ерийн хаяган хувьсагчийн адил аар доорх шингээр байгуулна.

```
employee *pptr ;
```

Ийм хаяган хувьсагчийг объектон буюу классан хаяган хувьсагч гэнэ. Байгаа объектын эхлэлийг (&) хаягийн оператороор авч тухайн объекттой класс ижилтэй объектон хаяган хувьсагч руу доор үзүүлсэн шигээр хийдэг.

```
employee emp ;  
pptr = &emp ;
```

Объектон хаяган хувьсагч, объект хоорондын харьцааг Зураг 7.1-д үзүүлснээр дүрсэлж болно.



Зураг 7.1: Объект, хаяган хувьсагч хоорондын харьцаа

Объектон хаяган хувьсагчаас классын функцийн руу 2 аргаар хандана. Тухайлбал,

- Шүүд сонголтын цэг (.) оператор хэрэглэх
  - (`*pptr`).`getdata()` ;
  - (`*pptr`).`showdata()` ;
- Шүүд бус сонголтын сум ( $\rightarrow$ )<sup>42</sup> оператор хэрэглэх
  - `pptr->getdata()` ;
  - `pptr->showdata()` ;

Ер нь, сонголтын цэг болон сум операторын зүүн талд бүтиэн эсвэл классан хаяган хувьсагчийн нэр заавал байх ёстой. Иймд доор жагсаах

```
pptr.getdata() ;  
pptr.showdata() ;
```

командуул дурмийн хувьд алдаатай юм.

Объектон хаяган хувьсагчаар дамжуулан гишүүн функцийг дуудах жишээ програмыг доор үзүүлсэн шигээр бичнэ.

<sup>42</sup> Сум оператор нь цэг операторыг бодвол гишүүнчтэлийн харьцааг илүү тод зааж чадлаг.

```
//Program #7.1
//Finding the gross pay of an employee using
//pointer to an object

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void getdata(void);
    void showdata(void);
};

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);
    cout << "Enter the Basic Pay: ";
    cin >> basicpay;
    cout << "Enter the Allowance: ";
    cin >> allowance;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance;

    cout << endl;
    cout << setw(20) << name;
    cout << setw(8) << basicpay;
    cout << setw(12) << allowance;
    cout.width(8);
    cout << grosspay;
}

void heading()
{
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
}

void main()
{
    clrscr();
    employee emp;
    employee *pptr;
    pptr = & emp;
    pptr->getdata();

    heading();
}
```

```

pptr->showdata() ;
getch() ;
}

Enter the Employee Name: George.
Enter the Basic Pay: 9000.
Enter the Allowance: 2000.

Employee Name    Basic    Allowance   Gross
          George      9000        2000     11000

```

Өмнөх програмын main() функцийн дотор employee классын emp объект болон pptr объектон хаяган хувьсагч байгуулж emp объектын хаягийг pptr хувьсагч руу дараах командаар хийнэ.

```
pptr = &emp ;
```

Объектон хаяган хувьсагчаас классын нь функцийг дуудаадаа сонголтын цэг эсвэл сум операторыг дараах байдлаар хэрэглэнэ.

```

pptr->getdata() ;
pptr->showdata() ;
(*pptr).getdata() ;
(*pptr).showdata() ;

```

Объектон хаяган хувьсагчийн утга нь дээр үзүүлсэн шигээр тодорхой нэр бүхий объектоос гадна нэргүй объектыг зааж чадна. Ийм тохиолдолд нэргүй объектыг байгуулахдаа ойн зориулалтын new оператороор байгуулдаг.

## 7.2 Класс ба new, delete оператор

Шинэ объектод ой бэлдэхэлээ new оператор хэрэглэж болох ба уг оператор объектод бэлдээн ойн эхэл хаягийг буцаана. Жишээ нь,

```
employee *pptr = new employee ;
```

Энэ команд нь нэргүй, тодорхой гарсаны утгагүй объектод ой бэлдээд объектын эхлэлийн хаягийг Зураг 7.1-д үзүүлсэн шигээр pptr хаяган хувьсагч руу хийдэг. Мөн доор үзүүлсэн шигээр нэргүй объект үүсгэж түүнд гарсаны утга оноож болно.

```
employee *pptr = new employee("George", 9000, 0);
```

Ийм нэргүй объектын өгөгдөл рүү хандаадаа сонголтын цэг эсвэл сум оператор хэрэглэнэ.

```

pptr->showdata() ;
(*pptr).showdata() ;

```

new оператороор үүсгэсэн объектын эзэмшилд байгаа ойг delete оператороор чөлөөлнө.

```
delete pptr ;
```

Дээрх хоёр операторын хэрэглээг хамтад нь Program #7.2-т үзүүлснээр програмчилна.

```

//Program #7.2
//Finding the gross pay of an employee using
//pointer to an object

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <string.h>

```

```

#include <conio.h>
class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void getdata(void);
    employee();
    ~employee();
    employee(char *, int, int);
    void showdata(void);
};

employee::employee()
{
}

employee::~employee()
{
}

employee::employee(char *n, int b, int a)
{
    strcpy(name, n);
    basicpay = b;
    allowance = a;
}

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);
    cout << "Enter the Basic Pay: ";
    cin >> basicpay;
    cout << "Enter the Allowance: ";
    cin >> allowance;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance;
    cout << endl;
    cout << setw(20) << name;
    cout << setw(8) << basicpay;
    cout << setw(12) << allowance;
    cout.width(8);
    cout << grosspay;
}

void heading()
{
    cout << endl;
    cout << setw(20) << "Employee Name";
    cout << setw(8) << "Basic";
    cout << setw(12) << "Allowance";
    cout << setw(8) << "Gross";
}

```

```

void main()
{
    clrscr();
    employee *pptr;
    pptr = new employee();
    pptr->getdata();
    employee *pptr1 = new employee("Bill", 8000, 5000);
    heading();
    pptr->showdata();
    pptr1->showdata();
    delete pptr;
    delete pptr1;
    getch();
}

```

Enter the Employee Name: George  
 Enter the Basic Pay: 9000.  
 Enter the Allowance: 2000.  

Employee Name	Basic	Allowance	Gross
George	9000	2000	1100
Bill	8000	5000	13000

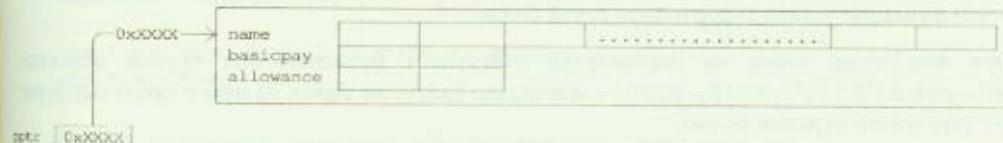
Program #7.2 нь Program #7.1-ийнхтэй ижил үр дүнг огно. Энэ програмын main() функцийн

```

employee *pptr;
pptr=new employee();

```

хөр команд дараалан хийгдсэнээр иэргүй, гарааны утгагүй объектод ой бэлдэж эхлэлийн хамгийг нь pptr хаяган хувьсагчид хадгална (Зураг 7.2).

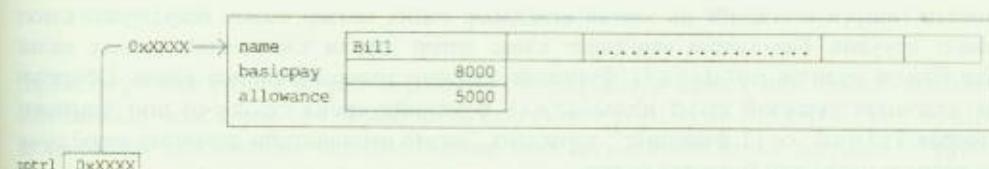


Зураг 7.2: Иэргүй объект, хаяган хувьсагч хоорондын харыцаа

Харин доорх

```
employee *pptr1=new employee("Bill", 8000, 5000);
```

команд нь иэргүй объектод зориулан ой бэлдээд түүнд гарааны утга онооно (Зураг 7.3).



Зураг 7.3: Гарааны утгатай иэргүй объект, хаяган хувьсагч хоорондын харыцаа

### 7.3 Хаяган хувьсагчийг класс дотор хэрэглэх

Нээн төрлийн дараалсан олон егөгдөл хэрэглэх ойг бэлдэхэд хамгийн сайн тохирдог бүтэц бол хүснэгт юм. Дараалсан 1000 бүхэл тоо эсвэл 20 тэмдэгт хадгалах хүснэгтийг

```

int num [1000] ;
char name [20] ;

гэж үүсгэнэ. Бүтвэрийн тоо, хэмжээ нь урдаас мэдэгдэж байгаа хүснэгтийг дээрх аргаар
байгуулж болдог нь хүснэгтийн сүл талын нэг юм. Тэгэхлээр,
    cin >> size;
    int num[size];

```

шиг командаар хүснэгт байгуулж болохгүй. Иймийн учир, employee класс дотор name
өгөглийн тодорхойлохдоо түүний хэмжээг тодорхой заах шаардлагатай болдог.

```

class employee
{
private:
    char name[20];
}
;
```

Шинээр үүсгэх employee терлийн объект бүрийн name өгөгдөл авсан есөөс уртгүй нэр
хадгалах ёстой. Ийм тогтсон хэмжээ нь зарим тохиолдолд багадаж болох тул ямар ч урт
изэрнд хангалттай хүрэлцэхээр name өгөглийн хэмжээг бас тодорхойлж болно. Энэ
тохиолдолд ихэнх объектын хувьд олон байт ой ашиглагдахгүй үлдэж ойг хэмнэлтгүй
хэрэглэхэд хүргэх талтай. Ийм байдлаас гарах хамгийн зөв арга бол хаяган хувьсагч
хэрэглэж объект бүрд шаардагдах ойг new оператороор бэлдээд хэрэглэж дуусаад delete
оператороор чөлөөлөх юм.

Классын тодорхойлолт дотор зарим гишүүн өгөглийн оронд хаяган хувьсагч хэрэглэж
тухайн өгөгдөл шаардагдах ойг new оператороор бэлдэхэд түүнийг програмчлахтай
холбоотойгоор шийдэх асуудал гардаг. Тухайлбал, байгуулагч функцийн дотор new операторыг,
устгач функцийн дотор delete операторыг хэрэглэх шаардлагатай. Тэгэхлээр, байгуулагч,
устгач функцийн заавал тодорхойлох ёстой болно.

Дээрх employee класс нь параметртэй байгуулагч функцийтэй бол түүний объектыг
employee e("Bill", 9000, 2000); командаар байгуулж болох ба ийм e объектыг Зураг
7.4-т үзүүлснээр дүрсэлж болно.

name	Bill	..	..	
basicpay	9000			
allowance	2000			

Зураг 7.4: employee төрлийн e объект

Хүснэгтэн гишүүн өгөглийг нь хаяган өгөгдлөөр солих замаар омнох employee класс
өөрчлөлт оруулна. Бас хаяган өгөглийг класс дотор хэрхэн хэрэглэхийг судлах ажлыг
хялбар болгох үүднээс getdata() функцийтэй классын тодорхойлолтоос хасна. Объектын
утыг дэлгэштхүүрэгтэй void showdata() функцийн оронд cout<<е шиг команда
зовшоорөх friend <<() функцийг<sup>43</sup> хэрэглэнэ. Эдгээр өөрчлөлтийн дараагаар employee
класс доор үзүүлсэн шиг бүтэцтэй болно.

```

class employee
{
private:

```

<sup>43</sup> friend буюу ийз функцийн талаар "Нийз функци, оператор дахин тодорхойлох" бүлгээс үзж болно.

```

    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void);
    ~employee(void);
    employee(char *n, int b, int a);
    friend ostream & operator <<(ostream &os, employee &e);
} ;

```

Классын тодорхойлолт доторх name хаяган хувьсагч нь new оператороор бэлдэх ойн хаягийг хадгалах юм. Тэгэхлээр, объектын иэр нь объект дотроо бус, харин түүний гадна талд бэлтгэх ойд хадгалагдана. Ой бэлдэхдээ new операторыг байгуулагч дотор хэрэглэнэ. Объект устгах үед түүний эзэмшиж байсан ойг устгагч дотроос delete оператороор чөлөөлийн. Эдгээр гишүүн функцийг дараах байдлаар бөрчлон бичье.

```

employee::employee()
{
    name = new char[20] ;
    strcpy(name,"");
    basicpay = 0 ;
    allowance = 0 ;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name,n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye, " << name << endl ;
    delete name;
}

friend ostream & operator <<(ostream &os, employee &e)
{
    int grosspay = e.basicpay + e.allowance ;
    os << setw(20) << e.name ;
    os << setw(8) << e.basicpay ;
    os << setw(12) << e.allowance ;
    os.width(8);
    os << grosspay ;
    return (os) ;
}

```

Параметргүй байгуулагч функция employee(void) дотор бэлдэх ойн хэмжээ тогтмол 20 байт байхаар зааж өгсөн бол параметртэй байгуулагч employee(char \*n, int b, int a) дотор бэлдэх ойн хэмжээг олохдоо гаднаас авах аргументийн уртыг strlen() функцийн тооцуулж авна. Гэхдээ strlen() функция нь цуваа тэмдэгтийн уртыг олохдоо түүний тогсгол заагчийг хасж тооцох учир уг уртыг нэгээр нэмж new оператороор бэлдэх ойн хэмжээт (new char[strlen(n)+1]) гэж зааж өгнө. Энэ нэмэлт байтыг цувааны тогсгол заагчид хэрэглэнэ.

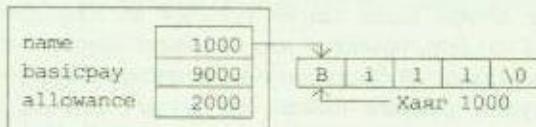
Дээрх маягаар тодорхойлогдох классын объектыг З янзаар үүсгэнэ. Тухайлбал,

```

o employee e("Bill", 9000, 8000);
o char *name = "Bill";
  employee(name, 9000, 8000) ;
o char *name[] = "Bill";
  employee(name, 9000, 8000);

```

Эдгээр аргын аль нэгээр байгуулж болох ё объектын ерөнхий бүтцийг Зураг 7.5-д узүүлсэн шигээр дүрсэлж болно.



Зураг 7.5: Гишүүн огогдол нь хаяган хувьсагч бүхий ё объектын бүтэц

Дээр үзүүлсэн employee классын тодорхойлолт, түүний функцийдийн тодорхойлолтыг нэгтгэн дараах байдлаар програмл хэрэглэнэ.

```

//Program #7.3

#include <iostream.h>
#include <string.h>
#include <conio.h>

class employee
{
private:
    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void);
    ~employee(void);
    employee(char *n, int b, int a);
    friend ostream & operator << (ostream &os, employee &e) ;
};

employee::employee()
{
    name = new char[20] ;
    strcpy(name,"") ;
    basicpay = 0 ;
    allowance = 0 ;
}
employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}
employee::~employee()
{
    cout << "\nBye, bye, " << name ;
    delete name;
}

```

```

ostream & operator <<(ostream &os, employee &e)
{
    int grosspay = e.basicpay + e.allowance ;
    os << "\n" ;
    os.width(20) ;
    os << e.name ;
    os.width(8) ;
    os << e.basicpay ;
    os.width(12) ;
    os << e.allowance ;
    os.width(8) ;
    os << grosspay ;
    return (os) ;
}

void main()
{
    clrscr() ;
    employee e1("Bill", 9000, 8000) ;
    cout << e1 ;
    employee e2("George", 9000, 7000) ;
    cout << e2 ;
}

```

Bill 9000 8000 17000  
George 9000 7000 16000

Bye, bye, George  
Bye, bye, Bill

Класс дотор хаяган өгөгдөл хэрэглэх тохиолдолд жинхэнэ өгөгдөл нь объектын гадна талд new оператороор бэлдэх ойд, түүний хаяг нь объектын доторх хаяган огогдэлд хадгалагдаж байна. Ийм өгөгдлөтгий объектын хувьл доор үзүүлсэн шиг веер объектоор гарааны утга эсвэл програмын явцад утга оноох үед зөрчил үүсч болох талтай.

```

employee e1 = e2 ;           //гарааны утга оноох
e1 = e2 ;                   //утга оноох

```

Гарааны утга оноох үйлдэл нь програмын явцад утга оноохоос ялгаатай. Эхнийх нь объект үүсх үед нэг удаа хийгдэнэ. Нэг объектод веер объектоор гарааны утга оноож байх тохиолдолд анхдагч байгуулагч дуудагдахгүй. Үүний оронд гишүүн өгөгдөл бүрийг ээлж дараалуулан бүрэн хуулна. Ингэхдээ утга оноох операторын баруун талд байгаа объектын өгөгдөл бүрийг уг операторын зүүн талд байгаа объектын өгогдөл бүрд харгалзуулан хуулна. Харин програмын явцад утга оноох тохиолдолд байгаа хоёр объект хооронд хуулах үйлдэл хийгдэнэ. Ингэхдээ хоёр объектын гишүүн өгөгдөл бүрийг харгалзуулан хуулдаг.

### 7.3.1 Хаяган өгөгдлийн гарааны утгатай холбоотой үүсх алдаа

Объектод гарааны утга оноох тохиолдолд ямар алдаа гарч болохыг жишээ програм дээр авч үзье.

```

//Program #7.4
#include <iostream.h>
#include <string.h>
#include <conio.h>
class employee
{
private:

```

```

    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void);
    ~employee(void);
    employee(char *n, int b, int a);
    friend ostream & operator <<(ostream &os, employee &e) ;
};

employee::employee()
{
    name = new char[20] ;
    strcpy(name, "Hello, C++") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye, " << name << "\n" ;
    delete name;
}

ostream & operator <<(ostream & os, employee & e)
{
    int grosspay = e.basicpay + e.allowance ;
    os.width(20) ;
    os << e.name ;
    os.width(8) ;
    os << e.basicpay ;
    os.width(12) ;
    os << e.allowance ;
    os.width(8);
    os << grosspay ;
    os << "\n" ;
    return (os) ;
}

void main()
{
    clrscr() ;
    employee e1("Bill", 9000, 8000) ;
    cout << e1 ;
    employee e2("George", 9000, 7000) ;
    cout << e2 ;
    employee e = e1 ;
    cout << e ;
}

```

```

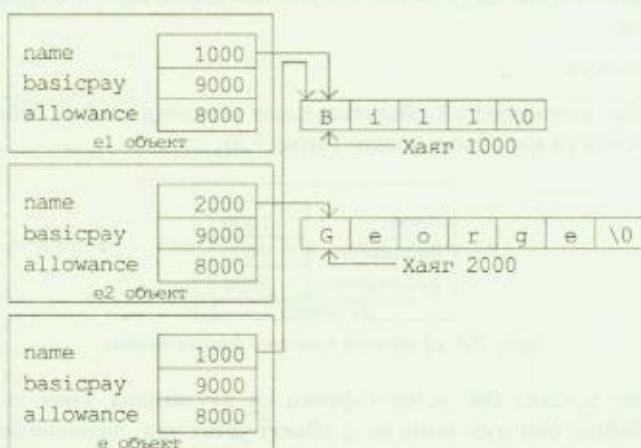
Bill    9000      8000   17000
George  9000      7000   16000
Bill    9000      8000   17000
Bye, bye, Bill
Bye, bye, George
xxxxxNull pointer assignment

```

Дээрх програмын үр дүнг дараах байдалар тайлбарлаж болно. Тухайлбал,

```
employee e1("Bill", 9000, 8000);
```

командаар `e1` объектыг Зураг 7.6-д үзүүлсэн шигээр байгуулна. Ингэхдээ "Bill" өгөглийг объектын гадна `new` оператороор бэлдэх ойд байгуулж уг ойн мужийн эхлэлийг, жишээ нь 1000 гэсэн хаягийг `e1` объектын `name` өгогдолд хийнэ.



Зураг 7.6: Program #7.4-өөр байгуулагдах объектууд; хаягийн утга нь зохиомол

Үүнтэй төстэйгээр, `e2` объектыг

```
employee e2("George", 9000, 7000);
```

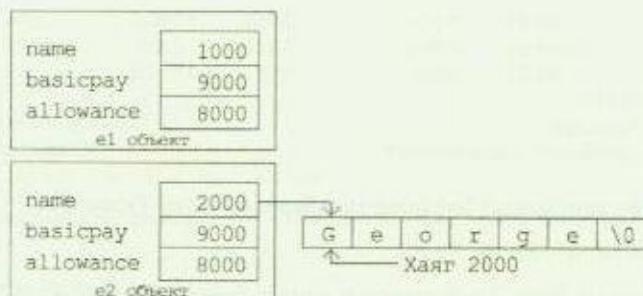
командаар Зураг 7.6-д үзүүлсэн шигээр байгуулна. Ингэхдээ "George" өгөглийг объектын нь гадна талд `new` оператороор бэлдэх ойд байгуулж уг ойн мужийн, жишээ нь 2000 гэсэн эхлэл хаягийг `e2` объектын `name` өгөгдолд хийнэ. Харин `employee e=e1;` командаар `e` объект байгуулж `e1` объектоор гарсан утга онооно. Ингэхдээ `e1` объектын гишүүн бүрийг `e` объектын харгалзах гишүүн рүү хуулах тул энэ хоёр объект ашиглан утгатай болно. Иймд `e, e1` объектууд дундаа 1000 гэсэн дугаартай ойг эзэмшин.

Байгуулагдсан бүх объект нь харьялагдах функци зэвэл програмын ажиллагаа дуусгавар болоход эзлж дараалан устахдаа харгалзах устгач функцийн дууддаг. Ингэхдээ хамгийн сүүлаа байгуулагдсан нь эзлж устана<sup>44</sup>. Тэгэхээр, эхлээд `e` объект устахад холбогдох устгач функци нь дуудагдан дараах мэдээллийг дэлгэц рүү бичнэ.

Bye, bye, Bill

Ингэснээр `e` объектын `name` өгогдолд хаяг нь байгаа ойг чөлөөлнө. Үлдэх хоёр объект нь Зураг 7.7-д үзүүлсэн шиг байдалд байна.

<sup>44</sup> Стек гэх өгөглийн бүтэй ингэж ажилладаг.



Зураг 7.7: e объект устсаны дараах байдал

Байгуулагдсан дарааллынхаа дагуу эхний e объектын дараагаар e2 объект устахад устгач функций нь дуудагдан

Bye, Bye, George

Гэсэн мэдээллийг бас дэлгэшлээд e2 объектын name өгөгдлөд хаяг нь байгаа ойт чөлөөлнө. Үүний дараагаар устаагүй нэг объект үлдэнэ (Зураг 7.8).

name	1000
basicpay	9000
allowance	8000
e1 объект	

Зураг 7.8: e2 объект устсаны дараах байдал

Сүүлийн e1 объект устахад бас устгач функций нь дуудагдана. Гэвч уг объектын name өгөгдлөд хаяг нь байгаа ойн муж ёмне нь e объект устах үед чөлөөлгдсөн байх тул тэнд байгаа тохиолдлын мэдээллийг дэлгэц рүү бичнэ. Мен чөлөөлох зүйл байхгүй тул дараах алдааны мэдээллийг

Null pointer assignment (хоосон хаяг оноолт)

долгэццээд энэст нь e1 объект устана.

### 7.3.2 Хаяган өгөгдлийн утгатай холбоотой үүсэх алдаа

Програм дотроос объектод утга оноох үед ямар алдаа гарч болохыг жишээ програмаар авч үзье.

```
//Program #7.5
#include <iostream.h>
#include <string.h>
#include <conio.h>

class employee
{
private:
    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void) ;
    ~employee(void) ;
    employee(char *n, int b, int a) ;
}
```

```

        friend ostream & operator <<(ostream &os, employee &e) ;
    } ;

employee::employee()
{
    name = new char[20] ;
    strcpy(name, "Hello, C++") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye, " << name << "\n" ;
    delete name;
}

ostream & operator <<(ostream &os, employee &e)
{
    int grosspay = e.basicpay + e.allowance ;
    os.width(20) ;
    os << e.name ;
    os.width(8) ;
    os << e.basicpay ;
    os.width(12) ;
    os << e.allowance ;
    os.width(8) ;
    os << grosspay ;
    os << "\n" ;
    return (os) ;
}

void main()
{
    clrscr() ;
    employee e1("Bill", 9000, 8000) ;
    cout << e1 ;
    employee e2("George", 9000, 7000) ;
    cout << e2 ;
    employee e ;
    e = e1 ;
    cout << e ;
}

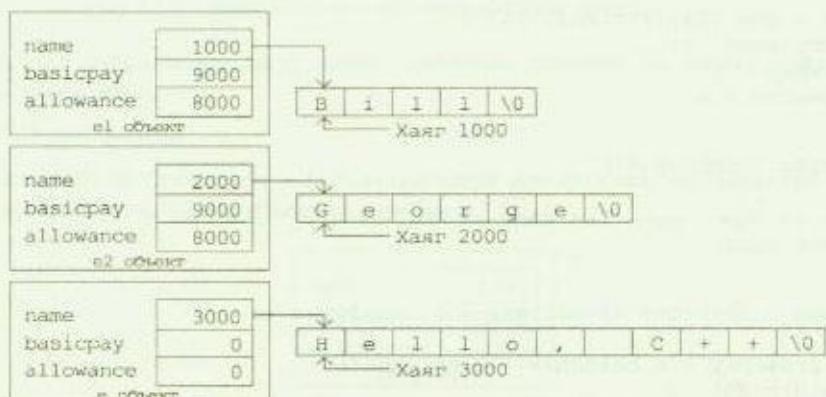
```

 Bill 9000 8000 17000  
 George 9000 7000 16000  
 Bill 9000 8000 17000  
 Bye, bye, Bill  
 Bye, bye, George  
 BillNull pointer assignment

Программын үр дүнг дараах байлаар тайлбарлаж болно. Тухайлбал,

```
employee e1("Bill", 9000, 8000) ;
employee e2("George", 9000, 7000) ;
employee e ;
```

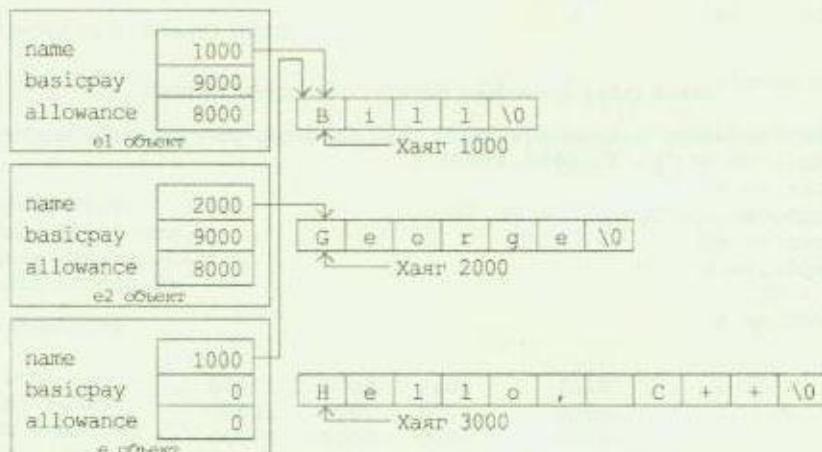
гурван командаар e1, e2, e объектуудыг Зураг 7.9-д үзүүлсэн шигээр байгуулж тэдгээрт гарсаны утга онооно. Гэхдээ e=e1; командаар e1 объектын гишүүн бүр e объектын харгалзах гишүүн рүү хуулагдах тул тэдгээр объектын утга ижил болж дундаа 1000 гэсэн дугаартай ойт эзэмшинэ. Утга оноох энэ үйлдлийн дараагаар объектуудын хамаарал Зураг 7.10-д үзүүлсэн шиг бүтэктэй болно.



Зураг 7.9: Объектоор утга оноох үйлдлийн омнох байдал; хаягийн утгууд зохиомол

Бүх байгуулагдсан объект нь програм дуусгавар болоход ээлж дараалан устахдаа харгалзах устгагч функцийн дуудвна. Ингэхдээ хамгийн сүүлл байгуулагдсан нь эхэлж устана. Тэгэхлээр, эхлээд e объект устахад устгагч функцийн дуудагдан дараах мэдээллийг дэлгэц рүү бичнэ.

Bye, bye, Bill

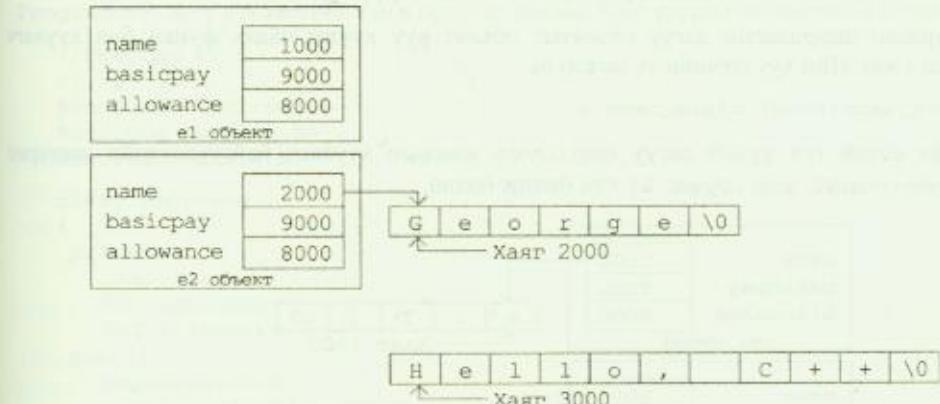


Зураг 7.10: Объектоор утга оноох үйлдлийн дараах байдал;  
хаягийн утгууд зохиомол

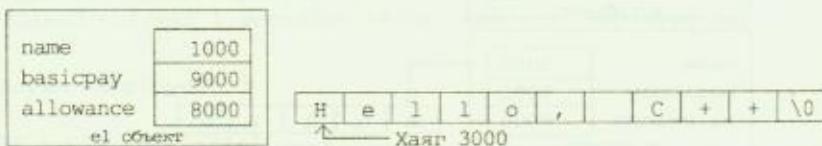
Дараагаар нь е объектын name өгөгдөл хаяг нь байгаа ойг чөлөөлнө. Ингэснээр Зураг 7.11-д үзүүлсэн шигээр хоёр объект үлдэнэ. Ээлжит устах объект нь e2 болох ба харгалзах устгач функци нь дуудагдан

Bye, bye, George

мэдээллийг дэлгэц рүү бичиж e2 объектын name өгөгдолд хаяг нь байгаа ойг чөлөөлнө. Үүний дараагаар устаагүй нэг объект, тухайлбал e1 үлдэнэ (Зураг 7.12).



Зураг 7.11: e объект устсаны дараах байдал; хаягийн утгууд зохиомол

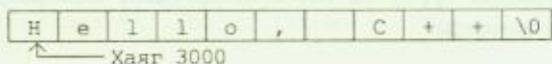


Зураг 7.12: e2 объект устсаны дараах байдал; хаягийн утга зохиомол

Сүүлийн e1 объект устахад харгалзах устгачи функци нь дуудагдана. Гэвч энэ объектын name өгөгдэл хаяг нь байгаа ой омно нь e объект устах үед чөлөөлгэдсэн байх тул тэнд байгаа тохиолдлын мэдээллийг дэлгэц рүү бичнэ. Мөн чөлөөлөх зүйл байхгүй тул дараах алдааны мэдээллийг

Null pointer assignment

дэлгэшийэд энэст нь e1 объект устах боловч Зураг 7.13-д үзүүлсэн зүйл устаагүй үлдэх ба эн нь "эзэнтэй ой" шигээр компьютерийг унтрааж асаах хүртэл чөлөөлгэхгүй.



Зураг 7.13: Бүх объект устсаны дараах байдал

### 7.3.3 Хаяган өгөгдлийн гарааны утгатай холбоотой үүсэх алдааг засах

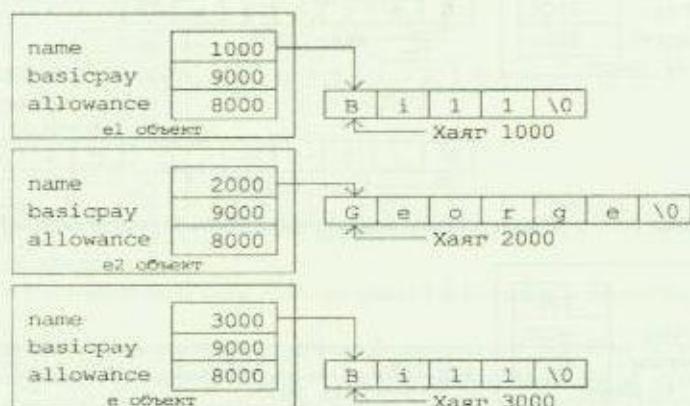
Program #7.4 доторх `employee e=e1;` командаар e объект байгуулж e1 объектоор гарааны утга оноход e1 объектын гишүүн бүрийг e объектын харгалзах гишүүн рүү хуулах тул энэ хоёр объектын утга ижил болно. Ингэснээр, e, e1 объектууд дундаа 1000 гэсэн дугаартай ойг эзэмших болсон нь алдаа гарах гол шалтгаан юм. Тухайлбал, оороор зааж өгөөгүй бол утга оноох оператор нь түүний баруун талын объектоос гишүүн өгөгдэл бүрийг

нь зүүн талын объект руу хуулдагтай холбоотойгоор алдаа гарна. Иймд `employee e=e1;` командын дараагаар Зураг 7.14-д үзүүлсэн шиг үр дүн гардаг байхаар хуулах функцийн бичих шаардлагатай болно.

Хуулах функцийн нь "Bill" үгийг хадгалах ойг эхэлж бэлдээд хуулна. Шинээр бэлдэх ойг хаягийг `e` объектын `name` өгөгдөл заана. Эх үгийн байгаа ойн хаягийг `e1` объектын `name` өгөгдөл агуулж байдаг. Бусад `e1.basicpay`, `e1.allowance` өгөгдлүүдийг `e` объектын харгалзах өгөгдөл болох `e.basicpay`, `e.allowance` рүү хуулна.

Дээр дурдсан шаардлагын дагуу объектыг объект руу хуулж чадах функцийн бол хуулагч байгуулагч юм. Ийм хуулагчийн эх загвар нь

```
class_name(const class_name &)
shig baih jstoy tul zyinii daguu employeee klassyn huuлагch baiгуулагчийн загварыг
employeee(const employeee &) gž bichиж bolno.
```



Зураг 7.14: Хуулагч байгуулагчийн ажиллуулсны дараах байдал

Класс нь хуулагч байгуулагчтай бол C++ компайлер түүнийг дараах 3 тохиолдолд дуудаж ажиллуулна. Тухайлбал,

- Объектын гаралтын утга нь объект байх
- Функцийн авах утга нь объект байх
- Функцийн буцах утга нь объект байх

Эхний тохиолдолд, жишээ нь `employee e=e1;` команда хийгдэх үед `e` объектоос хуулагч байгуулагчийг дуудахлаа түүнд `e1` объектыг аргумент хэлбэрээр дамжуулна. Тэгэхээр `employee e=e1;` команда гэсэн хоёр командаар оруулж болно. Энэ `employeee` классын хуулагч байгуулагчийг доор үзүүлсэн шигээр тодорхойлно.

```
employeee e ;
e.employeee(e1) ;

employeee::employeee(const employeee &e)
{
    name = new char[strlen(e.name)+1] ;
    strcpy(name, e.name);
    basicpay = e.basicpay ;
```

```
    allowance = e.allowance ;
}
```

Дээрх шиг хуулагч байгуулагч нь employee e=e1; командын хувьд е объектоос дуудагдах ба e1 объект түүний аргумент нь болно. Иймээс хуулагч байгуулагчийн доторх

```
name = new char[strlen(e.name)+1];
```

командын хувьд name нь е объектын өгөгдөл, e.name нь e1 объектын өгогдол юм.

Program #7.4-т хуулагчийг нэмж оруулах замаар доор үзүүлснээр өөрчлөн бичиж болно.

```
//Program #7.6
//
#include <iostream.h>
#include <string.h>
#include <conio.h>

class employee
{
private:
    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void);
    ~employee(void);
    employee(char *n, int b, int a);
    employee(const employee &e);
    friend ostream & operator <<(ostream &os, employee &e);
};

employee::employee()
{
    name = new char[20] ;
    strcpy(name, "Hello C++") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::employee(const employee &e)
{
    name = new char[strlen(e.name)+1];
    strcpy(name, e.name);
    basicpay = e.basicpay ;
    allowance = e.allowance;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye,\t" << name << "\n" ;
    delete name;
}
```

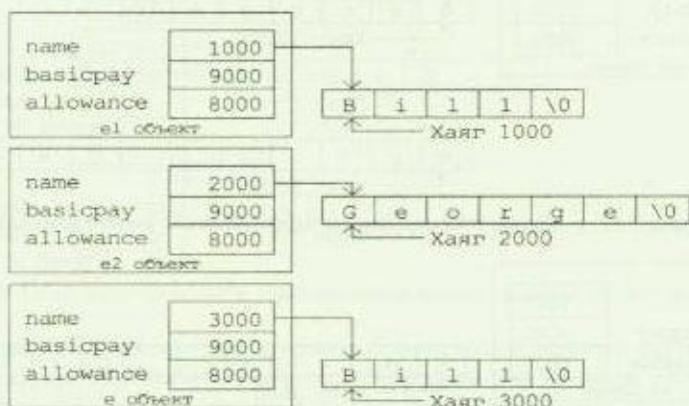
нь зүүн талын объект руу хуулдагтай холбоотойгоор алдаа гарна. Иймд `employee e=e1;` командын дараагаар Зураг 7.14-д үзүүлсэн шиг үр дүн гардаг байхаар хуулах функцийг бичих шаардлагатай болно.

Хуулах функцийг "Bill" үгийг хадгалах ойг эхэлийг бэлдээд хуулна. Шинээр бэлдэх ойн хаягийг `e` объектын name огогдол заана. Эх үгийн байгаа ойн хаягийг `e1` объектын name егегдэл агуулж байдаг. Бусад `e1.basicpay`, `e1.allowance` огогдлуудийг `e` объектын харгалзах өгөгдөл болох `e.basicpay`, `e.allowance` рүү хуулна.

Дээр дурдсан шаардлагын дагуу объектыг объект руу хуулж чадах функцийг бол хуулагч байгуулагч юм. Ийм хуулагчийн эх загвар нь

```
class_name(const class_name &)
```

шиг байх ёстой тул үүний дагуу `employee` классын хуулагч байгуулагчийн загварыг `employee(const employee &)` гэж бичиж болно.



Зураг 7.14: Хуулагч байгуулагчийг ажиллуулсны дараах байдал

Класс нь хуулагч байгуулагчтай бол C++ компайлер түүнийг дараах 3 тохиолдолд дуудаж ажилтуулна. Тухайлбал,

- Объектын гараны утга нь объект байх
- Функцийн авах утга нь объект байх
- Функцийн буцах утга нь объект байх

Эхний тохиолдолд, жишээ нь `employee e=e1;` команд хийгдэх үед `e` объектоос хуулагч байгуулагчийг дуудахлаа түүнд `e1` объектыг аргумент хэлбэрээр дамжуулна. Тэгэхээр, `employee e=e1;` командыг

```
employee e ;  
e.employee(e1) ;
```

гэсэн хоёр командаар орлуулж болно. Энэ `employee` классын хуулагч байгуулагчийг доор үзүүлсэн шигээр тодорхойлно.

```
employee::employee(const employee &e)  
{  
    name = new char[strlen(e.name)+1] ;  
    strcpy(name, e.name);  
    basicpay = e.basicpay ;
```

```
    allowance = e.allowance ;
```

Дээрх шиг хуулагч байгуулагч нь employee e=e1; командын хувьд е объектоос дуудагдах ба e1 объект түүний аргумент нь болно. Иймээс хуулагч байгуулагчийн доторх

```
    name = new char[strlen(e.name)+1];
```

командын хувьд паме нь е объектын огогдол, e.name нь e1 объектын өгөгдөл юм.

Program #7.4-т хуулагчийг нэмж оруулах замаар доор үзүүлсэнээр өөрчлөн бичиж болно.

```
//Program #7.6
//
#include <iostream.h>
#include <string.h>
#include <conio.h>

class employee
{
private:
    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void);
    ~employee(void);
    employee(char *n, int b, int a);
    employee(const employee &e);
    friend ostream & operator <<(ostream &os, employee &e);
};

employee::employee()
{
    name = new char[20] ;
    strcpy(name, "Hello C++") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::employee(const employee &e)
{
    name = new char[strlen(e.name)+1];
    strcpy(name, e.name);
    basicpay = e.basicpay ;
    allowance = e.allowance;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye,\t" << name << "\n" ;
    delete name;
}
```

```

ostream & operator <<(ostream &os, employee &e)
{
    int grosspay = e.basicpay + e.allowance ;
    os.width(20) ;
    os << e.name ;
    os.width(8) ;
    os << e.basicpay ;
    os.width(12) ;
    os << e.allowance ;
    os.width(8);
    os << grosspay ;
    os << "\n" ;
    return (os) ;
}

void main()
{
    clrscr() ;
    employee e1("Bill", 9000, 8000) ;
    cout << e1 ;
    employee e2("George", 9000, 7000) ;
    cout << e2 ;
    employee e = e1;
    cout << e ;
}

```

Bill	9000	8000	17000
George	9000	7000	16000
Bill	9000	8000	17000
Bye, bye,	Bill		
Bye, bye,	George		
Bye, bye,	Bill		

#### 7.3.4 Хаяган өгөгдлийн утгатай холбоотой үүсэх алдааг засах

Program #7.5 доторх `e=e1;` команд өмнө нь байгуулагдсан `e` объектод `e1` объектоор утга оноохдоо `e1` объектын гишүүн бүрийг `e` объектын харгалзах гишүүн рүү хуулах тул энэ хоёр объектын утга ижил болно. Ингэснээр, `e, e1` хоёр объект дундаа 1000 гэсэн дугаартай ойт эзэмшиж, харин 3000 гэсэн дугаартай ойт байгаа "Hello, C++" тэмдэгт мөр аль ч объектын харьяалалгүйгээр үлдэж байгаа нь алдаа гарах гол шалтгаан болно.Өөрөөр зааж өгөөгүй бол утга оноох оператор нь түүний баруун талын объектоос гишүүн өгөгдэл бүрийг нь зүүн талын объект руу харгалзуулан хуулдагтай холбоотойгоор алдаа гарна. Иймд `e=e1;` командын дараагаар гарах үр дүн нь Зураг 7.10-д үзүүлсэн шиг биш харин Зураг 7.14-ийн шиг байх шаардлагатай.Үүний тулд дараах зүйлийг хийж, тухайлбал

- "Hello, C++" морийн эзэмшиж байгаа ойт чөлөөлж,
- new оператороор ойт бэлдээд түүнд `e1` объектын name огогдолд хаяг нь байгаа ойт утгыг хуулж,
- Шинээр бэлдэх ойн эхлэл хаягийг `e` объектын name огогдолд хийж

чадах `=()` оператор функцийг шинээр нэмж бичихдээ уг функцийн<sup>45</sup> загварыг классын нь тодорхойлолт дотор

<sup>45</sup> Энэ функци нь найдуулж болохгүй, учир нь = операторыг найдуулж болохгүй.

```

employee & operator =(employee &e);
гэж бичээд дараах байдлаар тодорхойлно.

employee & employee :: operator =(employee &e)
{
    if(this == &e)
        return (*this) ;
    delete name ;
    name = new char[strlen(e.name)+1] ;
    strcpy(name, e.name);
    basicpay = e.basicpay ;
    allowance = e.allowance ;
    return (*this) ; //or return (e) ;
}

```

Объект руу объект хуулах =() оператор функций нь e=e; командын хувьд е объектоос дуудагдах ба e1 объект түүний аргумент болно. Иймд e=e; команд нь e.operator =(el) командтай ижил юм. Дахин тодорхойлсон уг функцийн

```
name = new char[strlen(e.name)+1] ;
```

командын хувьд name нь e объектынх бол e.name нь el объектын өгөгдөл юм.

Program #7.5-д =() оператор функцийг нэмж бичих замаар доор үзүүлснээр өөрчлөн бичиж болно.

```

//Program #7.7
#include <iostream.h>
#include <string.h>
#include <conio.h>
class employee
{
private:
    char *name;
    int basicpay ;
    int allowance ;
public:
    employee(void) ;
    ~employee(void) ;
    employee(char *n, int b, int a) ;
    employee & operator =(employee &e) ;
    friend ostream & operator <<(ostream &os, employee &e) ;
} ;

employee & employee::operator =(employee &e)
{
    if(this == &e)
        return (*this) ;
    delete name ;
    name = new char[strlen(e.name)+1] ;
    strcpy(name, e.name);
    basicpay = e.basicpay ;
    allowance = e.allowance ;
    return *this ; // or return e ;
}
employee::employee()
{
    name = new char[20] ;
}
```

```

strcpy(name, "Hello C++") ;
basicpay = 0 ;
allowance = 0 ;
}

employee::employee(char *n, int b, int a)
{
    name = new char[strlen(n)+1] ;
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee::~employee()
{
    cout << "Bye, bye,\t" << name << "\n" ;
    delete name;
}

ostream & operator <<(ostream &os, employee &e)
{
    int grosspay = e.basicpay + e.allowance ;
    os.width(20) ;
    os << e.name ;
    os.width(8) ;
    os << e.basicpay ;
    os.width(12) ;
    os << e.allowance ;
    os.width(8);
    os << grosspay ;
    os << "\n" ;
    return (os) ;
}

void main()
{
    clrscr() ;
    employee e1("Bill", 9000, 8000) ;
    cout << e1 ;
    employee e2("George", 9000, 7000) ;
    cout << e2 ;
    employee e;
    e = e1;
    cout << e ;
}

```

Bill 9000 8000 17000  
 George 9000 7000 16000  
 Bill 9000 8000 17000

Bye, bye, Bill  
 Bye, bye, George  
 Bye, bye, Bill

#### 7.4 this хувьсагч

Систем нэмж тодорхойлдог this хувьсагч нь гишүүн функцийг хэрэглэж байгаа объектын хаягийг заадаг. Энэ нь объектон хаяган хувьсагч юм. Объектоос гишүүн функцийг дуудах үед систем тухайн объектын хаягийг уг хувьсагч руу хийсний дараагаар дуудагдсан функцияа програмын удирдлагыг шилжүүлдэг. Ингэснээр гишүүн функцияа дуудагдсан объектынхоо

хаягийг мэдэх, улмаар гишүүн функци бүр this хувьсагчийг хэрэглэх боломжтой болно. Объектын хаягийг this хувьсагчаар дамжуулж авах жишээ програмыг доор үзүүллээ.

```
//Program #7.8
//Finding Address of object through this pointer

#include <iostream.h>
#include <iomanip.h>
#include <stdio.h>
#include <conio.h>

int k = 1 ;
class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    void showdata()
    {
        cout << "\nThe object emp" << k << "'s address is: " << this << endl ;
        k++ ;
    }
};

void main()
{
    clrscr() ;
    employee emp1, emp2 ;
    emp1.showdata() ;
    emp2.showdata() ;
    getch() ;
}

The object emp1's address is: 0x8fb5ffde
The object emp2's address is: 0x8fb5ffc6
```

Program #7.8-ын main() функци дотор employee классын emp1, emp2 гэсэн хоёр объект байгуулж тэдгээрээс showdata() функцийг дуудснаар дээр үзүүлсэн үр дүн гарна.

