Արմինե Նավասարդյան

C++
ԾՐԱԳՐԱՎՈՐՄԱՆ ԼԵԶՈՒ

ուսումնամեթոդական ձեռնարկ
հեռակա ուսուցմամբ ուսանողների համար

Երևան
Հեղինակային հրատարակչություն
2010

ՀՏԴ 681.3/.5:004 (07)
ԳՄԴ 32.973 g7
Ն 380

Երաշխավորված է հրատարակման
Խ Աբովյանի անվան ՀՊՄՀ-ի գիտական խորհրդի կողմից

Գրախոսներ՝ Ռ.Արմենակյան
Հայաստանի պետական տնտեսագիտական համալսարանի «Ինֆորմացիոն տեխնոլոգիաների և համակարգերի» ամբիոնի վարիչ, տ.գ.թ., դոցենտ՝

Է.Հայկազյան
ՍԱՐՏԻԳ Արտաքին տնտեսական կապերի համալսարանի «Ինֆորմասցիոն տեխնոլոգիաների և համակարգերի» ամբիոնի դասախոս. տ.գ.թ.

Հ.Շարխաթունյան
Կրթության ազգային ինստիտուտի «Ինֆորմատիկայի և տեղեկատվական տեխնոլոգիաների» բաժնի վարիչ

Խմբագիր Ա.Նավասարդյան

Նավասարդյան Արմինե
Ն 380 C++ ծրագրավորման լեզու: Ուս.մեթ ձեռնարկ հեռակա ուսուցման ուսանողների համար: - Եր.: Հեղինակային հրատարակչություն, 2010թ-72էջ:

Ձեռնարկը նախատեսված է ՀՊՄՀ –ի հեռակա ուսուցման ուսանողների համար: Այստեղ շարադրված են C++ լեզվի կարևորագույն հասկացությունները, որոնց հետ ծանոթանալով ուսանողը ձեռք է բերում սեփական ծրագրերի և նախագծերի կազմման անհրաժեշտ կարողություններ: Այն կարող է օգտակար լինել նաև ծրագրավորման հարցերով զբաղվողներին:

ՀՏԴ 681.3/.5:004(07)
ԳՄԴ 32.973g7

ISBN 978-9939-53-610-1

Նախաբան

Ձեռնարկը կազմված է 3 գլուխներից, գրականության ցանկից, բովանդակությունից և 3 հավելվածներից:

Առաջին գլուխը նվիրված է C++ լեզվում ծրագրերի մշակման հիմնական էտապների, ծրագրերի կազմության, գրելաձևի և հիմնական կառուցվածքների նկարագրման խնդիրներին: Առանձին պարագրաֆներում հնարավորինս հակիրճ ձևով տրված են որոշ կառուցվածքների մանրամասն բացատրություններ, որոնք ամրագրված են պարզ օրինակներով:

2-րդ գլուխը նվիրված է նոր տիպերի ստեղծմանը: Նախորդ գլխում ծանոթանալով հիմնական տվյալների և ֆունկցիայի գաղափարի հետ, այստեղ ընթերցողին հնարավորություն է ընձեռվում առանց հավելյալ բարդությունների ծանոթանալու օբյեկտային կողմնորոշման հիմնական տարրերի հետ՝ այն է կլասի և օբյեկտի գաղափարներին: Այս գլխում մեկը մյուսի հետևից դիտարկվում են կառուցվածքներ, որոնք միանման ձևով են աշխատում ինչպես բազային տիպերի այնպես և կլասների հիմքերի վրա հայտարարված օբյեկտների համար, որը թույլ է տալիս ընթերցողին մտածել օբյեկտի մակարդակով: Այստեղ նաև բացատրվում է օբյեկտային կողմնորոշման հիմնական դրույթներից 1-ը՝ տվյալների ինկապսուլացիայի գաղափարը:

Վերջին գլուխը նվիրված է օբյեկտային կողմնորոշման հիմնական գաղափարախոսության վերջին 2 դրույթներին: Աստեղ ընթերցողը ծանոթանում է ինչպես ժառանգականության մեխանիզմներին, անպես և պոլիմորֆիզմի հետ:

Նկատենք, որ ողջ ձեռնարկում կարմիր թելի պես ձգվում է բարդ հասկացությունների պարզաբանումը հնարավորինս պարզ օրինակների քննարկմամբ գործելաոճը: Ձեռնարկի նպատակն է ընթերցողին տալու ինքնուրույն աշխատելու հնարավորություն, ոստի հիմնական շեշտադրումը ոչ թե կայանում է բարդ խնդիրների ձևակերպման և լուծումների առաջարկման մեջ, այլ ընդհակառակը:

Վերջում բերված են թվով 3 հավելվածներ, որոնցից 1-ը նվիրված է ծրագրերի կազմման, կոմպիլացման և կատարողական ֆայլերի ստացման ու դրանց թողարկման հարցերին, 2-րդը՝ առավել հաճախ հանդիպող, ներդրված որոշ ֆունկցիաների (մասնավորապես մաթեմատիկական և տողերի հետ աշխատող ֆունկցիաների) գրելաձևերին, իսկ վերջին հավելվածում բերված են ստուգողական աշխատանքների մի քանի օրինակներ:

ԳԼՈՒԽ 1 ՀԻՄՆԱԿԱՆ ՀԱՍԿԱՑՈՒԹՅՈՒՆՆԵՐ

§ 1-1 C++ լեզվում ծրագրերի մշակման էտապները

C++ -ում ծրագրերի մշակման պարտադիր էտապներն են՝

1. Ելակետային տվյալներով ֆայլի ստեղծում, որը պետք է ունենա .cpp ընդլայնումը:
2. Ելակետային ֆայլի կոմպիլացիա, որի արդյունքում պետք է ստանալ .obj ընդլայնմամբ օբյեկտային ֆայլ
3. .obj ընդլայնմամբ ֆայլերի, անհրաժեշտ գրադարանների հետ, հավաքագրում, որի արդյունքում պետք է ստանալ կատարողական ֆայլ:

Ծրագրերի կազմման, կոմպիլացման և կատարողական ֆայլերի ստացման ու դրանց թողարկման համար անհրաժեշտ գործողությունների գործընթացը բերված է հավելված 1-ում

§ 1-2 Դուք արդեն ծրագիր եք գրում

Դիտարկենք 1.2.1-ում բերված ծրագրային կոդը

Ծրագրային կոդ 1.2.1

```
1   #include <iostream.h>
2   // Այստեղ իրականացվում է main() ֆունկցիայի կանչ
3   int main()
4    /* main()  ֆունկցիան համարվում է գլխավոր բոլոր ծրագրերի համար
5   և կանչվում է ավտոմատ կերպով օպերացիոն համակարգի կողմից*/
6   {
7        cout<<"Hello world! \n";
8        cout<<"Here is 5:" << 5 << "\n";
9        cout<<"Good bye\n";
10       return 0;
11  }
```

Թողարկման արդյունքը `

```
Hello World
Here is 5:"   5
Good bye
```

#include-ը նախապրոցեսորի հրաման է, որը նշանակում է գտնել նրանից հետո գրված ֆայլը /այստեղ *iostream.h*-ը/ և նրա պարունակությունը տեղավորել ծրագրի հենց այդ հատվածում: (<) և

(>)–սիմվոլները ցույց են տալիս, որ այդ ֆայլը անհրաժեշտ է փնտրել բոլոր այն թղթապանակներում, որոնք պատասխանատու են և պարունակում են նման ֆայլեր: Կոմպիլյատորը նախապրոցեսորի հրամանների հետ աշխատում է նախքան ծրագրի կոմպիլացիան:

Ծրագրի հիմնական կողը սկսվում է 3-րդ տողից, որտեղ իրականացված է ***main() անվանմամբ ֆունկցիայի կանչ***:

C++-ում յուրաքանչյուր ծրագիր պարտադիր պարունակում է մեկ main() ֆունկցիա: Ֆունկցիաներին կանդրադառնանք ավելի ուշ, առայժմ բավական է գիտենալ որ նրանք կանչվում են այլ ֆունկցիաների միջից, բացառությամբ main() ֆունկցիայի, որը առանձնահատուկ ֆունկցիա է և ավտոմատ կերպով կանչվում է ծրագրի թողարկման ժամանակ օպերացիոն համակարգի կողմից: main բառից առաջ գրված int բառը նշանակում է, որ այդ ֆունկցիան պետք է ***վերադարձնի*** ամբողջ տիպի մեծություն, որն էլ իրականացվում է ֆունկցիայի ավարտին ***return*** հրամանի օգնությամբ /տե՛ս տող 10-ը, վերադարձվող մեծությունը՝ 0 /:

Այն ամենը, ինչը տեղավորված է բացվող և փակվող ձևավոր փակագծերի միջև, կոչվում է ***ֆունկցիայի մարմին***: Ծրագրի ֆունկցիոնալությունը ամփոփված է 7, 8, 9 տողերում, որոնք ***cout օբյեկտի*** օգնությամբ էկրան են դուրս բերում թողարկման արդյունքի 3 տողերը: Ընդ որում որևէ արտահայտություն էկրան դուրս բերելու համար անհրաժեշտ է cout բառից հետո դնել << օպերատորը, /որին անվանում են ելքային հոսքի օպերատոր/, այնուհետև էկրան դուրս բերվող արտահայտությունը տեղավորել կրկնակի չակերտների մեջ, իսկ որևէ մեծության դուրս բերման համար՝ << օպերատորից հետո անհրաժեշտ է պարզապես գրել այդ մեծությունը:

Տեքստերի խմբագրման համար cout օբյեկտի հետ գործածվել է “\n” հատուկ սիմվոլը, որը նշանակում է տողի ընդհատում: Հատուկ սիմվոլների այլ օրինակներ բերված են աղ.1.2.1-ում

Աղյուսակ 1.2.1

No	Սիմվոլ	Նշանակությունը	No	Սիմվոլ	Նշանակությունը
1	\t	Տաբուլացիա	4	\’	Մեկ չակերտ
2	\b	Հետադարձ մեկ դիրքով	5	\?	Հարցական նշան
3	\’’	Կրկնակի չակերտ	6	\\	Հետադարձ թեք գիծ

Ծրագրերի ընթեռնելիության բարձրացման նպատակով նրանցում գործածվում են մեկնաբանություններ /տես տող 2, 4,5/: Ընդ որում 2-րդ տողում գործածված է կրկնակի թեք գծով (//) մեկնաբանություն, որը կոմպիլյատորին տեղեկացնում են որ անհրաժեշտ է արհամարհել այն

ամենը, ինչ գրված է այդ սիմվոլից հետո մինչև այդ տողի ավարտը, իսկ 4-5 տողերում գործածված է մեկ թեք գիծ աստղիկով (/*) մեկնաբանություն, որոը կոմպիլյատորին տեղեկացնում են, որ պետք է արհամարհել այն ամենը, ինչ գրված է նրանից հետո մինչև մեկնաբանության ավարտը նշող սիմվոլը, այն է՝ (*/): Այս դեպքում մեկնաբանությունները կարող են տեղավորվել լինել մեկից ավելի տողերում: Արդյունքում բերված ծրագիրը համարժեք է 1.2.1 ծրագրային կոդին

Ծրագրային կոդ 1.2-2

```
1   #include <iostream.h>
2   int main()
3   {
4       cout<<"Hello world! \n";
5       cout<<"Here is 5:" << 5 << "\n";
6       cout<<"Good bye\n";
7       return 0;
8   }
```