Энэ this хувьсагчийг гишүүн функци дотроос далд, ил хоёр янзаар хэрэглэж болно. Эхний тохиолдолд объектын утга буюу гишүүн өгөгдөл рүү хандахдаа this хувьсагчийг илээр шууд зааж өгдөгтүй. Удаах тохиолдолд this хувьсагчийт илээр гол төлөв сонголтын сум операторын хамт хэрэглэнэ. Үүнийг дараах програмаар харууллаа.

```
//Program #7.9
//Finding gross pay using this pointer

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <stdio.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
```

```

int basicpay ;
int allowance ;
public:
void showdata() ;
void getdata()
{
    cout << "\nEnter the Employee Name: " ;
    gets (this->name);
    cout << "Enter the Basic Pay: " ;
    cin  >> this->basicpay;
    cout << "Enter the Allowance: " ;
    cin  >> this->allowance;
}
} ;
void employee::showdata()
{
cout << endl ;
cout << setw(20) << name ;
cout << setw(8)   << basicpay ;
cout << setw(12) << allowance ;
}

void heading()
{
cout << endl ;
cout << setw(20) << "Employee Name" ;
cout << setw(8)   << "Basic" ;
cout << setw(12) << "Allowance" ;
}

main()
{
clrscr() ;
employee emp ;
emp.getdata() ;
heading() ;
emp.showdata() ;
getch() ;
}

 Enter the Employee Name: George.
Enter the Basic Pay: 9000.
Enter the Allowance: 8000.

Employee Name      Basic      Allowance
      George        9000        8000

```

this хувьсагчар дамжуулан гишүүн өгөгдөл хандах боломжийг Program #7.9-д үзүүллээ. Объектын утга буцаах гишүүн функцийн дотор, мөн рекурсив классын<sup>46</sup> тодорхойлолт дотор this хувьсагчийг бас хэрэглэж болно.

### 7.5 Статик өгөгдөл

Классын гишүүн өгөгдэл статик шинжтэй байж болно. Ийм статик өгөгдөл бусад гишүүн өгөгдлөөс ялгагдах шинжтэй. Байгуулагч функцийн статик биш өгөгдлийн хуулбарыг объект

<sup>46</sup> Оюроо верийгээ хэрэглэх классыг рекурсив класс гэнэ.

бүрд үүсгэх бол статик өгөгдлийг классын бүх объект дундаа хэрэглэдэг байхаар нэг удаа сөрөнхий үүсгэнэ. Иймд энэ өгөгдлийг бас классын өгөгдөл гэнэ.

Статик өгөгдлийг тодорхойлохдоо классын нь тодорхойлолт дотор түүний нэрийн өмнө static гэдэг тусгай уг бичнэ. Статик өгөгдлийн гарынны утгыг классын нь тодорхойлолтын гадна талд Зураг 7.15-д үзүүлсэн шигээр зааж огно. Гэхдээ static тусгай үгийг зөвхөн классын тодорхойлолт дотор хэрэглэнэ.

```
class class_name
{
private:
    data members;
    static int number; → өгөгдөл
                           → тусгай тулхүүр уг
public:
    member_functions;
};

int class_name::number = 0; → static хувьсагчид
                           → гарынны утга оноох
```

Зураг 7.15: static гишүүн өгөгдэл тодорхойлах загвар

Статик өгөгдлийг, жишээ нь классын хэдэн объект байгуулагдсанас хэд нь үлдээд байгааг тооюу тоолуураар хэрэглэхэд нэн тохиромжтой байдаг. Үүнийг доорх жишээ програмаар үзүүлүүлэх боломжтой.

```
//Program #7.10
//static class data members

#include <iostream.h>
#include <conio.h>
#include <string.h>

class employee
{
private:
    static int number ;
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee() ;
    ~employee() ;
    employee(char *n, int b, int a) ;
    employee(employee &e) ;
    void show_number() ;
};

int employee :: number = 0 ;

employee::employee()
{
    strcpy(name, "Hello C++") ;
    basicpay = 0 ;
    allowance = 0 ;
}
```

```

        number++ ;
    }

employee::employee(char *n, int b, int a)
{
    strcpy(name, n);
    basicpay = b;
    allowance = a;
    number++ ;
}

employee::employee(employee &e)
{
    strcpy(name, e.name);
    basicpay = e.basicpay ;
    allowance = e.allowance ;
    number++ ;
}

employee::~employee()
{
    cout << "\nBye, bye, " << name << " ";
    number-- ;
    cout << number << " Objects left behind" ;
}

void employee::show_number()
{
    cout << "\nNumber: " << number ;
}

void main()
{
    clrscr() ;
    employee e1("George", 100, 100) ;
    e1.show_number() ;
    employee e2("Bill", 200, 200) ;
    cout << endl ;

    e1.show_number() ;
    e2.show_number() ;
    employee e3("Jack", 300, 300) ;
    cout << endl ;

    e1.show_number() ;
    e2.show_number() ;
    e3.show_number() ;

    employee e = e1 ;
    cout << endl ;
    e1.show_number() ;
    e2.show_number() ;
    e3.show_number() ;
    e.show_number() ;
    e3 = e2 ;
    cout << endl ;
    e1.show_number() ;
    e2.show_number() ;
    e3.show_number() ;
    e.show_number() ;
}

```

```

cout << endl ;
getch() ;
}

Number: 1
Number: 2
Number: 2

Number: 3
Number: 3
Number: 3

Number: 4
Number: 4
Number: 4
Number: 4

Number: 4
Number: 4
Number: 4
Number: 4

Bye, bye George 3 Objects left behind
Bye, bye Bill 2 Objects left behind
Bye, bye Bill 1 Objects left behind
Bye, bye George 0 Objects left behind

```

Дээрх програмыг авч үзье. Статик өгөгдлийн тодорхойлолтын эхийд static тусгай үгийг жишээ програмд үзүүлсэн шигээр нэмж бичих замаар статик өгөгдлийг зарлана.

```
static int number ;
```

Классын тодорхойлолт дотор статик өгөгдөл гарсаны утга оноож болохгүй учир классын нь тодорхойлолтын гадна талд доор үзүүлснээр бичиж өгнө.

```
int employee :: number = 0 ;
```

Бүх байгуулагч дотор number++; морийг нэмж оруулснаар шинэ объект үүсэх бүрд түүнийг тох боломжтой болно. Харин устгач функци доторх number--; команд нь объект устгах бүрд статик өгөгдлийн утгыг нэгээр хорогдуулж, бусад команд нь устах объектын иэр, үлдэх объектын тоог хэвлэнэ. Програмд тодорхойлсон employee классын хэдэн ч объектыг доор үзүүлсэн шигээр үүсгэж болно.

```
employee e1("George", 100, 100) ;
employee e2("Bill", 200, 200) ;
employee e3("Jack", 300, 300) ;
```

Зураг 7.16-д үзүүлснээр e1, e2, e3 объектууд дундаа хэрэглэх срөнхий number өгөгдолтэй. Ер нь, классын бүх объект дундаа хэрэглэх шинжийг static байхаар тодорхойлох боломжтой.

## 7.6 Статик функци

Классын гишиг функци статик шинжтэй байж болно. Ийм функци класст нэг л байж болох ба түүнийг тухайн классын бүх объект дундаа хэрэглэнэ. Гэхдээ статик функци классынхаа зөвхөн статик өгөгдлийг боловсруулах үүрэгтэй. Иймд статик функци классынхаа статик биш гишиг рүү хандаж чадахгүй. Бас статик функци дотроос this хувьсагч руу хандахгүй. Статик функци нь const төрлийнх байж болдогтүй.

number 3							
name	G	e	o	r	g	e	\0
basicpay	100						
allowance	100						
e1							

name	B	i	l	l	\0	
basicpay	200					
allowance	200					
e2						

name	J	a	s	k	\0	
basicpay	300					
allowance	300					
e3						

Зураг 7.16: Ерөнхий статик number огогдол бүхий объектууд

Статик функцийн ажиллагаатай танилцахын тулд Program #7.10-д байгаа

```
void show_number();
merийг static void show_number(); гэсэн мөрөөр сольж бичье.
```

Статик функцийг классын тодорхойлолт дотор зарлахдаа static тусгай угийт ганц удаа хэрэглэнэ. Түүнийг функцийн тодорхойлолтын толгойн хэсэгт огт хэрэглэхгүй. Тэгэхлээр, доор үзүүлсэн

```
static void employee::show_number()
{
    -----
}
```

тодорхойлолт нь дүрмийн алдаатай бол харин

```
void employee::show_number()
{
    -----
}
```

нь алдаагүй тодорхойлолт юм.

Статик функцияа классынхаа аль ч объектод бэхлэгдээгүй байх бөгөөд түүнийг бүх объект дундаа хэрэглэх тул хаанаас ч дуудаж болно. Жишээ нь, ямар ч классын статик гишүүд рүү тухайн классын гадна талаас хандахдаа аль нэг объектоор нь дамжуулан static функцийг доор үзүүлсэн шигээр дуудна.

```
employee e1("George", 100, 100);
e1.show_number();
employee e2("Bill", 200, 200);
e1.show_number();
e2.show_number();
```

Бас статик функцийг тодорхой объектоор дамжуулахгүйгээр шууд дуудаж болдог. Ингэхдээ классын нэр, үйлчлэх хүрээний ( :: ) операторыг хэрэглэнз.

```
employee e1("George", 100, 100) ;
employee::show_number() ;
employee e2("Bill", 200, 200) ;
employee::show_number() ;
```

Статик функцийг бас классын функц гэнэ. Статик функц хэрэглэх жишээг доор үзүүллээ.

```
//Program #7.11
//static class data members

#include <iostream.h>
#include <conio.h>
#include <string.h>

class employee
{
private:
    static int number ;
    char name[13] ;
    int basicpay ;
    int allowance ;
public:
    employee() ;
    ~employee() ;
    employee(char *n, int b, int a) ;
    employee(employee &e) ;
    static void show_number() ;
} ;

int employee :: number = 0 ;

employee::employee()
{
strcpy(name, "Hello C++") ;
basicpay = 0 ;
allowance = 0 ;
number++ ;
}

employee::employee(char *n, int b, int a)
{
strcpy(name, n) ;
basicpay = b ;
allowance = a ;
number++ ;
}

employee::employee(employee &e)
{
strcpy(name, e.name) ;
basicpay = e.basicpay ;
allowance = e.allowance ;
number++ ;
}

employee::~employee()
{
cout << "\nBye, bye, " << name << " " ;
number-- ;
```

```
    cout << number << " Objects left behind" ;
}

void employee::show_number()
{
    cout << "\nNumber: " << number ;
}

void main()
{
    clrscr() ;
    employee e1("George", 100, 100) ;
    e1.show_number() ;
    employee::show_number() ;

    employee e2("Bill", 200, 200) ;
    cout << endl ;

    e1.show_number() ;
    e2.show_number() ;
    employee::show_number() ;

    employee e3("Jack", 300, 300) ;
    cout << endl ;

    e1.show_number() ;
    e2.show_number() ;
    e3.show_number() ;
    employee::show_number() ;

    employee e = e1 ;
    cout << endl ;
    e1.show_number() ;
    e2.show_number() ;
    e3.show_number() ;
    e.show_number() ;
    employee::show_number() ;
    cout << endl ;
    getch() ;
}
```

```
□ Number: 1
Number: 1

Number: 2
Number: 2
Number: 2

Number: 3
Number: 3
Number: 3
Number: 3
```

```

Number: 4
Number: 4
Number: 4
Number: 4
Number: 4

Number: 4
Number: 4
Number: 4
Number: 4
Number: 4

Bye, bye George 3 Objects left behind
Bye, bye Bill 2 Objects left behind
Bye, bye Bill 1 Objects left behind
Bye, bye George 0 Objects left behind

```

#### Мэдлэгээ шалгах асуулт

- 7.1. Эхлээд a, дараа нь b объект байгуулагдах бол програм дуусгавар болоход тэдгээр нь ямар дарааллаар устах
- 7.2. Объектон хүснэгтийг 10 бүтвэртэй байхаар classA aa[10]; гэж байгуулах бол програм ажиллаад дуусгавар болоход тэдгээр нь ямар дарааллаар устах
- 7.3. Статик хувьсагч гэж юу болох, статик, статик бус хувьсагч хоорондын ялгааг жишээн дээр тайлбарлах
- 7.4. Статик гишүүн өгөгдөл гэж юу болох, статик, статик бус өгөгдөл хоорондын ялгааг жишээн дээр тайлбарлах
- 7.5. Ямар онцгой шинж static классын гишүүн өгөгдлөтэй холбогдох
- 7.6. static классын гишүүн өгөгдлийг хэрхэн гаранаы утгатай болгох; гаранаы утгатай болгох командыг програмын аль хэсэгт бичиж өгөх
- 7.7. Статик функцийг хэрхэн тодорхойлох, онцлог шинж нь юу болох
- 7.8. Объектон хаяган хувьсагчийг яаж байгуулах, онцлог шинж нь юу болох, түүнийг объект той хэрхэн уях
- 7.9. this хувьсагч гэж юу болох, түүнийг хэрхэн хэрэглэхийг жишээн дээр тайлбарлах
- 7.10. Хаяган хувьсагчийг класс дотор яаж хэрэглэх, түүнтэй холбоотойгоор үүсч болох зөрчилтэй байдлыг жишээн дээр тайлбарлах
- 7.11. Классын static функци this хувьсагч руу хандаж чадах эсэх
  - a) true
  - b) false

#### Жишээ бодлого

- 7.12. Дурын урттай тэмдэгт мөрийг загварчлах string класс тодорхойлох; Мөрийг объектын гадна талд, түүний эхлэл хаягийг объект дотор тус тус хадгалах ёстой бол тохиорх байгуулагч, устгагч функцийг тодорхойлох
- 7.13. Бодлого 7.12-т тодорхойлох string классын хувьд байгаа s1 объектоор нь шинээр байгуулах з объектод нь гаранаы утга оноох (`string s=s1;`) боломжтой байхаар хуулагч байгуулагчийг нэмж тодорхойлох

# НАЙМДУГААР БҮЛЭГ

## НАЙЗ ФУНКЦ, ОПЕРАТОР ДАХИН ТОДОРХОЙЛОХ

- Найз функци, 239
- Найз класс, 247
- Оператор дахин тодорхойлох, 248
  - Ганц операндын операторыг дахин тодорхойлох, 249
  - Хос операндын операторыг дахин тодорхойлох, 253
  - Хоёр тэмдэгт мөрийг залгах, 256
  - Хоёр объектыг жишиг, 257
  - Хоёр объектын тэнцүү эсэхийг шалгах, 259
  - Нэмээд буцааж утга оноох нийлмэл операторыг дахин тодорхойлох, 260
- Найз функци ба дахин тодорхойлох оператор, 261
- << операторыг дахин тодорхойлох, 265
- Оператор дахин тодорхойлоход анхаарах зүйл, 269
- Өгөгдөл хөрвүүлэг, 270
  - Суурь ба зохиомол төрөл хоорондын хувиргалт, 271
  - Зохиомол төрлийн объект хоорондын хувиргалт, 271



## НАЙМДУГААР БҮЛЭГ

### НАЙЗ ФУНКЦ, ОПЕРАТОР ДАХИН ТОДОРХОЙЛОХ

C++ хэлний гол хоёр шинж бол битүүмжлэл ба өгөгдөл далдлал юм. Өгөгдөл, түүнийг боловсруулах аргыг цутг нь програмчлалын хийсвэр нэгж хэлбэрээр тодорхойлохыг битүүмжлэл гэнэ. Харин өгөгдөл далдлал гэдэгт *private* өгөгдөл рүү классын биш функцизэс хандаж болдоггүй чанарыг ойлгоно. *Private* өгөгдөл рүү зөвхөн классын нь *public* гишүүн функцизээр дамжуулж хандана. Ингэж өгөгдөл рүү хандах хандалтанд хязгаар тогтоож хямгадах нь өгөгдөл устах, санамсаргүйгээр өөрчлөгдохеосс сэргийлэхэд тустай байдаг. Гэхдээ ийм аргаар өгөгдлийг хамгаалах нь заримдаа тийм ч тохиromжтой байдаггүй байна. Иймээс C++ хэлний өгөгдөл далдлалын шинж нь (*friend*) найз функцийг бий болгоход хүргэжээ. Амьдрал дээр хэвшил болсон зүйлс заримдаа зөрчигдөх явдал байдаг. Жишээ нь, өглөө бүр гүйж сурсан хүн өвчний улмаас гүйж чадахаа хэсэг хугацаанд болих нь бий. Үүнтэй адил найз функци нь өгөгдөл далдлалын шинжийг зөрчиж буй шинж, боломж юм.

#### 8.1 Найз функци

Класс изгтэй хоёр объектын нийлбэрийг гишүүн функцийн нь оролцоотойгоор дараах турван аргын аль нэгээр олж болно.

- `e3.addpay(e1, e2);`
- `e3 = e1.addpay(e2);`
- `e = addpay(e1, e2);`

Эхний `e3.addpay(e1,e2);` командаын `addpay()` нь `employee` классын гишүүн функци байх ба классынхаа хоёр объектыг аргумент хэлбэрээр аваад нийлбэрийг нь олж идэвхтэй<sup>47</sup> байгаа `e3` объект руу хадгалах үүрэгтэй. Удаах `e3=e1.addpay(e2);` командаын `addpay()` нь идэвхтэй объект, гаднаас авах объект хоёрын нийлбэрийг олж `e3` объект руу хуулах функци юм. Сүүлийн `e=addpay(e1,e2);` командаын `addpay()` нь ямар ч классын биш функци байх ба гаднаас авах хоёр объектын нийлбэрийг олж бушаана.

Ямар нэгэн функцизээр олон объектын нийлбэр олох дээрх шиг ажлыг найз функцизээр бас хийлгэж болно. Найз функци бол классынхаа *private* өгөгдөл рүү хандаж чаддаг гишүүн функци биш. Найз функци нь гол төлөв классын биш функци байна. Олон классын *private* өгөгдөл рүү хандах функци хэрэгтэй болоход ихэвчлэн найз функцийг хэрэглэнэ.

Өмнөх бүлэгт хэрэглэсэн `employee` классын тодорхойлолтыг найз функци нэмж оруулах замаар доор үзүүлснээр өөрчлөн бичье.

```
class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    employee();
    ~employee();
    void showdata();
}
```

<sup>47</sup> Гишүүн функцийнг тухайн агшинд хэрэглэж байгаа объект бол идэвхтэй объект юм.

```
    void getdata() ;
    employee(char *n, int b, int a) ;
    friend employee addpay(employee e1, employee e2) ;
};
```

Дээрх тодорхойлолтын сүүлийн мөрөнд байгаа

```
    friend employee addpay(employee e1, employee e2);
```

командаар addpay() гэдгээ нийз функцийг зарлаж байна. Энд хэрэглэж байгаа friend ийн addpay() бол employee классын нийз функц болохыг C++ компайлерт хэлж өгөх үүрэгтэй тусгай үг юм. Нийз функцийг классын тодорхойлолт дотор зарласан ч тэрээр классын гишүүн биш функц юм. Нийз функц нь түүнийг тодорхойлолт дотроо зарласан классын нийз функц нь болно. Тэгэхлээр, класс нь ямар нэгэн функштэй "найз" байхын тулд түүнийг тодорхойлолт дотроо "найз" гэж зарлах ёстой. Дээрх тодорхойлолтын addpay() ийн employee классын нийз функц болно.

Гишүүн функц классынхаа private өгөгдөл рүү хандаж чадлагийн адил нийз функц бас нийз классынхаа private өгөгдөл рүү хандаж чадна. Гэхдээ гишүүн функц классынхаа тодорхой объектоор дамжуулж өгөгдөл рүү доор үзүүлсэн шигээр

```
employee e1, e2;
e1.getdata();
e2.showdata();
```

хандах бол нийз функцийг тодорхой объектоор дамжуулж дуудах шаардлагагүй байдаг. Ийм функц рүү объектыг зөвхөн параметр хэлбэрээр дамжуулж өгнө.

```
employee e1, e2, e3 ;
e3 = addpay(e1, e2) ;
```

Нийз функцийг гишүүн функцээс өөрөөр тодорхойлно. Энэ ялгааг харуулах үүднээс омни тодорхойлсон employee классын зарим гишүүн, нийз функцийг дараах байдлаар дахин тодорхойльб.

```
employee::employee(char *n, int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

void employee::showdata()
{
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
}

employee addpay(employee e1, employee e2)
{
    employee temp ;
    temp.basicpay = e1.basicpay + e2.basicpay ;
    temp.allowance = e1.allowance + e2.allowance ;
    strcpy(temp.name, "Total") ;

    return (temp) ;
}
```

Найз функцийн тодорхойлолтод friend үгийг хэрэглэхгүй. Түүнийг зөвхөн классын тодорхойлолт дотор найз функцийг зарлаадаа хэрэглэнэ. Найз функц нь ямар ч классын гишүүн биш функц байх учир үйлчлэх хүрээний (::) операторыг түүний тодорхойлолтод хэрэглэдэггүй. Гишүүн функц зөвхөн классынхаа объектуудын хүрээнд үйлчлэх тул түүнтэй нэр ижилтэй функц веер класс дотор байж болдог. Харин найз функц нь тодорхойлогдсон файлынхаа хэмжээнд үйлчилнэ. Иймийн учир ижил нэртэй олон найз функц нэг файл дотор байж болохгүй. Найз, гишүүн функцуудийн хооронд байх веер нэг ялгаа бол тэдгээрээс егегдэл хандах арга юм. Энэ ялгааг олж хараахын тулд employee класст байж болох

```
void addpay(employee e1, employee e2) ;  
гишүүн функцийн доорх тодорхойлолтыг авч үзье.
```

```
void employee::addpay(employee e1, employee e2)  
{  
    basicpay = e1.basicpay + e2.basicpay ;  
    allowance = e1.allowance + e2.allowance ;  
}
```

Дээрх функц нь тодорхой объектоос, жишээ нь

```
e3.addpay(e1, e2);
```

тэж дуудагдана. Функцийн тодорхойлолт дотор объектын нэргүйгээр хэрэглэгдэж байгаа basicpay, allowance нь гишүүн функцийт хэрэглэх e3 объектын өгөгдлүүд болно. Харин e1.basicpay, e2.basicpay, e1.allowance ба e2.allowance нь уг функц рүү параметр хэлбэрээр дамжуулж өгөх e1, e2 гэсэн хоёр объектынх юм. Харин найз функц нь basicpay, allowance гэсэн хувьсагчуудгүй, зөвхөн employee төрлийн temp дотоод хувьсагч хэрэглэсэн байна. Найз функцийг програм дотроос, тухайлбал main() функц дотроос

```
e3 = addpay(e1, e2);
```

тэж дуудах бөгөөд найз функцизэс буцаах temp объектын уттыг e3 объектын утга болгож хуулна.

Өмнө тодорхойлсон addpay() найз функцийн тодорхойлолтыг дараах байдлаар өөрчлөн бичиж болно.

```
employee addpay(employee e1, employee e2)  
{  
    int basicpay = e1.basicpay + e2.basicpay ;  
    int allowance = e1.allowance + e2.allowance ;  
    return employee("Total", basicpay, allowance) ;  
}
```

Функц нь гаднаас параметр хэлбэрээр авах e1, e2 гэсэн хоёр объектын гишүүн өгөгдлүүдээр болдот хийж хариугаа basicpay, allowance гэсэн хоёр дотоод хувьсагч руу хадгална. Эдгээр завсрлын хувьсагч нь employee төрлийн объект үүсгэхэд хэрэглэгдэх бөгөөд үүсгэх объектыг функцизэс буцаах үед e3 объект руу "баръж" авна. Ийм найз функцийн хэрэглээг дараах програмаас харж болно.

```
// Program #8.1  
//Finding the gross pay of an employee using friend function  
  
#include <iostream.h>  
#include <iomanip.h>  
#include <stdio.h>
```

```

#include <conio.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    void getdata(void) ;
    void showdata(void) ;
    employee();
    ~employee();
    employee(char *n, int b, int a) ;
    friend employee addpay(employee e1, employee e2);
} ;

employee::employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::~employee()
{
}

employee :: employee(char *n, int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee addpay(employee e1, employee e2)
{
    employee temp ;
    temp.basicpay = e1.basicpay + e2.basicpay;
    temp.allowance = e1.allowance + e2.allowance ;
    strcpy(temp.name, "Total");
    return (temp) ;
}

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);
    cout << "Enter the Basic Pay: ";
    cin  >> basicpay ;
    cout << "Enter the Allowance: ";
    cin  >> allowance ;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance ;
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
}

```

```

    cout << setw(12) << allowance ;
    cout.width(8);
    cout << grosspay ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

void main()
{
clrscr() ;
employee e1, e2, e3 ;
e1.getdata() ;
e2.getdata() ;
heading() ;
e1.showdata() ;
e2.showdata() ;
e3 = addpay(e1, e2) ;
e3.showdata() ;
getch() ;
}

```

 Enter the Employee Name: George.  
Enter the Basic Pay: 8000.  
Enter the Allowance: 9000.  
Enter the Employee Name: Bill.  
Enter the Basic Pay: 8500.  
Enter the Allowance: 7000.

Employee Name	Basic	Allowance	
George	8000	9000	17000
Bill	8500	7000	1550
Total	16500	16000	32500

Дээр тодорхойлсон наиз функцийн хувьд доорх шиг түрвэн өөр загвар байж болно.

- friend employee addpay(employee e1, employee e2);
- friend employee addpay(employee &e1,employee &e2);
- friend employee addpay(const employee &e1, const employee &e2);

Одгээрээс сүүлийн 2 загвар функции параметрээ заалтаар дамжуулж авна. Ийм арга нь ой хэмиж программын хурлыг нэмэх зэрэг сайн талтай болохыг билд омни үзсн. Гэхдээ ийм параметрийн хувьд цаад эх огогдол рүү шууд хандаж болох учир түүнийг функцияа дотроос өөрчлөх боломжтой байдаг. Иймд егөгдлийг өөрчлөх шаардлагагүй тохиолдолд санамсаргүй өөрчлөлтөөс const тусгай уг хэрэглэн сэргийлж болно.

Найз функцийг дараах байдлаар заалтын оператор хэрэглэн зарлаж, улмаар тодорхойлох боломжгүй байдаг.

- friend employee & addpay(employee e1, employee e2);
- friend employee & addpay(employee &e1, employee &e2);
- friend employee & addpay(const employee &e1, const employee &e2);

Найз функцийг ихэвчлэн олон классын private өгөгдөл рүү зэрэг хандах функц хэрэгтэй болоход хэрэглэн.

```
//Program #8.2
#include <iostream.h>
#include <string.h>

class wife;

class husband
{
private:
    char name[20];
    int age;
    int income;
public:
    husband(void);
    ~husband(void);
    husband(char * string, int a, int c);
    void set_husband_data(char *string,int a, int c);
    void show_husband_data(void);
    friend int total_income(husband h, wife w);
};

husband::husband(void)
{
}

husband::~husband(void)
{
}

husband::husband(char *string, int a, int c)
{
    strcpy(name, string);
    age = a ;
    income = c ;
}

void husband::set_husband_data(char *string, int a,int c)
{
    strcpy(name, string);
    age = a ;
    income = c ;
}

void husband::show_husband_data(void)
{
    cout << "Name of husband: " << name << "\n" ;
    cout << "Age of husband: " << age << "\n" ;
    cout << "Income of husband: " << income << "\n" ;
}

class wife
{
private:
    char name[20];
    int age;
    int salary;
public:
    wife(void);
```

```

~wife(void);
wife(char *string, int a, int sal);
void set_wife_data(char *string, int a, int sal);
void show_wife_data(void);
friend int total_income(husband h, wife w);
};

wife::wife(void)
{
}

wife::~wife(void)
{
}

wife::wife(char * string, int a, int sal)
{
    strcpy(name, string);
    age = a ;
    salary = sal ;
}

void wife::set_wife_data(char *string, int a, int sal)
{
    strcpy(name, string);
    age = a ;
    salary = sal ;
}

void wife::show_wife_data(void)
{
    cout << "Name of wife: " << name << "\n" ;
    cout << "Age of wife: " << age << "\n" ;
    cout << "Salary of wife: " << salary << "\n" ;
}

int total_income(husband h, wife w)
{
    return(h.income+w.salary);
}

void main()
{
    husband h("Jack", 30, 15000) ;
    wife w("Dunia", 28, 12000) ;
    cout << "\nData about husband:\n";
    h.show_husband_data() ;
    cout << "\nData about wife:\n";
    w.show_wife_data() ;
    cout << "\nTotal family income: ";
    cout << total_income(h, w) ;
}

```

**[ ]** Data about husband:  
 Name of husband: Jack  
 Age of husband: 30  
 Income of husband: 15000

Data about wife:  
 Name of wife: Dunia  
 Age of wife: 28  
 Salary of wife: 12000

Total family income: 27000

Дээрх програмд *wife* классыг түүний тодорхойлтоос нь омно байх *husband* классын тодорхойлолт дотор хэрэглэх тул *class wife*; гэж эхэлж зарлана. Ингэснээр найз функцийг *husband* класс дотор хэрэглэх боломжтой болно. Хэрэв *wife* классын тодорхойлолт *husband* классынхаас өмнө байвал *husband* классыг *class husband*; гэж эхэлж зарлах ёстой. Энд байгаа *int total\_income(husband h, wife w)* бол *husband, wife* гэсэн хоёр өөр классын объектуудыг гаднаас авч хэрэглэх ерийн C++ функциюм.

Найз функцийг нь зөвхөн ерийн C++ функцийг бус, бас классын гишүүн функцийг байж болно. Жишээ нь, *wife* классын *total\_income* функцийг *husband* класс дотор найз функцизэр зарлах замаар түүний *private* өгөгдэл рүү шууд хандах эрхтэй болгож болно. Гэхдээ энэ шинжийг C++ хэлний зарим компайлдер, тухайлбал Turbo C++, MS Visual C++ зэрэг нь зөвшөөрдөггүй байна. Классын гишүүн функцийг өөр классын найз функцийг байх боломжийг доор үзүүллээ.

```
//Program #8.3
//Turbo C++ does not allow to use friend member function

#include <iostream.h>
#include <string.h>

class husband;

class wife
{
private:
    char name[20];
    int age;
    int salary;
public:
    wife(void);
    ~wife(void);
    wife(char *string, int a, int sal);
    void set_wife_data(char *string, int a, int sal);
    void show_wife_data(void);
    int total_income(husband &h);
};

int wife::total_income(husband &h)
{
    return(salary+h.income);
}

class husband
{
friend int wife::total_income(husband &h);
private:
    char name[20];
    int age;
    int income;
public:
    husband(void);
    ~husband(void);
    husband(char * string, int a, int c);
    void set_husband_data(char *string, int a, int c);
    void show_husband_data(void);
};
```

```

класс void main()
{
    wife w;
    husband h;
    cout << w.total_income(h);
    getch();
}

```

## 8.2 Найз класс

Классын гишүүн, гишүүн бус функцийн өөр классын найз байж болох талаар өмнөх бүлэгт үзсэн. Мөн `friend` үгийг классын түвшинд хэрэглэснээр тухайн класс бүхэлдээ бусад классын найз класс болно. Найз класс тодорхойлоходоо доор жагсаасан дурмийг мөрдөнө.

- `friend` үгийг классын тодорхойлолт дотор хэрэглиз.
- Класс оорийнхөө бүх найз классыг мэдэж байх ёстой.
- `public` биш огогдолтгай класс бусад классыг найзаа гэж зарлаж болно. Харин `public` огогдолтгай класс бусыг оорийнхөө найз гэж зарлах шаардлага байхгүй, учир нь ийм огогдолд программын аль ч хэсгээс шууд хандаж болдог. Класс нь бусад классыг найз гэж зарласнаар бусдаас хамгадах ёстой "нандин зүйлдээ", өгөгдлийн хандах давуу эрхийг бусад класст олгож байгаа юм.
- Класс өөрийгөө бусдын найз гэж зарлаж чадахгүй.
- Найз класс түүнийг найз гэж зарласан классынхаа омни, хойно хаана ч байж болно, учир нь тэдгээрийн дараалал чухал бус. Гэхдээ ийм тохиолдоод C++ объектыг тодорхойллоос нь өмнө хэрэглэх бол заавал зарлах ёстой байдаг дурмийг баримтална.
- Найз классаас удамшиж класс нь эх классынхаа найз классын найз биш, оороор хэлбэр эх классын найз нь удамшиж классын найз биш байдаг. Энэ нь авв, эсэжийн нь найз тэдний хүүгийн, охины нь найз биш байдагтай ижил юм.
- Тодорхойлолтдоо бусад классыг найз гэж зарласнаар түүний `private` огогдолд найз класс нь шууд хандаж боломжтой болно. Харин эсрэг хандалт байхгүй, тухайлбал класс найз классынхаа огогдолд хандаж чадахгүй. Классууд огогдолдоо харилцан ханддаг байхын тулд биеz харилцан найз гэж зарлах шаардлагатай байдаг.

Найз классын тодорхойлохтой холбогдох жишээг доор үзүүллээ.

```

class aClass
{
private:
    float value;
public:
    aClass ()
    {
        value=3.14;
    }
};

class bClass
{
private:
    aClass A;
public:
    void show(void)
    {
        cout << A.value ;
    }
};

```

Энд bClass классын private өгөгдол нь aClass классын A объект юм. Эдгээр класс хоорондоо харилцан холбоогүй учир aClass классын гишүүн функциятоос bClass классын A объект руу түүний өгөгдөл рүү хандаж чадахгүй, харин aClass нь bClass классыг доор үзүүлсэн шигээр өөрийн наиз гэж зарласнаар private, protected өгөгдлөө хандах эрхийг түүнд олгоно.

```
class aClass
{
    friend class bClass;
private:
    float value;
public:
    aClass ()
    {
        value=3.14;
    }
};
```

Мен хоёр класс доор үзүүлсэн шигээр харилцан бие биээ наиз гэж зарлаж болох ба ингэнээр өөрсдийн private, protected өгөгдлөө хандах эрхийг харилцан бие биедээ олгоно.

```
class aClass
{
    friend class bClass;
};

class bClass
{
    friend class aClass;
};

friend
```

тусгай үгийг хэрэглэснээр OOP технологийн гол шинжүүдийн нэг болох бусдаас огогдлийг далдалж зөвхөн класс дотроо хэрэглэх шинжийг ямар нэгэн хэлбэрээр зөрчиж байгаа юм. Энэ нь модулийн аргаар програм бичиж байгаа ч goto команд хэрэглэхтэй адил юм. Иймд OOP технологи хэрэглэж байгаа бол friend үгийг аль болохоор хэрэглэхгүй байх ёстой.

### 8.3 Оператор дахин тодорхойлох

C++ хэлний int, char гэх зэрэг дотоод суурь төрлийн өгөгдөл дээр +, -, \*, +=, ... гэх мэтнийн операторыг хэрэглэн одон зүйлийн бодолт хийж бодлог. Иймээс C++ программын доторх

```
int ii, ii1=10, ii2=18 ;
ii = ii2 + ii3 ;
double dd, dd1, dd2 ;
dd1 = 10.6 ;
dd2 = 15.5 ;
dd = dd1 + dd2 ;
```

мэтийн командыг C++ компайлэр шууд тайлж уншаад холбогдох үйлдлийг хийдэг байна. Ингэхдээ ямар ч ишмэлт заавар, тайлбар, командыг програм хөгжүүлэгчээс шаардлагтуй. Харин хэрэглэгчийн тодорхойлсон зохиомол төрлийн хувьд C++ хэлний үндсэн операторуудыг дээрхийн адиллаар хэрэглэж бодлогтуй. Тухайлбал, хэрэв програмд

```
employee e1, e1, e2 ;  
e = e1 + e2 ;
```

мэтийн команд байвал түүнийг C++ компайлер тайлж уншиж чадахгүй бөгөөд энэ тухайгаа буцааж мэдээлнэ. Иймээс, жишээ нь

```
o e3.addpay(e1, e2) ;  
o e3 = e1.addpay(e2) ;  
o e = addpay( e1, e2) ;
```

түрвэн боломжоос аль нэгийг нь сонгож хэрэглэх замаар e1, e2 гэсэн хоёр объектын нийлбэрийт олох шаардлага гардаг. Эхний e3.addpay(e1, e2); командин addpay() нь employee классын гишүүн функц байх ба гаднаас авах хоёр объектын нийлбэрийт олж идэвхтэй байгаа e3 объект руу хийнэ. Удаах e3=e1.addpay(e2); командин addpay() нь идэвхтэй объект, гаднаас авах объект хоёрын нийлбэрийг олж гуравдаахаа e3 объект руу хуулах үүрэгтэй гишүүн функц юм. Сүүлийн e=addpay(e1, e2); командин addpay() нь ямар ч классын гишүүн бус функц байх ба гаднаас авах хоёр объектын нийлбэрийг олж буцаана.

Дээрх түрвэн боломжийн аль нь ч e=e1+e2; илэрхийллийн адил уншууртай бишээс гадна зохиомол төрлийн хувьд e=e1+e2; шиг илэрхийллийг эгэл нехцэлд C++ компайлер зөвшөөрдөгтүй. Харин операторыг дахин тодорхойлохтой холбогдох C++ хэлний арга технологийг хэрэглэн (+) нэмэх операторын гүйцэтгэх үүргийг дахин тодорхойлж өгснөөр доор үзүүлсэнтэй адил үйлдлийг зохиомол төрлийн өгөгдлийн хувьд хийх боломжтой болно.

```
e = e1 + e2 ;  
char s1[100] ;  
char s2 [] = "Test" ;  
char s3 [] = "C++" ;  
s1 = s2 + s3 ;  
int num1[10], num2[10], num[10] ;  
num = num1 + num2 ;
```

### 8.3.1 Ганц operandын операторыг дахин тодорхойлох

C++ хэлэнд байдаг ганц operandын ++, -- хоёр оператор нь operandынхаа утгыг харгалзан нэгээр нэмэгдүүлэх, корогдуулах үүрэгтэй. Ийм операторыг класс дотор хэрхэн хэрэглэж болохыг дараах програмаар харуулъя.

```
//Program #8.4  
//use of unary operators  
  
#include <iostream.h>  
class counter  
{  
    private:  
        int count;  
    public:  
        counter(void) ;  
        ~counter(void) ;  
        int get_count(void) ;  
        void increment_count(void) ;  
};  
counter::counter(void)  
{  
    count = 0 ;  
}
```

```

counter::~counter(void)
{
}

int counter::get_count(void)
{
    return (count) ;
}

void counter::increment_count(void)
{
    count++ ;
}

void main(void)
{
    counter c1, c2;
    cout << "\nc1=" << c1.get_count() ;
    cout << "\nc2=" << c2.get_count() ;
    cout << endl ;
    c1.increment_count() ;
    c1.increment_count() ;
    c1.increment_count() ;

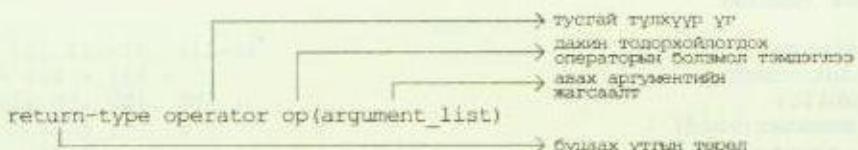
    c2.increment_count();
    cout << "\nc1=" << c1.get_count() ;
    cout << "\nc2=" << c2.get_count() ;
}

```

 c1=0  
c2=0  
  
c1=3  
c2=1

Дээрх программын main() функц доторх c1.increment\_count(); бол c1 объектын count гишүүний уттыг нэгээр иzmэгдүүлэх үүрэгтэй функц юм. Тэгвэл түүний орона c1++; гэж бичиж болох уу? Учир нь, c1++; нь бичихэд энгийнээс гадна c1 объектын уттыг нэгээр иzmэгдүүлэх гэсэн санааг илэрхийлэх талаар дээрх функцээс илүү юм. Зохиомол төрлийн объектын хувьд c1++; шиг үйлдэл хийдэг байхын тулд ++ операторыг дахин тодорхойлох замаар гүйцэтгэх ажил үүргийг нь өөрчлөх шаардлагатай болдог.

Оператор дахин тодорхойлоход зарим нэгэн иzmэлт зүйл, иzmэлт функц хэрэгтэй. Тухайлбал, тусгай operator гишүүн функцийг Зураг 8.1-д үзүүлсэн загварын дагуу заавал хэрэглэн.



Зураг 8.1: Ганц операндын операторыг дахин тодорхойлох загвар

Ганц операндын ++ операторыг дахин тодорхойлох гишүүн функцийг дээрх загварын дагуу void operator ++(void); гэж зарлах бөгөөд ийм мерийг тодорхойлолт дотор нь иzmж оруулах замаар омнох counter классыг доор үзүүлснээр өөрчлөн бичиж болно.

```

//Program #8.5
#include <iostream.h>
class counter
{
private:
    int count;
public:
    counter(void) ;
    ~counter(void) ;
    int get_count(void) ;
    void operator ++(void) ;
};

counter::counter(void)
{
    count = 0 ;
}

counter::~counter(void)
{
}

int counter::get_count(void)
{
    return (count) ;
}

void counter::operator ++(void)
{
    count++ ;
}

```

Дээрх counter классыг хэрэглэх main() функцийг дараах байдлаар бичнэ.

```

void main(void)
{
    counter c1, c2;
    cout << "\nc1=" << c1.get_count() ;
    cout << "\nc2=" << c2.get_count() ;
    cout << endl ;
    c1++ ;
    c1++ ;
    c1++ ;
    c2++ ;
    cout << "\ncl=" << c1.get_count();
    cout << "\nc2=" << c2.get_count();
    //c2=c1++ ;
}

```

 c1=0  
c2=0

c1=3  
c2=1

counter классын c1 объектын хувьд c1++; үйлдэл хийгдэх үед void operator ++(void) гишүүн функци дуудагдана. Иймд c1++; командын мөрийг c1.operator ++(); гэж бичиж өгсөнтэй ижил юм. Мөн ++c1; үйлдлийн үед void operator

`++(void);` гишүүн функци бас дуудагдах тул `++c1` командин мөрийг `c1.operator ++();` гэж бичиж болно.

Дээрхээс үзэхэд `c1++` ба `++c1` хоёрын хооронд ялгаа огт байхгүй. Харин дээрх програмд тайлбар болгосон байгаа `c2=c1++;` шиг командин хувьд зөв бичих дурмийн алдаа гардаг. Учир нь, `operator` функцийг `void` утга буцаадаг байхаар тодорхойлсон байгаа, гэтэл (=) утта оноох операторынхоор бол `operator` функцийн `counter` төрлийн утга буцах ёстой. Иймийн учир, дээрх шигээр дахин тодорхойлсон `++` операторыг `c2=c1++` шиг командин хувьд хэрэглэх боломжгүй, зөвхөн дангаар нь `c1++, c2++` шиг командин хувьд хэрэглэж болно. Харин, жишээ нь `c2=c1++;` командинх шиг үйлдэл хийх боломжтой байхын тулд `operator` гишүүн функцийг өөрчлөх шаардлагатай. Тухайлбал, `operator` функци нь `counter` утга буцаадаг байхаар `void operator ++(void)` мөрийг `counter operator ++(void)` болгож өөрчлох шаардлагатайг дараах программаас харж болно.

```
//Program #8.6
#include <iostream.h>
class counter
{
private:
    int count ;
public:
    counter(void) ;
    ~counter(void) ;
    int get_count(void) ;
    counter operator ++(void) ;
};

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

Дээрх классын бусад гишүүн функцийн тодорхойлолт өмнөх Program #8.5-д байгаатай адилхан. Харин `counter` классыг хэрэглэх `main()` функцийг дараах байдлаар өөрчилж бичье.

```
void main(void)
{counter c1, c2;
 cout << "\nc1=" << c1.get_count();
 cout << "\nc2=" << c2.get_count() ;
 cout << endl ;
 c1++ ;
 c1++ ;
 c1++ ;
 c2++ ;
 cout << "\nc1=" << c1.get_count() ;
 cout << "\nc2=" << c2.get_count() ;
 c2=c1++ ;
 cout << endl ;
 cout << "\nc1=" << c1.get_count() ;
 cout << "\nc2=" << c2.get_count() ;
}
```

```
c1=0  
c2=0  
  
c1=3  
c2=1  
  
c1=4  
c2=4
```

Програмын үр дүнгийн хэсэгт байгаа `c2=4` мөрийн хувьд C++ хэлний `++` операторын дүрэмтэй холбоотойгоор ойлгомжгүй зөрчилтэй байдал үүсч болох талтай. Уг хэлний хувьд `++` нь бусад оператортой хамт хэрэглэгдэх тохиолдолд хийгдэх дарааллаасаа болж утвар (`prefix`), дагавар (`postfix`) гэсэн хоёр янз байдаг. Гэтэл `void operator ++(void)` гишүүн функцийн хувьд утвар, дагавар `++` нь адил юм. Иймээс `c2=c1++;` командаын хувьд C++ хэлний зарим дүрэм зорчигддэг байна.

Дээрх програмд хэрэглэх `operator` гишүүн функцийн counter `operator ++(void);` тодорхойлолтыг бөөрөөр

```
counter counter::operator ++(void)  
{  
    count++ ;  
    return counter(count) ;  
}
```

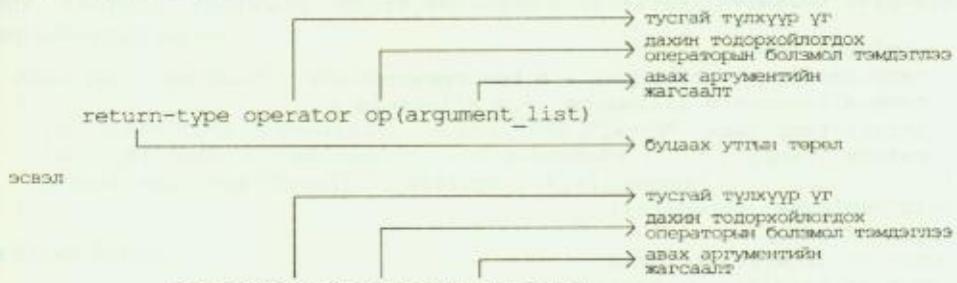
гэж бичиж болох ба ингэснээр командаын мөрийн тоо 4 байснаа 2 болж багасна. Энэ арга нь завсрлын нэргүй объект байгуулж түүнийг буцаана. Гэхдээ үүнтэй холбогдуулан параметртэй байгуулагч функцийг `counter` классын тодорхойлолтод

```
counter(int x);  
counter::counter(int x )  
{  
    count = x ;  
}
```

гэж нэмж оруулах шаардлагатай.

### 8.3.2 Хос operandын операторыг дахин тодорхойлох

Хос operandын операторыг ганц operandын операторынх шигээр Зураг 8.2-т үзүүлсэн загварын дагуу дахин тодорхойлж болно.



Зураг 8.2: Хос operandын операторыг дахин тодорхойлох загвар

Энд `employee` классын хоёр объектын нийлбэрийг олж гуравдахъ объектод `e0=e1+e2;` командаар хийх боломжтой байхаар `+` операторыг дахин тодорхойлохын тулд `employee`

operator +(employee e) функцийг employee классын тодорхойлолтод иzmж оруулна.  
Үүний хэрхэн програмчлахыг доорх програмл үзүүллээ.

```
//Program #8.7
//Creating default constructor with arguments

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a) ;
    employee() ;
    ~employee() ;
    void showdata() ;
    employee operator +(employee e) ;
};

employee::employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}
employee::~employee()
{
}

employee::employee(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

employee employee::operator +(employee e)
{
    employee temp ;
    temp.basicpay = basicpay + e.basicpay ;
    temp.allowance = allowance + e.allowance ;
    strcpy(temp.name, "Total") ;
    return (temp) ;
}

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
```

```

cout << setw(20) << "Employee Name" ;
cout << setw(8)  << "Basic" ;
cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee e1("George", 9000, 8000);
    employee e2("Bill", 8000, 9000);
    employee e0 ;
    heading() ;
    e1.showdata() ;
    e2.showdata() ;
    e0 = e1 + e2 ;
    e0.showdata() ;
    getch() ;
}

```

Employee Name	Basic	Allowance
George	9000	8000
Bill	8000	9000
Total	17000	17000

C++ компайлер дээрх програмын main() функцийн `e0=e1+e2;` командыг гүйцэтгэхдээ түүний + операторыг employee классын хоёр объект хооронд нэмэх үйлдэл хийх гэж тайлж уншина. Гэхдээ классын тодорхойлолт дотор байгаа

```
employee operator +(employee e);
```

функци нь e1 объектоос дуудагдахдаа e2 объектыг аргумент хэлбэрээр дамжуулж авна. Функцийн буцах утга нь e0 объектынх болно. Тэгхэлэр, `e0=e1+e2;` команда нь `e0=e1.operator +(e2);` командтай адил юм.

C++ компайлер `e1+e2` шиг үйлдлийг зөвшөөрдөг байхаар хос операндын операторыг дахин тодорхойлох тохиолдолд уг операторын зүүн талын объект e1 нь operator функцийг дуудаж хэрэглэнэ. Харин уг операторын баруун талын объект e2 нь operator функцийн аргумент болно.

Дээрх програмд хэрэглэсэн employee operator +(employee e); функцийн тодорхойлолтыг өөрөөр

```
employee::operator +(employee e)
{
    int t_basicpay = basicpay + e.basicpay ;
    int t_allowance = allowance + e.allowance ;
    return employee("Total", t_basicpay, t_allowance) ;
}
```

гэж бичиж болно.

Еренхийдээ, ийм operator функцийг дараах гурван янзаар

- o employee operator +(employee e) ;
- o employee operator +(employee &e) ;
- o employee operator +(const employee &e) ;

гэж тодорхойлох боломжтой.

### 8.3.3 Хоёр тэмдэгт мөрийг залгах

Өгөгдсөн хоёр тэмдэгт мөрийг хооронд нь залгаж шинэ тэмдэгт мөр гаргаж авах тохиолдолд + операторыг дахин тодорхойлох string классыг шинээр үүсгэж ийм класс бүхий програмыг доор үзүүлсэнээр бичнэ.

```
//Program #8.8
//C++ program to create a string class

#include <iostream.h>
#include <string.h>

const int SIZE = 100 ;

class string
{
private:
    char str[SIZE] ;
public:
    string(void) ;
    string(char *ss) ;           //string(char[]) гэж бичиж болно.
    void showstring(void) ;
    string operator +(string s);
};

string::string(void)
{
    str[0] = '\0' ;
}

string::string(char *ss)
{
    strcpy(str, ss) ;
}

void string::showstring(void)
{
    cout << str << endl ;
}

string string::operator +(string s)
{
    string temp ;
    if((strlen(str) + strlen(s.str)) < SIZE)
    {
        strcpy(temp.str, str) ;
        strcat(temp.str, s.str) ;
    }
    else
        cout << "String overflow" << endl ;
    return temp ;
}

void main()
{
    string s1 = "Hello world, " ;
    string s2 = "Hello beautiful world " ;
    string s3 = s1 + s2 ;
    cout << "\ns1= " ;
```

```

    s1.showstring() ;
    cout << "s2= " ;
    s2.showstring() ;
    cout << "s3= " ;
    s3.showstring() ;
}

└─ s1= Hello world,
   s2= Hello beautiful world
   s3= Hello world, Hello beautiful world

```

### 8.3.4 Хоёр объектыг жиших

Хоёр объектыг хооронд нь харьцуулахын тулд < операторыг дахин тодорхойлох шаардлагатай. Ингэснээр, жишээ нь employee классын e1, e2 гэсэн хоёр объектын хооронд

```

if( e1 < e2)
{
    // ...
}

```

шиг харьцуулалт хийх боломжтой болно. Үүний тулд,

```
boolean48 operator <(employee e);
```

гэсэн гишүүн функцийг employee классын тодорхойлолтод нэмж оруулна. Ийм функцийн бүхий employee классыг хэрэглэх жишээ програмыг дараах байдлаар бичнэ.

```

//Program #8.9
// C++ program to create a string class with
// operator function

enum boolean {false, true} ;

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a) ;
    employee();
    ~employee();
    void showdata() ;
    boolean operator <(employee e) ;
};

employee::~employee()
{

```

---

<sup>48</sup>boolean нь нэрлэсэн тогтмол: enum boolean {false, true};

```

employee::employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}
employee::employee(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}
boolean employee::operator <(employee e)
{
    int t1 = basicpay + allowance ;
    int t2 = e.basicpay + e.allowance ;
    return (t1<t2) ;
}
void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8) << basicpay ;
    cout << setw(12) << allowance ;
}
void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8) << "Basic" ;
    cout << setw(12) << "Allowance" ;
}
main()
{
    clrscr() ;
    employee e1 ("George", 9000, 8000) ;
    employee e2 ("Bill", 8000, 6000) ;

    heading() ;
    e1.showdata() ;
    e2.showdata() ;
    if(e1 < e2)
        cout << "\ne1 is less than e2" ;
    else
        cout << "\ne1 is greater than or equal to e2" ;
    getch() ;
}

```

	Employee Name	Basic	Allowance
	George	9000	8000
	Bill	8000	6000

e1 is greater than or equal to e2

### 8.3.5 Хоёр объектын тэнцүү эсэхийг шалгах

C++ хэлний тэмдэгт мөрүүд хоорондоо ижил эсэхийг `s1 == s2` шиг илэрхийллээр шалгаж болдоггүй. Харин омнох бүлэгт тодорхойлсон `string` класс дотор `==` операторыг дахин тодорхойлж хэрэглэснээр уг үйлдлийг хийж болохыг дараах програмаар харууллаа.

```
//Program #8.10
//C++ program to check whether two strings are identical

enum boolean {false, true} ;

#include <iostream.h>
#include <string.h>

const int SIZE = 100 ;

class string
{
private:
    char str[SIZE] ;
public:
    string(void) ;
    string(char *ss) ;
    void showstring(void) ;
    void getstring(void) ;
    boolean operator ==(string s);
} ;

string::string(void)
{
    str[0] = '\0' ;
}

void string::getstring(void)
{
    cin.get(str, SIZE) ;
}

string::string(char *ss)
{
    strcpy(str, ss) ;
}

void string::showstring(void)
{
    cout << str << endl ;
}

boolean string::operator ==(string s)
{
    return (strcmp(str, s.str) == 0 ? true : false) ;
}

void main()
{
    string s1 = "yes" ;
    string s2 = "no" ;
    string s ;
    cout << "Enter 'yes' or 'no' at the keyboard: " ;
    s.getstring() ;
```

```

if( s == s1)
    cout << "You typed 'yes'" ;
else
    if(s == s2)
        cout << "You typed 'no'" ;
    else
    {
        cout << "You typed " ;
        s.showstring() ;
    }
}

```

Enter 'yes' or 'no' at the keyboard: yes  
You typed 'yes'

### 8.3.6 Нэмээд буцааж утга оноох нийлмэл операторыг дахин тодорхойлох

Нэмээд буцааж утга оноох нийлмэл операторыг дахин тодорхойж employee объектол хэрэглэх жишээ програмыг (+=) операторын жишээн дээр үзүүллээ.

```

//Program #8.11
//Creating operator +=() function

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20] ;
    int basicpay ;
    int allowance ;
public:
    employee(char n[], int b, int a) ;
    employee();
    ~employee();
    void showdata() ;
    void operator +=(employee e) ;
};

employee::employee()
{
    strcpy(name, "") ;
    basicpay = 0 ;
    allowance = 0 ;
}

employee::~employee()
{
}

employee::employee(char n[], int b, int a)
{
    strcpy(name, n) ;
    basicpay = b ;
    allowance = a ;
}

```

```

void employee::operator+=(employee e)
{
    basicpay += e.basicpay ;
    allowance += e.allowance ;
}

void employee::showdata()
{
    cout << endl ;
    cout << setw(20) << name ;
    cout << setw(8)  << basicpay ;
    cout << setw(12) << allowance ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee e1 ("George", 9000, 8000);
    employee e2 ("Bill", 8000, 6000);
    heading() ;
    e1.showdata() ;
    e2.showdata() ;
    cout << endl ;
    e1 += e2 ;
    e1.showdata() ;
    e2.showdata() ;
    getch() ;
}

```



	Employee Name	Basic	Allowance
	George	9000	8000
	Bill	8000	6000
	George	17000	14000
	Bill	8000	6000

#### 8.4 Найз функци ба дахин тодорхойлох оператор

Операторыг дахин тодорхойлоход хэрэглэх operator функци нь классын гишүүн функци байж болдгийг өмнөх бүлгүүдэд үзсэн билээ. Ийм оператор функцийн бараг ихэнхийг бас найз функци шигээр хэрэглэж болно. Энэ тохиолдолд ганц операндын оператор функци нэг параметр, хос операндынх хоёр параметр гаднаас авах хэрэгтэй болно.

Program #8.6-д тодорхойлсон counter классын counter operator ++(void); функцийг friend counter operator ++(counter c); гэсэн функцийн замаар уг програмыг доор үзүүлснээр өөрчлөн бичнэ.

```

//Program #8.12
class counter
{
    private:
        int count;

```

```

public:
    counter(void) ;
    ~counter(void) ;
    int get_count(void) ;
    friend counter operator ++(counter c) ;
};

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

```

Энэ классын бусад бүх гишүүн функцийн тодорхойлолт өмнөх Program #8.6-д байгаатай ижил. Харин counter классыг хэрэглэх main() функцийг дараах байдлаар тодорхойлно.

```

void main(void)
{
    counter c1, c2;
    cout << "\nc1 = " << c1.get_count() ;
    cout << "\nc2 = " << c2.get_count() ;
    cout << endl ;
    c1++ ;
    c1++ ;
    c1++ ;
    c2++ ;
    cout << "\nc1 = " << c1.get_count() ;
    cout << "\nc2 = " << c2.get_count() ;
    c2=c1++ ;
    cout << endl ;
    cout << "\nc1 = " << c1.get_count() ;
    cout << "\nc2 = " << c2.get_count() ;
}

c1 = 0
c2 = 0

c1 = 0
c2 = 0

c1 = 0
c2 = 1

```

C++ компайлер main() функци доторх c1++; команд counter объектын утсыг нэгзэр нэмэгдүүлэх ёстой гэдгийг тайлж уншаад ийм зориулалтын гишүүн функци counter классын тодорхойлолтод байгаа эсэхийг эхэлж тодруулна. Ингэхдээ, counter класс ийм гишүүн функцийг, харин ийм зориулалтын friend функци олдвол түүнийг c1 аргументтэйгээр дуудна. Тэгэхлээр, c1++; команд бол operator ++(c1) командтай ижил юм.

```

c1 = 0
c2 = 0

c1 = 3
c2 = 1
c1 = 4
c2 = 4

```

Энэ програмын үр дүнг Program #8.6-гийнхтай харьцуулж үзэхэд ялгаатай байна. Эдгээр програмын main() болон operator ++() функцийдийн бүх команд утгын хувьд харгалзан ижил хэдий ч ялгаатай үр дүн гарч байгаа нь ямар нэгэн зүйл буруу байгааг харуулж байна. Гарах хариуны ийм зорөттэй байдал нь friend оператор функцийн загвар юм уу тодорхойлолт алдаатай эсвэл операторыг дахин тодорхойлоход friend функцийг огт хэрэглэж болохгүйтэй холбоотой байж болох юм. Гэвч програмыг нарийвчлан шалгаж нягталснаар friend оператор функци логик алдаатайг тогтоож болно. Тухайлбал, дээр дурдсаныар c1++; команд нь operator ++(c1) командтай ижил. Гэвч оператор функцийн тодорхойлолтод зааж отгний дагуу c1 объектыг утгаар нь дамжуулах тул энэ оператор функци уг объектын хуулбарын утгыг c.count++ командаар нэмэгдүүлнэ. Иймээс эх объектын утга нь c1++ командаар огт өөрчлөгдхүй. Энэ асуудлыг friend оператор функцийн загвар, тодорхойлолтод дараах өөрчлөлтийг, тухайлбал функцийн загварт

```
friend counter operator ++(counter &c);  
функцийн тодорхойлолтод  
counter operator ++(counter &c)  
{  
    c.count++;  
    counter temp;  
    temp.count = c.count ;  
    return (temp);  
}
```

гэж оруулах замаар шийдэж болно. Ингэж тодорхойлогдох оператор функцийн параметр нь эх объектын заалт (counter &c) байх тул объектын утгыг c1++ командаар өөрчлөх боломжтой болно. Ингэснээр, дээрх програмын үр дүн Program #8.6-гийнхтай ижил болно.

Сүүлд тодорхойлсон friend оператор функцийн загварыг

```
friend counter & operator ++(counter &c);  
тодорхойлолтыг  
counter & operator ++(counter &c)  
{  
    c.count++;  
    return (c);  
}
```

гэж өөрөөр бичиж болно.

Энэ аргын дагуу заалтыг аргумент хэлбэрээр авч заалт буцаах ямар ч функцийг тодорхойлох бололцоотой.

Хос операндын оператор функцийг мөн friend функци хэлбэрээр тодорхойлж болно. Ингэхдээ, жишээ нь employee operator +(employee e); оператор гишүүн функцийн оронд доор үзүүлэх гурван загварын аль нэгний нь дагуу тодорхойлогдох friend функци хэрэглэнэ.

```
friend employee operator +(employee e1, employee e2) ;  
жэвэл  
friend employee operator +(employee &e1,employee &e2);  
эсвэл
```

```
friend employee operator +(const employee &e1, const employee &e2) ;
```

Эхний загварын дагуу тодорхойлогдох friend функц бүхий employee классын жишээ програмыг дараах байтлаар бичин.

```
//Program #8.13
//Creating default constructor with arguments

#include <iostream.h>
#include <conio.h>
#include <iomanip.h>
#include <string.h>

class employee
{
private:
    char name[20];
    int basicpay;
    int allowance;
public:
    employee(char n[], int b, int a);
    employee();
    ~employee();
    void showdata();
    friend employee operator +(employee e1, employee e2);
};

employee::employee()
{
    strcpy(name, "");
    basicpay = 0;
    allowance = 0;
}

employee::~employee()
{
}

employee::employee(char n[], int b, int a)
{
    strcpy(name, n);
    basicpay = b;
    allowance = a;
}

employee operator +(employee e1, employee e2)
{
    employee temp;
    temp.basicpay = e1.basicpay + e2.basicpay;
    temp.allowance = e1.allowance + e2.allowance;
    strcpy(temp.name, "Total");
    return (temp);
}

void employee::showdata()
{
    cout << endl;
    cout << setw(20) << name;
    cout << setw(8) << basicpay;
    cout << setw(12) << allowance;
}
```

```

void heading()
{
    cout << endl ;
    cout << setw(20) << "Employee Name" ;
    cout << setw(8)  << "Basic" ;
    cout << setw(12) << "Allowance" ;
}

main()
{
    clrscr() ;
    employee e1 ("George", 9000, 8000) ;
    employee e2 ("Bill", 8000, 9000) ;
    employee e0 ;
    heading() ;
    e1.showdata() ;
    e2.showdata() ;
    e0 = e1 + e2 ;
    e0.showdata() ;
    getch() ;
}

```

Employee Name	Basic	Allowance
George	9000	8000
Bill	8000	9000
Total	17000	17000

C++ компайлер  $e0=e1+e2$ ; команда гүйцэтгэхдээ нэмэх үйлдлийг employee классын хоёр объект дээр хийхээ "мэдэж" ийм зориулалтын оператор гишүүн функцийг классын тодорхойлолтоос хайна. Гэвч ийм функцийн оронд friend оператор функцияа олдох тул түүнийг  $e1$ ,  $e2$  гэсэн хоёр аргументтэй ажиллуулна. Иймд  $e0=e1+e2$ ; нь  $e0=operator+(e1, e2)$ ; командтай адил юм. Энд нэмэх операторын зүүн талын объект бол эхний аргумент, баруун талынх нь удаах аргумент нь болно.

### 8.5 << операторыг дахин тодорхойлох

Өмнө үзсэн бүх жишээ програмд классын объектын утгыг дэлгэц рүү гаргахдаа ийм ажил үүргийг хийдэг зориулалтын гишүүн функцийг нь хэрэглэсэн билээ. Тухайлбал, Program #8.8-д string классын  $s1$  объектын утгыг  $s1.showstring()$  функцияар хэвлэдэг. Тэгвэл уг командын оронд  $cout << s1$ ; шиг командыг хэрэглэж болох уу?  $cout$  бол ostream.h толгой файлд тодорхойлогдсон ostream классын объект юм. Эл классын тодорхойлолт дотор  $<<$  операторыг C++ хэлэнд байх дотоод суурь төрөл бүрийн хувьд дахин тодорхойлсон байдаг. Тэгэхлээр, int аргумент авах operator функцияа double аргумент авах operator функцияа байх жишээтэй. Харин cout нь зохиомол төрлийн объектыг танидаг байхын тулд шинэ operator функцияа нэмэх замаар ostream классын хуулбар үүсгэж болно. Гэвч стандарт толгой файлыг << operator функцияар string класс дотор нэмж оруулах юм. Гэвч ийм функцийг объектоор дамжуулж дуудах учир  $s1 << cout$ ; гэж хэрэглэх шаардлагатай болох ч энэ хэлбэр нь бидний хүсч байгаа  $cout << s1$  хэлбэрээс ялгаатай байна. Бас friend функцияа хэрэглэх замаар энэ асуудлыг шийдэж болох багаад ийм функцийн дараах гурван загвар байж болох талтай.

- o friend void operator <<(ostream os, string s) ;
- o friend void operator <<(ostream &os, string &s) ;

```
o friend void operator <<(ostream os,const string s);
```

Эдгээрээс эхний нь хэрэглэхэд илүү тохиromжтой энгийн загвар тул түүнийг доор үзүүлсэн шигээр тодорхойльб.

```
void operator <<(ostream os, string s)
{
    os << s.str << "\n" ;
}
```

Дээрх шигээр тодорхойлон operator функц нь s1 << cout командыг боловсруулж чадах эсэхийг Program #8.14-өөр шалгай.

```
//Program #8.14
//C++ program to create a string class

#include <iostream.h>
#include <string.h>

const int SIZE = 100 ;

class string
{
private:
    char str[SIZE] ;
public:
    string(void) ;
    string(char *ss) ;           //string(char()); гэж бичиж болно.
    void showstring(void) ;
    friend void operator <<(ostream os, string s);
};

string::string(void)
{
    str[0] = '\0' ;
}

string::string(char *ss)
{
    strcpy(str, ss) ;
}

void string::showstring(void)
{
    cout << str << endl ;
}

void operator <<(ostream os, string s)
{
    os << s.str << "\n" ;
}

void main()
{
    string s1 = "Hello world, " ;
    string s2 = "Hello beautiful world " ;
    cout << "s1= " ;
    cout << s1 ;
    cout << "s2= " ;
    cout << s2 ;
    cout << "\n" ;
```

```
//cout << s1 << s2 ;  
}  
  
Compiler could not generate copy constructor for class 'ostream' Type  
mismatch in parameter 'os' in call to 'operator << (ostream, string)'
```

C++ компайлер Program #8.14-т operator функцийн параметрийг утгаар нь дамжуулж байгаатай холбоотойгоор дээрх алдааны мэдээллийг дэлгэцэнэ. Энэ асуудлыг шийдэхийн тулд доорх зүйлийг хийнэ. Тухайлбал, классын тодорхойлолт доторх

```
friend void operator <<(ostream os, string s);  
мерийг  
friend void operator <<(ostream &os, string &s);  
жээл  
friend void operator <<(ostream &os, const string &s);  
гэж өөрчилнэ. Мен operator функцийн тодорхойлолтыг дээрхтэй харгалзуулан  
void operator <<(ostream &os, string &s)  
{  
    os << s.str << "\n" ;  
}  
жээл  
void operator <<(ostream &os, const string &s)  
{  
    os << s.str << "\n" ;  
}
```

Болгож өөрчлөх замаар, жишээ нь Program #8.14A програмыг гарган авч ажиллуулахад дараах үр дүн гарна.

```
//Program #8.14A  
//C++ program to create a string class  
  
#include <iostream.h>  
#include <string.h>  
  
const int SIZE = 100 ;  
  
class string  
{  
private:  
    char str[SIZE] ;  
public:  
    string(void) ;  
    string(char *ss) ;           //string(char[]); гэж бичиж болно.  
    void showstring(void) ;  
    friend void operator <<(ostream &os, string &s) ;  
};  
  
string::string(void)  
{  
    str[0] = '\0' ;  
}  
  
string::string(char *ss)  
{
```

```

strcpy(str, ss) ;
}

void string::showstring(void)
{
    cout << str << endl ;
}

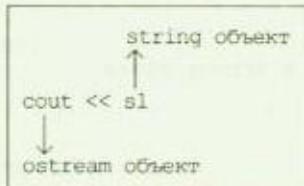
void operator <<(ostream &os, string &s)
{
    os << s.str << "\n" ;
}

void main()
{
    string s1 = "Hello world, " ;
    string s2 = "Hello beautiful world " ;
    cout << "s1= " ;
    cout << s1 ;
    cout << "s2= " ;
    cout << s2 ;
    cout << "\n" ;
    //cout << s1 << s2 ;
}

s1= Hello world,
s2= Hello beautiful world

```

Дээр үзүүлснээр дахин тодорхойлогдох operator функцийн нь << операторын зүүн талд ostream төрлийн объект, баруун талд нь string төрлийн объект байхыг шаардлаг (Зураг 8.3). Иймд уг оператор функцийн нь cout << s1; шиг нэг cout объект, нэг string объект бүхий командын хувьд зөв ажиллана.



Зураг 8.3: Дахин тодорхойлогдох << оператор

Нэг cout объектыг олон string объектын хувьд cout<<s1<<s2; мэтээр угсаагаар хэрэглэх тохиолдолд эл команд зүүнээсээ эхэлж хийгдэх тул энэ нь (cout<<s1)<<s2; гэж бичихтэй ижил юм. Иймд эхлээд хаалт доторх cout<<s1 хэсэг хийгдэнэ. Гэтэл дээр тодорхойлсон operator <<() функцийн cout объект буцахгүй учир cout<<s1<<s2; командын эхний хэсэг хийгдсний дараагаар үлдэх хэсэг нь (void)<<s2 гэсэн бүтэцтэй болж улмаар системийн алдаа гарна (Зураг 8.4).

```

(cout << s1) << s2;
      ↓
(void) << s2;

```

Зураг 8.4: Нэг cout объектыг олон string объекттой хэрэглэхэд үүсч байдаг

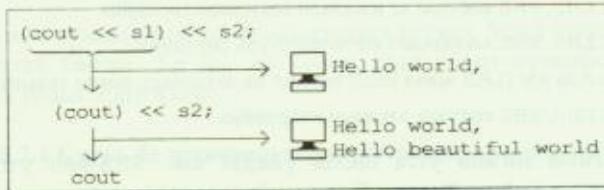
Иймд эхний cout илэрхийлэл нь ostream төрлийн cout объект буцааж cout<<s1<<s2; команд Зураг 8.5-д үзүүлсэн шиг дарааллаар хийгддэг байхаар operator <<() функцийг өөрчлөх ёстай. Үүний тулд friend operator функцийн загвар, тодорхойлолтыг дараах байдлаар өөрчлөх шаардлагатай. Тухайлбал,

- string классын тодорхойлолт доторх функцийн загварыг  

```
friend ostream & operator<<(ostream &os, string &s);
```
- Функцийн тодорхойлолтыг  

```
ostream & operator <<(ostream &os, string &s)
{
    os << s.str << "\n";
    return (os);
}
```

тож тус өөрчлөх ёстай. Дурдсан өөрчлөлтийг Program #8.14A-д хийгээд main() функцийн сүүлийн мөрийг буцааж командын мөр болгосны дараагаар ажиллуулахад үр дүн нь Зураг 8.5-д үзүүлсэнтэй адил болохыг харж болно.



Зураг 8.5: Нэг cout объектыг аллон string объекттой хэрэглэх

Program #8.14A-д хэрэглэж байгаа friend ostream & operator <<(ostream &os, string &s) функц нь зөвхөн string классын хувьд найз болохоос ostream классын хувьд найз биш юм. Иймээс ч operator функц ostream объектыг аргумент хэлбэрээр гаднаас авч байгаа юм. Энэ функц найз нь биш тул ostream классын private өгөгдөл хандаж чадахгүй ба ингэж хандах ч шаардлага Program #8.14A-гийн хувьд байхгүй байсан. Харин operator функц найз нь тул string классын private өгөгдөл хандаж чадах ба ийм боломжийг Program #8.14A-гийн хувьд ашигласан байна.

## 8.6 Оператор дахин тодорхойлоход анхаарах зүйл

Операторыг дахин тодорхойлоход тодорхой хязгаарлалт байна. Тухайлбал,

- Ойт байхгүй операторыг шинээр үүсгэж болохгүй.
- Дахин тодорхойлоходо эх операторын дүрмийг зорчиж болохгүй.
- Дахин тодорхойлогдох операторын ядаж нэг операнд нь зохиомол төрлийн байна.

Гэхдээ доор жагсаасан операторыг, тухайлбал

- . шууд сонголтын цг оператор.
- .\* гишүүний дам утгын од оператор.
- :: үйлчлэх хүрээний оператор.
- ?: нөхцөл шалгах оператор.
- sizeof объектын хэмжээг олж буцаах макро функц

зэргийг дахин тодорхойлж болдогтүй байна.

Харин = утга оноох, () функцийн, [] хүснэгтийн, → шууд бус сонголтын сүм оператор зэргийг зөвхөн operator гишүүн функц хэрэглэн дахин тодорхойлж болно.

### 8.7 Өгөгдөл хөрвүүлэг

C++ хэлэнд аль ч төрлийн утгыг түүнээс өөр төрлийн хувьсагч руу хийж болох ба ингэхдээ утга оноох операторын зүүн гар талд (LHS-Left Hand Side) тухайн хэлний дотоод суурь эсвэл хэрэглэгчийн зохиомол төрлийн хувьсагч байх ёстой. Харин уг операторын баруун гар талд (RHS-Right Hand Side) дараах

- суурь юм уу зохиомол төрлөн утга эсвэл тогтмол
- суурь юм уу зохиомол төрлөн хувьсагч
- суурь эсвэл зохиомол төрлөн хариутай илэрхийлэл

гэсэн гурван зүйлээс аль нэг нь байна.

Утга оноох үйлдлийн хувьд дараах 5 тохиолдол байж болдог. Тухайлбал,

- тохиолдол 1: LHS, RHS хоёулаа яг ижилхэн суурь төрлийнх
- тохиолдол 2: LHS, RHS хоёулаа яг ижилхэн зохиомол төрлийнх
- тохиолдол 3: LHS, RHS хоорондоо ялгаатай суурь төрлийнх
- тохиолдол 4: Аль нэг (LHS эсвэл RHS) талынх нь зохиомол, негээ талынх нь суурь төрлийнх
- тохиолдол 5: LHS, RHS хоёулаа зохиомол төрлийнх

**Toхиолдол 1.** Хамгийн энгийн утга оноох үйлдэл юм. RHS-ийн уттыг ямар ч төрөл хувиргалтуйгээр RHS-д хийнэ. Тасдаж богиносгох юм уу нарийвчлах үйл хийгдэхгүй тул мэдээлэл алдагдахгүй. Жишээг доор үзүүллээ.

```
int x = 100 ;  
char x = 'a' ;  
double x = 10.5 ;
```

**Toхиолдол 2.** LHS, RHS хоёулаа яг ижил зохиомол төрлийнх байна. employee объектыг түүнтэй класс иэгтэй өөр объект руу хуулах тухай өмнөх бүлгүүдэд үзсэн билээ. Энэ үед RHS дэх объектын өгөгдөл бүрийг LHS дэх объектын харгалзах өгөгдөл рүү хуулиа. LHS, RHS хоёулаа ижил төрлийнх байх тул суурь юм уу зохиомол гэдгээс шалтгаалахгүйгээр эн үйлдлийг компайлэр түвэргүйгээр гүйцэтгэдэг.

**Toхиолдол 3.** Далд, ил төрөл хувиргалт шаардгдана. Дараах жишээг авч үзье.

```
long ll = 8 ;           //long <- int хувиргалт  
double dd = 10 ;        //double <- int хувиргалт  
int ii = 3.56 ;         //int <- double хувиргалт  
float ff = 6 ;          //float <- int хувиргалт
```

Дээрх бүх жишээний хувьд утга оноохын өмнө RHS өгөгдлийн төрлийг LHS өгөгдлийн төрөл рүү эхэлж хөрвүүлийн. Иймэрхүү хувиргалт нь програмчлалд их тохиолддог. C++ компайлэр ийм хөрвүүлгийг хийдэг бэлэн програмын кодтой байх тул шаардлага гарахад түүний дуудаж хэрэглэнэ. Гэхдээ ийм шаардлага гарах эсэх нь утга оноох операторын хоёр талын өгөгдлийн төрлийг нягталж үзснээр тодорхой болдог байна.

Зарим тохиолдолд дээрх шиг хөрвүүлгийн явцад мэдээлэл алдагдах явдал гарч болно. Жишээ нь, int ii=3.56; командын дараагаар ii нь 3 болж 0.56 алдагдана.

Жишээ болгон дээр зааж өгсон хөрвүүлэг нь програм хөгжүүлэгчийн оролцоогүйгээр далд буюу автоматаар явагдах төрөл хувиргалт юм. Утга оноох операторын хоёр талын өгөгдлийн

төрлийн хувьд хоорондоо нийцж байх тохиолдолд далд төрөл хувиргалтыг хэрэглэх боломжтой байдаг. Дээрх бүх жишээнд янз бүрийн тоог хэрэглэсэн байгаа. Тэгвэл,

```
int *p =100;
```

шиг командыг авч үзье. Энд LHS нь хаяг, RHS нь тоо байна. Хаяг нь компьютерийн дотоод дурслэлийн хувьд тэмдэггүй бүхэл тоо хэдий ч жинхэнэ утгаараа тоо биш юм. Энэ нь хаягийн арифметик бүхэл тооныхоос ялгаатай байдагтай холбоотой юм. Хаягийг үржиж, хувааж болдоггүй. Иймээс хаягийн хувьд далд төрөл хувиргалт байж болохгүй тул омнох нь алдаатай команд юм. Иймд төрөл хувиргалтыг илээр int \*p = (int \*)100; гэж хийх ба ингэснээр хаяган хувьсагчийн утга нь 100 гэсэн хаяг болно.

Өгөгдлийн нэг төрлөөс негеө рүү хөрвүүлэг хийхийг компайлерт хэлж өгөхдөө төрөл хувиргалтыг хэрэглэнэ. Жишээ нь, болит тоог бүхэл тоо руу хөрвүүлэх бол

```
ii =(int) ff ;  
юм уу  
ii=int(ff);
```

гэж бичиж өгөх ба ийм хувиргалтыг ил төрөл хувиргалт гэнэ. Үүнд програм хөгжүүлэгчийн оролцоо шаардлагатай байдаг. Ер нь, ил болон далд төрөл хувиргалтын үед ижилхэн стандарт програмын кодыг хэрэглэдэг.

### 8.7.1 Суурь ба зохиомол төрөл хоорондын хувиргалт

Утга оноох үйлдлийн доровдэх нь суурь болон зохиомол төрөл хоорондох хувиргалтын үед тохиолдоно. Хоёр зүйлийн хувиргалт байж болно. Тухайлбал,

```
user_defined_type = basic_type  
basic_type = user_defined_type
```

Эхний нь суурь төрлөөс зохиомол төрөл рүү, удаах нь зохиомол төрлөөс суурь төрөл рүү хувиргалт хийнэ. Эдгээр хувиргалтын үед стандарт програмын код хэрэглэж болохгүй тул програм хөгжүүлэгч нь өөрийн гэсэн функцийн бичих шаардлагатай болдог.

### 8.7.2 Зохиомол төрлийн объект хоорондын хувиргалт

Утга оноох үйлдлийн тавдахь тохиолдол нь хоёр зохиомол төрлийн хооронд хувиргалт хийхэд гарна. Ийм хувиргалтын үед утга оноох операторын баруун талд нь “эх” объект, зүүн талд нь “авах” объект байна. Жишээ нь,

```
classA objA ;  
classB objB ;  
  
objA = objB //objB-эх объект objA-авах объект  
-----  
objB = objA //objA-эх объект objB-авах объект  
-----
```

Хоёр зүйлийн хувиргалтын код хэрэгтэйгээс нэг нь (objB-objA) objB-ээс objA руу, негеө нь (objA-objB) objA-aас objB рүү хувиргахад хэрэглэгдэнэ. Хувиргалтын кодууд нь эх класс эсвэл авах классын аль нэгэнд байж болно. Жишээ болгож rect, polar гэсэн

хоёр классыг авч үзье. Эхний класс нь цэгийг тэгш өнцөгтийн солбилцолд дүрслэнэ. Удаах polar класс нь адилхан цэгийг туйлын солбилцолд дүрслэнэ. Эдгээр классын объектуудыг лараах шигээр байгуулж болно.

```
rect r;
polar p;
```

Хоёр хувиргалт байж болно, тухайлбал

```
r = p;           //r-p хувиргах кодыг дуудах
p = r;           //r=r хувиргах кодыг дуудах
```

Хувиргах кодыг хаана байршуулхтай хамаатай хоёр боломж бий. Тухайлбал,

- Хувиргах кодын хэсэг нь эх класс дотор байна. Ингэхдээ r-p хувиргах код polar, r-p хувиргах код rect класс дотор байршина.
- Хувиргах кодын хэсэг нь "авах" класс дотор байна. Ингэхдээ r-p хувиргах код rect, r-p хувиргах код polar класс дотор байршина.

Дээрх rect, polar хоёр классын тодорхойлолтыг Program #8.15-д үзүүлээ. Хувиргалтын код эх класс дотор байна.

```
//Program #8.15
#include <iostream.h>
#include <math.h>

const double rad_to_deg=57.2957795130823;
class polar;

class rect
{
private:
    double xx;
    double yy;
public:
    rect(void);
    ~rect(void);
    rect(double x, double y);
    void show_rect(void);
    operator polar();           //r-p хувиргалт
};

rect::rect(void)
{
}

rect::~rect(void)
{
}

rect::rect(double x, double y)
{
    xx = x;
    yy = y;
}

void rect::show_rect(void)
{
    cout << "(" << xx << ", " << yy << ")\n";
}
```

```

/*
rect::operator polar()
{
    double rad = sqrt(xx*xx+yy*yy) ;
    double ang = rad_to_deg*atan2(yy, xx) ;
    return polar(rad, ang) ;
}
*/
class polar
{
private:
    double radius;
    double angle;
public:
    polar(void);
    ~polar(void);
    polar(double rad, double ang);
    void show_polar(void);
    operator rect();           //p-r хувиргалт
};

polar::polar(void)
{
}

polar::~polar(void)
{
}

polar::polar(double rad, double ang)
{
    Radius = rad ;
    angle = ang ;
}

void polar::show_polar(void)
{
    cout << "(" << radius << ", " << angle << ")\n" ;
}

polar::operator rect()
{
    double x = radius*cos(angle/rad_to_deg) ;
    double y = radius*sin(angle/rad_to_deg) ;
    return rect(x, y);
}

rect::operator polar()
{
    double rad = sqrt(xx*xx+yy*yy) ;
    double ang = rad_to_deg*atan2(yy, xx) ;
    return polar(rad, ang) ;
}

void main(void)
{
    rect r ;
    polar p ;
    rect r1(10, 10) ;
    p = polar(r1) ;

```

```

cout.precision(4) ;
cout << "rect(10, 10) = polar";
p.show_polar() ;
polar p1(15, 45) ;
r = rect(p1) ;
cout << "polar(15, 45) = rect";
r.show_rect() ;
rect r2(25, 15) ;
p = r2 ;
cout << "rect(25, 15) = polar";
p.show_polar() ;
polar p2(20, 60) ;
r = p2 ;
cout << "polar(20, 60) = rect";
r.show_rect() ;
}

rect(10, 10) = polar(14.1421, 45)
polar(15, 45) = rect(10.6066, 10.6066)
rect(25, 15) = polar(29.1548, 30.9638)
polar(20, 60) = rect(10, 17.3205)

```

Дээрх програм дотор `operator polar()` функцийн тодорхойлолт хоёр байгаагас эхнийхийг нь кодын хэсэг програмын дотор хаана байршиж байгаа нь чухал болохыг харуулах үүднээс тайлбар болгосон байгаа. Хэрэв классын тодорхойлолт дотор бичигдсэн дарааллаар ийн гишүүн функцийг тодорхойлдог хэвшмэл гэмээр жаягийг дагавал `operator polar()` функцийн тодорхойлолт нь тайлбар болгосон тэр газарт байх ёстой. Иймд эхний тодорхойлолтыг тайлбаргүй, хоёрдахийг нь тайлбар болгоод шалгавал хорвүүлгийн алдаа гарна. C++ компайлер `return polar(rad, ang);` командын мөрөнд хүрч очих үед `polar` класс тодорхойлогоогүй байх тул системийн алдаа гарна. Ийм алдааг засах шийдэл нь дээрх програмд үзүүлсэн шигээр `operator polar()` функцийн тодорхойлолтыг `polar` классын дараагаар байршуулах явдал юм.

Програмын `main()` функцийн `r=polar(r1);` команд хийгдэх үед `r`-р хувиргалт хийх функцийг хайж улмаар `rect` класс дотроос `r`-р хувиргалтын `operator polar()` функцийг олж авч түүнийг `r1` объектоос дуудаж ажиллуулна. Тэгэхлээр, `r=polar(r1);` нь `r=r1.operator polar();` гэсэнтэй дүйнэ.

Харин `main()` функцийн `r=rect(p1);` команд биелэгдэх үед `r`-г хувиргалт хийх кодын хэсгийг олох ёстой. Гэвч `rect` класс дотроос `r`-р биш зөвхөн `r`-р хувиргалтын функцийг олох тул хайлтыг `polar` класс дотор үргэлжлүүлэн хийж тэндээс `r`-г хувиргалтын `operator rect()` функцийг олоод түүнийг `r1` объектоос дуудна. Тэгэхлээр, `r=rect(p1);` нь `r=p1.operator rect();` гэсэнтэй дүйдэг.

Бас `r=r2;` команд биелэгдэх үед `r`-г хувиргалт хийх кодын хэсгийг хайж улмаар `rect` класс дотроос хувиргалтын `operator polar()` оператор функцийг олж түүнийг дараа нь `r2` объектоос дуудах тул `r=r2;` нь `r=r2.operator polar();` гэсэнтэй адил юм.

Эцэст нь `r=r2;` команд биелэгдэх үед `r`-г хувиргалт хийх кодын хэсгийг хайж улмаар `polar` класс дотроос хувиргалтын `operator rect()` операторыг олж түүнийг `r2` объектоос дуудаж ажиллуулах тул `r=r2;` нь `r=r2.operator rect();` гэсэнтэй дүйдэг.

Аравтын таслалын дараах хэдэн цифрийг дэлгэшлэхээ cout объектын precision() функцийн зааж өгдөг. Дээрх програмд аравтын таслалын ард 4 цифртэй байлгахын тулд cout.precision(4); гэж хэрэглэсэн.

Хувиргалтын функцийн авах класс дотор байх програмын жишээг доор үзүүллээ.

```
//Program #8.16
#include <iostream.h>
#include <math.h>

const double rad_to_deg = 57.2957795130823 ;
class polar;

class rect
{
private:
    double xx ;
    double yy ;
public:
    rect(void) ;
    ~rect(void) ;
    rect(double x, double y) ;
    void show_rect(void) ;
    double get_xx(void) ;
    double get_yy(void) ;
    rect(polar p) ; //p-r хувиргалт
};

rect::rect(void)
{
}

rect::~rect(void)
{
}

rect::rect(double x, double y)
{
    xx=x;
    yy=y;
}

void rect::show_rect(void)
{
    cout << "(" << xx << ", " << yy << ")" \n" ;
}

double rect::get_xx(void)
{
    return xx ;
}

double rect::get_yy(void)
{
    return yy ;
}

/*
rect::rect(polar p)
{
    double rad=p.get_radius() ;
```

```

        double ang=p.get_angle() ;
        xx=rad*cos(ang/rad_to_deg) ;
        yy=rad*sin(ang/rad_to_deg) ;
    }
}

class polar
{
private:
    double radius ;
    double angle ;
public:
    polar(void) ;
    ~polar(void) ;
    polar(double rad, double ang) ;
    void show_polar(void) ;
    double get_radius(void) ;
    double get_angle(void) ;
    polar(rect r) ; //r-p хувиргалт
};

polar::polar(void)
{
}

polar::~polar(void)
{
}

polar::polar(double rad, double ang)
{
    radius=rad;
    angle=ang;
}

void polar::show_polar(void)
{
    cout << "(" << radius << ", " << angle << ")\n" ;
}

double polar::get_radius(void)
{
    return radius ;
}

double polar::get_angle(void)
{
    return angle ;
}

polar::polar(rect r)
{
    double x = r.get_xx() ;
    double y = r.get_yy() ;
    radius = sqrt(x*x+y*y) ;
    angle = rad_to_deg*atan2(y, x) ;
}

rect::rect(polar p)
{
    double rad = p.get_radius() ;
}

```

```

double ang = p.get_angle() ;
xx = rad*cos(ang/rad_to_deg) ;
yy = rad*sin(ang/rad_to_deg) ;
}

void main(void)
{
    rect r1(10, 10) ;
    polar p = r1 ;
    cout << "rect(10, 10) = polar" ;
    //cout.precision(4) ;
    p.show_polar() ;
    polar p1(15, 45) ;
    rect r = p1 ;
    cout << "polar(15, 45) = rect" ;
    r.show_rect() ;
    rect r2(25, 15) ;
    p = r2 ;
    cout << "rect(25, 15) = polar" ;
    p.show_polar() ;
    polar p2(20, 60) ;
    r=p2;
    cout << "polar(20, 60) = rect" ;
    r.show_rect() ;
}

```

 rect(10, 10) = polar(14.1421, 45)  
polar(15, 45) = rect(10.6066, 10.6066)  
rect(25, 15) = polar(29.1548, 30.9638)  
polar(20, 60) = rect(10, 17.3205)

Дээрх програмын кодын зарим хэсгийг тайлбар болгож тэмдэглэсэн нь програмын эл хэсгээс өмнө polar классын тодорхойлолт байхгүй байгаатай холбоотой юм. Иймд энхүү кодын хэсгийг polar классын тодорхойлолтын дараагаар main() функцийн тодорхойлолтын өмнө дахин бичиж егсен.

Програмын main() функцийн polar p=r1; команд биелэгдэх үед ганц аргумент бүхий polar(r1) байгуулагч, үүнтэй төстэйгээр rect r=p1; команд биелэгдэх үед ганц аргумент бүхий rect(p1) байгуулагч тус тус дуудагдана. Эшээт нь r=p2; команд биелэгдэх үед ганц аргумент бүхий rect(p2) байгуулагчийг дуудна.

#### Мэдлэгээ шалгах асуулт

- 8.1. friend функц гэж юу болох, ийм функцийг хэрэглэх шаардлага яагаад гардаг
- 8.2. friend функцийг класс дотор хэрхэн зарлаж болохыг жишээгээр үзүүлэх
- 8.3. friend функцийн тодорхойлолтын эхлэл ямар байдгийг жишээгээр үзүүлэх

#### Жишээ болдого

- 8.4. func(c) функц classA, classB хоёр классын private огогдолд ханддаг байхаар тэдгээр классыг тодорхойлох
- 8.5. Рациональ тоог загварчлах класс тодорхойлох; Эл классын гишүүн функц болох void show\_data(void); нь рациональ тооны аравтын уттыг, жишээ нь S гэсэн тоог 0.5

гэж хэвлэдэг байх; Рациональ тооны классын объектод хэрэглэгдэх + операторыг гишүүн функцийн хэрэглэн дахин тодорхойлох

- 8.6. Өмнөх бодлогын хувьд + операторыг `friend` функцийн хэрэглэн дахин тодорхойлох
- 8.7. `cout <<` командын рациональ тоог `xx/yy` хэлбэрээр хэвлэх боломжтой байхаар `<<` операторыг дахин тодорхойлох
- 8.8. Комплекс тоог загварчлах класс тодорхойлох; Эл классын гишүүн функцийн `void show_data(void);` нь комплекс тооны магнитудыг хэвлэдэг байх; Комплекс тооны классын объектод хэрэглэх + операторыг гишүүн функцийн хэрэглэн дахин тодорхойлох
- 8.9. Өмнөх бодлогын хувьд + операторыг `friend` функцийн хэрэглэн дахин тодорхойлох
- 8.10. `cout <<` командын комплекс тоог `r+iq` хэлбэрээр хэвлэх боломжтой байхаар `<<` операторыг дахин тодорхойлох
- 8.11. Бодлого 7.12-т тодорхойлох классын хувьд тэмдэгт мөрийг `cout << s`; командаар дэлгэц рүү бичих бололцоотой байхаар `<<` операторыг дахин тодорхойлох
- 8.12. Бодлого 7.12-т тодорхойлох классын хувьд хэрэв `s1` нь "Hello", `s2` нь "World" мөрийг тус тус хадгалах бол `s=s1+s2`; үйлдлээр `s` нь "Hello World" мөрийг хадгалахаар + операторыг дахин тодорхойлох. Хоёр мөрийн хооронд зайд нэмж оруулна.
- 8.13. Бодлого 7.12-т тодорхойлох `string` классын `s1`, `s2` хоёр объект байгаа бол `s1=s2`; шиг үйлдэл хийх бололцоотой байхаар утга оноох операторыг дахин тодорхойлох

# ЕСДҮГЭЭР БҮЛЭГ

## ОРОЛТ ГАРАЛТ БА ФАЙЛ

- Орох гарах урсгал, 281
- Оролт гаралтын буфер, 282
- *iostream.h* толгой файл, 282
- cout объект, 283
- << оператор, 284
- void \* төрөл, 286
- Оролт гаралтын чиглэл вэрчлэх, 286
- *ostream* арга *put()*, 287
- *ostream* арга *write()*, 288
- cout << хэрэглээ, 289
- Тооны суурь тоог вэрчилж хэвлэх, 290
- Талбарын ёргөний тохируулга, 291
- Ашиглагдаагүй зайд дүүргэх, 292
- Бутархайн нарийвчлал, 292
- Ут нийт бус тэгийг хэвлэх, 293

-Үргэлжлэл-



-Үргэлжлэл-

- *cin* объект, 293
- Оруулгын оператор *>>*, 294
- *cin >> нь* гарын буферийг хэрхэн унших, 296
- *istream* арга: *get(char &)*, 301
- *istream* арга: *get(void)*, 304
- *istream* арга: *get(char \*, int, char ='\\n')*, 305
- *istream* арга: *getline(char \*, int, char ='\\n')*, 307
- *istream* арга: *ignore()*, 308
- *istream* арга: *read()*, 309
- *istream* арга: *peek()*, 310
- *istream* арга: *gcount()*, 311
- *istream* арга: *putback()*, 312
- Файлын оролт гаралт, 313



## ЕСДҮГЭЭР БҮЛЭГ

### ОРОЛТ ГАРАЛТ БА ФАЙЛ

Програм ямар нэгэн хэрэгтэй зүйл хийдэг байхын тулд түүнд хэрэгтэй өгөгдлийг гаднаас авч боловсруулаад хариу үр дүнг гадагшaa өгдөг байх ёстой. Програм орох өгөгдлийг компьютерийн гараас, гадаад тохөөрөмж дээрх файлаас эсвэл бусад програмын үр дүнгээс авна. Програм гаргах мэдээллийн дэлгэц, хэвлүүр эсвэл гадаад тохөөрөмж дээрх файл руу бичиж чадна.

Оруулах гаргах үйлдэл хийх тусгай команд, жишээ нь Basic хэлний хувьд Input, Read, Print; Pascal хэлний хувьд readln, writeln юм. Эдгээртэй төстэй оролт гаралтын команд C++ хэлэнд байдагтуйг Хавсралт В дэх жагсаалтаас мэдэж болно. Омно үзсэн бүх жишээ програмд хэрэглэж байгаа cout, cin нь C++ хэлний команд биш, үнэндээ cout бол эх загвар нь iostream.h толгой файлд байх ostream классын объект бол cin нь эх загвар нь iostream.h толгой файлд байх istream классын объект юм. Эдгээр объект нь оруулах гаргах үйлийг гүйцэтгэх арга функц, дахин тодорхойлсон олон оператортой. Жишээ нь, омно үзсэн (<<) гаргах операторыг ostream класс дотор дахин тодорхойлсон байдал. Үүнтэй адил, (>>) нь istream класст дахин тодорхойлсон орууллын оператор юм.

#### 9.1 Орох гарах урсгал

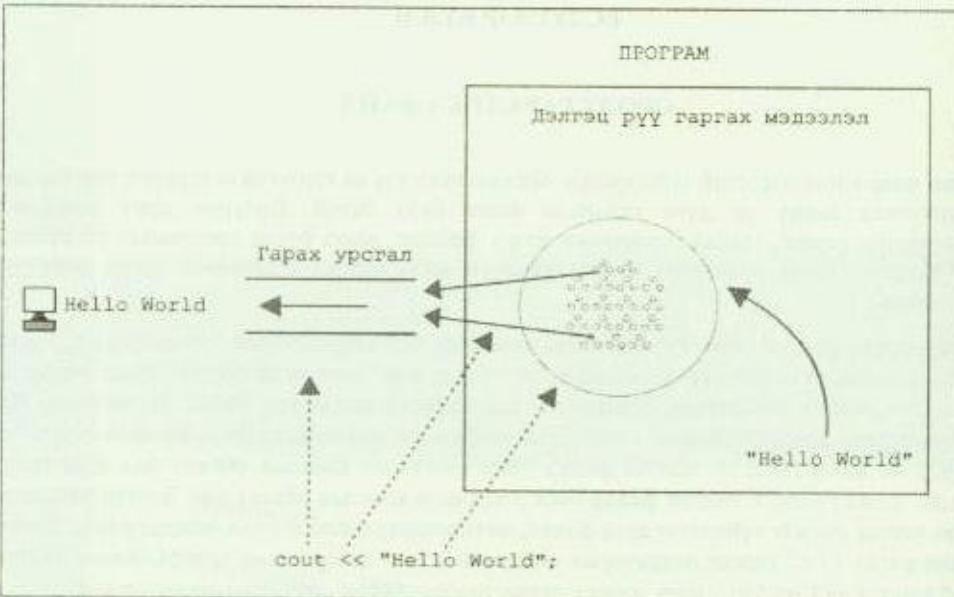
C++ програмын оролт гаралтыг өгөгдлийн урсгал мэтээр тосоолж болно. Ийм урсгалыг оруулах үйлдлийн үед өгөгдөл програм руу, гаргах үйлдлийн үед өгөгдөл програмаас "урсан" орох том хэмжээтэй заверын хуримтлуур сан гэж төсөөлж болно. Ийм сан нь компьютерийн гар, дэлгэц, машинд залгаатай хэвлүүр, эсвэл гадаад тохөөрөмж дээрх файл хэлбэрээр байж болно. Урсгалын нэг талд нь өгөгдлийн хуримтлуур, ногөө талд нь програм байна. Өгөгдлийн урсгал байгаар хэмжигдэх бөгөөд ийм байт бүрд нэг ASCII тэмдэгт харгалзана.

Янз янзын урсгалыг програмд холбож болдог. Өөрөөр зааж өгөөгүй бол анхнаасаа орох урсгал нь компьютерийн гартай, гарах урсгал нь дэлгээтгэй холбогдоно. Програм дэлгэшлэх өгөгдлөө дэлгээтгэй холбоотой гарах урсгал руу хийнэ. Ийм дэлгэшлэх үйлийг Зураг 9.1-д үзүүлсэн шигээр дүрсэлж болно.

C++ програм нь "Hello World" гэсэн тэмдэгт мөрийг дэлгэшлэхдээ түүнийг дэлгээтгэй холбогдох гарах урсгал руу хийх ёстой. Үүний тулд дэлгээцийн гарах урсгалыг төлөөлөх cout объект бүхий cout<<"Hello World"; командыг хэрэглэнэ. Гаргах оператор нь "Hello World" мөрийг гарах урсгал руу нэмж хийнэ. Яагаад уг операторыг утгаа (insertion) оруулгын гэж нэрлэх болсон, эн нь операторын болзмол тэмдгийн заах чигтэй хэрхэн холбогддог зэрэг нь Зураг 9.1-ээс тодорхой харагдана.

Гарах урсгалын байтууд гаралтын хуримтлуурт очоод тэндээ боловсруулагдана. Жишээ нь, дэлгээтгэй холбоотой гарах урсгал руу нэмэгдэж орох байтууд дэлгэшлэгдэн. Хэвлүүртэй холбогдох гарах урсгалл нэмж орох байтууд хэвлэгдэн. C++ програм нь дэлгэц, хэвлүүр хоёрны алийг нь ч илүүд үздэггүй тул мэдээллийг дэлгэц, хэвлүүр рүү адил аргаар гаргаж чадна.

Орох урсгалын байтуудыг уг урсгалд холбоотой хуримтлуураас уншина. Орох мэдээллийг мөн гадаад тохөөрөмж дээрх файлаас авч болно. C++ програм нь хаанаас авч байгаагаар нь мэдээллийг ялгаж үздэггүй тул түүнийг компьютерийн гараас эсвэл файлаас авахыг адилхан түвшинд авч үзэх боломжийг програм хөгжүүлэгчид олгодог байна.



Зураг 9.1: Дэлгээзэх үзэдэл

## 9.2 Оролт гаралтын буфер

Завсрын ой болох буфер хэрэглэснээр оролт гаралтын ажил үйлийг ихээхэн бүтээмжтэй гүйцэтгэж болдог. Буфер бол оруулах үйлдлийн үед төхөөрөмжөөс програм руу, гарах үйлдлийн үед програмаас төхөөрөмж рүү дамжих мэдээлэлд завсрын хадгалуур хэлбэрээр хорглэх ойн хэсэг юм. Диск мэтийн төхөөрөмж 512 байтаар багцлагдах мэдээллийг хурдтайгаар дамжуулж чадна. Гэтэл програм нэг удаа даа нэг байж мэдээлэл боловсрууна. Ийм ялгаатай хурдыг хооронд нь зохицуулахад оролтын буфер туслах бөгөөд буфер дамжуулал нь мэдээллийн ямар ч алдагдалгүйгээр хэрэгжиж чаддаг. Програм буфериийн төгсгэлл хүрч очиход шинэ багц мэдээлэл дискийн төхөөрөмжөөс буфер рүү дамжиж орно.

Гарах үйлдлийн үед програм эхлээд буферийг мэдээллээр дүүргэнэ. Буфер дүүрэхэд доторхыг нь диск рүү нэг бодино үйлдлээр дамжуулна. Үүнийг буфериийн "албадмал" хадгалалт гэнэ. Албадмал гаралтыг бусад үед ч бас хэрэглэнэ. Дэлгээний гарах буфер рүү шинэ мөрийн тэмдэгт "\n" нэмэгдэж ороход албадмал дэлгэшлэл болдог.

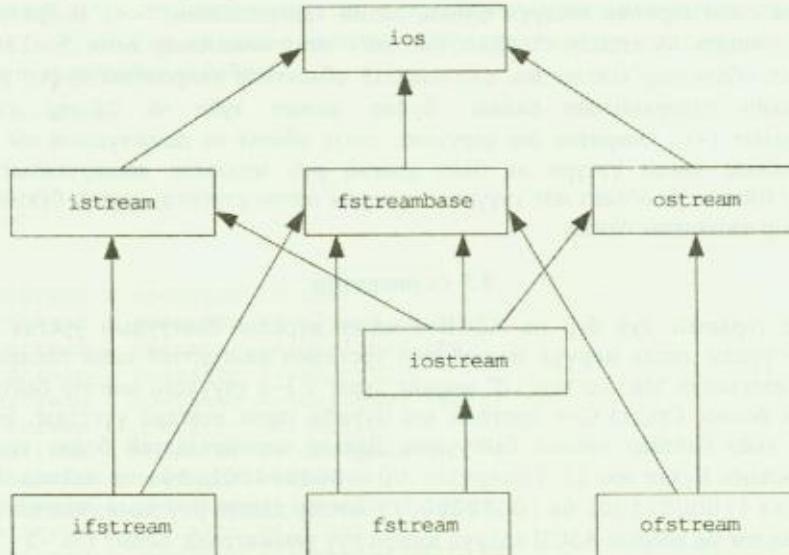
Гар бол CPU-тэй харьцуулахад удаан төхөөрөмж юм. Иймд түүнээс авах зүйлийг буферлэх шаардлага байдагтуй ч түүнийг буферлэх боломж бий. Ийм оруулгыг дамжуулахын омно засч болно. Гарын оруулгыг ↲ товчоор дуусгавар болгоход буферлэгдсэн оруулга явь албадлагаар програм руу дамжигдана.

## 9.3 iostream.h толгой файл

ANSI Си хэлний хувьд оруулах гарах үйлийг хийхэд `stdio.h` толгой файл дотор тодорхойлсон стандарт сангийн функцийийг хэрэглэнэ. C++ хэлэнд оролтын төхөөрөмжөөс авч компютерийн ой руу хийх эсвэл ойгос авч гаралтын төхөөрөмж рүү гарах өгөгдлийн цуваар `stream` буюу урсгал гэх ба ийм урсгалыг класс `iostream` программчилсан байдаг.

Өгөгдлийн урсгалтай хамаатай олон классын тодорхойлолт `iostream.h`, `fstream.h` гэсэн хоёр толгой файлд байдаг. Гэхдээ `fstream` нь `iostream` классаас удамших класс тул

`fstream.h` файлыг програмд препроцессорын `#include` командаар зааж өгөх тохиолдолд `iostream.h` файлыг илээр зарлах шаардлага байдагтүй<sup>49</sup> (Зураг 9.2).



Зураг 9.2: Урсгал, класс хоорондын хамаарал

Оролт гаралтын үйлд холбогдох `cout`, `cin` объектууд болон бусад зүйл нь дээрх хоёр файлд байна. Тэгэхлээр, C++ хэлний хувьд оруулах гаргах үйл нь эдгээр файлд тодорхойлсон классуудаар удирдагдана. Ийм классын жишээ болгон `iostream.h` файлд байгаа зарим классыг доор жагсаалаа.

- **streambuf класс:** Завсрин буфер бэлдэнэ. Буферийг удирдах, тухайлбал өгөгдөл рүү нь хандах, өгөгдлөөр дүүргэж улмаар албадан дамжуулах үүрэгтэй гишүүн функциүүдтэй.
- **ios класс:** streambuf төрлийн объектын хаяг энэ классын нэг гишүүн өгөгдөл бодлог. Урсгал нь бичгийн юм уу хөбтүн файлынх эсвэл урсгалыг уншихаар юм уу бичихээр ирэх зөргээс хамааралгүйгээр урсгалын ерөнхий шинжийг харуулж чадлаг класс юм.
- **ostream класс:** ios классаас удамшина. Гаргах үйлдлийн аргуулттай.
- **istream класс:** ios классаас удамшдаг. Оруулах үйлдлийн аргуулттай.
- **iostreams класс:** istream, ostream классуудаас удамших тул тэдгээрийн шинжийг овалож авсан байдаг.

`iostream.h` толгой файл дотор `cout`, `cin`, `cerr`, `clog` зэрэг объектыг үүсгэсэн байдаг. Гэхдээ `cout` нь ostream классын объект бол `cin` нь istream классын юм. Харин `cerr` ба `clog` нь алдааны мэдээлэл гаргахад хэрэглэх стандарт урсгалууд юм. Тэдгээр нь стандарт гаралтын төхөөрөмж болох дэлгэцтэй бас холбогдсон байдаг. Тэгэхлээр, өөроор заагаагүй бол алдааны мэдээлэл дэлгэшгэдэн. Гэхдээ `clog` нь буферт урсгал юм.

#### 9.4 cout объект

Объект `cout` нь ostream классын стандарт гаралтын урсгалыг төлоөлио. Уг объект нь гаралттай хамаатай мэдээллийг хадгалж байх гишүүн өгөгдлөтэй. Ийм мэдээлэлд дэлгэнүүх

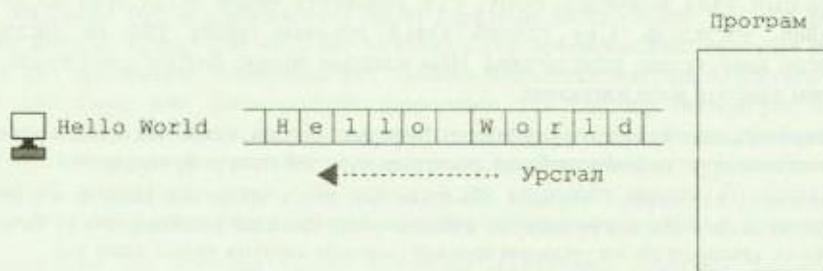
<sup>49</sup> Урсгалын эх класс бол ios юм. Файлын зориулалттай fstreambase нь бас ios классаас удамшина. Харин ifstream, ofstream зэрэг нь олон-нэг харьцаатай нийлмэл удамшины классууд юм.

мэдээллийн талбарын өргөн, тоон өгөгдлийн суурь тоо, таслалын оронгийн тоо, гаргах уйл хэрэглэх буферийг тодорхойлох урсгалын буферэн объектын хаяг зэрэг олон зүйл багтана.

`ostream` нь олон төрлийн гишүүн функцийн дахин тодорхойлсон (<<) оператортой. Энэ операторыг, жишээ нь `cout << "Hello World";` шиг командаар өгөх "Hello World" мөрийг `cout` объектоор хаяглагдаж `streambuf` объектоор удирдагдах буфер рүү хийдэг байхаар дахин тодорхойлсон байдаг. Буфер доторх зүйл нь OS-ээр дэлгэц рүү дамжуулагдахыг (<<) оператор бас харуулна. `cout` объект нь дамжууллын нэг үзүүрт нь байх програмаас нөгөө үзүүрт нь байх дэлгэц рүү мэдээлэл дамжуулахыг удирдаж зохицуулна. Иймээс уг объект нэг үзүүртээ програм нөгөө үзүүртээ дэлгэц бүхий гаралтын урсгал шигээр ажилладаг байна.

### 9.5 << оператор

C++ хэлэнд гаралтыг тус бүр нь ASCII тэмдэгт дүрслэх байтуудын урсгал гэж үзүүлэв. Програмаар үүсдэг гарах мөрүүд мэдээллийн урсгалын ойлголттой сайн тохирдог байна. Жишээ нь, дэлгэцлэх "Hello World" мөрийг Зураг 9.3-л үзүүлсэн шигээр байтын урсгал мэтээр үзэж болно. Гэхдээ C++ програм язсан бүрийн гарах өгөгдөл үүсгэдэг. Бүхэл тоон мэдээллийг хоёр байтаар зохион байгуулна. Давхар нарийвчлалтай болит тоог найман байтаар дүрслэнэ. Бүхэл тоо 12 (хоёртын 0000000000001100) нь найман битэн хоёр байтын урсгал ((00001100) ба (00000000)) мэтээр дэлгэц рүү дамжуулагдахгүй, харин түүний оронд нэг ба хоёртын ASCII кодууд дэлгэц рүү дамжигддана. Болит тоо -2.76 тус бүр нь уг тооны компьютерийн дотоод дүрслэл болох найман байтын урсгал бус харин '-', '2', '.', ',', '6' гэсэн таван ASCII тэмдэгтийн урсгал хэлбэрээр дэлгэцэгдэнэ.



Зураг 9.3 "Hello World" өгүүлбөрийг дэлгэцэх байдал

Дэлгэц рүү 12 гэсэн тоог бичихийн тулд `cout << 12;` командыг хэрэглэх ба үүнийг доор үзүүлсний адил хувьсагч хэрэглэн бичиж болно.

```
int ii = 12 ;  
cout << ii ;
```

Дээрхтэй адил -2.76 гэсэн тоог дэлгэцлэх бол `cout << -2.76;` гэсэн командыг хэрэглэхээс гадна бас доорх шигээр хувьсагч хэрэглэж болно.

```
double dd = -2.76 ;  
cout << dd ;
```

Тэгэхлээр, (<<) операторын гүйштгэх олон чухал үүргийн нэг бол `int`, `float`, `double` гэх мэт төрлийн тоон утгыг тэдгээрт харгалзах тэмдэгтийн урсгал руу буулгах явдал юм. Уул нь (<<) операторын үндсэн зориулалт бол бит шилжилт билээ. Баруун, зүүн гэсэн хоёр бит шилжилт байдаг. Жишээ нь, `x << 3;` нь х-ийн хоёртын дүрслэлийг зүүн тийш 3 бит удаа (3 битээр) шилжүүлэх тул  $2^{<<3}=16$  болно. Харин `x >> 3;` нь х-ийн хоёртын дүрслэлийг

баруун тийш 3 бит удаа (3 битээр) шилжүүлэх тул  $8 \gg 3 = 1$  болно. Энэ операторын үүргийг ostream класс дотор дахин тодорхойлоод гаргах оператор гэж ирлэдэг байна. Түүнийг C++ хэлний бүх суурь төрлийг таньдаг, ostream объектын заалтыг буцаадаг байхаар дахин тодорхойлжээ. Тухайлбал, ostream классын тодорхойлолт дотор operator функцийдийг доорх шигээр тодорхойлсон байдаг.

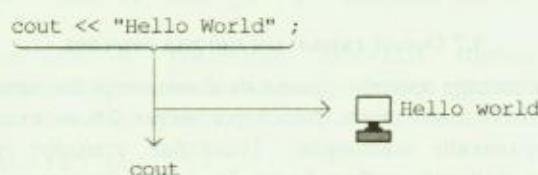
```
class ostream
{
public:

    ostream & operator <<(char) ;
    ostream & operator <<(signed char) ;
    ostream & operator <<(unsigned char) ;
    ostream & operator <<(short) ;
    ostream & operator <<(unsigned short) ;
    ostream & operator <<(int) ;
    ostream & operator <<(unsigned int) ;
    ostream & operator <<(long) ;
    ostream & operator <<(unsigned long) ;
    ostream & operator <<(float) ;
    ostream & operator <<(double) ;
    ostream & operator <<(long double) ;
    ostream & operator <<(const char *) ;
    ostream & operator <<(const signed char *) ;
    ostream & operator <<(const unsigned char *) ;
    ostream & operator <<(void *) ;

};

cout<<value; шиг команд хэрэв value нь бүхэл тоо бол ostream & operator <<(int) функцийг, харин value нь болит тоо бол ostream & operator <<(float) функцийг тус тус дуудах жишээтэй.
```

ostream классын функцийд нь тэдгээрийг дуудаж хэрэглэж байгаа объектын заалтыг буцаадаг байхаар тодорхойлогдсон байдаг. Иймд cout<<"Hello world"; команд нь "Hello world" мөрийг дэлгэшлэхээс гадна объектынхоо заалтыг буцаана. Үүнийг Зураг 9.4-т үзүүлснээр дүрсэлж болно.

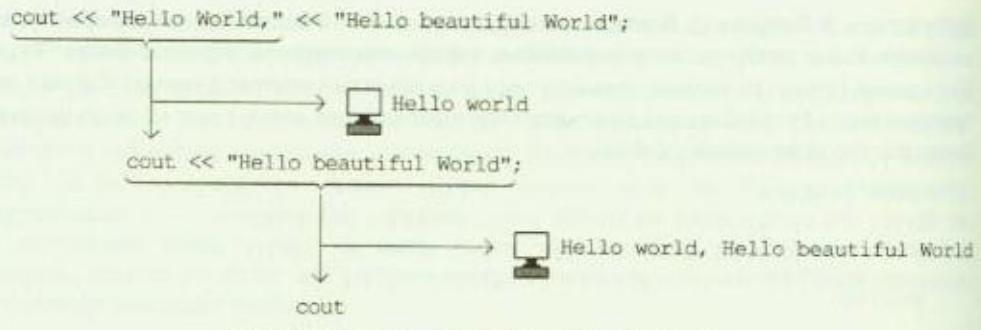


Зураг 9.4: cout объектын ажиллах зарчим

Нэр cout обьектоор

```
cout << val1 << val2 << ... << valn ;
```

шигээр хэд хэдэн (<<) операторыг угсрулан хэрэглэж болно. Үүнийг Зураг 9.5-д жишээ болгож харуулнаа.



Зураг 9.5: Олон << операторыг хэрэглэх боломж

## 9.6 void \* төрөл

Доорх програмыг авч үзье.

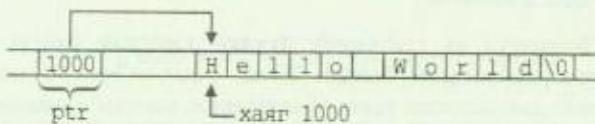
```

//Program #9.1
#include <iostream.h>

void main(void)
{
    char *ptr = "Hello world" ;
    cout << ptr << "\n" ;
    cout << (void *)ptr << "\n" ;
}
  
```

[monitor] Hello world  
0x85420004 (зохиомол хаяг)

Энэ програмд ptr бол "Hello world" гэсэн тэмдэгт мөрийн эхлэл хаягийг хадгалах тэмдэгтэн хаяган хувьсагч, удаах cout<<ptr; нь ptr-ын заах морийг дэлгэшлэх команд юм. Харин cout<<(void \*)ptr; нь ptr хувьсагчийн утга болох "Hello world" мөрийн эхлэл хаяг бүхий ойгоос эхэлж хоёртын тэг хүртэлхийг дэлгэшлэн (Зураг 9.6).



Зураг 9.6: char \*ptr = "Hello World"; командын үр дүнгийн дүрслэл

## 9.7 Оролт гаралтын чиглэл оорчлох

Өөрөөр заагаагүй бол стандарт оролтын ургал нь компьютерийн гартай, стандарт гаралтын ургал нь дэлгэцтэй холбогдоно. Unix, Dos зэрэг ихэнх OS нь стандарт оролт гаралтын чиглэл, холбоосыг оорчлохийг зөвшөөрдог. Тухайлбал, стандарт гаралтыг >, стандарт оролтыг < оператороор файлтай холбож болох ба ингэснээр өтгөгдлөө файлаас авах эсвэл файл руу гаргах боломжтой болно.

Програм нь үр дүнгээ дэлгэж рүү гаргадаг бол түүний чиглэлийг нь оорчлон файл руу гаргаж болно. Доор үзүүлсэн програмыг авч үзье.

```

//Program #9.2
#include <iostream.h>
  
```

```
void main(void)
{
    cout << "Hello standard output stream\n" ;
    cerr << "Hello standard error stream\n" ;
}
```

 Hello standard output stream  
 Hello standard error stream

Стандарт алдааны урсгал бас дэлгэцтэй холбогдоно. Иймээс програм нь дээрх хоёр мөрийг дэлгэц рүү бичсэн байна. Програмын кодыг "prog1.cpp" нэрээр диск рүү хадгалаад түүнээс "prog1.exe" машины програм гарган авч болно. Дараа нь түүнийг OS командын мөрөөс<sup>50</sup> prog1 >file.txt гэж дуудаж ажиллуулбал "file.txt" файл үүснэ. Програм бас "Hello standard error stream" мөрийг дэлгэц рүү бичнэ. Үүснээр

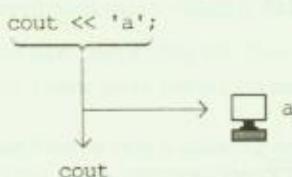
"Hello standard output stream"

тэсэн мөр "file.txt" лотор бичигдсэн эсэхийг DOS-ийн type (хойшид товчоор type гэнэ.) командаар эсвэл потераад програмаар шалгаж болно.

Гаралтын чиглэлийг > оператор нь стандарт гаралтын урсгал руу нэмж хийх мэдээллийг "file.txt" файл руу чиглүүлэх учраас дээрх мөр мэдээллийг уг файлд бичихэд хүргэдэг. Харин (cerr) стандарт алдааны урсгалын мэдээллийн гарах чиглэл > операторд хэвээр хадгалагдах учир түүн рүү гаргах мэдээлэл холбогдох төхөөрөмж нь болох дэлгэц рүү бичигдэнэ.

### 9.8 ostream арга put()

ostream класс нь дурын тэмдэгтийг дэлгэцлэх аргатай бөгөөд түүний загвар нь ostream & put(char) юм. Энэ функцийг, жишээ нь ostream обьектоос cout.put('a'); гэж дуудаж болох ба функц нь 'a' тэмдэгтийг дэлгэцлэн. Үүнийг Зураг 9.7-д үзүүлсэн шигээр дүрсэлж болно.



Зураг 9.7: cout.put('a'); команда нүр дүн

cout<<'a'; нь өгөгдсөн 'a' тэмдэгтийг дэлгэцлэх команд тул түүний оронд cout.put('a'); команда гэрэглэж болно. put() функци дуудагдах обьектынхоо заалтыг буцаах учраас ирг cout обьектын хувьд угсраагаар, жишээ нь

```
cout.put('a').put('b').put('c');
```

гэж холбож хэрэглэж болдог. Дэлгэц рүү "abc" гэж бичих энэ командан ажиллах зарчмыг Зураг 9.8-д үзүүлсэн байдлаар дүрсэлж болно.

put() функцийг бас тоон утгатайгаар хэрэглэж болно. Тухайлбал,

```
cout.put(65); // 'A'-г дэлгэцлэх
```

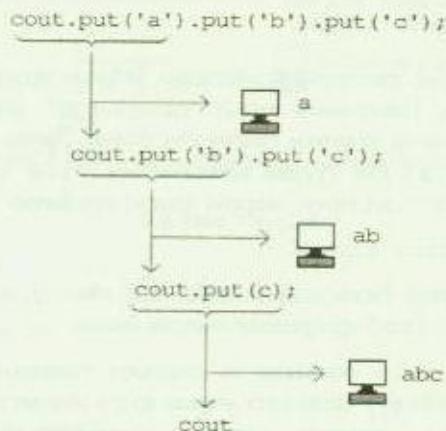
<sup>50</sup> Windows системд Start-Run горимоор

```

cout.put(65); // 'A'-г дэлгэцлэх
cout.put(66.3); // 'B'-г дэлгэцлэх
cout.put(66.8); // 'B'-г дэлгэцлэх

put() функций нь авах утгынхаа зөвхөн бүхэл хэсгийг боловцуулдгийг дээрх үр дүнгээс
харж болно.

```



Зураг 9.8: `cout.put('a').put('b').put('c');` командын ажиллах зарчмын

### 9.9 ostream арга write()

`ostream` класс нь тэмдэгт морийг бүхэлд нь эсвэл түүний хэсгийг дэлгэцүү рүү гаргах аргатай. Энэ арга нь дараах хоёр загварынх байна.

```

ostream & write(const signed char *, int);
ostream & write(const unsigned char *, int);

write() функцийг хэрхэн хэрэглэхийг Program #9.3-т үзүүлэхэй.

```

```

//Program #9.3
#include <iostream.h>
void main(void)
{
    char *ptr1 = "operator";
    char *ptr2 = "overloading";
    cout.write(ptr1, 5);
    cout << "\n";
    cout.write(ptr1, 13) << "\n";
}

```

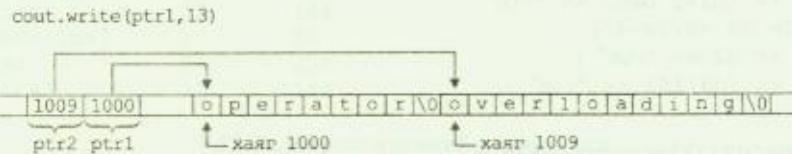
Энэ програмын `cout<<ptr1;` команда нь мөрийн эхнээс тэмдэгт, тэмдэгтээр нь хэвлэсөр тэгсголд нь байх null тэмдэгт дээр хүрч очиход дэлгэцлэх үйл дуусгавар болно. Гэтэл `cout.write()` нь null тэмдэгт дээр очоод автоматаар зогсдоггүй. Харин түүний дуудахдаа зааж өгөх тооны тэмдэгтийг ут тоо нь мөрийн бодит уртаас их байсан ч бүрэг гаргаж байж дуусгавар болно. Дээрх програмын

```

char *ptr1 = "operator";
char *ptr2 = "overloading";

```

хоёр команд нь ой бэлдэж түүнийг өгөгдсөн утгаар дүүргэх үүрэгтэй (Зураг 9.9). Харин cout.write(ptr1,13); нь ptr1(=хаяг 1000) хаягаас эхэлж байршиг 13 тэмдэгтийг хэвлээд дуусгавар болно. Программын үр дунгээс үзэхэд "operator" мөрийн төгсгөлийн null тэмдэгтийн оронд зайд хэвлэсэн байна.

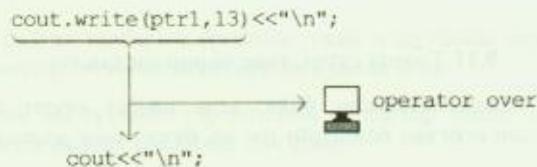


Зураг 9.9: Program #9.3-ын эхийн хоёр командын ажиллах зарчим

Дээр авч үзсэн write() функц түүнийг хэрэглэх объектын заалтыг буцаах учраас cout.write(ptr1,13); команда "operator over" мөрийг дэлгэцэлсний дараагаар

```

cout.write(ptr1,13) << "\n";
    |
    +--> cout << "\n"; болж богиносдог (Зураг 9.10).
  
```



Зураг 9.10: cout.write(ptr1,13)<<"\n"; командын ажиллах зарчим

### 9.10 cout << хэрэглээ

ostream классын (<<) гаргах оператор нь дэлгэшлэх зүйлээ эхлээд түүнд харгалзах тэмдэгт мөр рүү буулгаад дараа нь дэлгэц рүү гаргана. Өөрөөр зааж өгөөгүй үед cout<< нь дэлгэшлэх зүйлийг эхлээд доор үзүүлсний дагуу бэлддэг байна.

- Тэмдэгтэн утлыг, хэрэв хэвлэгдэг тэмдэгт бол нэг char урттай талбарт гаргах
- Бүхэл тоог, бас хасах тэмдэг тооны өмнө байвал түүнийг багтаах хангалттай өргөн талбарт хэвлэх
- Бодит тоог аравтын таслалын баруун талд б оронтойгоор гэхдээ утгат бус тэгүйгээр гаргах; тоог бэхжсэн цэгийн эсвэл тоон утгаас хамаарах тэмдэглэгээр тус тус харуулах; хэрэв тоо нь 1.68e07-оос их эсвэл 1.0e-04-ээс бага бол е тэмдэглэгээ хэрэглэх
- Тэмдэгт мөрийг түүний урттай дүйх хэмжээний талбарт гаргах

Янз бүрийн мэдээллийг гаргах жишээ програмыг Program #9.4-т үзүүллээ.

```

//Program #9.4
#include <iostream.h>
void main(void)
{
    cout << "abcdefghijklmnopqrstuvwxyz\n";
    char ch = 'a';
    int ij=102;

    cout << ch << ":\n";
    cout << ij << ":\n";
    cout << -ij << ":\n";
  
```

```

double d1 =1.600 ;
cout << d1 << ":\n" ;
cout << (d1+1.0/9.0) << ":\n" ;
double d2 =1.67e7 ;
cout << d2 << ":\n" ;
cout << (d2+2.0e5) << ":\n" ;
double d3 =2.3e-4 ;
cout << d3 << ":\n" ;
cout << (d3/10) << ":\n" ;
}

abcdefghijklmnopqrstuvwxyz
a:
102:
-102:
1.6:
1.711111:
16700000:
1.69e+07:
0.00023:
2.3e-05:
```

### 9.11 Тооны суурь тоог өөрчилж хэвлэх

*iostream.h* файлд тодорхойлогдсон байх *ios* класст оролт гаралттай холбоотой мэдээлийг хадгалах олон өгөгдөл байдгийн нэг нь бүхэл тоог дэлгэншлэхэд хэрэглэх суурь тоо юм.

*ostream* нь *ios* классаас удамших класс тул эх классынхаа бүх шинжийг өвлөж авсан байдаг. Мөн *cout* нь *ostream* классын объект, иймээс уг объект нь гаралтад хэрэглэх суурь тоо ямар байхыг зааж өгөхэд хэрэглэх гишүүн өгөгдлүүдтэй. Бүхэл тоог дэлгэшүүдээ түүний суурь тоог тусгай функц хэрэглэн хянаж болно. Гэхдээ ийм функц нь гишүүн биш функц юм. Иймийн учир тэдгээрийг ямар ч объектгүйгээр дуудаж болдог. Тоог аравтаар, арванзургаатар эсвэл наймтаар гаргахад *dec()*, *hex()*, *oct()* зэрэг функцээс тохиরхыг нь хэрэглэнэ. Жишээ нь, *hex(cout)*; команда суурь тоотой холбоотой *cout* объектын гишүүн өгөгдлийг арванзургаат болгож шинээр тогтоох ба энэ нь өөр суурь тоог сонгох хүртэл хүчинтэй байна. Үүний дараагаар програм тоог арванзургаатын системд хэвлэн. Суурь тоог удирдаж залах үүрэгтэй функцуудийг янз бүрээр хэрэглэдэг. Тухайлбад, *cout<<hex;* нь *hex(cout)*; функцийг дуудахтай адил юм. Иймд *cout<<hex* нь *hex(cout)* функциэр солигддог байхаар (*<<*) операторыг *ostream* класс дотор дахин тодорхойлсон байна. Дараах програм *dec*, *hex*, *oct* функцуудийн хэрэглээг харуулна.

```

//Program #9.5
#include <iostream.h>
void main(void)
{
    cout << "Enter a number:" ;
    int n ;
    cin >> n ;
    cout << "\nnumber base n      n*n\n" ;
    cout << "decimal" << "      " << n << "      " << n*n << "\n" ;
    cout << hex ;
    cout << "hexadecimal" << "      " << n << "      " << n*n << "\n" ;
    oct(cout) ;
    cout << "octal" << "      " << n << "      " << n*n << "\n" ;
```

```

cout << dec << "decimal" << "      " << n << "      " << n*n << "\n" ;
}

Enter a number:12
number base      n      n*n
decimal          12      144
hexadecimal     c      90
octal            14      220
decimal          12      144

```

### 9.12 Талбарын өргөний тохируулга

`ios` нь дэлгэцлэх талбарын өргөний талаарх мэдээллийг хадгалах гишүүн өгөгдлүүдтэй класс юм. Уг классын холбогдох гишүүн функцийдээр талбарын өргөнийг хянаж бас удирдаж болно. `ostream` нь `ios` классаас удамших класс учир түүний `cout` объект эх классынхаа бүх гишүүн функция рүү хандаж чадна. Талбарын өргөнтэй холбогдох дараах хоёр функци байна.

```

int width(void);
int width(int);

```

Эхнийх нь хэрэглэж байгаа талбарын өргөний тухай мэдээллийг олж буцаах бол удаах нь талбарын өргөнийг шинээр тогтоож өмнөхийг нь буцааж өгнө.

Дээрх `width()` функций нь түүнийг хэрэглэсний дараагаар хэвлэгдэх зөвхөн нэг л зүйлд нелөөлнө. Жишээ болгож дараах командыг авч үзье.

```

cout << '*' ;
cout.width(8) ;
cout << 12 << '*' << 22 << '*' ;

```

Эдгээр команд нь доорх мөрийг дэлгэцлэнэ.

```

*      12*22*

```

Энд 12 гэсэн тоо нь найман тэмдэгтийн өргөнтэй талбарын баруун хэсэгт бичигдэх ба үүнийг баруун тэгшлэл гэнэ. Дараа нь талбарын өргөн нь өмнөх байдалдаа буцаж орно. Иймээс хоёр од болон 22 гэсэн тоог өөрсдийнх нь бодит өргөнөөр бичсэн байна.

Хэрэв тогтоосон талбарын өргөн нь гаргах мэдээлэл багтахаар хангалттай бус бол C++ програм нь талбарын өргөнийг мэдээлэл багтахаар өргөсгөдөг. Жишээ нь, талбарын өргөн нь 2, бичих мэдээлэл нь 7 цифртэй бол өргөнийг 7 цифр багтахаар өргөсгөнө. `width()` функци хэрхэн ажилладгийг дараах програмын код, түүний үр дүнгээс үзж болно.

```

//Program #9.6
#include <iostream.h>
void main(void)
{
    char *name[3]={"Tom", "Dick", "Harry"} ;
    int income[3] = {1000,2000,3000} ;
    for(int i=0; i<3; i++)
    {
        cout.width (12);
        cout << name[i] << ":" ;
        cout.width (10) ;
        cout << income[i] << "\n" ;
    }
}

```



Tom:	1000
Dick:	2000
Harry:	3000

### 9.13 Ашиглагдаагүй зайл дүүргэх

Ашиглагдаагүй зайл нь өөрөөр зааж өгөөгүй бол зайн тэмдэгтээр дүүргэгдэнэ. Ийм зайл өөр тэмдэгтээр програмын замаар дүүргэх бол `fill()` функцийг хэрэглэдэг. Жишээ нь, функцийг `cout.fill('*');` гэж хэрэглэвэл зайл '\*' тэмдэгтээр дүүргэнэ. Үүнийг дараах програмаар үзүүллээ.

```
//Program #9.7
#include <iostream.h>
void main(void)
{
    char *name[3]={"Tom", "Dick", "Harry"} ;
    int income[3] = {1000,2000,3000} ;
    cout.fill('*') ;
    for(int i=0; i<3; i++)
    {
        cout.width(12);
        cout << name[i] << ":" ;
        cout.width (10) ;
        cout << income[i] << "\n" ;
    }
}
*****Tom:*****1000
*****Dick:*****2000
*****Harry:*****3000
```

### 9.14 Бутархайн нарийвчлал

Ховох цэгийн (аравтын таслалын) нарийвчлал гэдэг нь таслалын баруун талд бичигдэх цифрийн тоотой хамаатай ойлголт юм. Өөрөөр заагаагүй бол ийм цифрийн тоо нь б байх ба түүнийг `precision()` гишүүн функцээр өөрчилнэ. Жишээ нь, `cout.precision(4);` команд нь бодит тоог таслалын баруун талд 4 цифртэйгээр хэвлэх тохиргоо хийнэ. Сүүлд хийсэн тохиргоо нь `precision()` функцийг дахин хэрэглэх хүртэл хадгалагдаж үлдэнэ. Үүнийг дараах програм тодорхой харуулна.

```
//Program #9.8
#include <iostream.h>
void main(void)
{
    float f1 = 200.60 ;
    float f2 = 16.8+8.0/9.0 ;
    // cout.setf (ios::scientific) ;
    // cout.setf (ios::fixed) ;
    cout << "f1 = " << f1 << "\n" ;
    cout << "f2 = " << f2 << "\n" ;
    cout.precision (4) ;
    cout << "f1 = " << f1 << "\n" ;
    cout << "f2 = " << f2 << "\n" ;
}
f1 = 200.600006
f2=17.688889
```

```
f1=20.6  
f2=17.6889
```

Хэрэв cout объектын хувьд дэлгэцүү рүү тоо гаргах хэлбэрийг scientific эсвэл fixed гэгийн аль нэгээр зааж өгөх тохиолдолд (дээрх програм доторх тайлбар мөрүүд) precision() функцийн аргумент нь аравтын таслалын дараах цифрийн тоог заана. Харин дэлгэцүү рүү гаргах хэлбэрийг scientific эсвэл fixed гэж заагаагүй тохиолдолд аргумент нь нийт цифрийн тоо болно. Нарийвчлалын тохиргоо нь програм хэвлэх утгыг тайрахын оронд орон дэвшүүлэхэд хүргэдэг байна.

### 9.15 Утгат бус тэгийг хэвлэх

Заримдаа утгат бус тэгийг хэвлэх шаардлага гардаг. ostream классаас удамших ios класс нь гаргах зүйлийг бэлдэх утгыг хянах setf() гэсэн функцийтэй. Мөн setf() функцийн аргумент хэлбэрээр хэрэглэж болох хэд хэдэн тогтмол энэ класст байдаг.

Функцийг cout.setf(ios::showpoint); гэж дуудах нь cout объект утгат бус тэгийг хэвлэхэд хүргэнэ. Үүнийг дараах програмаас үзэж болно.

```
//Program #9.9  
#include <iostream.h>  
  
void main(void)  
{  
    float f1 = 20.60 ;  
    float f2 = 16.8+8.0/9.0 ;  
    cout.setf (ios::fixed ) ;  
    cout.setf (ios::showpoint) ;  
    cout << "f1 = " << f1 << "\n" ;  
    cout << "f2 = " << f2 << "\n" ;  
    cout.precision (4) ;  
    cout << "f1 = " << f1 << "\n" ;  
    cout << "f2 = " << f2 << "\n" ;  
}  
  
f1 = 20.600000  
f2 = 17.688889  
f1 = 20.6000  
f2 = 17.6889
```

### 9.16 cin объект

cin нь ios классаас удамших istream классын объект юм. Түүнд streambuf объектоор удирдагдах буферийг заах хаяган өгөгдөл байдаг. Мөн istream класс нь дахин тодорхойлсон (>) оруулгын оператортой.

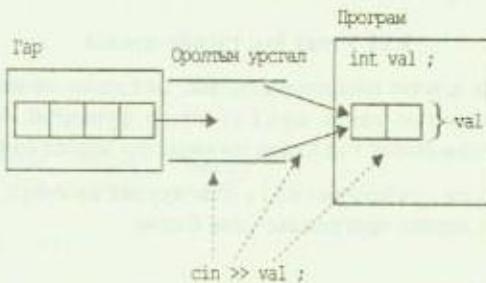
Объект cin нь стандарт оролтын урсгалыг төлөөлнө. Ер нь, оролтын урсгалыг гараар үүсгэнэ. Ингэхдээ, жишээ нь a, b, 1, 2 гэсэн цуврал товчийг товшиход 'a', 'b', '1', '2' гэсэн тэмдэгтийн урсгал бий болох ба тэдгээр нь анхандаа гарын буферт хадгалагдж байдаг.

Гараас бүхэл тоо оруулахдаа C++ програмд доорх шиг командыг хэрэглэх ёстой.

```
int val ;  
cin >> val ;
```

Энд val бол оруулсан тоо байрших хувьсагч юм. Оруулах бүхэл тоог хадгалах ойн байрлалыг заах хувьсагчийг байрших хувьсагч гэнэ. Оруулгын оператор гараас байт

мэдээлэл уншихын тулд OS-ийн тохирох функцийг дуудаж хэрэглэнэ. Буфер хоосон бол оруулга хийхийг сануулж дэлгэцийн заагуур анивчинаа. Гараас оруулах байт эхлээд гарын буферт хадгалагдана. Оруулгыг `<` товчоор дуусгавар болгоход буферийн доторх утга програм руу албадан дамжуулагдах бөгөөд оруулгын оператор тэдгээрийг `cin` объектоор удирдагдах буферт тухайлбал, байрших хувьсагчид хийнэ. Ингэхдээ уг оператор нь байрших хувьсагчийн төрөл ямар байхаас хамаарч цуваа байтад хувиргалт хийж гарах утгыг нь тухайн хувьсагч руу хийнэ. Ерийн оруулах үйлдлийн хийгдэх зарчмыг Зураг 9.11-д үзүүлсэн шигээр дүрслж болно.



Зураг 9.11: Оруулах үйлдлийн дүрслэл

### 9.17 Оруулгын оператор >>

Гараас 1, 2, 3 гэсэн цуврал гурван товчийг товшиход тэдгээр тэмдэгтийн дараалал нь байрших хувьсагч ямар төрлийн байхаас хамааран бүхэл тоо, бодит тоо эсвэл тэмдэг мөр мэтээр уншигдана. Иймээс оруулгын операторын хийх ажлын нэг бол байрших хувьсагчид нийшүүлэн төрөл хувиргалт хийх юм. Байт '1', '2', '3' гэсэн гурван байтын урсгал нь хэрэв `val` нь бүхэл тоон төрлийнх бол 123 гэсэн бүхэл тоог дараалсан 2 байт руу хөрвүүлж дүрслэнэ. Иймээс '1', '2', '3' гэсэн байтууд нь 123 гэсэн аравтын тооны хоёртын дүрслэлийн бага, ахлах байт болох (01111011) ба (00000000) гэсэн 2 байт руу хөрвөгдөж `val` хувьсагчийн хаяглах ойд хадгалагдана.

`istream` класс дотор (`>>`) операторыг доорх загвар бүхий `operator` функцийн дахин тодорхойлсон байдал.

```
<return-type> operator >>(argument_list) :
```

Янз бүрийн аргумент таньдаг, бас `istream` объектын заалтыг буцаадаг байхаар `operator >>()` функцийг `istream` классын тодорхойлолтод доор үзүүлсэн шигээр дахин тодорхойлж хэрэглэдэг.

```

class istream
{
public:
    istream & operator >>(char * );
    istream & operator >>(signed char * );
    istream & operator >>(unsigned char * );
    istream & operator >>(char & );
    istream & operator >>(signed char & );
    istream & operator >>(unsigned char & );
    istream & operator >>(short & );

```

```

istream & operator >>(int &) ;
istream & operator >>(long &) ;
istream & operator >>(unsigned short &) ;
istream & operator >>(unsigned int &) ;
istream & operator >>(unsigned long &) ;
istream & operator >>(float &) ;
istream & operator >>(double &) ;
istream & operator >>(long double &) ;

} ;

```

Дараах бүлэг командаар

```

int ii ;
cin >> ii ;

```

cin объект нь дээрх тодорхойлолтын дагуу istream & operator >>(int &) функцийг дудаж ажиллуулна. Иймээс cin>>ii; команд нь cin.operator >>(ii); гэж бичиж өгсөнтэй адил юм.

Дараах зүйлийг хийдэг байхаар бүхэл тоон аргумент бүхий operator >>() функцийг тодорхойлж болно. Тухайлбал, OS-ийн тохирох функцийт хэрэглэн тэмдэгтүүдийг гарас гарын буфер рүү уншина. Буфер албадан дамжуулагдахад тэмдэгтүүд син объектоор удирдагдах буфер рүү уншигдана. Гараас 1, 9, 6 гэсэн гурван товчийг товшсон гэж үзвэл тэдгээр тэмдэгт нь 196 гэсэн бүхэл тоо болж залгагдана. Иймд '1', '9', ба '6' гэсэн тэмдэгтүүд бүхэл тоон хувьсагчийн бага, ахлах байтууд шигээр хадгалагдах (11000100) ба (00000000) гэсэн хоёр байт утга болж хувирна.

Хэрэв ii нь double төрлийнх байсан бол operator >>() функц нь гарас цувруулан оруулж өгөх '1', '9', ба '6' гэсэн тэмдэгтүүдийг 196.0 гэсэн болит тоонд тухайлбал, уг тооны хоёртын дурслэл болох 8 байт мэдээлэлд хувиргана. Дараа нь тэдгээр байт мэдээллийг ii хувьсагчид хадгална.

Дараах

```

float ff ;
cin >> ff ;

```

командууд нь istream & operator >>(float &) функцийг син объект дуудахад хүргэх тул cin>>ff; команд нь cin.operator >>(ff); командын мөртэй адил юм. Харин доорх хоёр командын хувьд

```

char name[20] ;
cin >> name ;

```

cin объект, хэрэв C++ програм нь char төрлийг unsigned буюу тэмдэгтүй гэж үзэх бол istream & operator >>(unsigned char \*) функцийг дуудаж ажиллуулна. Харин C++ програм нь char төрлийг signed буюу тэмдэгтэй гэж үзэх бол istream & operator >>(signed char \*) функц дуудагдана. Иймд cin>>name; командыг cin.operator >>(name); гэсэн командаар орлуулж болно. Аргумент нь char төрлийн хаяг байх operator функц нь гарас оруулах тэмдэгтүүдийг name[] хүснэгт рүү хийнэ.

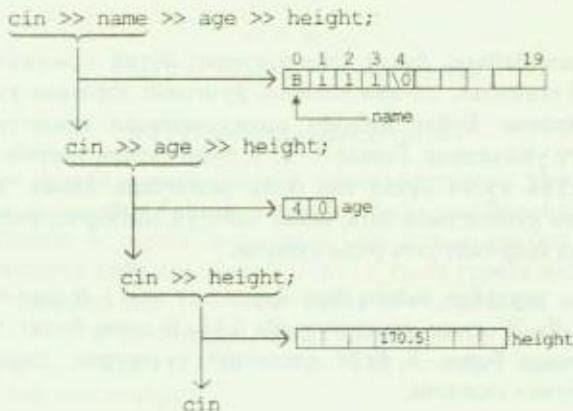
operator >>() функцийн терел, аргумент нь заалт байдаг. Аргумент нь заалт байснаар cin.operator >>(ii); функцийг дууддаг cin>>ii; шиг команд нь ii хувьсагчийн

хуулбар гэхэсээ жинхэнэ *ii*-тэй ажиллана. Иймд *cin* объект нь аргумент хэлбэрээр хэрэглэгдэж байгаа хувьсагчийн уттыг шууд өөрчлөх бололцоотой байна.

Дахин тодорхойлогдсон *operator >>()* функц тус бүрийн төрөл нь *istream* объектын<sup>51</sup> заалт байдаг. Ихвхтэй байгаа объектын заалтыг бушаадаг байхаар *operator* функцийг тодорхойлсон байдгийг өмнө үзүүлсэн *istream* классын тодорхойлтоос үзэж болно. Иймд *cin* объектын заалт *cin>>ii;* командаас бушна. Энэ нь цуврал оруулгыг цуврал гаралт шигээр нэгтих бололцоог програм хөгжүүлэгчид олгодог байна. Дараах командуудыг авч үзье.

```
char name[20] ;
int age ;
double height ;
cin >> name >> age >> height ;
```

Хэрэв компьютерийн гараас Bill 40 170.5-гэж бичиж оруулбал Зураг 9.12-т дүрсэлсэв шиг үйлдэл хийгдэнэ.



Зураг 9.12: *cin >> name >> age >> height;* командаын јр дун

Зураг 9.12-т үзүүлснээр *cin>>name* командаас буцах *cin* объект нь *age* хувьсагчтай холбогдох боловсруулалтыг хийх гэх мэтээр ажиллана.

### 9.18 *cin >>* нь гарын буферийг хэрхэн унших

C++ програм нь *cin>>val;* шиг командыг гүйцэтгэхдээ *cin* объектоос *operator>>()* функцийг дуудна. Тэгэхлээр, *cin.operator >>(val);* гэсэн дуудлага хийгдэнэ.

*operator()* функцийн бүх хувилбар дараах зүйлийг хийдэг байхаар бүтээгдсэн байдаг. Эхлээд гарын буферийг шалгана. Буфер хоосон бол програм түр зогсож дэлгэцийн заагуур анивчих ба энэ нь програм гараас оруулга хийхийг хүлээж байгааг илтгэнэ. OS-ээр дамжуулан гарын оруулгын функцийг дуудна. Оруулга нь ↲ товчийг товшиж дуусгавар болгох хүртэл буферлэгднэ<sup>52</sup>. Оруулга дуусгавар болоход оруулгын оператор буферэж уншиж авсан тэмдэгтүүдээ *cin* объектоор удирдагдах *val* рүү дамжуулахдаа уг хувьсагчийн төрөлд нийцэх утга болгож хувиргана.

<sup>51</sup> *istream* классын объект гэхийт товчоор *istream* объект гэж болно.

<sup>52</sup> Гарын буферт хадгалах, хадгалгахаа гэдгийг товчоор буферлэх, буферлэгдэх гэж болно.

Оруулгын оператор гарын буферийг хэрхэн уншдагийг авч үзье. Хувьсагч нь char төрлийнх байх тохиолдолд оруулгын оператор нь буферээс нэг тэмдэгт уншиж түүнийг байрших хувьсагч руу нь хийнэ. Иймд

```
char ch ;  
cin >> ch ;
```

шиг командын хувьд програмын биелэлтийн талбар нь `cin>>ch;` команд дээр шилжиж очиход гарын буферийг эхлээд шалгана. Буфер хоосон бол програм гараас оруулга хийхийг хүлээж тэндээ тур зогсоод дэлгэцийн заагуур анивчина. Оруулах тэмдэгтүүд буферлэгдэж улмаар ↓ товчоор дуусгавар болгоход оруулгын оператор нь буферийн эхний тэмдэгтийг бүрмэсөн уншиж аваад байрших хувьсагч руу нь хийнэ. Үүнийг дараах програмаас үзэж болно.

```
//Program #9.10  
#include <iostream.h>  
  
void main(void)  
{  
    char ch ;  
    cin >> ch ;  
    cout << "ch = " << ch << "\n" ;  
}
```

 abcdefgh..  
ch = a

`operator >>()` функц нь түүнийг хэрэглэж байгаа объектын заалтыг буцаах тул доор үзүүлсний адил цуврал оруулах үйлдлийг тодорхойлж болно.

```
char c1, c2, c3, c4 ;  
cin >> c1 >> c2 >> c3 >> c4 ;
```

Үүнийг хэрхэн програмчлахыг дараах програмд үзүүллээ.

```
//Program #9.11  
#include <iostream.h>  
  
void main(void)  
{  
    char c1, c2, c3, c4 ;  
    cin >> c1 >> c2 >> c3 >> c4 ;  
    cout << "\nc1 = " << c1 ;  
    cout << "\nc2 = " << c2 ;  
    cout << "\nc3 = " << c3 ;  
    cout << "\nc4 = " << c4 ;  
}  
  
 abcdefgh.. //Жишээ 1  
c1 = a  
c2 = b  
c3 = c  
c4 = d  
  
 ab cdefgh.. //Жишээ 2  
c1 = a  
c2 = b  
c3 = c  
c4 = d
```

```
ab..                                //Жишээ 3
c..
defgh..
c1 = a
c2 = b
c3 = c
c4 = d
```

Оруулгын оператор нь нэг тэмдэгтийн горимд цагаан зайл<sup>53</sup> хэрэгсэхгүй алгасдаг нь дээрх турван жишээнээс харагдана. Ингэхдээ цагаан зайл уншиж аваад буферээс арчиж дараа нь хэрэгсэхгүй орхидог байна. Дараах програм үүнийг үзүүлнэ.

```
//Program #9.12
#include <iostream.h>

void main(void)
{
    int count ;
    char ch ;
    count = 0 ;
    cin >> ch ;
    while (ch != '*')
    {
        cout << ch ;
        count++ ;
        cin >> ch ;
    }
    cout << "\n" ;
    cout << count << " characters read\n" ;
}

 abcd ef g h*ijk..                //Жишээ 1
abcdefgh..
8 characters read

abcd..                               //Жишээ 2
abcdeffgh..
efghijklm..
ijkl
11 characters read
```

Өмнөх хариуны бүдэг саарлаар бичигдсэн хэсэг нь cout<<ch; командын уишсан тэмдэгтүүд юм. Дээрх програмын while (ch != '\*') мөрийг while (ch != '\n') гэж өөрчлөөд програмыг дахин ажиллуулбал гар унших давталт тасралтгүй хийгдэх ба энэ нь cin >> команд нь цагаан зайл хэрэгсэхгүй алгасах учир зэлжит уншилтаар ch хэзээ ч '\n' болдогтүйтэй холбоотой юм.

Нэг тэмдэгтийн бус горимд оруулгын оператор нь оролтын урсалаас (гарын буферээс) нэг хэсгийг нь уншина. Оруулга хадгалах (cin >> дэх) хувьсагч нь тэмдэгтэн хүснэгтийн хаяг байх тохиолдолд оруулгын дунд байх цагаан зайлгаар нь оруулгын хэсгийг зааглаж уншина. Дараах програмыг авч үзье.

```
//Program #9.13
#include <iostream.h>
```

<sup>53</sup> Хэвтэй ба босоо догол, шинэ морийн тэмдэгт, зайл цагаан зайл гэно.

```
void main(void)
{
    char name[10] ;
    cin >> name ;
    cout << "name=" << name << "\n" ;
}
```

 abcde..  
name=abcd

Хүснэгтийн хэмжээнээс хэтрэх, тухайлбал 10-аас олон тэмдэгт оруулбал алдаа гарч програм хэвийн бусаар дуусгавар болно.

 abcdefghijklmn..  
abnormal program termination

Харин яг 10 тэмдэгт оруулбал дараах үр дүн гарна.

 abcdefghi..  
name=abcdefghijklm

Оруулга дунд цагаан зайн тэмдэгт байвал програмд хэрхэн нөлөөлөхийг доорх үр дүн харууна.

```
 abcde ef..           //зайн
name=abcd

 abcde ef..           //хэвтээ догол
name=abcd
```

Гарын буферийг цагаан зайн хүртэл уншиж тэмдэгтуүдийг зориулалтын хаяган хувьсагчийн заах ойгоос эхэлж хийнэ. null тэмдэгтийг төгсгөлд нь бас иэмж бичнэ. Дараагийн ушилт нь цагаан зайнгаас эхэлнэ. Дараах програмыг авч үзье.

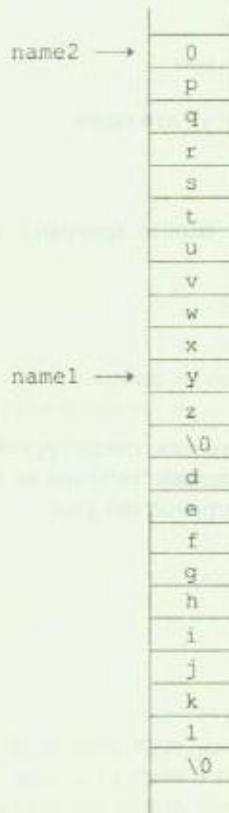
```
//Program #9.13A
#include <iostream.h>

void main(void)
{
    char name1[10] ;
    char name2[10] ;
    cin  >> name1   >> name2 ;
    cout << "name1=" << name1 << "\n" ;
    cout << "name2=" << name2 << "\n" ;
}

 abcdef..  
pqrstuv..  
  
name1=abcdef  
name2=pqrstuv..  
                                //оруулга дунд зайн оруулбал  
  
abcdef gh..  
name1=abcdef  
name2=gh..  
                                //хүснэгтийн хэмжээнээс хэтрэх мэдээлэл оруулбал  
abcdefgijkl..  
opqrstuvwxyz..  
name1=yz..
```

```
name2=opqrstuvwxyz  
abnormal program termination
```

Программын сүүлийн үр дүнг дараах байдалар тайлбарлаж болно. Зураг 9.13 нь `char` төрлийн `name1[10]` ба `name2[10]` гэсэн хоёр хүснэгтийн байрших ойг дүрслэх багаад орууллын оператор уг зурагт үзүүлсэн шигээр хүснэгтүүдийг дүүргэнэ. Иймд `cout << name1;` команда нь `name1` хаягаас `null` хүртэлх тэмдэгтүүдийг, тухайлбал Зураг 9.13-т үзүүлсний дагуу зөвхөн "yz" мөрийг дэлгэцлнэ. Үүнтэй адиллаар, `cout << name2;` нь `name2` хаягаас `null` хүртэлх тэмдэгтүүдийг, тухайлбал Зураг 9.13-т үзүүлсний дагуу "opqrstuvwxyz" мөрийг дэлгэцлнэ.



Зураг 9.13: `char name1[10]` ба `char name2[10]` хүснэгтүүдийн дүрслэл

Орууллын хувьсагч `int`, `float` гэх мэт төрлийнх байвал гарын буферээс унших үйл нь чагаан зал эсвэл хувьсагчийн төрөлтэй үл нийцэх тэмдэгт хүртэл, явагддагийг дараах программаар үзүүлээ.

```
//Program #9.14  
#include <iostream.h>  
  
void main(void)  
{  
    int val ;  
    cout << "\nEnter an integer value: " ;  
    cin  >> val ;
```

```

cout << "val=" << val << "\n" ;
}
123 ↴           //Жишээ 1
val=123

256.8 ↴         //Жишээ 2
val=256

123z ↴          //Жишээ 3
val=123

296 5 ↴         //Жишээ 4
val=296

```

Жишээ 2-т 256.8 нь 256 болж тайрагдсан биш, харин ямар ч бүхэл тоон өгөгдөл (.) цэг байх ёсгүй тул түүний өмнөхийг гарын буферээс уншиж цэг буферт үлдэнэ. Иймд ээлжит уншилт байвал тэрээр цэгээс эхэлнэ.

Жишээ 3-т 'z' тэмдэгт нь бүхэл тоонд байж болохгүй учир гарын буферээс 'z' хүртэлхийг уншиж харин 'z' нь буферт үлдэнэ. Үнэхэр тийм эсэхийг програмаас үзэж болно.

```

//Program #9.15
#include <iostream.h>
void main(void)
{
    int val = 100 ;
    char name[10] ;
    cout << "\nEnter an integer value and a string: " ;
    cin >> val >> name ;
    cout << val << "---" ;
    cout << name << "\n" ;
}

120 ↴           //Жишээ 1
abcd ↴
120---abcd

256.5 ↴         //Жишээ 2
256---.5

123xy ↴          //Жишээ 3
120---xy

120 xy ↴         //Жишээ 4
120---xy

xyz abcd ↴       //Жишээ 5
100---

```

Жишээ 5-д "xuz" нь програмын шаардлагад мөн таараахгүй учир энэ тохиолдолд оруулгын оператор нь val хувьсагчийн уттыг хэвээр үлдээж cin объектын оронд 0 буцаана.

### 9.19 istream арга: get(char &)

Гарын буферээс ээлжит тэмдэгтийг авчрах үүрэгтэй гишүүн функц istream класст байдаг ба түүний загвар нь istream & get (char &) юм. Аргумент нь заалтан торлийнх байх тул уг функц нь аргументийн хуулбартай бус харин өөртэй нь харьцаж ажилладаг. Энэ функц буферийн ээлжит тэмдэгт нь цагаан зайд байсан ч түүнийг авчирч аргументийн утга болгодог байна. Функцийг istream объектоор дамжуулан доор үзүүлсэн шигээр дуудна.

```
    cin.get(ch) ;  
  
get(char &) нь гарын буферийг шалгаж буфер хоосон бол дэлгэцийн заагуур анивчин  
гараас оруулга хийхийг хүлээнэ. Оруулах тэмдэгтүүд - товчоор оруулгыг дуусгавар болгох  
хүртэл буферэгдэж байлаг. Оруулга дуусгавар болоход эсвэл гарын буферийг шалгахад  
буфер хоосон биш байсан бол буфериин эхний тэмдэгтийг ch рүү уншиж буферээс арчина.
```

Энэ аргыг хэрэглэх жишээ програмыг доор үзүүллээ.

```
//Program #9.16  
#include <iostream.h>  
  
void main(void)  
{  
    int count = 0 ;  
    char ch ;  
    cout << "\nEnter a string: " ;  
    cin.get(ch) ;  
    while (ch != '\n')  
    {  
        cout << ch ;  
        count++ ;  
        cin.get(ch) ;  
    }  
    cout << "\n" ;  
    cout << count << "charactres read\n" ;  
}  
Hello world, Hello beautiful world!  
34 characters read
```

Програм эхлээд гарын буферээс 'H' тэмдэгтийг уншиж дэлгэнэлээд тоолуурын хувьсагчийн  
утгыг нэгээр ижмээр. Програм цагаан зайл оролцуулаад бүхий л тэмдэгтүүдийг уншина. Шинэ  
морийн тэмдэгт уншигдахад while давталт дуусгавар болно. Хувьсагч count нь гараас  
оруулсан (шинэ мөрийн тэмдэгтээс бусад) бүх тэмдэгтийг тоолж эсийн тоог буцаана.

get(char &) функция нь түүнийг дуудах cin объектын заалтыг бушаах тул олон get(ch)  
функцийт доор үзүүлсэн шигээр нэг cin объектын хувьд угсаагаар хэрэглэж болно.

```
char c1, c2, c3, c4 ;  
cin.get(c1).get(c2).get(c3) >> c4 ;
```

Команд хийгдэх үед get функциүүд Зураг 9.14-т үзүүлсэн шигээр нэг нэгээр хорогдоно.  
Оруулгын эхний тэмдэгт c1 рүү, удаах нь c2 рүү гэх мэтээр уншигдана. Үүнийг даразах  
жишээнүүдээр үзүүллээ.

1. Оруулга нь abcde бол

```
c1 ← a  
c2 ← b  
c3 ← c  
c4 ← d
```

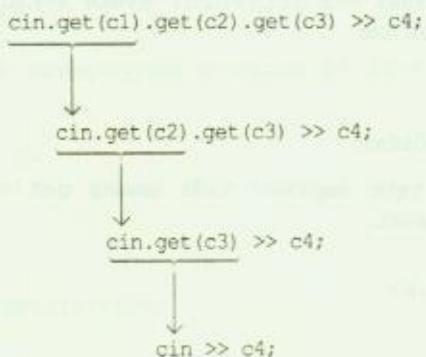
2. Оруулга нь ab cd бол

```
c1 ← a  
c2 ← b  
c3 ← (зайл)  
c4 ← c
```

3. Оруулга нь abc d- бол

```
c1 ← a  
c2 ← b  
c3 ← c  
c4 ← d
```

Оруулга 2, 3 нь хоорондоо ялгаатай. Оруулга 3-ын хувьд зайд бус харин 'd' уншигдана. Цагаан зайлж чаддаг нь `get(char &)` функцийн чухал шинж юм.



Зураг 9.14: `cin.get(ch).get(ch).get(ch) >> ch;` командын ажиллах зарчмын

Хэрэв `cin.get(ch)` нь файлын төгсгөл заах тэмдэгттэй<sup>54</sup> (бодит эсвэл зохиомлоор бий болгосны аль нь ч байж болно.) тааралдвал түүнийг `ch` рүү уншдагтүй. Үүнээс гадна, файлын төгсгөл болоход `get(char &)` функцийн түүнийг дуудсан `cin` объект бус харин 0 буцна. Зөв оруулгын үед өөрөөр хэлбэр `cin.get(ch)` функцийн буцах утга `cin` объект (энэ нь 0 бус) байх тохиолдолд `while` давталт үргэлжилнэ.

Файлаас тэмдэгт уншдаг байхаар Program #9.16-д бага зэрэг оорчлолт хийж болохыг Program #9.17-д үзүүллээ.

```
//Program #9.17  
#include <iostream.h>  
void main(void)  
{  
    int count = 0 ;  
    char ch ;  
    cout << "\nEnter characters: " ;  
    while (cin.get(ch))  
    {  
        cout << ch ;  
        count++ ;  
    }  
    cout << "\n" ;  
    cout << count << " characters read\n" ;  
}  
  
Hello world, Hello beautiful world [Ctrl]-[Z]I love you...  
34 characters read
```

<sup>54</sup> Үүнийг товчоор "файлын төгсгөл" гэж болно. DOS-д файлын төгсголийг `Ctrl+Z` гэсэн хоёр товчны нийтээмжээр зохиомлоор үүсгэж болно.

Файлын төгсгөл тавих Ctrl, Z хоёр товчны нийлэмжийг оруулгын дотор [Ctrl]-[Z] гэж бичиж өгсөн. Файлаас унших үйлдэл файлын төгсгөл гарах хүртэл үргэлжлэхийг дээрх код харуулна.

#### 9.20 istream арга: get(void)

Энэ гишүүн функци нь зэлжит гарын оруулгыг авчирч өгнө. Ийм оруулга нь цагаан зайд байж болно. Функцийн загвар нь int get(void) юм. Уншсан тэмдэгтээ get(char &) функцийнхөө аргумент рүү хийх бол get(void) функци дуудагдсан газраа буцааж өгдөг байна. Иймээс доор үзүүлсэн шиг

```
char ch ;
ch = cin.get() ;

үйлдлийг хийх боломжтой байдаг.
```

Program #9.16-д бага зэрэг өөрчлөлт хийх замаар get(void) функцийг хэрэглэх програмыг доорх шигээр бичин.

```
//Program #9.18
#include <iostream.h>

void main(void)
{
    int count = 0 ;
    char ch ;
    cout << "\nEnter a string: " ;
    ch = cin.get() ;

    while (ch != '\n')
    {
        cout << ch ;
        count++ ;
        ch = cin.get() ;
    }
    cout << "\n" ;
    cout << count << "charactres read\n" ;
}
```

 Hello world, Hello beautiful world[Ctrl]-[Z]I love you...
34 characters read

Файлын төгсгөлд хүрэхэд cin.get() нь файлын төгсгөл заах тэмдэгтийг (EOF<sup>55</sup>) буцаана. Дээрх Program #9.18-д өөрчлөлт хийх замаар файлаас унших програмыг доор үзүүлсэн шигээр бичиж болно.

```
//Program #9.19
#include <iostream.h>

void main(void)
{
    int count = 0 ;
    char ch ;
    cout << "\nEnter a string: " ;
    ch = cin.get() ;
    while (ch != EOF)
    {
```

<sup>55</sup>EOF - end of file - файлын төгсгөл

```
    cout << ch ;
    count++ ;
    ch = cin.get() ;
}
cout << "\n";
cout << count << " charactres read\n" ;
}

Hello world, Hello beautiful world[Ctrl]-[Z]I love you.
34 characters read
```

Мөн Program #9.19-ийг хялбаршуулан Program #9.20-д үзүүлсэн шигээр оорчилж болно.

```
//Program #9.20
#include <iostream.h>

void main(void)
{
    int count = 0 ;
    char ch ;
    while ((ch = cin.get())!=EOF)
    {
        cout << ch ;
        count++ ;
    }

    cout << "\n";
    cout << count << " charactres read\n" ;
}

Hello world, Hello beautiful world[Ctrl]-[Z]I love you.
34 characters read
```

get(void) функц нь бүхэл тоон утга буцаах тул уг функцийг доор үзүүлсэн шигээр угсуулан хэрэглэж болдоггүй.

```
cin.get().get().get()
```

Гэхдээ доорх хэлбэрээр хэрэглэх боломжтой юм.

```
char c1, c2, c3 ;
c3 = cin.get(c1).get(c2).get() ;
```

Оруулгын эхний тэмдэгтийг c1 рүү, удаахыг нь c2 рүү, гуравдахийг нь c3 рүү тус тус утга онооно.

#### 9.21 istream арга: get(char \*, int, char ='\\n')

istream классын аргууд болох

```
istream & get(char &),
int get(void)
```

функцийд оруулгын урсалаас нэг тэмдэгтийг нэг удаадаа уншдаг тухай өмнө үзсэн. Тэгвэл уг классын гишүүн болох get(char \*, int, char ='\\n') функцээр нэг үйлдлээр тэмдэгт мөр унших боломжтой юм. Энэ функц нь дараах загвартай.

```
istream & get(char *, int, char ='\\n') ;
```

Функцийн эхний аргумент нь оруулах мөрийг хадгалах ойн хаяг, хөрдахь нь унших тэмдэгтийн тооноос нэгээр их тоо байна. Энэ нэмэлт байтыг мөрийн төгсгөл заагч ('\0') тэмдэгтийг тавихад хэрэглэн. Гуравдахь аргументийн шууд авах гарааны утга нь зааглагч тэмдэгт болох ('\n') шинэ мөрийн тэмдэгт юм. Функцийг доор үзүүлсэн шигээр хэрэглэн.

```
char line[50] ;
cin.get(line, 50) ;
```

Команд нь оруулгын урсалаас эхний 50-1=49 эсвэл эхний ('\n') хүртэлх тэмдэгтийг унших үүрэгтэй. Уншсан зүйлээ line[] хүснэгт рүү хадгалаад төгсгөлд нь null тэмдэгтийг нэмж хийнэ. Зааж өгсон тооны тэмдэгтийг уншсан эсвэл зааглагч тэмдэгт дээр очсон үед уншилт зогсоно. Доорх cin.get(line, 50, '\*'); командаар оруулгын эхний 49 тэмдэгтийг эсвэл эхний (\*) хүртэлх тэмдэгтийг уншина.

```
char line[50] ;
cin.get(line, 50, '*') ;
```

get() функцийн ('\n') шинэ мөрийн эсвэл (\*) зааглагч тэмдэгтийг оруулгын буферийн үлдээхэн. Тэгэхдээр, гэлгээр нь элжит уншилтын эхний тэмдэгт болно.

Энэхүү istream & get(char \*, int, char ='\n') функцийн хэрэглээг доор үзүүллээ.

```
//Program #9.21
#include <iostream.h>

void main(void)
{
    char name[10] ;
    cin.get(name, 8) ;
    cout << "name=" << name << "\n" ;
}


name=abcde
abcdefghijklmn.
name=abcdefg
```

Program #9.22 нь дээрх функцийн өөр хувилбарыг хэрэглэх жишээ юм.

```
//Program #9.22
#include <iostream.h>

void main(void)
{
    char name1[10] ;
    char name2[10] ;
    cin.get(name1, 10, '*') ;
    cin.get(name2, 10) ;
    cout << "\nname1=" << name1 ;
    cout << "\nname2=" << name2 ;
}


name1=abcdef
name2=*ghijk

abcdefghijklmn.
name1=abcdefghi
```

```
name2=jk1*mn
abcdef..
ghijkl..
name1=abcdef
gh
name2=ijkl
```

Дээрх програмыг бага зэрэг өөрчилж `cin.get(name1,10,'*');` командын оронд `cin.get(name1, 10);` командыг хэрэглэвэл дараах үр дүн гарна.

 abcdef..
name1 = abcdef
name2 =

Эхний `cin.get(name1,10);` нь шинэ мөрийн тэмдэгт хүртэлхийг (өөрийг нь оруулахгүйгээр) унших тул уг тэмдэгт нь оруулгын буферт үлдэнэ. Иймд буфер хоосон биш байх тул дараагийн `cin.get(name2,10);` командаар уншигдах зүйл буферт байхгүй ч дэлгээний заагуур анивчихгүй.

#### 9.22 istream apra: `getline(char *, int, char ='\n')`

Тэмдэгт мөр унших функц доор үзүүлсэн шиг загвартай байна.

```
istream & getline(char *, int, char = '\n')
```

Энэ функц нь өмнө үзсэн `get(char *, int, char ='\n')` функцтэй төстэй. Гэхдээ тэдгээрийн хооронд байх гол ялгаа гэвэл `getline()` функц түүний хоёрдахь аргумент болох дуусгаврын тэмдэгтийг оруулгын буферт үлдээдэггүй явдал юм. Уг тэмдэгтийг уншсан мөрийнхөө төгсгөлд бичнэ. Түүний дараагаар `null` тэмдэгт бас бичигдэнэ. Дараах програмууд `getline()` функцийн хэрэглээг харуулна.

```
//Program #9.23
#include <iostream.h>
void main(void)
{
    char name[10] ;
    cin.getline(name, 10) ;
    cout << "\nnname = " << name << "\n" ;
}
```

 abcde..
name = abcde
abcdefghijklmn..
name = abcdefghi

```
//Program #9.24
#include <iostream.h>
void main(void)
{
    char name1[10] ;
    char name2[10] ;
    cin.getline(name1, 10, '*') ;
    cin.getline(name2, 10) ;
    cout << "\nnname1 = " << name1 ;
    cout << "\nnname2 = " << name2 ;
}
```

```


    abcdef*ghijk...           //Жишээ 1
    name1 = abcdef*
    name2 = ghijk

    abcdefghijkl*mn...      //Жишээ 2
    name1 = abcdefghi
    name2 = jkl*mn

    abcdef...                //Жишээ 3
    ghijkl...
    name1 = abcdef
    gh
    name2 = ijk

    abcdef...                //Жишээ 4
    ghijkl...
    name1 = abcdef

    name2 = ijk

```

Сүүлийн програмын `cin.getline(name1,10,'*');` командын оронд

```
    cin.getline(name1,10);
```

командыг хэрэглэх замаар бага зэргийн өөрчлөлт оруулбал Жишээ 4-т үзүүлсэн шиг үр дүн гарах ба түүний сүүлийн хоёр мөрийн хооронд давхар зайд байна. Энэ нь эхний `cin.getline(name1, 10);` команд нь шинэ мөрийн тэмдэгт бүхий мөрийг гарын буферээс унших тул уг тэмдэгт хоёр мер хооронд нэмэлт зайд гаргасантай холбоотой юм. Харин `cin.getline(name2, 10);` команд хийгдэх үед буфер хоосон байх тул дэлгэцийн заагуур анивчина. Иймд дээр үзүүлсэн `ghijkl...` ээлжит оруулгыг хийсэн байна.

### 9.23 istream арга: ignore()

Энэ арга нь `istream & ignore(int=1, int=EOF)` гэсэн загвар бүхий функц юм. Функц хоёр аргументтэйгээс эхнийх нь унших тэмдэгтийн тоо хэмжээг, хоёрдахь нь дуусгаврын тэмдэгтийн тус тус заана. Функцийг `cin.ignore(80, '\n')` гэж дуудвал оруулгын эхнээс 80 тэмдэгт эсвэл '\n' хүртэлхийг хэрэгсэхгүй алгасна. Хоёр аргументийн нь авах утта өөрөөр зааж огоогүй байвал харгалzan 1, EOF байна.

Энэ функцийн хэрэглээг дараах хоёр програмаар үзүүллээ.

```

//Program #9.25
#include <iostream.h>

void main(void)
{
    char name[10] ;
    cin.ignore(10, '\n') ;

    cin.get(name, 10) ;
    cout << "\nnname = " << name << endl ;
}

```

```

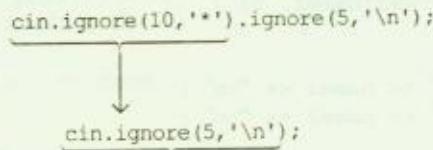

    abcdefghijklmn.
    name = klmn

```

```
//Program #9.26
#include <iostream.h>
void main(void)
{
    char name[10] ;
    cin.ignore(10, '*') ;
    cin.get(name, 10) ;
    cout << "\nname=" << name << endl ;
}
```

 abcde\*fghijklmn.  
name=fghijklmn

Олон ignore() функцийг нэг cin объектын хувьд угсуулан хэрэглэж болдгийг Зураг 9.15 болон Program #9.27-гоос үзэж болно.



```
cin.ignore(10, '*').ignore(5, '\n');
          ↓
    cin.ignore(5, '\n');
```

Зураг 9.15: `cin.ignore(10, '*').ignore(5, '\n')` командын ажиллах зарчим

```
//Program #9.27
#include <iostream.h>
void main(void)
{
    char name[10] ;
    cin.ignore(10, '*').ignore(5, '\n') ;
    cin.get(name, 10) ;
    cout << "\nname=" << name << endl ;
}
```

 abcde\*fghijklmn.  
name=klmn

## 9.24 istream арга: read()

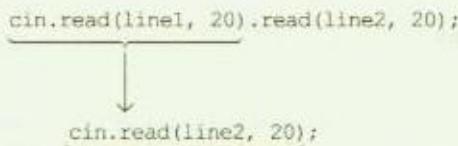
read() функция нь доор үзүүлсэн загвартай.

```
istream & read (char *, int);
```

Эхний аргумент нь унших мөрийг хадгалах ойн хаяг, удаах нь стандарт оруулгаас унших тэмдэгтийн тоо юм. Гарын буферээс 20 тэмдэгт уншиж `line[]` хүснэгт рүү хадгалах командыг доор үзүүлсэн шигээр бичин.

```
char line[20] ;
cin.read(line, 20) ;
```

Энэ функция хүснэгтийн төгсгөлд `null` тэмдэгтийг нэмж бичдэггүйгээрээ өмнө үзсэн `get()`, `getline()` функциүүдээс ялгаатай юм. Буцаах зүйл нь `istream` классын заалт (`istream &`) байх тул `read()` функцийг угераагаар хэрэглэж болдгийг доор Зураг 9.16-д үзүүллээ.



Зураг 9.16: `cin.read(line1,20).read(line2,20);` командын ажиллах зарчим

Функций доорх програмд үзүүлсэн шигээр хэрэглэнэ.

```

//Program #9.28
#include <iostream.h>

void main(void)
{
    char name1[10] ;
    char name2[10] ;
    cin.read(name1, 5).read(name2, 5) ;
    name1[5] = '\0' ;
    name2[5] = '\0' ;
    cout << "\nname1 = " << name1 << "\n" ;
    cout << "\nname2 = " << name2 << "\n" ;
}

 abcdefghijklmn.
name1 = abcd
name2 = fghij
  
```

#### 9.25 istream арга: peek()

`peek()` нь `int peek(void);` гэсэн загвартай функци юм. Функци гарын буфериийн эхний тэмдэгтийг буфферээс нь гаргахгүйгээр уншина. Тэгэхлээр, `peek()` нь оруулгын буфериийн эхнд байгаа тэмдэгтийг шалгах боломжийг програм хөгжүүлэгчид олгодог. Функцийн хэрхэн хэрэглэхийг доорх програмд үзүүллээ.

```

//Program #9.29
#include <iostream.h>
void main(void)
{
    char name[10] ;
    char ch ;
    ch = cin.peek() ;
    cout << "\nch = " << ch ;
    ch = cin.peek() ;
    cout << "\nch = " << ch ;
    cin.get(name, 10) ;
    cout << "\nname = " << name ;
}

 abcdefghijklmn.
ch = a
ch = a
name = abcdefghi
  
```

Програмын удирдлага `ch=cin.peek();` команд дээр очиход програм түр зогсч дэлгэшний заагуур анивчина. Энэ нь гарын буфер хоосон байгаатай холбоотой юм. Гараас отгөдөл оруулсны дараагаар эхний `ch=cin.peek();` команд хийгдэж буфериийн эхний тэмдэгтийг

ch рүү хийнэ. Түүнийг дараагийн нь cout командаар дэлгэц рүү гаргана. Хоёрдахь ch=cin.peek(); командаар програмын ажиллагаа түр зогсдоггүй нь гарын буфер хоосон биштэй холбоотой юм. Энэ команд буфериин эхний тэмдэгт болох 'a'-г дахин уншиж ch рүү хийснийг дараагийн нь cout командаар дэлгэшлэнэ. Ээлжит get(name,10); командаар 9 тэмдэгт уншигдаж, ингэснээр дээр үзүүлсэн шиг үр дүн гарна.

peek() функцийн өөр хэрэглээг доор үзүүллээ.

```
//Program #9.30
#include <iostream.h>

void main(void)
{
    char name[10] ;
    char ch ;
    int i = 0 ;
    while((ch = cin.peek()) !='*')
        cin.get(name[i++]) ;
    name[i] = '\0' ;
    cout << "\nname = " << name ;
}

💻 abcdef*ghijklmn.
name = abcdef
```

### 9.26 istream арга: gcount()

gcount() нь int gcount(void); загвартай бөгөөд хамгийн сүүлд хэрэглэсэн

```
istream& get(char *,int,char='\n')
```

эсвэл

```
istream& getline(char *, int,char='\n')
```

функцийн уншсан тэмдэгтийн тоог олж буцаадаг функц. Гар унших get(name,80); функцийг хэрэглэх тохиолдолд шинэ мөрийн тэмдэгт хаана байгаагас хамаарч 79-оос ихгүй тэмдэгт уншиж чадна. Уншсан тэмдэгтийн тоог олох өөр нэг арга бол strlen() функцийг strlen(name); гэж хэрэглэх юм. Гэвч оруулгын урсгалаас дөнгөж уншсан тэмдэгтуүдийн тоог gcount() функцийн мэдэж авах нь хамгийн тохиромжтой арга юм. Энэ функцийг хэрхэн хэрэглэхийг дараах програмаар үзүүллээ.

```
//Program #9.31
#include <iostream.h>

void main(void)
{
    char name[10] ;
    int count ;
    cin.get(name, 10) ;
    cout << "\nname = " << name << endl ;
    count = cin.gcount() ;
    cout << count << " characters read\n" ;
}

💻 abcdefghijklmn.
name = abcdefghi
9 characters read
```

### 9.27 istream арга: putback()

Функцийн загвар нь `istream & putback(char);` юм. Энэ функцийн огогдсон тэмдэгтийг орууллын буфер рүү хийнэ. Буферт нэмж оруулсан тэмдэгт нь орох урсгалын эхний тэмдэгт болдог. Функцийн авах нэг аргумент бол буфер рүү буцааж хийх тэмдэгт юм. Функцийн нь `istream &` төрлийнх. Энэ функцийн хэрэглээг доорх програмаас үзэж болно.

```
//Program #9.32
#include <iostream.h>

void main(void)
{
    char name[10] ;
    cin.get(name, 10) ;
    cout << "\nname = " << name ;
    cin.putback('*') ;
    cin.get(name, 10) ;
    cout << "\nname = " << name ;
}
abcdefghijklmn
name = abcdefghi
name = *jklmn
```

putback() аргыг хэрэглэх өөр жишээг доорх програмд үзүүллээ.

```
//Program #9.33
#include <iostream.h>

void main(void)
{
    char ch ;
    cin.get(ch) ;
    while (ch != '\n')
    {
        if(ch != '#')
            cout << ch ;
        else
            cin.putback('*') ;
        cin.get(ch) ;
    }
}
abcd#efgh_
abcd*efgh
```

Оруулга доторх '#' тэмдэгт нь '\*' тэмдэгтээр солигдоно. putback() функцийн буцаах утгы бол `cin` объектын заалт байна. Иймээс уг функцийг угсрулан хэрэглэж болох ба үүнийг доорх програмд харууллаа.

```
//Program #9.34
#include <iostream.h>
void main(void)
{
    char ch ;
    cin.get(ch) ;
    while (ch != '\n')
    {
        if(ch != '#')
```

```

        cout << ch ;
    else
        cin.putback ('*').putback ('*') ;
        cin.get (ch) ;
    }
}

abcd#efgh
abcd**efgh

```

Оруулга доторх '#' тэмдэгт нь '\*\*' тэмдэгтүүдээр солигдсоныг дээрх үр дүнгээс харж болно.

### 9.28 Файлын оролт гаралт

Файл бол янз бүрийн хадгалуур тохооромж дээр бичсэн байгаа байтын цуглуулбар юм. Тэдгээр байт нь бичгийн файлын хувьд ASCII тэмдэгтүүд болно. Байт бүр 8 битийнх байна. OS нь файл үүсгэж эрхлэх зэргээр файлтай холбоотой ажил үйлийг гүйцэтгэнэ. Тухайлбал, файлын хэмжээ, түүний байршилыг хадгалж файлын замналын бүртгэлийг нь хотелиө. Програмчлалын хэл нь OS-ийн файлтай ажиллах чадварыг хэрэглэх өөрийн хэрэгсэлтэй байдаг. Мөн файлыг програмд холбож файл үүсгэж түүн рүү мэдээлэл бичиж, уншиж чадлаг.

C++ хэлний хувьд файлын оролт гаралт нь стандарт оролт гаралт шиг явагдана. Стандарт гаралт руу бичихийн тулд, жишээ нь `ostream` классын `cout` объектыг хэрэглэнэ. Энэ класс нь гаргах зүйлийг дэлгэцлэх үүрэг бүхий `put()`, `write()` гэх мэт арга болон (`<<`) гаргах оператортой. Үүнтэй адил, стандарт оролтын тохооромжөөс мэдээлэл уншиж авахын тулд `istream` классын `cin` объектыг хэрэглэнэ. Энэ класс нь `get()`, `getline()` гэх мэт арга болон (`>>`) оруулгын оператортой. Файлд бичих унших үйлийг ижил арга замаар хийдэг байхын тулд файлын урсгалын объект үүсгэж хэрэглэдэг.

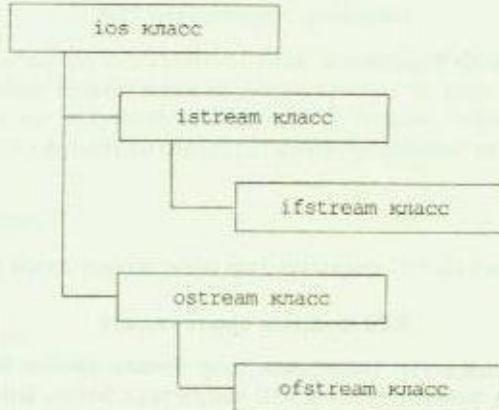
#### 9.28.1 `fstream.h` файл

C++ хэлний хувьд файлтай холбоотой оролт гаралтын үйлийг `fstream.h` толгой файл дотор тодорхойлж огсон классуудаар эрхэлнэ. Энэ файл нь өөр дотроо `iostream.h` файлыг ачаалах тул `fstream.h` файлыг хэрэглэж байгаа програмд `iostream.h` файлыг бас зааж өгөх шаардлагагүй. `fstream.h` файлд тодорхойлсон зарим классыг доор жагсааж үзүүллээ.

- `ifstream` класс. `istream` классаас удамшиж учир `ifstream` классын объект нь `istream` класс дотор тодорхойлогдсон `get()`, `getline()` гэх зэрэг аргыг шууд хэрэглэж чадна.
- `ofstream` класс. `ostream` классаас удамшиж тул `ofstream` классын объект нь `ostream` классын `put()`, `write()` гэх зэрэг аргыг мөн хэрэглэх боломжтой.
- `fstream` класс. `ifstream`, `ofstream` классуудаас удамшиж класс юм.

Дээр дурдсан `istream`, `ostream` класс хоёулаа `ios` классаас удамшдаг. Ер нь, стандарт оролт гаралттай холбоотой классын шатлалын бүдүүвчийг Зураг 9.17-д байгаа шигээр дурсэлж болно.

Файл нь стандарт оролт гаралтыг бодвол нарийн удирдлага шаарддаг. Файл шинээр нээж түүнд мэдээлэл бичиж, уншиж болно. Байгаа файлыг зөвхөн уншихаар нээж болно. Файлын тогсголд нэмж бичихээр бас нээх бололцоотой. Файл дотор хойш урагш шилжиж дурын байршилаас нь унших юм уу бичиж болно. Файлын доторхыг бүхэлд нь эсвэл хэсэгчлэн сольж бичиж болно. Файлыг бичгийн болон хоёртын горимд нээж болно. Ингэхлээр, файлтай ажиллах нь олон асуудлыг дагуулдаг, ихээхэн нарийн удирдлага шаарддаг зүйл юм.



Зураг 9.17: Урсгалын классын шатгалын бүдүүрүч

Файл шинээр нээж гарах файлын урсгал үүсгэхийн тулд `ofstream` классыг хэрэглэж файлын нэрийт уг урсгалтай холбож огно. Файлаас унших бол орох файлын урсгалын объект үүсгэн файлын нэртэй холбож өгөхдөө `ifstream` классыг хэрэглэнэ. Файлыг унших, бичих горимоор хэрэглэхийн тулд орох гарах файлын урсгалын объект үүсгэн файлын нэртэй холбож өгөх ба ингэхдээ `fstream` классыг хэрэглэнэ.

### 9.28.2 Шинэ файл нээж мэдээлэл бичих

Мэдээлэл бичихээр шинэ файл нээхийн тулд гарах файлын урсгалын объект үүсгэж файлын нэртэй холбож өгөх ба ингэхдээ `ofstream` классыг хэрэглэнэ. Үүний тулд `fstream.h` файлыг эхлээд програмдаа `#include` командаар зааж өгөөд доор үзүүлсэн шиг командааг хэрэглэнэ.

```
ofstream & fout("test.dat");
```

Энд `fout` бол гарах файлын урсгалын объект, "test.dat" бол үүсгэх файлын нэр юм. Иймд "fout" нь срийн C++ ялгац нэр болно. Объект `fout` нь `cout` объект шиг `streambuf` классаар удирлагдах гаралтын буфер ашиглана. Дээрх команд нь хэрэв ажлын хавтас дотор "test.dat" файл байхгүй бол хоосон файл шинээр, харин байвал түүнийг эхэлж устгаад дахин шинээр үүсгэнэ.

Дээр дурдсанаяа `ofstream` класс нь `ostream` классаас удамших тул эх классын нь `put()`, `write()` зэрэг арга, хэлбэршүүллийн `precision()`, `width()` зэрэг функци. (`<<`) гарах оператор зэргийг `fout` объектод хэрэглэж болно. Гарах оператор нь бүхэл тоо, бодит тоо гэх мэт тоон мэдээллийг тэдгээрт харгалзах тэмдэгт мөр рүү буулгаад дэлгэнэлнэ. Объект `fout` нь програмаас ирэх мэдээллийг буфертээ цуглуулж байгаад буфер дүүрэх эсвэл `close()` функци дуудагдахад буферийн доторхыг OS-ийн тохирох функцийн файлд бичдэг.

Дараах C++ програм нь файл шинээр нээж түүнд мэдээлэл бичих зориулалттай.

```
//Program #9.35
#include <fstream.h>

void main(void)
{
    ofstream fout("test1.dat");
    fout << "Hello world, ";
```

```
    fout << "Hello beautiful world" ;
}
```

Програмыг ажиллуулахад "test1.dat" файл ажлын хавтас дотор үүсдгийг, бас файлын хэмжээ 34 байт болохыг, жишээ нь DOS-ийн dir командаар нягталж type командаар доторхыг нь дэлгэшлэн харж болно. Мөн файлд бичих хоёр морийн нийт тэмдэгтийн тоо 34 болох нь програмын кодоос тодорхой харагдана. Тэгхэлээр, энэ тоо нь файлын хэмжээтэй тохирч байна.

Дээрх програмын сүүлийн командыг нь '\n' тэмдэгт бүхий

```
fout << "Hello beautiful world\n";
```

командаар сольж бичвэл файлд бичигдэх нийт тэмдэгтийн тоо 36 болно. Програмыг доор узүүлээ.

```
//Program #9.36
#include <iostream.h>

void main(void)
{
    ofstream fout("test2.dat");
    fout << "Hello world, ";
    fout << "Hello beautiful world\n";
}
```

Програмыг ажиллуулахад "test2.dat" файл ажлын хавтас дотор үүсдэг бөгөөд файлын хэмжээ нь 36 байт болно. OS нь шинэ морийн тэмдэгтийн оронд CR (Carriage Return буюу морийн эх рүү шилжих), LF (Line Feed буюу мөр хөөх) гэсэн 2 тэмдэгтийг бичих тул файлын хэмжээ нь програм дотор байгаагаас хоёрроор нэмэгднэ. Файлаас унших үед CR, LF хоёр тэмдэгтийн нийлэмж нь нэг шинэ морийн тэмдэгтээр солигддог.

Програм дуусгавар болоход устгагч функцийн fout объект устана. Ингэхдээ устгагч нь файлыг хаана. Хэрэв програм дуусгавар болохоос өмнө файлыг хаах бол close() функцийг илэр fout.close(); гэж хэрэглэн. void close(void) гэсэн загвартай close() нь fstream.h файл дотор тодорхойлогдсон байх fstreambase классын гишүүн функций юм. Түүнчлэн олон түвшинг удамшилаар энэ функцийг ifstream, ofstream, fstream обьектууд шууд хэрэглэх бололцоотой болсон байна.

Файлыг хаах үйл нь нээсэн файл ба файлын урсгалын объект хоорондын холбоог салгадаг. Гэхдээ хаах файл харгалзах урсгалын обьектоос салах ч урсгалын объект хэвээр хадгалагдаж үлдэх тул түүнийг програмд дахин хэрэглэж болно.

Гарах урсгалын хоёр обьектыг өмнөх хоёр програмд

```
ofstream fout("test1.dat");
ofstream fout("test2.dat");
```

гэсэн командуудаар тус тусад нь үүсгээд тэдгээрт "test1.dat" ба "test2.dat" файлуудыг харгалзуулсан холбосон. Үүнийг хоёр алхамт үлдлээр бас гүйцэлдүүлж болно. Ингэхдээ, гарах файлын урсгалын fout обьектыг эхэлж үүсгээд дараа нь нээх файлын нэрийг түүнтэй холбоодо open() гишүүн функцийг хэрэглэн. Энэ open() нь ifstream, ofstream, fstream зэрэг классын гишүүн функций юм. Гэхдээ класс бүрд өөр өөр бичдэстэй байна. Өмнөх Program #9.35-ыг доор узүүлснээр өөрчлон бичье.

```

//Program #9.37
#include <iostream.h>

void main(void)
{
    ofstream fout ;
    fout.open ("test3.dat") ;
    fout << "Hello world, " ;
    fout << "Hello beautiful world" ;
}

```

Програм шинээр "test3.dat" файлыг үүсгээд өгөгдсөн хоёр морийг түүнд бичнэ. Омнө үзсэний дагуу програм дуусгавар болоход файл дацуур хаагдана. Файлыг илээр хаагаад гарах файлын урсгалын объектыг өөр файлтай холбож хэрэглэх жишээг доор үзүүлээ.

```

//Program #9.38
#include <iostream.h>
void main(void)
{
ofstream fout ;
fout.open ("test4.dat") ;
fout << "Hello world, Hello beautiful world" ;
fout.close () ;
fout.open ("test5.dat") ;
fout << "Hello world, Hello beautiful world, " ;
fout << "I love you" ;
}

```

Програмыг ажиллуулбал шинээр "test4.dat", "test5.dat" гэсэн хоёр файлыг үүсгээд

```

Hello world, Hello beautiful world
Hello world, Hello beautiful world, I love you

```

гэсэн мөрүүдийг харгалзуулан бичсэнийг, жишээ нь түре командаар шалгаж болно. Энэ програмын `fout.close()`; командын омни нь ("//") давхар гэдрэг зураас тавих замаар тайлбар болгоод програмыг дахин ажиллуулахад зөвхөн "test4.dat" файлыг үүсгэдэг. Үүхээр тийм эсэхийг нягтлах бол "test4.dat", "test5.dat" файлуудыг эхэлж устгаад програмыг дахин ажиллуулж болно.

### 9.28.3 Файлаас мэдээлэл үнших

Файлаас мэдээлэл үншихын тулд орох файлын урсгалын объект үүсгээд түүнд иэх файлын исрийг холбож өгөх ба ингэхдээ `ifstream` классыг хэрэглэнэ. Үүнийг нэг мэр командаар

```
ifstream fin("test1.dat")
```

гэж бичнэ. Энд `fin` нь програм болон "test1.dat" файлыг хооронд нь холбох үүрэгтэй орох урсгалыг төлоөлөх объект юм. Энэ объект `sin` объектынх шиг `streambuf` классаар удирдагдах оролтын буфер ашиглана. Ер нь, "fin" бол хэрэглэгчийн тодорхойлох ерийн ялгац нэр юм.

Дээрх команд нь "test1.dat" файл ажлын хавтас дотор байвал түүнийг иэнэ. Ажлын хавтас дотор байхгүй файлын хувьд нээх үйл бүтэлгүйтэж `fin` нь тэг утгатай болно. Файл иэх үйл хэвийн явагдвал `fin` нь тэг биш утгатай байна.

`fin` нь `istream` классаас удамшиж `ifstream` классын объект тул эх классынхаа доорх аргууд болон (`>>`) оруултын операторыг хэрэглэж чадна.

```
istream & get(char &) ;
int get (oid) ;
istream & get(char *, int, char='\n') ;
istream & getline(char *, int, char='\n') ;
```

Оруулгын оператор нь тэмдэгт мөр хэлбэрт байгаа бүхэл тоо, бодит тоо гэх мэтийг байрших хувьсагчийн нь төрөлд нийцүүлэн тоон хэлбэрт буулгадаг байхаар дахин тодорхойлогдсон байдаг. Иймээс доорх шиг командуудыг бичиж болно.

```
char ch ;
fin >> ch ; //Тэмдэгт унших ('N')
char buff[80] ;
fin >> buf ; // "Hello" үгийг унших
fin.getline(buff,80); // "world, Hello beautiful world" мөрийг унших
```

Дээр үүсгэх "test1.dat" файлыг уншиж дэлгэшлэх програмыг доор үзүүлээ.

```
//Program #9.39
#include <iostream.h>

void main(void)
{
    const int MAX = 80 ;
    char buff[MAX] ;
    ifstream fin("test1.dat") ;
    while(!fin.eof ()) 
    {
        fin.getline (buff, MAX) ;
        cout << buff ;
    }
}
```



Hello world, Hello beautiful world

Дээрх програмыг "test2.dat" файлын хувьд бас хэрэглэж болох бөгөөд '\n' нь CR, LF хоёр тэмдэгт болж бичигддэг тухай омно үзсэн билээ. Үүгээр "test1.dat", "test2.dat" хоёр файл хоорондоо ялгагддаг байна.

Дээрх програмыг бага зэрэг өөрчилж дараах байдлаар бичиж болно.

```
//Program #9.40
#include <iostream.h>

void main(void)
{
    const int MAX = 80 ;
    char buff[MAX] ;
    ifstream fin("test2.dat") ;
    while(fin)
    {
        fin.getline (buff, MAX) ;
        cout << buff ;
    }
}
```



Hello world, Hello beautiful world

Програмыг "test1.dat" файлын хувьд хэрэглэж болдогтүй. Энэ програм нь нэг удаадаа нэг мөр унших ба файлын төгсгөлд хүрч очиход fin тэг болж улмаар while давталт дуусгавар болно.

Program #9.39-9.40 гэсэн хоёр програмд орох файлын урсгалын fin объект үүсгэж файлын нэргэй холбоодо нэг командын мөр хэрэглэснийг хоёр командын мөрөөр оруулан

```
ifstream fin  
fin.open("test1.dat") ;  
  
гэж хэрэглэх боломжтой.
```

#### 9.28.4 Тэмдэгтийн оролт гаралт

Тэмдэгт морийг файлд бичиж уншиж болохоос гална дан тэмдэгтийг мөн файлд бичиж уншиж болно. Тэмдэгт бичих бол ostream классын ostream::put(char) аргыг, унших бол istream классын istream::get(char &) аргыг тус тус хэрэглэнэ.

Файлд тэмдэгт мэдээлэл бичих програмыг дараах байдалар бичнэ.

```
//Program #9.41  
#include <iostream.h>  
#include <string.h>  
  
void main(void)  
{  
    char message[] = "Every dose of radiation is"  
                  " an overdose of radiation";  
  
    ofstream fout("test6.dat");  
    for(int i=0; i<strlen(message); i++)  
        fout.put(message[i]);  
}
```

Програм "test6.dat" файлыг үүсгээд зааж өгсөн морийг түүнд бичнэ. Үүнийг шалгахдаа жишээ нь type командаас гадиз дараах програмыг хэрэглэж болно.

```
//Program #9.42  
#include <iostream.h>  
void main(void)  
{  
    char ch ;  
    ifstream fin ;  
    fin.open ("test6.dat");  
    while(fin)  
    {  
        fin.get (ch);  
        cout << ch ;  
    }  
}
```



Every dose of radiation is an overdose of radiation

Дээрх програмын while давталтыг дараах энгийн командаар оруулан

```
while(fin.get(ch))  
    cout << ch ;
```

гэж бичих боломжтой.

### 9.28.5 Бичгийн файл

Омно үзсэн `put()`, `get()` нь ASCII файлын зориулалттай функцийд юм. Эдгээрээс `put()` нь мөрийн төгсгөлийн '\n' тэмдэгтийн оронд харгалzan мөрийн эх рүү болон шинэ мөрөнд шилжүүлэх үүрэгтэй 0x0D, 0x0A гэсэн арванзурагатын хоёр кодыг файлд бичих тул DOS орчинд `type` командаар эсвэл `notepad.exe` шиг програмаар хэвийн ушигдах файл гаргаж авах боломжтой. Харин хойно үзэх `write()`, `read()` нь хоёртын файлын зориулалтын функцийд юм. Эдгээр нь объектын мэдээллийг дараалсан байтын цуваа хэлбэрээр файлд бичих унших учир ийм файл DOS орчинд хэвийн ушигдахгүй. Уг функцийд мөрийн төгсгөлийн тэмдэгтийг файлд бичиж уншдагтуйгээрээ `put()`, `get()` функцийдээс ялгаатай юм.

Омнох бүлгүүдэл хэрэглэсэн бүх жишээ програмаар үүсэх файлууд цөм бичгийн буюу ASCII файлууд юм. Бичгийн файлын хувьд 8 битийн хэв бүр түүнд хадсан тодорхой угтатай байна. Файлын байт бүр ASCII тэмдэгт юм. Дараах програмыг авч үзье.

```
//Program #9.43
#include <iostream.h>

void main(void)
{
    char name[] = "John Milton";
    int age = 56;
    double height = 170.6;

    ofstream fout;
    fout.open ("test7.dat");

    fout << name;
    fout << "";
    fout << age;
    fout << "";
    fout << height;
}
```

Програм "test7.dat" файлыг үүсгээд өгөгдсон өгөгдлүүдийг түүнд бичиэ. Үүнийг `type` командаар шалгаж болохоос гадна файлын хэмжээ 20 байт болохыг `dir` командаар мэдэж болно.

Файлд бичилт хийхдээ (<<) гаргах оператор хэрэглэвэл мэдээлэл нь ASCII хэлбэрээр бичигдэнэ. Иймд бүхэл тоо 56 нь '5', '6' гэсэн хоёр тэмдэгт хэлбэрээр, бодит тоо 170.6 нь '1', '7', '0', '.', '6' гэсэн 5 өөр тэмдэгт хэлбэрээр файлд бичигдэнэ.

Файл доторхыг доорх програмд үзүүлсэн шигээр тэмдэгт тэмдэгтээр нь уншин дэлгэцэлж болох боловч тэдгээр дээр арифметик үйлдэл хийж болдогтүй. Үйлдэл хийдэг байхын тул мэдээллийг ASCII тэмдэгт хэлбэрээр бус харин тэдгээрт харгалзах бүхэл тоон, бодит тоон хэлбэрээр нь бичих унших програм хэрэгтэй. Үүнийг доор харууллаа.

```
//Program #9.44
#include <iostream.h>
void main(void)
{
    char name[20];
    int age;
    double height;
    ifstream fin;
```

```

fin.open ("test7.dat") ;
fin.getline (name, 12) ;
fin >> age ;
fin >> height ;
cout << "name    = " << name << "\n" ;
cout << "age     = " << age << " years\n" ;
cout << "height  = " << height << " cm\n" ;
cout << "2*age   = " << 2*age << "\n" ;
cout << "2*height= " << 2*height << "\n" ;
}

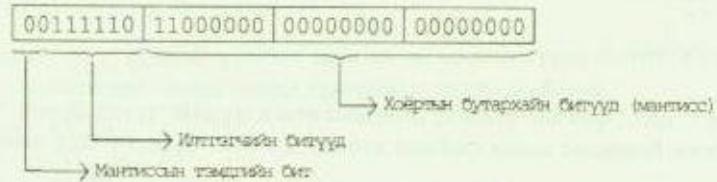
name      = John Milton
age       = 56 years
height    = 170.6 cm
2*age     = 112
2*height= 341.2

```

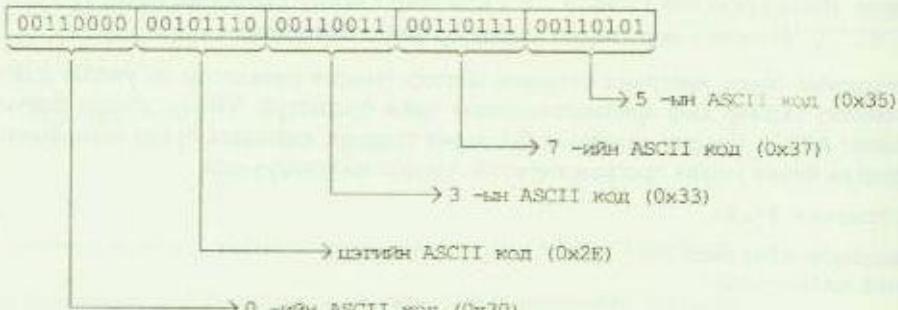
Бичгийн файлаас мэдээлэл авах нь стандарт оролтын тохиромжоос мэдээлэл авахтай адилхан. Оруулах оператор нь тэмдэгт мөрийн хэлбэрт байгаа тоон мэдээллийг харгалзах тоон хэлбэр рүү хувиргана.

#### 9.28.6 Хоёртын файл

Бичгийн файлд мэдээлэл бичихэд ASCII тэмдэгт хэлбэрээр бичигддэг тухай өмнө үзсэн билээ. Иймд, жишээ нь бичгийн файлд  $-2.324216e+07$  гэсэн тоог бичих гэвэл 13 тэмдэг болно. Мэдээллийг бас хоёртын хэлбэрээр файлд бичиж болдог. Ямар ч тооны компьютерийн дотоод дүрслэл нь хоёртын файлд хадгалагдана. Жишээ болгож 0.375 гэсэн тоог авч үзье. Энэ тооны хоёртын дүрслэлийг Зураг 9.18-д, ASCII дүрслэлийг Зураг 9.19-д тус тус үзүүлээ.



Зураг 9.18: 0.375 тооны хоёртын дүрслэл



Зураг 9.19: 0.375 тооны тэмдэгтэн (ASCII) дүрслэл

Тоог бичгийн файлд бичвэл Зураг 9.19-д, хоёртын файлд бичвэл Зураг 9.18-д тус тус үзүүлсэн шингэр бичигдэнэ.

Бичгийн болон хоёртын файл тус бүр өөрийн гэсэн сайн талтай. Бичгийн файлыг уншихад энгийн. Түүнийг потерад шиг энгийн засуур програмаар уншиж өөрчилж болно. Түүнчлэн ийм файлыг нэг системээс ногое рүү хялбар шилжүүлж болдог. Хоёртын файлын хувьд төрөл хувиргалт шаардлагагүй тул мэдээллийг хурдтай уншиж болно. Хоёртын хэлбэр нь ихэвчлэн бага зай шаардана. Жишээ нь, 0.375 гэсэн тоог бичгийн файлд бичихэд 5 байт, хоёртын файлд бичихэд 4 байт шаардагдана.

Мэдээллийг хоёртын файлд бичихэд ostream классын write() аргыг хэрэглэнэ.

```
ostream & write (const signed char *, int) ;
ostream & write (const unsigned char *, int) ;
```

Хоёртын файлаас уншихад istream классын istream & read(char \*, int); аргыг хэрэглэнэ.

Хоёртын файлыг шинээр үүсгэх жишээ програмыг доор үзүүллээ.

```
//Program #9.45
#include <fstream.h>

class person {
protected:
    char name[20] ;
    int age ;
    double height ;
public:
    void get_data(void) ;
};

void person::get_data (void)
{
    cout << "\nEnter name : " ;
    cin.get (name,20);
    cout << "Enter age : " ;
    cin  >> age ;
    cout << "Enter height: " ;
    cin  >> height;
}

void main(void)
{
    person p ;
    p.get_data () ;

    ofstream fout ;
    fout.open ("test8.dat") ;
    fout.write ((char *) &p, sizeof(p)) ;
}
```

 Enter name : John Milton-
Enter age : 65-
Enter height: 170.6.

Дээрх програм нь "test8.dat" файлыг үүсгээд person классын р объектын утгыг түүнд бичнэ. Уг объектын хэмжээ 30 (20+2+8) байт болно. Үнэхэр файлын хэмжээ 30 байт байгаа эсэхийг, жишээ нь dir командаар шалгаж болно.

Доорх програм нь омни үүсгэсэн "test8.dat" файлыг унших зориулалттай.

```

//Program #9.46
#include <iostream.h>

class person
{
protected:
    char name[20];
    int age;
    double height;
public:
    void show_data(void);
};

void person::show_data (void)
{
    cout << "\nname = " << name;
    cout << "\nage = " << age << " years";
    cout << "\nheight = " << height << " cm";
}

void main(void)
{
    person p;
    ifstream fin;
    fin.open ("test8.dat");
    fin.read ((char *) &p, sizeof(p));
    p.show_data ();
}

 name = John Milton
age = 65 years
height = 170.6 cm

```

Бичгийн файл дахь цуваа байт нь ASCII тэмдэгтийн код бол хоёртын файл дахь цуваа байт нь янз бүрийн 8 битэн код юм. DOS тэдгээр байтад ямар нэгэн тодорхой утга хадлагтүй. Хоёртын файлтай ажиллах зориулалтын програмаар нь тэдгээр байтыг тайлж уншина.

#### 9.28.7 Файлын горим

Файлын горим нь файлыг хэрхэн хэрэглэхийг заах үүрэгтэй. Файлыг зөвхөн уншихаар, зөвхөн бичихээр, унших бичихээр эсвэл файлын төгсгөлд измж бичихээр гэх мэтээс нээх хэрэглэж болно.

Файлтай холбоотой аль ч үйлийн үед файлын ургалын объект үүсгээд файлын нэртэй холбож өгөх командыг хэрэглэх ба ингэхдээ уг командын хоёрдахь аргументээр нь файлыг ямар горимоор нээхээ доор үзүүлсэн шигээр хоёр боломж тус бүрээс аль нэгийг сонгож зааж огч болно.

##### A. Зөвхөн уншихаар нээх үед

1. ifstream fin("test.dat", mode);
2. ifstream fin;
- fin.open("test.dat", mode);

##### B. Зөвхөн бичихээр нээх үед

1. ofstream fout("test.dat", mode);
2. ofstream fout;
- fout.open("test.dat", mode);

### C. Унших бичихээр нээх үсд

```
1. fstream finout("test.dat", mode) ;
2. fstream finout ;
   finout.open("test.dat", mode) ;
```

Өмнөх бүлгүүдэд файлыг зөвхөн уншихаар эсвэл бичихээр нээх талаар авч үзсэн бөгөөд бүх жишээ програмд зөвхөн нэг аргумент, тухайлбал файлын нэрийг хэрэглэсэн билээ. Энэ нь байгуулагч, open() функцийд шууд авах утга бүхий параметртэй байдагтай холбоотой. Тухайлбал, ifstream классын байгуулагч функцийн тодорхойлолт ба ifstream класс доторх open() функцийн загвар нь доор үзүүлсэн шиг бичвэртэй байдаг.

```
ifstream(const char*,int=ios::in,int=filebuf::openprot);
void open(const char*,int=ios::in,int=filebuf::openprot);
```

Дээрхтэй адил ofstream классын байгуулагч функцийн тодорхойлолт ба ofstream классын open() функцийн загвар нь доор шиг бичвэртэй.

```
ofstream(const char*,int=ios::out,int=filebuf::openprot);
void open(const char*,int=ios::out,int=filebuf::openprot);
```

Мөн fstream классын байгуулагч функцийн тодорхойлолт ба fstream классын open() функцийн загвар нь доор үзүүлсэн шиг бичвэртэй.

```
fstream(const char*,int,int=filebuf::openprot);
void open(const char*,int,int=filebuf::openprot);
```

fstream классын хувьд байгуулагч функци, open() функцийн аль алины хоёрдугаар параметр нь шууд авах утгагүй байна. Иймээс файлыг унших бичихээр нээх бол файлын горим болох хоёрдахь параметрийн утгыг зааж өгөх хэрэгтэй болдог.

Файлын горимыг заах бүхэл тоон тогтмол fstream.h файлд байдал. Янз бурийн горимын тогтмол байх ба тэдгээрийн зориулалтыг Хүснэгт 9.1-д жагсааж үзүүллээ.

Горимын нийлэмж хэрэглэх бол C++ хэлний бит OR буюу ( | ) операторыг хэрэглэнэ. Ингэснээр янз бурийн горим үүсгэх бололцоотой, жишээ нь

```
ios::in | ios::app
ios::in | ios::out |ios::app
```

Хүснэгт 9.1: open() функцийн авах аргумент

Аргументийн утга	Зориулалт
ios::in	Унших горим (ifstream классын шууд горим)
ios::out	Бичих горим (ofstream классын шууд горим)
ios::app	Файлын төгсгэлд залгаж бичих горим
ios::ate	Нээх файлын төгсгэлд очих
ios::trunc	Байгаа файлыг эхэлж тайрах
ios::binary	Хоёртын файл нээх
ios::nocreate	Нээх файл байхгүй бол алдааны мэдээлэл өгөх
ios::noreplace	Нээх файл байгаа бол алдааны мэдээлэл өгөх

#### 9.28.8 Файлыг унших бичихээр нээх

Өмнөх бүлгүүдэд файлыг зөвхөн уншихаар эсхүл зөвхөн бичихээр нээх жишээ програмыг авч үзсэн билээ. Тэгвэл файлыг бас унших бичихээр нээх боломжтой байдал. Ингэхдээ орох гарах файлын урсгалын объект үүсгээд түүнд файлын нэрийг холбоходо fstream классыг хэрэглэнэ. Үүний дараах хоёр арга замын нэээр хийж болно.

```
1. fstream finout("test.dat", ios::in | ios::out) ;
2. fstream finout ;
   finout.open("test.dat", ios::in | ios::out) ;
```

Дээрх хоёр командын аль аль нь "test.dat" нэртэй файл ажлын хавтас дотор байхгүй бол ийм нэр бүхий хоосон файлыг шинээр үүсгэнэ. Харин ажлын хавтас дотор байгаа файлыг нээвэл түүний доторхыг устгахгүй хэвээр үлдээнэ. Файлыг унших бичих горимоор<sup>56</sup> нээнэ. Зохих арга хэмжээг авахгүй бол файл руу шинээр бичих мэдээлэл нь омни байсан мэдээллийг дарж бичиз.

Дээрх хоёр боломжийн аль нэгийг хэрэглэн файлыг унших бичихээр нээхэд хоёр аргумент шаардлагатай. Эхийх нь нээх файлын нэр байх ба түүнийг програм дотроос заахдаа давхар хашилгад бичиж өгнө. Удаах нь файлын горим байх ба түүнийг орх гарах горимд

```
ios::in | ios::out  
гэж зааж өгнө.
```

fstream нь istream классаас удамших ifstream, бас ostream классаас удамших ofstream гэсэн хоёр классаас удамшдаг класс юм. Иймд fstream классын объект болох finout нь классынхаа эх классуудын бүх арга, дахин тодорхойлсон операторыг хэрэглэж болно. fstream объект нь оролт гаралтын хоёр буфертэй байна.

Унших бичих хоёр горимоор нэг файлыг нээх програмыг доор үзүүллээ.

```
//Program #9.47
#include <fstream.h>

void main(void)
{
    fstream finout ;
    finout.open ("test9.dat", ios::in|ios::out) ;
    finout << "Hello world, Hello beautiful world" ;
}
```

Ажлын хавтас дотор "test9.dat" файл байвал түүнийг эхэлж устгаад програмыг ажиллуулахад уг файл дахин шинээр үүсчхийг, файлын хэмжээ 34 байт болохыг, жишээ нь dir командаар шалгаж, бас файлыг, жишээ нь type командаар уншиж "Hello world, Hello beautiful world" мөр файлд бичигдсэнийг мэдэж болно.

"test9.dat" файлыг унших бичихээр нээгээд өгөгдсөн мэдээллийг бичих програмыг дараах байдлаар бичин.

```
//Program #9.48
#include <fstream.h>

void main(void)
{
    fstream finout ;
    finout.open ("test9.dat", ios::in|ios::out) ;
    finout << "Hello beautiful world" ;
}
```

Програмыг ажиллуулсны дараагаар "test9.dat" файлыг уншиж

"Hello beautiful worldautiful world"

<sup>56</sup> Ер нь, файлыг унших, бичих горимыг товчоор файлын горим гэж норалцго.

мер үнэхэр бичигдсэн эсэхийг шалгаж болно. Програм ажиллахдаа "test9.dat" файлыг эхэлж устгахгүйгээр унших бичих горимоор нээнэ. Файл руу бичих мэдээлэл нь файлын эхнээс байгаа мэдээллийг дарж бичигдэх тул дээрх үр дүн гардаг байна. Үүнийг Зураг 9.20-д үзүүлсэн байдлаар дүрсэлж болно.

```
test9.dat
• H e l l o   w o r l d ,   H e l l o   b e a u t i f u l   w o r l d
• H e l l o   b e a u t i f u l   w o r l d —→ new string (минэ тэмдэгт мэр)
```

Зураг 9.20: Program #9.47-ийн үр дүнгийн дүрслэл

### 9.28.9 Файлын заагуур

Файл нь дараалан бичигдсэн олон байт гэж төсөөлвэл тэдгээр байт нь 0, 1, 2, 3, ..., n-1 гэх мэтээр дугаарлагдана. Файлыг бичих горимоор нээх тохиолдолд эхний бичилт нь 0 дүгээр байтаас эхлэлдэг. Файлд бичих нэг тэмдэгт нь файлын 0 дүгээр байрлалаас бичигдэх тул дараагийн бичилт 1 дүгээр байрлалаас эхлэх болно. Хэрэв "world" үгийг хоёрдахь бичилтээр файлд бичвэл тэрээр 1, 2, 3, 4 ба 5 дугаар байрлалд бичигдэж зэлжит бичилт 6 дугаар байрлалаас эхлэх болно.

Гарах файлын заагуур<sup>57</sup> файлын дагууд шилжинэ гэвэл анх файлыг нээхэд заагуур нь байтын 0 дүгээр байрлалыг заана. Файлд нэг тэмдэгт бичсэний дараагаар заагуур 1 дүгээр байрлалыг заана. Дараа нь "world" гэсэн үгийг бичвэл уг уг байтын 1-ээс 5 дугаар байрлалд бичигдэж файлын заагуур нь 6 дугаар байрлал руу шилжинэ. Тэгэхлээр, гарах файлын заагуур нь эзлжит бичилт эхлэх байтын байрлалыг заана.

Үүнтэй төстэйгээр, файлыг зөвхөн уншихаар нээхэд унших үйл 0 дүгээр байтаас эхэлнэ. Иймд дараах

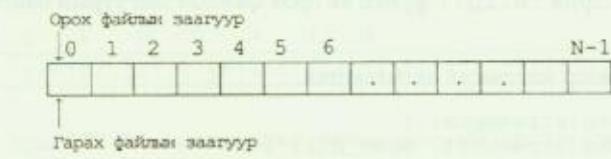
```
char ch ;
fin.get(ch) ;
```

командыг хэрэглэвэл байтын 0 дүгээр байрлалд буй тэмдэгтийг ch рүү уншина. Хоёрдахь уншилтаар, тухайлбал fin.get(ch); команд дахин хийгдэх үед 1 дүгээр байрлал дахь тэмдэгтийг ch рүү унших тул файлын заагуурын байрлал 2 болно. Дараагийн уншилтыг

```
char buffer[10] ;
fin.get(buffer, 10) ;
```

командаар хийвэл 2 дугаар байрлалаас эхлээд 9 тэмдэгт унших тул файлын заагуурын заах утга 11 болно. Ингэхлээр, орох файлын заагуур нь эзлжит уншилт эхлэх байтын байрлалыг заадаг байна.

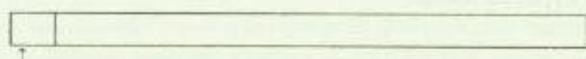
Файлыг унших бичихээр нээхэд орох файлын заагуур, гарах файлын заагуур гэсэн хоёр заагуур байна гэж үзвэл үүнийг Зураг 9.21-д үзүүлсэн шигээр дүрсэлж болно.



Зураг 9.21: Орох гарах файлтай холбогдох заагууруудын дүрслэл

<sup>57</sup> Файлын заагуур (file pointer) нь эзлжит бичилт, уншилт хаанаас эхлохийг заана.

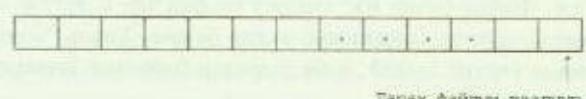
Файлтыг анх нээхэд хоёр заагуур хоёулаа 0 дүгээр байрлалыг заана. Эдгээр заагуур нь бие биесээс хамааралгүйгээр шилжинэ. Файлын заагуур нь файлыг нэх яз бүрийн горимтой хэрхэн холбогдохыг Зураг 9.22~26-д үзүүллээ.



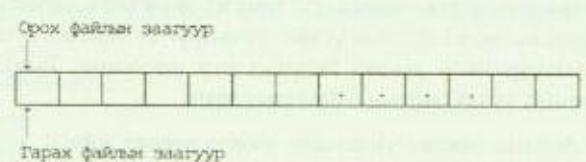
Зураг 9.22: `ofstream fout("test.dat")`, холбогдох файлын заагуур



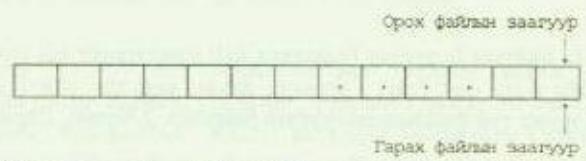
Зураг 9.23: `ifstream fin("test.dat")`, холбогдох файлын заагуур



Зураг 9.24: `ofstream fout("test.dat", ios::app)`, холбогдох файлын заагуур



Зураг 9.25: `fstream finout("test.dat", ios::in | ios::out)`, холбогдох файлын заагуур



Зураг 9.26: `fstream finout("test.dat", ios::ate)`, холбогдох файлын заагуур

### 9.28.10 seekg(), tellg() аргууд

`seekg()`, `tellg()` бол `istream` классын функцийд юм. Гэхдээ `fstream` классын объект тэдгээрийг удамшилаар дамжуулан хэрэглэнэ. `seekg()` функци орох файлын заагуурыг аргумент хэлбэрээр зааж өгөх байрлал руу шилжүүлэх тул ээлжит уншилт уг шинэ байрлалаас эхэлнэ. Харин `tellg()` функци нь орох файлын заагуурын байгаа байрлалыг олж буцаана.

`seekg()` функци нь доор жагсаасан эх загвартай.

```
istream & seekg(streampos) ;
istream & seekg(streamoff, seek_dir) ;
```

Дээрх `streampos`, `streamoff` нь `iostream.h` файлд доор үзүүлсэн шигээр тодорхойлсон өгөгдлийн зохиомол төрлүүд юм.

```
typedef long streampos ;
typedef long streamoff ;
```

Харин `seek_dir` бол `ios` класс дотор тодорхойлогдсон нэрлэсэн тогтмолон төрөл юм. Энэ хувьсагч дараах гурван утгын аль нэгийг авна.

```
ios::beg = 0
ios::cur = 1
ios::end = 2
```

Дээрх `seekg()` функцийн эхний тухайлбал, ганц параметртэй хувилбар нь файлын заагуурыг файлын эхнээс `streampos` байт зайд шилжүүлнэ. Тэгэхлээр, файлын заагуур, жишээ нь `fin.seekg(100);` командаар файлын эхнээс 100 дугаарх байт руу, `fin.seekg(0);` командаар файлын эх рүү тус тус шилжик очно.

Харин `seekg()` функцийн хоёрдахь, тухайлбал хоёр параметртэй хувилбар нь файлын заагуурыг гурван өөр байрлалаас, тухайлбал файлын эхнээс, эцсээс, байгаа газраас нь харьцангуйгаар шилжүүлнэ. Иймээс `seek_dir` нь `ios::beg`, `ios::cur`, `ios::end` гэсэн гурван тогтмол утгын аль нэгийг авч болно. Жишээ нь, `fin.seekg(30,ios::beg)` команд файлын заагуурыг файлын эхнээс 30 дугаарх байт руу, `fin.seekg(0,ios::end)` команд файлын эцсэг рүү, `fin.seekg(-1,ios::cur)` команд байгаа газраас нь 1 байтаар зүүн тийш тус тус шилжүүлж байршуулна.

`tellg()` функц нэг эх загвартай нь `streampos tellg(void)` юм. Энэ нь файлын заагуурын байгаа байрлалыг олж буцааж мэдээлэх үүрэгтэй функц юм.

Зөвхөн уншихаар нээх файлын хувьд файлын заагуурыг тодорхой байрлал руу шилжүүлж уг байрлалаас эхэлж унших үйлийг хэрхэн програмчлахыг дараах програм үзүүлнэ.

```
//Program #9.49
#include <fstream.h>

void main(void)
{
    char buffer [10] ;
    ifstream fin("test1.dat") ;
    fin.seekg (19) ;
    cout << "\nFile position after seekg(19) is: " << fin.tellg () ;
    fin >> buffer ;
    cout << "\nbuffer = " << buffer ;
}

File position after seekg(19) is: 19
buffer = beautiful
```

- Программаар үүсэх "test1.dat" файлд бичигдэх мэдээллийг Зураг 9.27-д үзүүлсэн шигээр дүрсэлж болно.

0	2	4	6	8	10	12	14	16	18																						
B	e	1	1	0		w	o	r	l	d	,	B	e	1	1	0	b	e	a	u	t	i	f	u	l		w	o	r	l	d

Зураг 9.27: Файлын заагуурыг удирдах функцийг хэрэглэх эсвэл

Дээрх програмын `fin.seekg(19);` команда нь файлын заагуурыг файлын эхнээс 19 дүгээрх байт руу шилжүүлнэ. Иймд `fin>>buffer;` командаар хийгдэх уншилт энэ шинэ байрлалаас эхэлж ээлжит цагаан зайнгаар дуусгавар болно.

### 9.28.11 seekp(), tellp() аргуул

seekp(), tellp() нь ostream класын функциүүд юм. Гэхдээ эдгээрийг ofstream класын объект удаашлаар дамжуулан хэрэглэж болно. seekp() нь гарах файлын заагуурыг файл доторх огогдсон байрлал руу шилжүүлэх тул дараагийн эзлжит бичилт уг шинэ байрлалаас эхэнэ. tellp() нь гарах файлын заагуурын байгаа байрлалыг олж буцаана.

seekp() функц нь доор үзүүлсэн шиг хоёр эх загвартай.

```
ostream & seekp(streampos) ;
ostream & seekp(streamoff, seek_dir);
```

Авах аргументийн нь төрөл болох streampos, streamoff ба seek\_dir зэрэг иэрэсэн тогтмолын талаар омнөх бүлгүүдэд үзсэн билээ. Нэг параметр бүхий seekp() функц нь гарах файлын заагуурыг файлын эхнээс streampos байт зайд шилжүүлнэ. Жишээ нь, fout гарах файлын заагуур нь fout.seekp(100); командаар файлын эхнээс 100 дугаарх байт руу, fout.seekp(0); командаар файлын эх рүү шилжих жишээтэй.

Хоёр параметрт seekp() функц гарах файлын заагуурыг файлын эхнээс, эцсээс эсвэл байгаа байрлалаас нь харьцангуйгаар streamoff байт зайд шилжүүлнэ. Иймээс seek\_dir нь

```
ios::beg
ios::end
ios::cur
```

гэсэн турван тогтмол утгын аль нэгийг нь авах ёстой. Жишээ нь, fout гарах файлын заагуур нь файлын эхнээс 30 дугаарх байт руу fout.seekp(30,ios::beg); командаар, файлын эцэс рүү fout.seekp(0,ios::end); командаар, fout.seekp(-1,ios::cur); командаар байгаа байрлалаас зуун тийш 1 байтвар тус тус шилжих жишээтэй байдал.

Зөвхөн бичих горимоор нээх файлын хувьд файлын заагуурыг тодорхой байрлал руу шилжүүлж уг байрлалаас бичих үйлийг хэрхэн програмчлахыг дараах програм харуулна. Файлын заагуурыг шилжүүлсний дараагаар бичих шинэ мэдээлэл нь файлд байгаа хуучин мэдээллийг дарж бичигдэн.

```
//Program #9.50
#include <fstream.h>

void main(void)
{
    ofstream fout("test10.dat");
    fout << "Hello world, Hello beautiful world";
    fout.seekp(6);
    cout << "\nFile position after seekp(6) is: " << fout.tellp();
    fout << "WORLD";
    cout << "\nFile position after fout<< \"WORLD\" is: " << fout.tellp();
    fout.seekp(29);
    cout << "\nFile position after seekp(29) is: " << fout.tellp();
}

File position after seekp(6) is: 6
File position after fout << "WORLD" is: 11
File position after seekp(29) is: 29
```

Дээрх програм эхлээд "test10.dat" файлыг үүсгээд түүнд "Hello world, Hello beautiful world" өгүүлбэрийг бичнэ. Дараа нь fout.seekp(6); командаар гарах файлын заагуурыг 6 дугаарх байт руу шилжүүлнэ. Энэ байрлалд файлын заагуур очсоныг эхний fout.tellp() команд мэдээлнэ. Эхний fout<<"WORLD"; команд нь 6 дугаар байтаас эхэлж бичигдсэн "world" үгийг "WORLD" үгээр дарж бичнэ. Ингэснээр файлын заагуур 11 дүгээрх байт руу шилжиж очно. Үүнийг хоёрдахь fout.tellp() команд мэдээлнэ. Дараа нь fout.seekp(29); командаар файлын заагуур нь 29 дүгээр байт дээр очно. Хоёрдахь fout<<"WORLD"; команд нь 29 дүгээр байтаас эхэлж бичигдсэн "world" үгийг бас "WORLD" үгээр дарж бичсэнийг Зураг 9.28-д дурслэн үзүүллээ.

0	1	2	3	4	5	6	7	8	9	11	13	15	17	19	21	23	25	27	29	31	33					
H	e	l	l	o	w	o	r	l	d	,	H	e	l	l	o	b	e	a	u	t	i	f	u	l	o	d
W	O	R	L	D																W	O	R	L	D		

Зураг 9.28: Program #9.49-ийн үр дүн

Ингэснээр, "test10.dat" файлд "Hello WORLD, Hello beautiful WORLD" гэсэн мөр бичигддэг.

### 9.28.12 Урсгал шалгах

Файлтай ажиллах явцад гарч болох янз бүрийн алдааны нохцел байдлын талаар өмнөх бүлгүүдэд бага анхаарсан билээ. Шинээр үүсгэх файл ямар ч бэрхшээлгүйгээр үүсч, уншихаар нээх файл заавал байгаа гэж үзэж байсан, үнэхээр тийм байсан тул үнэндээ файлтай холбогдох ямар нэгэн асуудал огт гарч байгаагүй. Гэвч програмчлалын бодит орчинд байдал өөр байдаг. Ажлын хавтас дотор байгаа гэсэн файл бодит байдал дээр байхгүй, дискийн сул зай шинээр нээх файлын хувьд хангалттай бус байж болно. Иймээс янз бүрийн алдаа гарч болох нохцел байдлыг шалгаж таарч тохирох зөв арга хэмжээг авах нь зайлшгүй байх ёстой зүйл юм.

Байхгүй файлыг уншихаар нээх гэж оролдоход алдаа гарах бөгөөд энэ алдааны талаар C++ компайлер ямар нэгэн байдааар хэрэглэгчид мэдээлэхгүй учир програм урьдаас үл мэдэх ажиллагаа үзүүлэн дуусгавар болно. Иймд алдаа гарч болох нохцел байдлыг урьдчилан шалгаж тохирох арга хэмжээг авах нь програм хөгжүүлэгчийн үүрэг юм.

ios класс нь int state; гишүүн өгөгдлөөс гадна гарч болох төлов байдлын талаар мэдээлэх дараах аргуутдтай.

```
int rdstate(void);           //урсгалын төлевийг унших
int eof(void);              //файлын төгсгөлд тэг биш
int fail(void);             //үйлдэл бүтэлгүйдвэл тэг биш
int bad(void);              //алдаа илэрвэл тэг биш
int good(void);             //аль ч төлевийн бит өөрчлөгдөөгүй бол тэг биш
void clear(int=0);          //урсгалын төлевийг гарааны байдалд оруулах
```

Файлын урсгалын объект нь дээрх гишүүн өгөгдэл болон аргуудыг удамшилаар дамжуулан авч хэрэглэдэг. Төловийн гишүүн өгөгдлийн бага байтыг төловийн байт гэх ба түүний битүүд тандах шаардлагатай янз бүрийн алдааны байр байдлыг заана. Үүнийг Зураг 9.29-д харууллаа.

Төловийн байтын бит бүр нэртэй байна.

Бит 0: eof бит/файлын төгсгөл бит  
Бит 1: fail bit/бүтэлгүйдэл бит

Бит 2: bad бит/хэвийн биш бит  
 Бит 3: hard fail bit/ноцтой бүтэлгүйдэл бит



Зураг 9.29: Төлөвийн байт, түүний битүүдийн зориулалт

Бүх зүйл хэвийн байвал төлөвийн байтын бүх битийн утга 0 байна. Оролт гаралтын үйлдлээр файлын заагуур файлын тэгсгэлд хүрвэл eof бит 1 болно. Унших бичих үйл бүтэлгүйтвэл fail бит 1 болно. Файлын эцэст аль хэдий нь очсон байхад унших мэт алдаатай үйлдэл хийвэл bad бит 1 гэсэн утга авна. Техникийн чанартай алдаа гарвал hard fail бит 1 болно.

Төлөвийн байдлыг мэдээлэх аргуудаас буцах утга нь янз бүрийн төлов байдлыг харуулна.

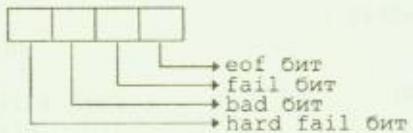
```
rdstate() төлөвийн байтыг буцаах
eof()      eof бит = 1 бол тэг биш утга буцаах
fail()     fail бит = 1 бол тэг биш утга буцаах
bad()      bad бит = 1 бол тэг биш утга буцаах
good()    омнөх бүх 4 бит тэг биш утга буцаах
```

Дараах C++ програм дээрх бүх шинжийг хэвлэж үзүүлнэ.

```
//Program #9.51
#include <fstream.h>
void main(void)
{
    ifstream fin("test1.dat");
    if(!fin)
        cout << "\nCan't open the file test1.dat";
    else
        cout << "\nOpened the file successfully";
    cout << "\nfin =\t" << fin;
    cout << "\nstate =\t" << fin.rdstate();
    cout << "\neof() =\t" << fin.eof();
    cout << "\nfail()=\t" << fin.fail();
    cout << "\nbad() =\t" << fin.bad();
    cout << "\ngood()=\t" << fin.good();
}
Opened the file successfully
fin = 0x0012FP40
state = 0
eof() = 0
fail()= 0
bad() = 0
good()= 1
```

Төлөвийн үг 0 байгааг дээрх үр дүнгээс үзэж болно. Энэ нь төлөвийн байтын бүх бит Зураг 9.30-д үзүүлсэн шигээр тэг байна гэсэн үг юм.

```
eof бит = 0 тул fin.eof() тэг буцаана.  
fail бит = 0 тул fin.fail() тэг буцаана.  
bad бит = 0 тул fin.bad() тэг буцаана.  
бүх бит = 0 тул fin.good() тэг буцаана.
```

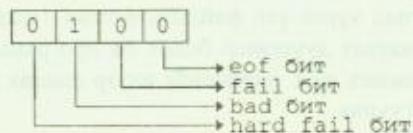


Зураг 9.30: Төлөвийн байт

Байхгүй файлыг хэрэглэхээр дээрх програмыг бага зэрэг өөрчилж, тухайлбал байхгүй "test.dat" файлыг нээх ifstream fin("test.dat"); командыг бичээд програмыг дахин ажиллуулбал дараах үр дүн гарна.

```
Can't open the file test.dat  
fin = 0x0  
state = 4  
eof() = 0  
fail() = 4  
bad() = 4  
good() = 0
```

Төлөвийн үг нь 4 гэсэн утгатай байгаа нь төлөвийн байтын битүүд Зураг 9.31-д үзүүлсэн шигээр тогтоогдсон байна гэдгийг заана.



Зураг 9.31: Төлөвийн байтын зарим битийн утга

Дээрх зургаар бол програмын үр дүн нь доор үзүүлсэн шиг байх ёстой.

```
fin = 0x0  
state = 4  
eof() = 0  
fail() = 0  
bad() = 1  
good() = 0
```

Гэтэл bad()=1 биш харин bad()=4 байгаагийн учрыг олохын тулд төлөвийг мэдүүлэх функцийдийн тодорхойлолтыг авч үзэх шаардлагатай. ios класс доорх нэрлэсэн тогтмолуудтай.

```
goodbit = 0x00  
eofbit = 0x01  
failbit = 0x02  
badbit = 0x04  
hardfail = 0x08
```

Түүнээс гадна төлөвийг мэдүүлэх аргуудыг доор үзүүлсэн шигээр тодорхойлсон байдаг.

```

1.
int ios::rdstate(void)
{
    return state ;
}
2.
int ios::eof(void)
{
    return state & eofbit ;
}
3.
int ios::fail(void)
{
    return state & (failbit | badbit | hardfail) ;
}
4.
int ios::bad(void)
{
    return state & (badbit | hardfail) ;
}
5.
int ios::good(void)
{
    return state == 0 ;
}

```

#### 9.28.13 Файлын төгсголийг хянах

Файлаас унших явцад түүний төгсгөлд хүрч очсон эсэхийг шалгаж тогтоохдоо хэрэглэх функцийг ifstream объектод байдал ба түүнийг while командтай хамт хэрэглэнэ. Унших бичих үйлээр файлын төгсгөлд хүрэх үед файлын төгсгөл (eof) гэсэн дохио програм руу илгээгдэж улмаар while давталт дуусгавар болох ба програмын удирдлага уг давталтын дараагаар байх мөр рүү шилжинэ. eof нохцелийг илээр шалгах бол while давталтыг доор ҮЗҮҮЛСЭН шигээр зохион байгуулна.

```
while (!ipfile.eof())
```

Програмл үүсгэх ipfile бол ifstream классын объект байх тул түүгээр дамжуулан eof() функцийг дуудна. Унших бичих үйл файлын төгсгөлд хүрч очсон байвал eof() функцийн үнэн (true) гэсэн утга бушна. Ингэснээр, ipfile.eof() үнэн болоход !ipfile.eof() илэрхийлийн хариу худал (false) болж, улмаар while давталт дуусгавар болно.

Урьдчилан нээсэн ipfile файлаас тэмдэгт мөр уншихдаа (>>) операторыг хэрэглэж болохгүй учир түүний оронд istream классын getline() функцийг хэрэглэнэ. Гэхдээ ifstream нь istream классаас удамших класс тул ipfile объектод getline() функцийг хэрэглэнэ. Энэ нь мөрийн төгсгелийг заах '\n' тэмдэгт хүргэлх тэмдэгтүүдийг файлаас уншиж эхнийхээ аргумент болох str хувьсагч руу хадгалах функций юм. Файлаас унших тэмдэгтийн тоог функцийн хоёрдахь аргумент нь заана. Униссан мөр бүрдийг cout<<str<<endl; командаар дэлгэц рүү бичнэ.

#### 9.28.14 Файлд нэмж бичих

Нэмж бичих горимоор файлыг нээхдээ дараах хоёр үйлдлийн аль нэгийг хийнэ.

```
1. ofstream fout("test11.dat", ios::app) ;
```

```
2. ofstream fout ;
   fout.open("test11.dat", ios::app) ;
```

Дээрх командууд нь нээх файл ажлын хавтас дотор байхгүй бол түүнийг шинээр үүсгэнэ. Харин, жишээ нь "test11.dat" нэртэй файл байвал түүнийг төгсгөлд нь мэдээлэл нэмж бичихээр нээнэ.

Доорх програм нь файлд нэмж бичих үйл явцыг хэрхэн зохион байгуулж болохыг харуулна.

```
//Program #9.52
#include <fstream.h>
void main(void)
{
ofstream fout("test11.dat") ;
fout << "Hello world" ;
fout.close() ;
ofstream fout1("test11.dat", ios::app) ;
fout1 << ", Hello beautiful world" ;
}
```

Програм "test11.dat" файлыг бичих горимоор нээж түүнд "Hello world" мөрийг бичээд хаана. Дараа нь файлыг нэмж бичихээр нээж түүнд ", Hello beautiful world" мөрийг бичихэд уг мөр файлын төгсгөлд нэмж бичигдэнэ. Үнэхээр тэгсэн эсэхийг type командаар

```
C:\>type test11.dat
```

гэж шалгана.

Хоёртын файлд нэмж бичих боломжийг дараах програмаар үзүүллээ.

```
//Program #9.52a
#include <fstream.h>
#include <stdlib.h>

class employee
{
private:
    char name [20];
    int age ;
    float height ;
public:
    void getdata(void) ;
    void showdata(void) ;
};

void employee::getdata(void)
{
    char ch ;
    cout << "Enter name: " ;
    cin.get(name,20).get(ch) ;

    cout << "Enter age: " ;
    cin  >> age ;

    cout << "Enter height: " ;
    cin  >> height;
    cin.get(ch) ;
}
```

```

void employee::showdata(void)
{
    cout << name << "/" << age << " years/" << height << " cm\n" ;
}

void main()
{
    employee emp ;
    long recnum;
    ifstream fin("test12.dat") ;
    if(fin.good() )
    {
        cout << "\nThe current content of the file test12.dat:\n" ;
        while(fin.read ((char*)&emp, sizeof(emp)))
        {
            emp.showdata() ;
        }
        fin.close() ;
    }
    ofstream fout("test12.dat",ios ::app| ios:: binary);
    if(fout. fail())
    {
        cout << "\nCannot open test12.dat for output:\n";
        exit(1) ;
    }

    cout << "\nEnter new data:\n" ;
    char ch='y' ;
    while(ch == 'y')
    {
        emp.getdata() ;
        fout.write((char*) &emp, sizeof(emp)) ;
        cout << "\nType 'y' to continue, any key to exit: " ;
        cin  >> ch ;
        cin.get() ;
    }

    fout.close() ;
    fin.open("test12.dat", ios::binary) ;
    if(fin.good())
    {
        cout << "\nHere are the new content of the file test12.dat:\n" ;
        while(fin.read ((char*)&emp, sizeof(emp)))
        {
            emp.showdata() ;
        }
        fin.close() ;
    }
}

```



Enter new data:  
 Enter name: George.  
 Enter age: 45.  
 Enter height: 180.  
 Type 'y' to continue, any key to exit: n  
 Here are the new content of the file test12.dat:  
 George/45 years/180 cm

Програмыг анх удаагаа ажиллуулж байгаа тохиолдолд файлыг шинээр үүсгээд өгөгдөл оруулах талаарх мэдээлийг дэлгэцэлдгийг дээрх үр дүнгээс харж болно. Дахин ажиллуулахад програм нь уг файлд байгаа мэдээлийг эхэлж дэлгэшид шинэ мэдээлэл оруулахыг хүсдэг. Үүний доорх үр дүнгээс харж болно.

 The current content of the file test test12.dat:  
George/45 years/180 cm  
Enter new data:  
Enter name: Bill.  
Enter age: 40.  
Enter height: 182.5.  
Type 'y' to continue, any key to exit: n  
Here are the new content of the file test12.dat:  
George/45 years/180 cm  
Bill/45 years/180.5 cm

#### 9.28.15 Санамсаргүй хандалтат файл

Санамсаргүй хандалттай файл нь түүнийг дэс дараалан уншихын оронд хүссэн байрлал руу нь шууд очиж унших боломжийг бүрдүүлнэ. Өнгөц харвал C++ хэлний хувьд бүх файл санамсаргүй хандалттай юм. Уншихаар нээх файлын хувьд түүний дурын хэсэг рүү очоод уншиж болно. Үүнтэй адил, зөвхөн бичихээр нээсэн файлын хувьд бас файлын дурын хэсэг рүү очоод бичиж болно. Гэхдээ ижил хэмжээ бүхий бичлэгүүдтэй файлын хувьд санамсаргүйгээр хандаж болно.

Файлыг санамсаргүй хандалтын горимд нээх бол дараах горимоос аль нэгийг нь хэрэглэнэ.

- o ios::in | ios::out
- o ios::in | ios::out | ios::ate
- o ios::in | ios::out | ios::ate | ios::binary

Өгөгдсөн "test12.dat" файлаас employee бичлэгийг санамсаргүйгээр олж өөрчлөх програмыг доор үзүүлснээр бичин.

```
//Program #9.53
#include <iostream.h>
#include <stdlib.h>

class employee
{
private:
    char name [20];
    int age ;
    float hieght ;
public:
    void getdata(void) ;
    void showdata(void) ;
};

void employee::getdata(void)
(char ch ;
cout << "Enter name: " ;
cin.get(name,20).get(ch) ;
cout << "Enter age: " ;
cin  >> age ;
cout << "Enter height: " ;
cin  >> height;
```

```

        cin.get(ch) ;
    }

void employee::showdata(void)
{
    cout << name << "/" << age << " years/" << height << " cm\n" ;
}

void main()
{
    char ch ;
    employee emp ;
    int snum=0 ;
    int recnum;
    int number ;

    fstream finout ("test12.dat", ios::in|ios::out|ios::ate|ios::binary) ;
    if(!finout.good())
    {
        cout << "Cannot open the file test12.dat" ;
        exit(1) ;
    }
    cout << "\nEnter the record number you want to change: " ;
    cin >> number ;
    cin.get(ch) ;

    finout.seekg(number*sizeof(emp)) ;
    finout.read((char *)&emp, sizeof(emp)) ;
    cout << "The content of the above record is:\n" ;
    emp.showdata() ;
    cout << "Make changes to the above record:\n" ;
    emp.getdata() ;
    finout.seekp(number*sizeof(emp)) ;
    finout.write((char *)&emp, sizeof(emp)) ;
    finout.seekg(0) ;

    while(finout.read((char *)&emp, sizeof(emp)))
    {
        cout << snum++ << ":" ;
        emp.showdata() ;
    }
}

```

 Enter the record number you want to change: 1  
 The content of the above record is:  
 George / 56 years / 170 cm  
 Make changes to the above record:  
 Enter name: Jack..  
 Enter age:40..  
 Enter height: 170..  
 0:Bill / 45 years / 180.5 cm  
 1:Jack / 40 years / 170 cm

#### 9.28.16 Объектын мэдээллийг файлд бичих жишээ програм

Объектын мэдээллийг файлд бичиж файлаас уншиж болох ба үүнийг дараах програмд үзүүлсэн шингэр програмчилна.

```

//Program #9.54
//writing an object to disk file
#include <iostream.h>
#include <conio.h>
#include <stdio.h>
#include <string.h>

class employee
{
private:
    char name [20];
    int basicpay ;
    int allowance ;
public:
    employee (char *n, int b, int a) ;
};

employee::employee(char *n, int b, int a)
{
    strcpy(name, n);
    basicpay = b ;
    allowance = a ;
}

void main()
{
    clrscr();
    employee emp("George", 9000, 8000) ;
    ofstream opfile("employee.dat", ios::binary);
    opfile.write ( (char *)&emp, sizeof(emp) );
}

```

Program #9.54-ийн main() функцийн эхэнд байх clrscr(); нь дэлгэц арчих функци, удаах employee emp("George", 9000, 8000); нь employee emp объектыг параметртэй байгуулагчаар байгуулах команд юм. Объектын утгыг файлд бичихэд хэрэглэх ofstream төрлийн opfile объектыг байгуулж "employee.dat" гэсэн файлтай уях командыг ofstream opfile("employee.dat", ios::binary); гэж бичнэ. Өврөөр заагаагүй бол ofstream урсгал нь бичгийн файлтай холбогддог учир хоёртын файлын нээх горимыг ios::binary гэж заадаг. Энэ ofstream нь ostream классаас удамших класс юм. Иймээс програмд хэрэглэх write() нь ostream классын гишүүн функци хэдий ч түүнийг ofstream классын opfile объектоор дамжуулан хэрэглэж болно. Уг функци нь объектын байгаа ойн хаяг, объектын хэмжээ хоёрыг аргумент хэлбэрээр гаднаас авна. Объектын мэдээллийг "employee.dat" файлд бичихдээ объектын хаягийг char төрлийнх болгож хувиргахаас гадна хэмжээг нь байтаар зааж огно.

#### 9.28.17 Объектын мэдээллийг унших жишээ програм

Объектын мэдээллийг файлаас уншихын тулд орох файлын урсгалын объект байгуулахад ifstream классыг хэрэглэж уг объектоор нээх файлын нэртэй холбож өгнө. Үүнийг доор үзүүлсэн шиг ганц командаар хийнэ.

```
ifstream ipfile("employee.dat", ios::binary);
```

Энд ipfile нь орох файлын урсгалын объект юм. сіп бол програмыг стандарт оролтын тохиромж болох гартай холбох оролтын урсгалын объектыг төлөөлдөг тухай омнө үзсэн билээ. Үүнтэй адил, ipfile нь програм болон "employee.dat" файлыг хооронд нь

холбох оролтын ургалыг заана. ifstream нь син объектынх шиг streambuf классаар удирдагдах оролтын буфер ашиглана.

Дээрх Program #9.54-т employee торлийн emp объектын мэдээллийг "employee.dat" файлд хэрхэн бичих талаар үзүүлсэн. Тэгвэл emp объектын утгыг "employee.dat" файлаас унших програмыг доор үзүүлсэн шигээр бичнэ.

```
//Program #9.55
//reading an object from disk file

# include <fstream.h>
# include <conio.h>
# include <stdio.h>
# include <string.h>

class employee
{
private:
    char name [20];
    int basicpay ;
    int allowance ;
public:
    employee (char *n, int b, int a) ;
    void showdata() ;
};

employee::employee(char *n, int b, int a)
{
    strcpy(name, n);
    basicpay = b ;
    allowance = a ;
}

void employee::showdata(void)
{
    cout << name << " / " << basicpay << " / " << allowance ;
}

void main()
{
    clrscr();
    employee emp ;
    ifstream ipfile("employee.dat",ios::binary);
    ipfile.read ( (char *)&emp, sizeof(emp) );
    emp.showdata() ;
    getch();
}
```



George / 9000 / 8000

Program #9.55-ын main() функцийн эхэнд байх clrscr(); нь дэлгэц арчих функцияа даах employee emp; нь employee торлийн emp хоосон объект байгуулах команд юм. Объектын тухай мэдээллийг файлаас авч emp объектын утга болгохын тулд ifstream торлийн ipfile объектыг байгуулж "employee.dat" файлтай уях командыг

```
ifstream ipfile("employee.dat",ios::binary);
```

гэж бичиж өгнө. Программ хэрэглэх `read()` нь `istream` классын гишүүн функц ч түүнийг `ifstream` классын `ipfile` обьектоор дамжуулан хэрэглэж болдог нь `ifstream` бол `istream` классаас удамшдагтай холбоотой. Энэ функц өмнө үзсэн `write()` функцийн нэгэн ажил обьектын байгаа ойн хаяг, обьектын хэмжээ хоёрыг аргумент хэлбэрээр гаднаас авна. Уг функц "employee.dat" файлд байх обьектын мэдээллийг уншиж `emp` обьектын огогдуулд рүү харгалзуулан хийснийг `showdata()` функцээр дэлгэцэд гаргаж харж болно.

#### 9.28.18 Объектын мэдээллийг файлд бичиж унших жишээ програм

Объектын мэдээллийг файлд бичих унших `write()`, `read()` хоёр функцийг өмнөх бүлгүүдэд үзүүлсэн шигээр тус тусад нь хэрэглэхээс гадна нэг программын хүрээнд хамт хэрэглэж болно. Ийм программыг доор үзүүлсэн шигээр бичнэ.

```
//Program #9.56
//writing an object to disk file, and reading back from disk file

# include <fstream.h>
# include <conio.h>
# include <stdio.h>
# include <string.h>

class employee
{
private:
    char name [20];
    int basicpay ;
    int allowance ;
public:
    employee (char *n, int b, int a) ;
    employee () ;
    void getdata() ;
    void showdata() ;
};

employee::employee(char *n, int b, int a)
{
    strcpy(name, n);
    basicpay = b ;
    allowance = a ;
}

employee::employee()
{
}

void employee::getdata()
{
    cout << "\nEnter the Employee Name: ";
    gets(name);
    cout << "Enter the Basic Pay: ";
    cin  >> basicpay ;
    cout << "Enter the Allowance: ";
    cin  >> allowance ;
}

void employee::showdata()
{
    int grosspay = basicpay + allowance ;
```

```

cout << endl ;
cout << setw(20) << name ;
cout << setw(8) << basicpay ;
cout << setw(12) << allowance ;
cout.width(8);
cout << grosspay ;
cout << endl ;
}

void heading()
{
    cout << endl ;
    cout << setw(20) << "Emp Name" ;
    cout << setw(8) << "Basicpay" ;
    cout << setw(12) << "Allowance" ;
    cout.width(8);
    cout << "Gross" ;
}

void main()
{
    clrscr();
    char c ;
    employee emp ;
    fstream iofile ;

    iofile.open("empl.dat", ios::app | ios::out | ios::in | ios::binary) ;
    do
    {
        emp.getdata() ;
        iofile.write( (char *)&emp, sizeof(emp) ) ;
        cout << "Do you enter data of another employee?\n" ;
        cout << "Enter y for YES and n for NO: " ;
        cin >> c ;
    } while (c == 'y');

    iofile.seekg(0) ;
    heading() ;
    iofile.read ( (char *)&emp, sizeof(emp) ) ;
    heading() ;
    while( !iofile.eof())
    {
        emp.showdata() ;
        iofile.read ( (char *)&emp, sizeof(emp) ) ;
    }
    iofile.close() ;
    getch();
}

```

 Enter the Employee Name: George  
 Enter the Basic Pay: 9000  
 Enter the Allowance: 2000  
 Do you enter data of another employee?  
 Enter y for YES and n for NO: y  
 Enter the Employee Name: Bill  
 Enter the Basic Pay: 8000  
 Enter the Allowance: 0000

```
Do you enter data of another employee?  
Enter y for YES and n for NO: n
```

Employee Name	Basic	Allowance
George	9000	2000
Bill	8000	0

Дээрх програмын main() функцийн employee emp; бол employee төрлийн emp объект байгуулах команд юм. Удаах fstream iofile; командаар fstream классын iofile объект байгуулж түүгээр дамжуулан хоёртын "empl.dat" файлыг (ios::app) нэмж бичих, (ios::out) бичих, (ios::in) унших гэсэн турван горимын нийлэмжээр нээнэ. Файл нээх open() функцийн нь fstream классынх. Энэ классын байгуулагч нь гаднаас авах бүхэл тоон аргумент дээрээ үндэслэн файл нээх үйлийг гүйцэтгэнэ. Энэ аргументийн авч болох утгуудыг ios класс дотор тодорхойлсон байдаг (Хүснэгт 9.1).

Гараас оруулах объектын утгыг "empl.dat" файлд бичин. Давталтын do-while команд дуусгавар болсны дараагаар файлын заагуур нь файлын төгсгэлд очсон байх тул эхнээс нь уншихын тулд уг заагуурыг файлын эхлэл рүү ofile.seekg(0); командаар шилжүүлнэ. Дараа нь файлыг уншиж дэлгэшлээд эзэст нь файлыг ofile.close(); командаар хаана. Энэ seekg(offset) функцийн авах (offset) харьцангуй утга нь файлын заагуурыг файлын эхнээс ямар байт зайд шилжүүлэхийт заана.

#### Мэдлэгээ шалгах асуулт

- 9.1. Толгой файл нь ямар файл болох, түүнийг C++ компайлер хэрхэн боловсруулах
- 9.2. Класс маягаар хэрэглэгдэх урсгалын ямар шатлал байдаг
- 9.3. Файл нээх гэдэг нь юу болох, үүнийг хэрхэн хийх
- 9.4. Файл нээх ямар арга байдаг
- 9.5. Файлын төгсгөлд хүрч очсоныг хэрхэн мэддэг
- 9.6. put(), get() функцийн зориулалт
- 9.7. Объектыг хэрхэн дискийн файлд бичих
- 9.8. sizeof() операторын хэрэглээ нь юу болох
- 9.9. Объектыг дискийн файлаас хэрхэн унших
- 9.10. ASCII болон binary (хоёртын) файлууд хоорондын ялгаа
- 9.11. Файлыг ямар ямар горимоор нээж болох
- 9.12. Дискийн файл хэрхэн программын ажлын файлтай холбогдох
- 9.13. getline() функцийн зориулалт нь юу болох
- 9.14. ostream классын write(), put() функцийн ялгаа
- 9.15. istream классын read(), get() функцийн ялгаа
- 9.16. Яагаад ostream классын бүх функцийг fstream классын объектод хэрэглэж болдог
- 9.17. Ямар класс ostream, istream классуудын эх класс болох
- 9.18. istream классын get(), getline() функцийн ялгаа

#### Жишээ бодлого

- 9.19. Алдаатай командуудыг олох
  - (a) cin.get().get()

```
(b) char ch ;
    cin.get(ch).get(ch) ;

(c) input buf[100], n = 10 ;
    cin.getline(buf, n, '\t').getline(buf, n, '\n');

(d) char *ch, *chl ;
    cout.put(ch).put(chl) ;

(e) char *addr) ;
    cin.read(addr) ;

(f) char *buf ;
    cout.write(buf).put('\n').put('\n') ;

(g) ifstream infile ;
    infile.open("dos.txt", ios::app)

(h) ifstream infile ;
    infile.close("dos.txt")
```

9.20. Дараах бүлэг команлын үр дүн юу болох

```
ofstream opfile("SAMPLE1.TXT") ;
opfile << "Necessity is the mother of invention" ;
```

9.21. Өмнөх 9.20 бодлогоор SAMPLE1.TXT дискийн файл үүссэн гэвэл дараах бүлэг команлын үр дүн юу болох

```
const max = 40 ;
char str[max] ;
ifstream ipfile ("SAMPLE1.TXT") ;
while (ipfile)
{
    ipfile.getline(str, max) ;
    cout << str << endl ;
}
```

9.22. Дараах бүлэг команд хийгдсэнээр диск рүү юу хадгалах, өмнөх 9.20 бодлогын SAMPLE1.TXT файлын агуулгаас ялгаатай эсэх

```
char str[] = "Necessity is the mother of invention";
ofstream opfile ("SAMPLE2.TXT") ;
for (int k=0; k<strlen(str); k++)
    opfile.put(str[k]) ;
```

#### Програм бичих бодлого

9.23. Жижиг үсгээр бичсэн зүйлийг дискийн файлаас уншиж том үсэг рүү шилжүүлээд оор файлд хадгалах

9.24. Оюутны нэр, үнэмлэхийн дугаар, таван хичээлийн дүн бүхий файл үүсгэх

9.25. Өмнөх 9.24-т үүсгэх файлыг ашиглан оюутны нэр, үнэмлэхийн дугаар, таван хичээлийн дүн, тэлгээрийн дундаж дүнг хадгалах тайлангийн файл үүсгэх

9.26. Өмнөх 9.25-т үүсгэх тайлангийн файлыг мөр мөрөөр уншиж дэлгэц рүү гаргах

9.27. Өмнөх 9.24-т үүсгэх файл руу шинэ бичлэг нэмж бичих

9.28. Өмнөх 9.24-т тодорхойлох файл руу шинэ объект нэмэх, унших

## ХАВСРАЛТ

- Хавсралт A: C++ хэлний оператор, 345
- Хавсралт B: Turbo c++ тусгай үтс, 347
- Хавсралт C: Онцгой тэмдэгт, 348
- Хавсралт D: Том программыг зохион байгуулах, 349
- Хавсралт E: Толгой файл, 353
- Хавсралт F: Зарим функцийн товч тайлбар, 356
- Хавсралт G: Turbo c++ програм, 363
- Хавсралт H: C++ командын товч лавлах, 373

## Хавсралт А

### C++ ХЭЛНИЙ ОПЕРАТОР

C++ хэлний операторуудыг хийгдэх дарааллаар нь Хүснэгт A.1-д үзүүлсэн шигээр 16 бүлэгт хувааж болно. Нэгдүгээр булгийн оператор нь хамгийн дээд түвшнийх, сүүлчийн бүлэг болох таслал оператор нь хамгийн доод түвшнийх. Нэг булгийн операторууд адил түвшинтэй байна. Ганц операндын (#2), нөхцөлт (#14) болон утга оноох (#15) операторууд баруунаас зүүн, бусад нь зүүнээс баруун чиглэлтэй байна.

*Хүснэгт A.1: C++ хэлний операторын жагсаалт*

Бүлэг №	Ганц Хос опе- ран- дых	Опе- ратор	Зориулалт
1		() [] -> :: .	Функцийн оператор, бүлэглэх оператор Хүснэгтийн оператор Дам сонголтын оператор (хаяг) Үйлчлэх хүрээний оператор Шууд сонголтын оператор (объект)
2		! ~ + - ++ -- & * sizeof new delete	Логик урвуу Бит үгүйсгэл (Нэгтийн гүйцээлт) Ганц операндын нэмэх (+ тэмдэг) Ганц операндын хасах (- тэмдэг) Нэмэгдүүлэх оператор (угтвар, дагавар) Хорогдуулах оператор (угтвар, дагавар) Хаягийн оператор Дам утга Хэмжээг байтаар тооцох оператор Динамик ой хуваарилах Динамик ой чөлеөлөх
3	X <sup>58</sup>	*	Уржих Хуваах Модуль авах (улдэгдэл)
4		.* ->*	Гишүүний дам утга (объект) Шууд бус гишүүний дам утга (хаяг)
5	X	+	Нэмэх
		-	Хасах
6	X	<< >>	Зүүн шилжилт Баруун шилжилт
7	X	< =< > =>	Бага Бага юм уу тэнцүү Их Их юм уу тэнцүү
8		== !=	Тэнцүү Тэнцүү бус
9	X	&	Битийн дагууд уржих (AND)
10	X	^	Битийн дагууд хоертын модулиар нэмэх (XOR)
11	X		Битийн дагууд нэмэх (OR)
12	X	&&	Логик уржих (AND)
13	X		Логик нэмэх (OR)

<sup>58</sup> X нь тухайн булгийн оператор нь хос операндынх гэдгийг заана.

14	?:	(a ? x:y нь if a then x, else y гэдэгтэй ижил утгатай)
15	=	Утга оноох
	*=	Үржвэрийг утга оноох
	/=	Ноогдворыг утга оноох
	%=	Модулийг утга оноох
	+=	Нийлбэрийг утга оноох
	-=	Ялгаврыг утга оноох
	&=	Битийн дагуух үржвэрийг утга оноох
	^=	Битийн дагуух хоёртын модулийн нийлбэрийг утга оноох
	=	Битийн дагуух нийлбэрийг утга оноох
	<<=	Зуун шилжилтийг утга оноох
	>>=	Баруун шилжилтийг утга оноох
16	,	Булэг илэрхийлэл

Дээрх хүснэгтэд байгаа (.) шууд сонголтын, (.\* ) гишүүний дам уттын, (:::) үйлчлэх хүрээний болон (? :) нэхцэлт оператороос бусдыг нь дахин тодорхойлж болдог.

## TURBO C++ ТУСГАЙ ҮГС

Тусгай үгийг бас нөөц үг гэнэ. Эдгээр үг нь програмчлалын хэлний шинжийг тодорхойлдог тул хувьсагчийн нэр зэрэг хэрэглэгчийн тодорхойлох зүйлд хэрэглэж болдоггүй.

asm	far	register
auto	_fastcall	return
break	float	_saveregs
case	for	_seg
cdecl	friend	short
char	goto	signed
class	huge	sizeof
const	if	_ss
continue	inline	static
_cs	int	struct
default	interrupt	switch
delete	_loadds	template
do	long	this
double	near	typedef
_ds	new	union
else	operator	unsigned
enum	pascal	virtual
_es	private	void
_export	protected	volatile
extern	public	while

Хавсралт С

ОНЦГОЙ ТЭМДЭГТ

Дараалал	Хийх үйлдэл	ASCII код
\a	Дохио дуугаргах	007
\b	Бүцаж арилгах	008
\f	Цаас хөөх	012
\n	Шинэ мөрийн эх рүү очих, шинэ мөрийн тэмдэгт	010
\r	Мөрийн эх рүү очих	013
\t	Доголдох	009
\\\	Бөхгөр зураас гаргах	092
\'	Дан хашилт гаргах	039
\"	Давхар хашилт гаргах	034
\xdd <sup>59</sup>	Арванзургаатын тоо гаргах	

<sup>59</sup> dd нь арванзургаатын тоо юм.

## ТӨМ ПРОГРАМЫГ ЗОХИОН БАЙГУУЛАХ

Хэмжээ томтой програмыг хянаж удирдаж болохуйц хэмжээтэй олон жижиг файлд хувааж бичиж болно. Ингэхдээ класс бүрийн тодорхойлолт, классын функциүүдийн тодорхойлолтыг тус тусад нь файлд хийнэ.

### D.1 Толгой файл

Классын тодорхойлолтыг .h өргөтгөл бүхий толгой файлд бичиж огно. Ийм файл нь классын тодорхойлолтоос гадна хэрэглэгчийн функцийн эх загвар, #define тогтмол, тогтмолон хувьсагч гэх мэтийг илэр зарлах мэдуулэгтэй байж болно. Ингэж тодорхойлох файлыг классын функцийн тодорхойлолт бүхий файлтай холбохдоо препроцессорын #include командыг хэрэглэнэ. Жишээ нь, x1x1x1...x1x1, x2x2x2...x2x2 зэрэг командын мөр бүхий header.h файлыг оор, тухайлбал file.cpp файлтай хэрхэн холбохыг Зураг D.1-д харууллаа.

<pre>header.h</pre> <pre>x1x1x1...x1x1 x2x2x2...x2x2 ----- xnxpxn...xnxpn</pre>	<pre>file.cpp</pre> <pre>#include "header.h" y1y1y1...y1y1 y2y2y2...y2y2 ----- yupup...yupup</pre>
---	--

*Зураг D.1: Толгой файл хэрэглэх*

C++ компайлер нь Зураг D.1-д үзүүлсэн шиг бүтэцтэй file.cpp файлыг боловсруулахын тулд эхлээд препроцессорын include командыг гүйцэтгэнэ. Программын #include<sup>60</sup> "header.h" команд нь хэрэглэгчийн header.h файлын доторхийг уг командын оронд нэмж оруулахыг препроцессорт үүрэг болгох ба ингэснээр "file.cpp" файл нь Зураг D.2-д үзүүлсэн шиг бүтэцтэй болно.

<pre>x1x1x1...x1x1 x2x2x2...x2x2 ----- xnxpxn...xnxpn y1y1y1...y1y1 y2y2y2...y2y2 ----- yupup...yupup</pre>
---

*Зураг D.2: Препроцессорын команд хийгдсэний дараах file.cpp файлын бүтэц*

### D.2 Оруулгын файлуудыг зохицуулах боломж

Оруулгын (include) олон файлыг зохицуулах арга механизмыг авч үзэхийн тулд Зураг D.3-д үзүүлсэн шиг агуулга бүхий "header1.h" файлыг үсгээ.

<sup>60</sup> Хэрэглэгчийн толгой файлын нийгээ давхар хашилтанд бичнэ.

```
XXXX-----XX
```

*Зураг D.3: header1.h файл*

Дараа нь дээрх "header1.h" файлыг оортөө холбох "header2.h" файлыг бас үүсгэнэ (Зураг D.4).

```
#include "header1.h"
```

```
YYYY-----YY
```

*Зураг D.4: header2.h файл*

Эцэст нь дээрх хоёр файлыг хэрэглэх "file.cpp" файлыг Зураг D.5-д үзүүлснээр үүсгэнэ. Эхний #include "header1.h" команд нь түүний "header1.h" файлын доторх зүйлээр сольж бичихийг препроцессорт даалгана. Ийм байдлаар #include "header2.h" команд мөн хийгдэнэ.

```
#include "header1.h"  
#include "header2.h"
```

```
ZZZZ-----ZZ
```

*Зураг D.5: file.cpp файл*

Ингэснээр омнөх файлууд Зураг D.6-д үзүүлсэн шиг бүтэцтэй болно.

header1.h

```
XXXX-----XX
```

header1.h

```
XXXX-----XX
```

header2.h

```
#include "header1.h"
```

```
YYYY-----YY
```

header2.h

```
XXXX-----XX
```

```
YYYY-----YY
```

file.cpp

```
#include "header1.h"  
#include "header2.h"
```

```
ZZZZ-----ZZ
```

file.cpp

```
XXXX-----XX  
XXXX-----XX
```

```
YYYY-----YY
```

```
ZZZZ-----ZZ
```

*Зураг D.6: Препроцессорын командууд хийгдсэнийг дараах байдал*

Препроцессорын команд хийгдэхийн омнөх байдлыг дээрх зургийн зүүн хэсэгт, препроцессорын команд хийгдсэнийг дараах байдлыг зургийн баруун хэсэгт тус тус үзүүлсэн. Программын file.cpp файл дотор

XXXX-----XX

гэсэн командын мөр хоёр удаа бичигдсэнийг Зураг D.6-аас харж болно. Хэрэв XXXX-----XX нь int i; гэсэн командын мөр гэж үзвэл нэг програмд ижил иэртэй хоёр хувьсагч байж болдоггүй лүрмийн дагуу алдаа гарах бөгөөд түүнийг засах бол "header1.h" файлыг Зураг D.7-д үзүүлсэн шигээр өөрчилж бичнэ.

```
#ifndef HEADER1  
#define HEADER1  
XXXX-----XX  
#endif
```

Зураг D.7: header1.h файл

HEADER1 бол ерийн ялгац нэр юм. Дараах командын мөр

```
#ifndef HEADER1
```

нь препроцессорыг дараах зүйлийг хийхэд нь чиглүүлийн. Хэрэв HEADER1 гэсэн ялгац нэр өмнө нь тодорхойлоогүй байвал энэхүү #ifndef HEADER1 ба #endif хоёр мөрийн хооронд байгаа зүйлийг боловсруулах ёстой.

Дээрх жишээнд HEADER1 нэрийг ямар нэгэн зориулалтаар өмнө нь огт тодорхойлоогүй, иймээс "header1.h" файл доторх #ifndef HEADER1 сонголтын командын хариу үнэн гарах тул препроцессор нь дараах хоёр мөр командыг

```
#define HEADER1  
XXXX -----XX
```

боловсруулах шаардлагатай болж улмаар түүний дараа "header1.h" нь Зураг D.8-д үзүүлсэн шиг бүтэцтэй болно.

header1.h

```
#ifndef HEADER1  
#define HEADER1  
XXXX-----XX  
#endif
```

header1.h

```
#define HEADER1  
XXXX-----XX
```

Зураг D.8: Препроцессорын команд хийгдсэний дараах байдал\_1

Препроцессорын команд хийгдэхийн өмнөх болон дараах байдлыг зургийн зүүн баруун хэсэгт харгалзуулан үзүүлсэн байгаа. Дээрхтэй төстэй байдлаар "header2.h" файлыг Зураг D.9-д үзүүлснээр өөрчилье.

```
#ifndef HEADER2  
#define HEADER2  
#include "header1.h"  
YYYY-----YY  
#endif
```

Зураг D.9: header2.h файлын бүтэц\_1

Үүний дараагаар холбогдох препроцессорын командууд нь хийгдсэний дараах "header2.h" файл нь Зураг D.10-д үзүүлсэн шиг бүтэцтэй болно.



*Зураг D.10: header2.h файлын бүтэц\_2*

Препроцессорын команд хийгдэхийн өмнөх дараах байдлыг дээрх зургийн зүүн баруун хэсэгт харгалзуулан үзүүлсэн болно. "header1.h", "header2.h" хоёр файлд хийсэн өөрчлөлт нь Зураг D.11-д үзүүлсэн шигээр file.cpp файлд тусгагдана.



*Зураг D.11: Препроцессорын команд хийгдсэний дараах байдал\_2*

Препроцессорын команд хийгдэхийн өмнөх дараах байдлыг дээрх зургийн зүүн баруун хэсэгт харгалзуулан үзүүлсэн бөгөөд програм дотор давхар мөр байхгүй байгааг Зураг D.11-ээс харж болно.

Олон файлыг нэгтгэн нэг програмын файл гаргаж авах үйлийг хэрхэн зохион байгуулах нь хэрэглэж байгаа OS болон програмчлалын системээс хамаардаг. Borland C++, Turbo C++ зэрэг системийн хувьд нэгтгэх файлуудын жагсаалт бүхий (project) төсөл файл үүсгэж хэрэглэх нь тохиромжтой байдаг бөгөөд энэ талаар эдгээр системийн заавраас үзэж болно.

## ТОЛГОЙ ФАЙЛ

Стандарт функц бол нийтлэг хэрэглэгддэг үйлдэл, тооцоог гүйцэтгэж чадах програмын багц код юм. Turbo C++ нь 300 гаруй стандарт функцийн сантай. Тэдгээрийн төрөлжсөн жагсаалт нь толгой файл гэгдэх олон файлд байна.

C++ програмыг шалгаж машины хэл рүү буулгах шатанд эх програмыг холбогдох толгой файлтай нь, улмаар стандарт функцийн сантай нь автоматаар холбож өгнө. Эх програм нь нэг файлынх байж болох ч машины програм ихэвчлэн эх програм болон хэд хэдэн толгой файлаас үүсдэг байна.

C++ компайлер нь хэдэн арван толгой файлтай. Жишээ нь, Turbo C++ компайлер дараах толгой файлуудыг хэрэглэн.

_defs.h	_null.h	alloc.h	assert.h	bcd.h
bios.h	complex.h	conio.h	constrea.h	ctype.h
dir.h	direct.h	dirent.h	dos.h	errno.h
fcntl.h	float.h	fstream.h	generic.h	graphics.h
io.h	iomanip.h	iostream.h	limits.h	locale.h
locking.h	malloc.h	math.h	mem.h	memory.h
new.h	process.h	search.h	setjmp.h	share.h
signal.h	stdarg.h	stddef.h	stdio.h	stdiostr.h
stdlib.h	string.h	strstrea.h	sys\stat.h	sys\timeb.h
sys\types.h	time.h	utime.h	values.h	varargs.h

Толгой файлууд тус бүрдээ олон арван функцийн төрөл зүйлийн тодорхойлолтыг агуулж байдаг. Тухайлбал, оролт гаралтын функцийн тодорхойлолт `iostream.h`, математик функцийн тодорхойлолт `math.h` файлд тус тус байх жишээтэй.

Толгой файлыг эх програмд нэмж оруулахдаа препроцессорын<sup>61</sup> гэгдэх `include` командаа програмын эхэнд хэрэглэнэ. Иймийн учир энэ сурах бичигт ашигласан ихэнх жишээ програмд дараах хоёр командыг байнга бичиж өгч байсан.

```
#include <iostream.h>
#include <conio.h>
```

Зарим толгой файлын тухай авч үзье.

### 1. `stdio.h`

Энэ файл нь оролт гаралтын үйлтэй холбоотой зарим нэгэн зүйлсийн тодорхойлолтыг агуулна. Тухайлбал, дараах стандарт оролт гаралтын функцуудийн тухай мэдээлэл түүнд байна.

<sup>61</sup> Препроцессорын командын мор ≠ тэмдэгтээр экзин.

clearerr	fclose	fcloseall	fdopen	feof	ferror
fflush	fgetc	fgetchar	fgetpos	fgets	fileno
flushall	fopen	fprintf	fputc	fputchar	fputs
fread	freopen	fscanf	fseek	fsetpos	ftell
fwrite	getc	getchar	gets	getw	perror
printf	putc	putchar	puts	putw	remove
rename	rewind	rmtmp	scanf	setbuf	setvbuf
sprintf	sscanf	strerror	_strerror	tempnam	tmpfile
tmpnam	ungetc	unlink	vfprintf	vfscanf	vprinnf
vscanf	vsprintf	vsscanf			

## 2. string.h

Тэмдэгт мөр, ойтой холбогдох боловсруулалт хийх олон функцийн тодорхойлолт энэ файлд байна.

_fmemccpy	_fmemchr	_fmemcmp	_fmemcpy	_fmemicmp
_fmemset	_fstrcat	_fstrchr	_fstrcmp	_fstrcpy
_fstrcsrn	_fstrupd	_fstricmp	_fstrlen	_fstrlwr
_fstrncat	_fstrncmp	_fstrnicmp	_fstrncpy	_fstrnset
_fstrpbrk	_fstrrchr	_fstrrev	_fstrset	_fstrspn
_fstrrstr	_fstrtok	_fstrupr	memccpy	memchr
memcmp	memcp	memicmp	memmove	memset
move data	movmem	setmem	stpcpy	strcat
strchr	strcmp	strcmpl	strcpy	strcspn
strdup	_strerror	strerror	stricmp	strlen
strlwr	strncat	strncmp	strncmpl	strncpy
strnicmp	strnset	strpbrk	strrchr	strrev
strset	strspn	strstr	strtok	strxfrm
strupr				

## 3. math.h

Энэ файл нь математик тооцоо хийх, бодолтын явцад гарах алдааг засах функциүүдийн тодорхойлолтыг агуулна.

abs	acos	acosl	asin	asinl	
atan	atanl	atan2	atan2l	atof	_atold
cabs	cabsl	ceil	ceill	cos	cosl
cosh	coshl	exp	expl	fabs	fabsl

floor	floorl	fmod	fmodl	frexp	freexpl
hypot	hypotl	labs	ldexp	ldexpl	
log	logl	log10	log10l	matherr	_matherrl
modf	modfl	poly	polyl	pow	powl
powl0	powl0l	sin	sinl	sinh	sinhl
sqrt	sqratl	tan	tanl	tanh	tanh1

#### 4. stdlib.h

Зарим стандарт функцтэй хамаатай тодорхойлолт энэ толгой файлд байна.

abort	abs	atexit	atof	atoi
atol	bsearch	calloc	div	ecvt
exit	_exit	fcvt	free	_fullpath
gcvt	getenv	itoa	labs	ldiv
lfind	_lrotl	_lrotr	lsearch	ltoa
_makepath	malloc	max	mblen	mbtowc
mbstowcs	min	putenv	qsort	rand
random	randomize	realloc	_rotl	_rotr
_searchenv	_splitpath	rand	strtod	strtol
_strtold	strtoul	swab	system	time
ultoa	wctomb	wctombs		

#### 5. iostream.h

Энэ файл нь зарим оролт гаралтын функцтэй хамаатай тодорхойлолтыг агуулна.

bad	clear	eof	fail	flush
get	getline	good	ignore	open
peek	put	putback	rdstate	read
seekg	seekp	tellg	tellp	write

#### 6. iomanip.h

C++ ургал, оролт гаралтай холбогдох боловсруулалтын тодорхойлолт, параметржсэн боловсруулалт хийх макро функци зэрэг зүйл энэ файлд байна.

dec	hex	oct	resetiosflags	setbase
setfill	setiosflags	setprecision	setw	

## ЗАРИМ ФУНКЦИЙН ТОВЧ ТАЙЛБАР

Функци	Буцаах утгын терел	Хийх үйлдэл	Толгой файл
abort ()	void	Процессыг хэвийн бусаар төгсгөх	stdlib.h
abs (i)	int	i-ийн абсолют утгыг олох	math.h
acos (d)	double	d-ийн арккосинусыг (acos) олох	math.h
arc (x,y,s,e,r)	void	(x,y) төв, s эхлэл өнцөг, e төгсгэл өнцөг, r радиустай нум зурах	graphics.h
asin (d)	double	d-ийн арксинусыг (arcsin) олох	math.h
atan (d)	double	d-ийн арктангесыг (arctan) олох	math.h
atof (s)	double	s тэмдэгт мөрийг давхар нарийвчлалтай бодит тоонд буулгах	stdlib.h
atoi (s)	int	s тэмдэгт мөрийг бүхэл тоонд буулгах	stdlib.h
atol (s)	long	s тэмдэгт мөрийг long бүхэл тоонд буулгах	stdlib.h
bar (l,t,r,b)	void	Зүүн дээд орой нь (l, t), баруун доод орой нь (r, b) цэгт тус тус байх дервэлжин зурах	graphics.h
ceil (d)	double	d-ээс бага биш хамгийн бага бүхэл тоог олох	math.h
circle (x,y,r)	void	(x,y) төв, r радиустай тойрог зурах	graphics.h
cleardevice ()	void	Зургийн дэлгэц арчих	graphics.h
clearviewport ()	void	Идэвхтэй (байгаа) зургийн цонхыг арчих	graphics.h
close (h)	int	h-тэй холбоотой файлыг хаах	io.h
closegraph ()	void	Зургийн горимыг дуусгавар болгох	graphics.h
clreol ()	void	Бичгийн цонхны мөрийн төгсгөл хүртэлхийг арчих	conio.h
clrscr()	void	Бичгийн цонх арчих	conio.h
cos (d)	double	d-ийн косинусыг олох	math.h
cosh (d)	double	d-ийн гипербол косинусыг олох	math.h
cprintf (o, . . .)	int	o хэвийн дагуу бэлдэх мөрийг дэлгэцлэх	conio.h
cputs(s)	int	s тэмдэгт мөрийг дэлгэцлэх	conio.h
cscanf (i, . . .)	int	i хэвийн дагуу тараас оруулга хийх	conio.h
ctime (&t)	char*	Огноо, цагийг t тэмдэгт мөрөнд буулгах	time.h
delay (m)	void	Процессыг m миллисекундээр зогсох	dos.h

delline ()	void	Заагчийн байгаа мөрийг устгаж арши бух мөрийг нь нэг мөрөөр дээш татах	conio.h
detectgraph (d,m)	void	Дэлгэцийн удирдлагын техник хангамжийт шалгах замаар d зургийн (драйвер) удирдах програм, т ажлын горимыг тогтоох	graphics.h
drawpoly (n,p)	void	n олон өндөгт зурах	graphics.h
ecvt (v,n,&d,&s)	char*	n оронтой v бодит тоог тэмдэгт мөрөнд буулгах; d болон s нь бухэл тоон төрлийн хувьсагчид	stdlib.h
ellipse (x, y, s, e, xr, yr)	double	(x,y) төв, s эхлэл өндөг, e төгсгэл өндөг, x-ийн хэвтээс радиус нь xr, y-ийн босоо радиус нь yr байх эллипс зурах	graphics.h
eof (f)	int	f файлын төгсгелийг шалгах	io.h
exit (u)	void	Бүх файл болон буферийг хааж програмыг дуусгавар болгох; (u нь дуусгаврын төлөв болгох функцизэс буцаах утга)	stdlib.h
exp (d)	double	e (e=2.718...) тоог d зэрэгт дэвшүүлэх	math.h
fabs (d)	double	d бодит тооны абсолют утрыг олох	math.h
fclose (f)	int	f файлыг хаах; Файл зөв хаагдвал 0 буцаах	stdio.h
fcloseall ()	int	Нээлттэй байгаа бүх файлыг хаах	stdlib.h
fcvt (v,n,&d, &s)	char*	v бодит тоог тэмдэгт мөрөнд буулгах; d ба s нь бухэл тоон төрлийн хувьсагчид. Зөв буулгалтыг n оронгоор нарийвчлах	stdlib.h
feof (f)	int	f файлын хувьд end-of-file буюу файлын төгсгөлд хурч очсон эсэхийг тогтоох; Хэрэв тийм бол 0 биш, угүй бол 0 утга буцаах	stdio.h
fgetc (f)	int	f урсгалаас нэг тэмдэгт авах	stdio.h
fgetchar ()	int	Стандарт оролтын төхөөрөмжөөс тэмдэгт авах	stdio.h
fgetpos (f, p)	int	f идэвхтэй файлын байрлал заагчийн байрлалыг авах	stdio.h
fgets (s, n, f)	char*	f файлаас n-1 тэмдэгт уншиж тэмдэгтэн s рүү хийх	stdio.h
filelength (f)	long	f файлын хэмжээг (байтаар) авах	io.h
fillellipse (x, y)	void	(x, y) төвтэй битүү эллипс зурж будах	graphics.h
fillpoly (n, p)	void	n орой, p талтай олон өндөгт зурж будах	
floodfill (x, y, c)	void	(x,y) цэг дотор нь орших битүү талбайг с өнгөөр будах	graphics.h
floor (d)	double	d -ээс ихгүй хамгийн их бухэл тоог олох	math.h

fmod (x,y)	double	Х-ийг у-т хувааж үлдэгдлийг олох	math.h
fopen (f, m)	FILE*	f файлыг ийн горимд нээх; Шинээр нээсэн урсгалын хаягийг буцах	stdio.h
fprintf(f, ...)	int	Бэлдсэн зүйлийг f файлд бичих	stdio.h
fputc (c, f)	int	f файл руу нэг тэмдэгт бичих	stdio.h
fputchar (c)	int	Стандарт гаралтын төхөөрөмж руу нэг с тэмдэгтийг гаргах	stdio.h
fputs (s, f)	int	f файл руу s тэмдэгт мөр гаргах	stdio.h
fread (p,m, n, f)	int	Тус бур нь ийн хэмжээтэй, н тооны багцыг f файлаас р хаяг руу унших	stdio.h
frexp (x, y)	double	Бодит тоог мантисс ба илтгэгчээр нь салгах	stdio.h
facanf (f,...)	int	f файлаас тодорхой хэвийн дагуу унших	stdio.h
fseek (f, o, w)	int	f файлын заагчийг w байрлалаас о зайд шилжүүлэх	stdio.h
isetpos (f, p)	int	f файлын заагчийг р байрлалц аваачих	stdio.h
ftell (f)	long	f файлын заагчийн байрлалыг авах	stdio.h
fwrite (s,m,n, f)	int	Ийн хэмжээтэй н тооны зүйлийг s хаягаас авч гаралтын f файлд нэмж бичих	stdio.h
gcvt (v, n ,a)	char*	n утгат оронтой v тоог s тэмдэгт мөр руу хөрвүүлэх	stdlib.h
getc (f)	int	f файлаас нэг тэмдэгт авах	stdio.h
getch ()	int	Гараас нэг тэмдэгт дэлгэшлэхгүйгээр авах	conio.h
getchar ()	int	Стандарт оролтын урсгалаас нэг тэмдэгтийг дэлгэшлэхгүйгээр авах	stdio.h
getche ()	int	Гараас нэг тэмдэгт авах; Түүнийг дэлгэцлэх	conio.h
getcolor ()	int	Хэрэглэж байгаа өнгийг олох	graphics.h
getdate (d)	void	date бүтцэн d-г системийн цагаар дүүргэх	dos.h
getimage(l,t,r,b)	void	Зүүн дээд орой нь (l, t), баруун доод орой нь (r, b) цэгт орших тэгш өндөгтээр хүрээлсэн зургийн хэсгийг ойл хадгалах	graphics.h
getpass (s)	char*	s нууц уг унших	conio.h
getpixel (x, y)	unsigned	(x,y) солбилцлын цэтийн онго авах	graphics.h
gets (s)	char*	Стандарт оролтын төхөөрөмжөөс s тэмдэгт мөрийг авах	stdio.h
gettext(l,t,r,b,x)	int	(l,t) цэгт зүүн дээд орой нь, (r,b) цэгт баруун доод орой нь тус тус орших тэгш өндөгтээр хүрээлгэсэн зүйлийг x-ийн руу хуулах	conio.h
gettime (t)	void	time бүтцэн t-г системийн цагаар дүүргэх	dos.h

getx ()	int	х тэнхлэгийн одоогийн утгыг авах	graphics.h
gety ()	int	у тэнхлэгийн одоогийн утгыг авах	graphics.h
gotoxy (x, y)	void	Заагчийг (x,y) цэг рүү шилжүүлэх	conio.h
highvideo ()	void	Тэмдэгт бичих өндөр эрчим тогтоох	dos.h
imag (x)	double	х комплекс тооны хуурмаг хэсгийг авах	complex.h
initgraph (d,m,p)	void	d удирдлага, m горим, p замналтай зургийн горимыг идэвхжүүлэх	graphics.h
insline ()	void	Бичгийн цонхонд заатч байгаа байрлалд шинэ мөр оруулах	conio.h
isalnum (c)	int	с нь үсэг-цифр мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isalpha (c)	int	с нь үсэг мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isascii (c)	int	с нь ASCII тэмдэгт мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
iscntrl (c)	int	с нь ASCII удирдах тэмдэгт мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isdigit (c)	int	с нь аравтын цифр мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isgraph (c)	int	с нь хэвлэгддэг ASCII график тэмдэгт мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
islower (c)	int	с нь жижиг үсэг мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isprint (c)	int	с нь хэвлэгддэг ASCII тэмдэгт мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
ispunct (c)	int	с нь цэг цэглэл мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isspace (c)	int	с нь цагаан зайн тэмдэгт мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isupper (c)	int	с нь том үсэг мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
isxdigit (c)	int	с нь 16-тын цифр мөн эсэхийг тогтоох; Хэрэв тийм бол 0 биш, эсрэгээр бол 0 утга буцаах	ctype.h
itoa (i, s, b)	char*	Бүхэл тоон i-г s тэмдэгт мөр	stdlib.h

РҮҮ В СУУРЬТАЙГААР БУУЛГАХ			
labs (l)	long int	1-ын абсолют утгыг олох	math.h
line (x, y, u, v)	void	(x,y) болон (u,v) цэгүүд хооронд шулуун зурах	graphics.h
linerel (x, y)	void	Заагч байгаа байрлалаас харьцангуйгаар (x,y) зай хүртэл шулуун зурах	graphics.h
lineto (x, y)	void	Заагч байгаа байрлалаас (x,y) цэг рүү шугам татах	graphics.h
log (d)	double	d-ийн натураг логарифмыг олох	math.h
log10 (d)	double	d-ийн 10-тын логарифмыг олох	math.h
lowvideo ()	void	Дэлгэцлэх тэмдэгтийн нам эрчим тогтоох	conio.h
ltoa (l, s, b)	char*	long төрлийн l-ийг s тэмдэгт мөрөнд в суурьтайгаар буулгах	stdlib.h
modf (x, i)	double	double x тоог 2 хэсэгт хувааж бүхэл хэсгийг i-д авч бутархайн хэсгийг буцаах	math.h
modfl (x, i)	double	long double төрлийн x тоог 2 хэсэгт хувааж бүхэл хэсгийг i-д авч бутархайн хэсгийг буцаах	math.h
movetext (l,t,r,b,d1,dt)	int	(l,t) зуун дээд, (r,b) баруун доод орой бүхий тэгш ёнцгэтийн доторхыг түүнтэй адил хэмжээстэй багасал (dt,d1) зуун дээд оройтой тэгш ёнцгэтийн рүү хуулах	conio.h
movevto (x, y)	void	Заагчийт (x,y) цэг рүү аваачих	graphics.h
norm (x)	double	x комплекс тооны абсолют утгын квадратыг авах	complex.h
normvideo ()	void	Дэлгэцлэх тэмдэгтийн хэвийн эрчмийт сэргээх	conio.h
nosound ()	void	Чанга яригийт унтраах	dos.h
open (f, a)	int	f файлыг a горимтойгоор нээх	io.h
outtext (s)	void	s тэмдэгт мөрийг зургийн горимд долгэцлэх	graphics.h
outtextxy (x,y,s)	void	s тэмдэгт мөрийг (x,y) цэгээс эхэлж дэлгэцлэх	graphics.h
pieslice (x,y,s,e,r)	void	(x,y) төв, s эхлэл болон e төгсгэлийн ёнцег, r радиустай сектор зурах	graphics.h
pow (x, y)	double	x -ийг y чарарт дэвшүүлэх	math.h
printf (...)	int	Бичгийн горимд стандарт гаралтын төхөөрөмж рүү өгөгдэл гаргах	stdio.h
putc (c, f)	int	f файл рүү с тэмдэгтийг гаргах	stdio.h
putch (c)	int	дэлгэц рүү с тэмдэгтийг гаргах	conio.h
putchar (c)	int	стандарт гаралтын төхөөрөмж рүү с тэмдэгтийг гаргах	stdio.h
putimage (l,t,i,c)	void	Өмнө нь getimage() функцийн хадгалсан i зургийг түүний зүүн дээд ёнцег нь (l,t) цэг дээр байхаар дэлгэц рүү буцааж гаргах; с нь цэгийг яаж зурахыг заах	graphics.h

<code>putpixel (x,y,c)</code>	<code>void</code>	(x, y) цэгийг с өнгөөр хатгах	<code>graphics.h</code>
<code>puts (s)</code>	<code>int</code>	стандарт гаралтын төхөөрөмж рүү з тэмдэгт мөрийт гаргах	<code>stdio.h</code>
<code>puttext (l,t,r,b,s)</code>	<code>int</code>	(l,t) ба (r,b) цэгүүдээр тодорхойлогдох дэлгэцийн хэсгийг з ойн муз рүү хуулах	<code>conio.h</code>
<code>random (i)</code>	<code>int</code>	0-осс i-1 хооронд байх нэг санамсаргүй зөврөг бүхэл тоог авчрах	<code>stdlib.h</code>
<code>randomize ()</code>	<code>void</code>	санамсаргүй тоон үүсгүүрийг ажилд бэлдэх	<code>stdlib.h</code>
<code>read (f, b, n)</code>	<code>int</code>	f файлаас n тооны байтыг b буфер рүү унших	<code>io.h</code>
<code>real (x)</code>	<code>double</code>	x комплекс тооны болит хэсгийг нь авах	<code>complex.h</code>
<code>rectangle (l,t,r,b)</code>	<code>void</code>	(l,t) ба (r,b) цэгэн орой бүхий тэгш ёнцегт зурах	<code>graphics.h</code>
<code>rename (o, n)</code>	<code>int</code>	o файлын нэрийг n болгох	<code>stdio.h</code>
<code>rewind (f)</code>	<code>void</code>	Файлын заагчийг f файлын эх рүү аваачих	<code>stdio.h</code>
<code>sector (x,y,s,e,xr,yr)</code>	<code>void</code>	(x,y) төв, s эхлэл болон e төгсгэлийн ёнцег, r радиус, xr хэвтээ болон уг боссоо тэнхлэгийн радиустай сектор зурах	<code>graphics.h</code>
<code>setbkcolor (c)</code>	<code>void</code>	Дэлгэцийн дэвсгэр өнгийг с болгох	<code>graphics.h</code>
<code>setcolor (c)</code>	<code>void</code>	Зурах өнгийг с болгох	<code>graphics.h</code>
<code>setfillstyle (p,c)</code>	<code>void</code>	Дүүргэлтийн хэвийг p, өнгийг с болгох	<code>graphics.h</code>
<code>setlinestyle (l,p,t)</code>	<code>void</code>	Шулуун зурах горимыг l хэлбэр, p хэвтэй, t зузаантайгаар тогтоох	<code>graphics.h</code>
<code>setpalette (p, c)</code>	<code>void</code>	Будгийн p өнгийг с болгох	<code>graphics.h</code>
<code>settextjustify (h,v)</code>	<code>void</code>	Вичиг тэгшлэх (h хэвтээ, v боссоо тэнхлэг)	<code>graphics.h</code>
<code>settextstyle (f,d,s)</code>	<code>void</code>	Вичгийн усгийн хэв f, чиглэл d, хэмжээ s байхаар тогтоох	<code>graphics.h</code>
<code>setwritemode (m)</code>	<code>void</code>	Зургийн горимд шулуун зурах горимыг m болгох	<code>graphics.h</code>
<code>sin (d)</code>	<code>double</code>	d-ийн синусыг авах	<code>math.h</code>
<code>sinh (d)</code>	<code>double</code>	d-ийн гипербол синусыг авах	<code>math.h</code>
<code>sleep (s)</code>	<code>void</code>	Программыг s секундээр зогсоох	<code>dos.h</code>
<code>sound (f)</code>	<code>void</code>	Чанга яригчийг f давтамжтай дуугаргах	<code>dos.h</code>
<code>sqrt (d)</code>	<code>double</code>	d -ээс квадрат язгуур авах	<code>math.h</code>
<code>strcat (d, s)</code>	<code>char*</code>	Тэмдэгт мөр d-ийг тэмдэгт мөр d-ийн араас залгах	<code>string.h</code>
<code> strchr (s, c)</code>	<code>char*</code>	z тэмдэгт мөрөөс с олдтол унших	<code>string.h</code>
<code>strcmp (s1, s2)</code>	<code>int</code>	Хоёр тэмдэгт мөрийг усгийн том, жижгийг ялгаж харьцуулах; Хэрэв s1<s2 бол серег, s1=s2 бол 0, s1>s2 бол зөврөг утга буцаах	<code>string.h</code>
<code>strcmpi (s1, s2)</code>	<code>int</code>	Хоёр тэмдэгт мөрийг усгийн том	<code>string.h</code>

		жижгийг алтажуй өөрчүүлах; Хэрэв $s1 < s2$ бол сөрөг, $s1=s2$ бол 0, $s1>s2$ бол сөрөг утга буцаах	
strcpy (s1, s2)	char*	s2 тэмдэгт мөрийг s1 руу хуулах	string.h
strlen (s)	int	s тэмдэгт мөрийн уртыг олох	string.h
strrchr (s,c)	char*	c тэмдэгт олдтол s тэмдэгт мөрийг араас нь урагш унших	string.h
strrev (s)	char*	s тэмдэгт мөрийг эргүүлэх	string.h
strset (s, c)	char*	s тэмдэгт мөрийн бүх тэмдэгтийг с болгох	string.h
strstr (s1, s2)	char*	s1 тэмдэгт мөрийг түүнээс s2 тэмдэгт мөрийн эхний илэрц олдтол унших	string.h
strtod (s, d)	double	s тэмдэгт мөрийг double терлийн d руу хөрвүүлэх	stdlib.h
strtol (s, l, b)	long	s тэмдэгт мөрийг b сууртайгаар long терлийн l тоо руу хөрвүүлэх	stdlib.h
system (s)	int	OS руу s командыг агах; Команд зөв хийгдээл 0, бусад үед 0 биш утга буцаах	stdlib.h
tan (d)	double	d-ийн тангенс утгыг авах	math.h
tanh (d)	double	d-ийн гипербол тангенс утгыг авах	math.h
tellg ()	long	Орох файлын заагчийн байрлалыг авах	math.h
tellp()	long	Гарах файлын заагчийн байрлалыг авах	math.h
textbackground (c)	void	Бичгийн дэвсгэр өнгийг с болгох	conio.h
textcolor (c)	void	Бичгийн өнгийг с болгох	conio.h
textmode (m)	void	Дэлгээний горимыг т болгох өөрчлөх	conio.h
time (b)	long int	Системийн цагийг секундээр тооцож авах	time.h
tcascii (c)	int	c -г ASCII хэлбэр руу буулгах	ctype.h
tolower (c)	int	c -г жижиг үсэг руу буулгах	ctype.h
toupper (c)	int	c -г том үсэг руу буулгах	ctype.h
wherex ()	int	Заагчийн байгаа хэвтээ байрлалыг авах	conio.h
wherey ()	int	Заагчийн байгаа босоо байрлалыг авах	conio.h
window (l,t,r,b)	void	Бичгийн горимд ажлын цонх тодорхойлох; Тэгш өнцөгийн хүрээг (l,t) болон (r,b) хоёр цэгээр тогтоох	conio.h
write (f, b, l)	int	l ширхэг байтыг b буферээс авч f файл руу бичих	io.h

## 1. TURBO C++ ПРОГРАМ

Turbo C++ нь Borland International фирмийн бүтээсэн C/C++ хэлний компайлер програм, гэхдээ бас засуур ба дэбагер<sup>62</sup> програмыг өөртөө багтаасан цогц систем юм. Програмын эх кодыг компьютерт оруулж диск рүү хадгалах, дуудаж засахад засуур програмыг, компайлер програмаар гаргаж авсан машины программыг туршиж зүгшрүүлэхэд дебагер програмыг тус тус хэрэглэнэ.

Олон програмын кодыг тус тусад нь эсвэл нэг програмын кодыг хэсэглэн хэсэг бүрийг өөр өөр цонхонд харж засаж болно. Хулгана хэрэглэснээр цонхон дотор, цонх хооронд шилжих, командын цэсээс сонгох мэтийн ажлыг хурдтай хийх боломжтой.

Turbo C++ нь хайх мэдээллийг хурдан олж авах бололцоог хангадаг хоорондоо логик холбоотой олон дэд хэсэг бухий заавартай. Мен стандарт функцийг хэрхэн хэрэглэхийг харуулсан жишээ энэ зааварт байх ба түүнийг завсрлын ойгоор<sup>63</sup> дамжуулан програмын ажлын цонхонд хуулж ажиллуулах юм уу өөрийн програмд нэмж оруулж болно.

### 1.1. Хулгана хэрэглэх

Turbo C++ програмын командаас сонгоход компьютерийн гарын оронд хулгана хэрэглэж болно.

Дэлгээний тодорхой байрлал, тодорхой объект дээр хулганыг аваачаад зүүн товчийг нь нэг товшсоноор тухайн байрлал юм уу объект сонгогдоно. Хулганы зүүн товчийг дараалуулан хоёр товших нь объект сонгогдонона дараагаар Enter товчийг товшихтой адил юм. Үүнээс гадна зүүн товч дараастай байхад хулганыг чирэх замаар хэсэг мэдээллийг будаж тэмдэглээд устгах юм уу өөр байрлалд хуулж, зөөж болно.

### 1.2. Гар хэрэглэх

Хулганаар хийж болох үйлдлийг дан товч, хос товчоор хийж болно. Ийм 3 бүлэг товчийг Turbo C++ програмд хэрэглэнэ.

- Hot keys буюу шуурхай товч
- Control keys буюу удирдах товч
- Function keys буюу функцийн товч

**Hot keys.** Тодорхой үүрэг бүхий дан юм уу хос товч юм. Жишээ нь, Alt товч дараастай байхад S товчийг товшиход хайлтын цонх идэвхжих бөгөөд үргэлжлүүлэн F товчийг товшиход Find (Хайх) команд сонгогдоно<sup>64</sup>.

**Control keys.** Гарын Ctrl болон бусад товчны хослол юм. Програмын ажлын цонхонд энэ булийн товчийг гол төлөв хэрэглэнэ.

**Function keys.** Компьютерийн гарын F1+F12 гэсэн товчууд нь функцийн товч юм. Функцийн товч бүрд тодорхой үүрэг оноож өгсөн байдаг. Жишээ нь, програмын зааврын цонхыг нэхэд F1, командын цэсийн мөрийг сонгоход F10 товчийг тус тус хэрэглэх жишээтэй.

<sup>62</sup> Debugger гэхийт дебагер гэж монголчилов.

<sup>63</sup> Clipboard гэлгийг завсрлын ой гэж монголчилов.

<sup>64</sup> Alt товчны хослолыг, жишээ нь Alt+C, тэмдэглэнэ.

## **2. Turbo C++ програмыг шинээр суулгах**

Turbo C++ програмыг машинд суулгахдаа, жишээ нь шахаж бэлдсэн `turbos.exe` програмыг хэрэглэнз.

## **3. Turbo C++ програмын үндсэн цонх**

Turbo C++ програм нь командын цэс, ажлын цонх, төлөвийн мөр гэсэн үндсэн 3 хэсэгтэй. Командын цэс нь доош нээгдэх цонхонд жагсаасан нэр бүхий олон командтай.

Командын цэсийг нээж харуулах 3 боломж байна.

- Нээх цэсийн нэр дээр хулганыг аваачаад зүүн товчийг нь товших
- F10 товчоор командын цэсийн мөрийг сонгоод сүмтэй товчоор нээх цэсийн нэр дээр очоод товших; Мон нэрийн эхний тол үсэгт харгалзах товчийг товших; Жишээ нь, File цэсийг нээх бол F10 товчны дараагаар F товч дээр товших
- Alt товчийг сонгох цэсийн изрийн эхийн тол үсэгт харгалзах товчны хамт товших; Жишээ нь, File цэсийг нээх бол Alt+F товчны хослолыг хэрэглэнз.

Turbo C++ програм олон цонхтой зэрэг ажиллаж чадна. Ийм 4 язнын цонх байдаг нь Editor, Message, Watch, Help цонх юм.

### **3.1. Editor цонх**

Editor цонхонд програмын эх кодыг гараас оруулж, диск рүү бичиж, улмаар дуудаж засаж болно. Олон файлыг тусад нь олон цонхонд зэрэг нээх юм уу нэг файлыг олон цонхонд хэсэглийн харж болно. Сүүлийн тохиолдолд аль нэг цонхонд хийсэн засвар үлдсэн бусад цонхонд шууд орно.

### **3.2. Message цонх**

Програмыг шалгах юм уу програмыг ажилшуулах явцад илэрсон алдааны талаарх мэдээллийг энэ цонхонд харуулиа. Алдааны мэдээлэлээс аль мөрөнд ямар шалтгааны улмаас алдаа гарсныг мэдэж болно.

### **3.3. Watch цонх**

Програм ажиллах явцад хувьсагчийн утга яаж өөрчлөгдж байгааг харах шаардлага гарвал Watch цонхыг хэрэглэнз. Хувьсагч програмын аль хэсэгт ямар утга авч байгааг хянах замаар програмын хаана ямар алдаа гарсныг олж болно.

### **3.4. Help цонх**

C++ командыг хэрэглэх зааврыг Help цонхонд харуулна. Командын зааврыг тухайн командыг яаж хэрэглэхийг харуулах жишээний хамт F1 товчоор харж болно. Ийм жишээг ажлын цонх руу хуулж улмаар судалж өөрийн болгох нь анхаасаа алдаа багатай програм бичиж сурахад ихээхэн тустай.

### **3.5. Доош нээгдэх нээ**

Turbo C++ командын цэс нь програмын боловсруулалтанд хэрэглэх олон командтай. Тэдгээрийг доош нээгдэх цонхонд торолжуулэн жагсаасан байдаг.

### **3.6. Төлөвийн мөр**

Энэ мөрөнд хэрэглэгчийн сонголтын тухай товч мэдээллийг харуулна.

#### 4. Turbo C++ програмын бүтэц

C++ програм нь функцийн тогтоно. Програмыг олон функцийд хувааж бичих нь хоёр гол давуу талтай. *Изгүүсэрт*, програмын үр дүн буруу байвал буруу ажиллаж байгаа функцийт нь түр тусгаарлаад програмыг ажиллуулж болно. Програмын аль функцийн буруу ажиллаж байгааг илрүүлэх зорилгоор хийж байгаа үйлийг програмын туршилт, зүгшрүүлэлт гэх бөгөөд энэ ажлыг хенгечвчлоход Turbo C++ лебагер програмыг хэрэглэнэ. *Хоёрдугаарт*, програм бүхэлдээ зөв ажиллаж байвал түүний зарим хэсгийг өөр програмд шууд авч хэрэглэж болно. Ийм маягаар зүгширсэн бэлэн функцийн сан үүсгэн ашиглаж болох бөгөөд энэ нь дараа дараагийн програмыг бичих хугацааг богиносгох сайн талтай.

C++ програмд `main()` функцийн тодорхойлолт нь функцийн толгойн ба командын гэсэн хоёр хэсэгтэй.

##### 4.1. Функцийн толгойн хэсэг

Функцийн төрөл буюу буцаах утгын төрөл, нэр, авах параметрийн жагсаалт зэрэг нь функцийн толгойн хэсэгт байна. Функцийн бүр бусдаас ялгагдах нэргийн толгойн хэсэгтэй байна.

##### 4.2. Функцийн командын хэсэг

Хийх үйлдэл нь тохирох багц командыг функцийн командын хэсэгт бичиж огно. Нэг ч командгүй хоосон функцийн толгойн хэсэгтэй байна.

#### 5. Turbo C++ програмыг ажиллуулах

Turbo C++ програмын файл үүсгэж, шинэ програм оруулж улмаар ажиллуулахад хийх үйлдэл нь доор үзүүлсэн шиг дараалалтай байна.

- Turbo C++ (`tc.exe`) програмыг эхэлж дуудах
- File цэсийн цонхыг Alt+F товчоор нээх
- File|New командааг сонгох

Turbo C++ програмын ажлын цонх нээгдэнэ. Доор үзүүлсэн програмыг гараас оруулна. Оруулгын явцад алдаа гаргавал Backspace буюу буцаж арчих товчоор алдаатай тэмдэгтийг арчаад зөв тэмдэгтийг оруулна. Мөр бүрийг (`;`) цэгтэй таслааар төгсгөөд Enter товчоор дараагийн шинэ морийн эхэнд заагчийг шилжүүлнэ.

```
#include <stdio.h>
void main()
{
    int x, y, z ;
    printf("\nPlease enter an integer") ;
    scanf("%d", &x) ;

    printf("\nPlease enter a second integer") ;
    scanf("%d", &y) ;
    z = x + y ;
    printf("\nTotal is: %d", z) ;
}
```

Програмын кодыг оруулж дуусаад диск рүү, жишээ нь “`first.cpp`” нэрээр хадгалахдаа F2 товчоор харгалзах цонхыг нээнэ. Програмын файлын нэр голдуу 8 тэмдэгтийн урттай байх ба нэрийн өргөтгөл нь .cpp байна. Гэхдээ .c өргөтгөл хэрэглэхийг Turbo C++ мөн зөвшөөрдөг.

## 6. File цэс

Файлын зориулалттай олон команд **File** цэст байна. Тухайлбал, **New**, **Open**, **Save**, **Save as**, **Save all**, **Change dir**, **Print**, **DOS shell**, **Quit** гэсэн 10 язын команд энэ цэст байна.

**New.** Программын шинэ цонх нээх бөгөөд энэ нь **NONAMExx.C (PP)** тэсэн нэртэй байна<sup>65</sup>. Уг нэр түр зуурын тул файлыг хадгалах үед нэр өгөхийг систем сануулдаг.

**Open.** Программын файлыг дискээс уншиж **Turbo C++ Editor** программын ажлын цонхонд оруулахад хэрэглэнэ.

**Save.** Харж байгаа цонхонд байгаа файлыг диск рүү бичихэд хэрэглэнэ. Командыг **F2** товчоор дуудаж болно.

**Save as.** Харж байгаа цонхонд байгаа файлыг оөр нэрээр зөвлөх хуучин нэрээр нь оөр газар руу бичихэд тусална. Хуучин файл нь хэвээр үлдэнэ.

**Save all.** Өөрчлөлт орсон бүх файлыг цонх нь идэвхтэй байгаа эсэхийг харгалзахгүйгээр диск рүү бичиз.

**Change dir.** Программын файл байгаа газрыг сонгоход хэрэглэнэ.

**Print.** Харж байгаа цонхонд байгаа программын кодыг хэвлэнэ. Программаас хэсэглэн хэвлэх бол мужийг эхэлж сонгоно. Хэвлэх мужийн эхэнд заагчийг аваачаад **Ctrl+K+B** товчоор "эхлэл" тэмдэглээг тавиад заагчийг мужийн тогсголд аваачаад **Ctrl+K+K** товчоор мужийн "төгсгөл" тэмдэглээг хийнэ. Ингэснээр, программын энэ хэсэг бусдаас ялгарч будагдах ба түүнийг **Ctrl+K+P** товчоор хэвлэнэ. Гэхдээ хэвлүүр бэлэн эсэхийг эхэлж шалгах шаардлагатай.

**DOS shell.** **Turbo C++** программаас түр гарч DOS орчинд ажиллах боломж олгоно. Программ буцаж орох бол DOS командын мөрөнд "exit" гэж бичээд **Enter** товчийг товшино.

**Quit.** **Turbo C++** программаас бүрмесон гарахад хэрэглэнэ. Энэ командыг **Alt+X** товчоор мон сонгож болно.

## 7. Edit цэс

Программын файл үүсгэж улмаар түүнд өөрчлөлт хийх зэрэгт хэрэглэх олон команд бүхий цэс юм. Уг цэсийн командаар программын кодын хэсгийг арчих, өөр газар хуулах, цонх хооронд зөөх юм уу хуулж болно. Цэсийн аль ч командыг хэрэглэхийн омно зөөх, хуулах юм уу арчих хэсгийг сонгож будаж тэмдэглэсэн байна. Сонголтыг хийхэд компьютерийн гар эсвэл хулгана хэрэглэнэ.

### 7.1. Компьютерийн гараар сонгох

Зөөх, хуулах эсвэл устгах кодын хэсгийг программаас сонгож тэмдэглэхэд компьютерийн гар хэрэглэх бол

- **Shift** товчийг аль нэгэн сумтай товчны хамт хэрэглэнэ.
- Сонгох мужийн эхлэх байрладл заагчийг аваачаад **Ctrl+K+B** товчоор "эхлэл" тэмдэглээ хийнэ. Дараа нь мужийн тогсгол хэсэгт нь заагчийг аваачин **Ctrl+K+K** товчоор "тогсгол" тэмдэглээ хийнэ. Ингэснээр программын муж тодоор будагдана.
- Хэрэв нэг уг сонгох бол заагчийг түүн дээр аваачаад **Ctrl+K+T** товч хэрэглэжээл уг будагдана.

<sup>65</sup> xx бол 0-99 хооронд байх бүхэл тоо.

- Хэрэв нэг мөр сонгох бол заагчийг түүн дээр аваачаад Ctrl+K+L товч хэрэглэвэл мөр будагдана.

## 7.2. Хулганаар сонгох

Програмын тодорхой мужийг хулганаар сонгож тэмдэглэхэд дараах үйлдлийг хийнс.

- Тэмдэглэх мужийн эхлэл юм уу төгсгөл хэсэгт хулгандын заагчийг аваачаад түүний зүүн товчийг дараастай хэвээр хулганаа чирж мужийн төгсгөл юмуу эхлэлт хэсэгт очоод эзест нь зүүн товчийг чөлөөлнө.
- Нэг мөр сонгох бол заагчийг түүн дээр аваачаад хулгандын зүүн товчийг хоёр удаа товшино.

## 7.3. Edit цэсийн команд

Edit цэс нь Undo, Redo, Cut, Copy, Paste, Clear, Copy Example, Show Clipboard гэсэн 8 командатай.

**Undo.** Сүүлд өөрчилсөн эсвэл устгасан зүйлийг бушааж сэргээх команд юм. Ажлын цонхонд хийсэн аливаа өөрчлөлтийг хуучнаар нь сэргээнэ.

**Redo.** Undo-командын сүүлчийн үр дүнг хүчингүй болгоно.

**Cut.** Тэмдэглэсэн програмын мужийг хайчилж Clipboard буюу түр санаах завсрлын ойд хийнэ. Үүнийг дараагаар нь програмын өөр хэсэг рүү юм уу өөр цонх руу хуулж болно. Командыг Shift+Del товчоор бас дуудна.

**Copy.** Тэмдэглэсэн програмын мужийн хуулбарыг түр ойд хийнэ. Үүнийг дараа нь програмын өөр хэсэг рүү юм уу өөр цонх руу хуулж болно. Turbo C++ заавраас жишээ програм хуулж авахад энэ командыг бас хэрэглэнэ. Ингэхдээ жишээг сонгоод Copy командаар түр санаах ой руу хуулна. Ctrl+Ins товчийг бас хэрэглэнэ.

**Paste.** Түр санаах завсрлын ойд байгаа зүйлийг заагч байгаа газарт хуулахад хэрэглэнэ. Хуулах зүйлийг эхлээд Cut эсвэл Copy командаар завсрлын ойд хийсэн байна. Командыг дуудаадаа Shift+Ins товчийг бас хэрэглэнэ.

**Clear.** Cut командатай төстэй. Гэхдээ энэ команд хайчилсан зүйлээ завсрлын ойд хийдэгтүй. Түүнийг C + Del товчоор дуудаж болно.

**Copy Example.** Turbo C++ заавраас жишээ програм хуулж авахад хэрэглэнэ.

**Show Clipboard.** Түр санаах завсрлын ойд байгаа зүйлийг Clipboard цонхонд харуулна. Сүүлд хэрэглэсэн Cut эсвэл Copy командаын үр дүн завсрлын ойд байна.

## 8. Help цэс

Заавартай холбогдох командын цэсийг өмнө дурдсан З аргаас гадна F1 эсвэл Ctrl+F1 товчны нийлмэжээр нээнэ. Help цэс нь Turbo C++ програмын талаар эсвэл хэрэглэж байгаа команд, функцийн талаар дэлгэрэнгүй мэдээлэл, заавар авах боломж олгоно. Turbo C++ зааврыг хэрэглэхэд холбоотой Contents, Index, Topic Search, Previous Topic, Help on Help ба About гэсэн 6 команд энэ цэст байна.

**Contents.** Turbo C++ зааварт багтсан сэдвийн жагсаалтыг харуулна.

**Index.** Turbo C++ командын эсвэл тодорхой сэдвийн талаарх зааврыг авахад хэрэглэх тулхүүр үсгийн жагсаалтыг цагаан толгойн үсгийн дараалалд харуулна.

**Topic Search.** C++ хэлний талаарх заавар харах бол хэрэглэнэ. Командыг дуудаадаа Ctrl+F1 товчийг бас хэрэглэнэ.

**Previous Topic.** Омнө харсан зааврын цонхыг дахин харах бол хэрэглэнэ. Turbo C++ сүүлийн 20 цонхыг санаж байдаг. Энэ командыг Alt+F1 товчоор дуудаж болно.

**Help on Help.** Turbo C++ зааврыг хэрхэн хэрэглэх талаарх зааврыг авч болно.

**About.** Turbo C++ програмын үйлдвэрлэгч, хувилбарын дугаар зэрэг мэдээллийг эндээс авч болно.

## 9. Run иэс

Идэвхтэй цонхонд харж байгаа програмыг ажиллуулахад энэ цэсийн командыг хэрэглэнэ. Ихэнх команд нь програмын зөв бичих дүрмийг шалгаж .obj файл гаргаж улмаар машины .exe/.com програмын файл үүсгээд програмыг ажиллуулна. Run, Program Reset, Go to Cursor, Trace Into, Step Over, Arguments гэсэн 6 команд энд байна. Дебагер програмын туслалцаатайгаар програмыг туршихад эдгээр командыг хэрэглэнэ.

**Run.** Харж байгаа цонхны програмыг ажиллуулахад хэрэглэнэ. Програмыг шалгаж .obj улмаар .exe/.com програмын файл үүсгээд програмыг ажиллуулна. Шууд ажиллуулах бол Ctrl+F9 товчийг хэрэглэнэ.

**Program Reset.** Програмыг алхмаар ажиллуулж байгаад програмыг дахин эхлүүлэх шаардлага гарвал хэрэглэнэ. Энэ командыг Ctrl+F2 товчоор шууд ажиллуулж болно.

**Go to Cursor.** Програмыг туршиж байхад хэрэглэж болно. Програмыг заагч байгаа мөр хүртэл ажиллуулаад тур зогсоно. Түүний оронд F4 товчийг мөн хэрэглэж болно.

**Trace Into.** Програмыг мөр мөрөөр ажиллуулна. Дебагер програм нь функцийг дуулагдсан мөрөнд очвол уг функцийн бүх мөрийт нэг бүрчлэн гүйцэтгэнэ. Функцийг ажиллаж дуусах үед уг функцийн дуудсан, жишээ нь main() функцийг рүү буцаж очно. Энэ командын оронд F7 товчийг мөн хэрэглэж болно.

**Step Over.** Trace Into командын адилгаар програмыг мөр мөрөөр ажиллуулна. Гэхдээ дуудагдсан функцийг нэг команд шигээр гүйцэтгэгээрээ Trace Into командаас ялгаатай. Функцийн F8 товч энэ командыг орлоно.

**Arguments.** Эхэлж ажиллахдаа гаднаас өгөгдөл заавал авах програмын хувьд хэрэглэнэ. OS-ийн командын мороос оруулж өгөх мэдээллийг энд зааж огне.

## 10. Debug иэс

Програмын логик алдааг илрүүлэхэд энэ цэсийн командыг хэрэглэнэ. Дебагер програмыг хэрэглэхэд туслах Inspect, Evaluate/modify, Call stack, Watches, Toggle breakpoint, Breakpoints гэсэн 6 командтай.

**Inspect.** Өгөгдлийн уттыг шалгах бол хэрэглэнэ. Ийм өгөгдөл нь тэмдэгт, хаяг юм уу заалт, хүснэгт, функцийн, класс эсвэл бүтээгдэхийн төрлийнх байж болно. Командыг мөн Alt+F4 товчоор дуудаж болно.

**Evaluate/modify.** Хувьсагчийн уттыг шалгаж шаардлага гарвал өөрчлөхөд хэрэглэнэ. Ctrl+F4 товчоор уг командыг шууд ажиллуулна.

**Call stack.** Програм ажиллах явцад дуудагдаж ажилласан функцийн жагсаалтыг харуулна. Командыг Ctrl+F3 товчоор шууд дуудаж болно.

**Watches.** Утгыг нь хянах хувьсагч, хүснэгт мэтийн объектын жагсаалт үүсгэхтэй холбоотой Add watch, Delete watch, Edit watch, Remove All watches гэсэн 6 команд бүхий цэсийг харуулна.

**Add watch.** Дебагер програмын туслалцаатайгаар програмыг алхам алхмаар ажиллуулж байх үед тодорхой хувьсагчийн утгыг шалгах шаардлага гарвал хэрэглэнэ.

**Delete watch.** Хувьсагчийг жагсаалтаас арчихад хэрэглэх команд юм.

**Edit watch.** Хувьсагчийн нэрийг засах мэтээр өөрчлөлт оруулах боломж олгоно.

**Remove All watches.** Хувьсагчийн жагсаалтыг арчина.

**Toggle breakpoint.** Програмын ажиллагааг түр зогсоох командын мөрийг тэмдэглэх эсвэл омнох тэмдэглээг болиулах бол хэрэглэнэ. Тэмдэглэсэн мөр дээр програм очоод дараах 3 товчоос аль нэгийг товших хүртэл түр зогсоно.

F7 (Trace Into) Функцийн бүх командыг нэг бүрчлэн гүйцэтгэнэ.

F8 (Step over) Функцийг нэг команд мэтээр гүйцэтгэнэ.

Ctrl+F9 (Run) Програмыг дахин ажиллуулна.

Түр зогсолт хийх мөрийг тэмдэглэх, болиулах үүрэгтэй Toggle breakpoint командыг мөн шууд ажиллуулж болно.

## 11. Search цэс

Тодорхой үг, функцийн тодорхойлолт, алдаатай командын мөр зэргийг ажлын цонхонд харж байгаа програмын файлаас хайж олоход хэрэглэнэ. Энэ цэст Find, Replace, Search again, Go to line number, Previous error, Next error, Location function гэсэн 7 команд байна.

**Find.** Find командаыг Ctrl+Q+F товчоор эхлүүлж болно. Хайлт хийхэд дагах хэд хэдэн шалгуур бүхий цонхыг нээнэ.

- Case sensitive – Том жижиг үсгийг ялгаатай үзэх
- Whole words only – Зөвхөн бүтэн үзээр дүйлгэх
- Regular expression – Тодорхой бүтцэл (хэв шинжит) илэрхийлэл
- Direction – Хайлтыг урагш хойш хийхийг тогтоох
- Scope – Хайлтын хамрах хүрээг тогтоох; ийм хүрээ нь нэг файл юм уу бүх файлыг хамарч болно.
- Origin – Хайлт хаанаас эхлэхийг заах; ийм эхлэл нь заагчийн байрлал эсвэл файлын эхлэл, файлын тогсгел байж болно.

**Replace.** Файлаас олж солих үтийг энэ командаар нээгдэх цонхны харгалзах талбарт оруулах; нээгдэх цонхонд байх мэдээлэл нь солих ажилтай холбоотой сонголтыг эс тооцвол Find цонхныхтай бараг ижил; Replace командаыг ажиллуулахдаа Ctrl+Q+A товчийг бас хэрэглэнэ.

**Search again.** Find юм уу Replace командаын үед зааж огсон зүйлийг хайх юм уу солих ажлыг үргэлжлүүлэх бол хэрэглэнэ. Энэ командаыг Ctrl+L товчоор мөн эхлүүлж болно.

**Go to line number.** Гараас оруулах дугаар бүхий командын мөрийг харуулах команд юм.

**Previous error.** Заагч байгаагаас омнох алдаатай юм уу санамжтай мөрийг програмаас олоход тусална. Message цонхонд ямар нэгэн мэдээлэл байх тохиолдолд энэ командыг хэрэглэнэ. Командыг Alt+F7 товчоор мөн ажиллуулж болно.

**Next error.** Ээлжит алдаатай юм уу санамжтай мөрийг програмаас олоход тусална. Message цонхонд ямар нэгэн мэдээлэл байх тохиолдолд хэрэглэнэ. Next error командыг Alt+F8 товчоор эхлүүлж болно.

**Location function.** Функцийн нэрийг олоход туслах энэ команд зөвхөн дебагер програмыг хэрэглэж байхад идэвхжин.

## 12. Window ижс

Turbo C++ програмын цонх ямар байхыг Window цэсийн командаар тогтооно. Цонхны харагдацыг өөрчлөх олон команд бий. Тухайлбал, Size/Move, Zoom, Tile, Cascade, Next, Close, Close All, Message, Output, Watch, User Screen, Register, Project, Project notes, List гэсэн 14 команд байна.

**Size/Move.** Ажлын цонхыг зоох юм уу хэмжээг нь өөрчлөхөд хэрэглэнэ. Хэмжээг өөрчлөхөд Shift товчийг аль нэгэн сумтай товчтой хамт хэрэглэнэ. Энэ командыг Ctrl+F5 товчоор шууд ажиллуулж болно.

**Zoom.** Харж байгаа цонхыг дэлгэн дүүртэл томсгох юм уу багасгахад хэрэглэнэ. Унтраалга шиг сэлгүүр команд тул том цонхыг багасгаж, бага цонхыг томсгоно. Энэ командыг F5 товчоор оруулан хэрэглэнэ.

**Tile.** Нээлттэй цонхнуудыг дэлгэнэд багтахаар нь эгнүүлж харуулна. Энэ командаын дараагаар давхассан цонх байхгүй болно.

**Cascade.** Нээлттэй цонхнуудыг давхцуулж харуулна. Энэ командаын дараа цонх бүрийн тодорхой хэсэг харагдаж байдаг.

**Next.** Ээлжит цонхыг идэвхтэй цонх болгоно. Цонх хооронд шилжиж харах бол хэрэглэнэ. F6 товчийг мөн хэрэглэж болно.

**Close.** Харж байгаа, өөрөөр хэлбэл хамгийн дээр байгаа цонхыг хаана. Энэ командаын оронд F3 товчийг хэрэглэж болно.

**Close All.** Бүх цонхыг хааж түүхчилсэн бүртгэлийг арчина.

**Message.** Энэ командаар Message цонхыг нээнэ. Гэхдээ энэ цонхыг омно нь нээсэн байвал түүнийг идэвхтэй цонх болгоно.

**Output.** Програмын үр дүнгийн цонхыг харуулна.

**Watch.** Watch цонхыг нээж түүнийг идэвхтэй болгоно. Програмыг дебагер програмын туслацаатайгаар турших үед хэрэглэнэ. Энэ цонхонд утгыг нь хянах хувьсагчийн тухайн утга зэргийг жагсааж харуулна.

**User Screen.** Програмын үр дүнгийн цонхыг харуулна. Цонхыг томсгож харах шаардлагагүй бол Output командаыг хэрэглэнэ.

**Register.** Програмын туршилтын үед төв процессорын регистрүүдийн утгыг харах шаардлагатай бол хэрэглэнэ.

**Project.** Програмын тосолд оруулсан файлын жагсвалтыг харна.

**Project notes.** Програмын тосол бүрийн хувьд хар дэвтэр хөтөлж өөртөө зориулан төрөл бүрийн тэмдэглээ хийх бол хэрэглэнэ.

**List.** Бүх нээлттэй файлын жагсаалтыг харуулна. Энэ командыг Alt+O товчоор орлуулж болно.

### 13. Options цэс

Turbo C++ програмын тохируулга хийх олон команд энэ цэст байна. Програм ажиллахдаа дагах тохируулгыг Application, Compiler, Transfer, Make, Linker, Librarian, Debugger, Directories, Environment, Save гэсэн олон командын тусламжтайгаар хийнэ. Turbo C++ програмыг сайн эзэмших хүртлээ түүний стандарт тохируулгыг хэвээр хэрэглэх нь хамгийн зөв шийдэл юм.

**Application.** Turbo C++ програмын тохируулгын зарим хэсгийг харуулна.

**Compiler.** Компайлерын ажиллагаатай холбогдох тохируулгыг энд хийдэг.

**Transfer.** System цэст шинэ програм нэмж оруулах юм уу байгаа программыг арчихад хэрэглэнэ.

**Make.** Ямар нохцөл үүсвэл .exe программын файл гаргаж авах ажлыг зогсоож болохыг Make командаар нээх цонхонд зааж өгнө.

**Linker.** .obj файлуудыг нэгтгэж .exe программын файл гаргаж авах тохируулгыг уг командаар нээх тохируулгын цонхонд зааж өгнө.

**Librarian.** Системийн болон хэрэглэгчийн сантай холбоотой тохируулгыг энд хийнэ.

**Debugger.** Дебагер програм хэрхэн ажиллахыг тогтооход хэрэглэнэ.

**Directories.** Компайлер нь хаанаас .h ба .lib файлуудыг олж авч болохыг энэ командаар нээгдхэх цонхонд зааж өгнө. Мөн .c (.pp) файлыг хаана хадгалааха энэ цонхонд зааж өгнө.

**Environment.** дебагер, засуур програм хэрхэн ажиллах зэргийг зааж өгнө.

**Save.** Turbo C++ тохируулгыг байнга хэрэглэх бол энэ командаар хадгална.

### 14. Compile цэс

Энэ цэсийн команд нь Turbo C++ програм ба программын тослийг<sup>66</sup> шалгаж машины програм гаргаж авах зориулалттай. Энд Compile, Make, Link, Build ALL, Information, Remove messages гэсэн команд байдаг.

**Compile.** Програмыг шалгаж .obj файл гаргана.

**Make.** Програмыг шалгаж ямар нэгэн алдаа гараагүй бол .obj улмаар .exe файл үүсгэнэ. Гэхдээ эх кодын файл ба түүний .obj файлын огноог жишиж байж тухайн программыг дахин шалгах эсэхээ шийддэг. Тухайлбал, программын өөрчлөлтийн дараагаар .obj файл үүсгээгүй байвал программыг заавал шалгана.

**Link.** Өмнө үүсгэсэн .obj файлууд ба .lib<sup>67</sup> файлуудыг холбож .exe файл гаргаж авахад хэрэглэнэ.

<sup>66</sup> Програмын тесол бол хоорондоо холбоотой, хамтдаа ажиллах програмууд юм. Ийм багцыг Project цэсийн командаар үүсгэнэ.

<sup>67</sup> lib бол library гэдэг англи үтийн товчлол юм. Стандарт функцийн сангийн файлыг заана.