§ 1-3 Փոփոխականներ և հաստատուններ

Փոփոխականների և հաստատունները նախատեսված են ինֆորմացիայի պահպանման համար: Գործածումից առաջ նրանք պետք է նախ հայտարարվեն: Փոփոխականներում պահպանվող ինֆորմացիոն հետագայում ենթակա է փոփոխման, իսկ հաստատուններում պահպանվողները՝ ոչ:

➢ Փոփոխականների բազային տիպերը

Փոփոխականների հայտարարումը կոմպիլյատորին տեղեկացնում է նրանց տիպի մասին՝ արդյունքում հիշողության մեջ համապատասխան քանակությամբ տեղ է պահեստավորվում: C++--ում գործածվող փոփոխականների բազային տիպերը բերված են աղյուսակ 1.3.1-ում

Աղյուսակ 1.3.1.

Հ/Հ	Անվանումը	Բացատրություն	Չափ	Արժեքների միջակայքը
1	**2**	**3**	**4**	**5**
1	Bool	Բուլյան	1	True կամ False
2	unsigned short int	Առանց նշանի կարճ ամբողջ	2	0 - 65 535
3	short int	կարճ ամբողջ	2	-32 768 – 32 767
4	unsigned long int	Առանց նշանի երկար ամբողջ	4	0 – 4 294 967 295

5	long int	Երկար ամբողջ	4	-2 147 483 648 – 2 147 483 647
6	int /16 կարգանի/	ամբողջ	2	-32 768 – 32 767
7	int /32 կարգանի/	ամբողջ	4	-2 147 483 648 – 2 147 483 647
8	unsigned int /16 կարգանի/	առանց նշանի ամբողջ	2	0 – 65 535

1	2	3	4	5
9	unsigned int /32 կարգանի/	առանց նշանի ամբողջ	4	0 – 4 294 967 295
10	Char	սիմվոլային	1	256 սիմվոլներ
11	Float	ստորակետով	4	1.2e-38 – 3,4e38
12	Double	կրկնակի	8	2,2e-308 – 1,8e308

Եթե առանց նշանի փոփոխականի առավելագույն արժեքին գումարվի 1, արդյունքում կստացվի 0, իսկ եթե նշանով փոփոխականի առավելագույն արժեքին գումարվի 1 կստացվի տվյալ տիպի համար նվազագույն բացասական արժեքը /- 32 768 կամ – 2 147 483 648/: Ան երևույթը անվանում են գերլցում:

➢ Փոփոխականների հայտարարումը

Հայտարարման սխեման հետևյալն է

փոփ_տիպ փոփ_անվ_1, փոփ_անվ_2, …, փոփ_անվ_n ;

որտեղ **փոփ_տիպ-ը** հայտարարվող փոփոխականներիի տիպն է, **փոփ_անվ_i-ն** օգտագործողի կողմից տրված i-րդ փոփոխականի անվանումն է: **;** - ը ցույց է տալիս հրամանի ավարտ:

Օրինակ՝

```
unsigned int x,y; // հայտարարված են unsigned int տիպի x և y փոփոխականները
bool k;   // հայտարարված է բուլյան տիպի k փոփոխականը
```

Փոփոխականների տիպերի անվանումները կարող են փոխարինվել կեղծանուններով, այսպես

typedef փոփ_տիպ կեղծանուն ;

որտեղ ***typedef***-ը պահեստավորված բառ է, ***փոփ_տիպ***-ը փոփոխականի տիպն է, ***կեղծանունը*** ՝ վերանվանված տիպի անվանում է: Օրինակ

```
typedef unsigned short int USHORT;      //unsigned short int տիպը վերանվանվել է USHORT-ի
```

➢ Փոփոխականներին արժեքների վերագրումը

Փոփոխականներին արժեքների վերագրումը իրականացվում է {=} վերագրման օպերատորի օգնությամբ և դա կարող է կատարվել ինչպես փոփոխականների հայտարարման տողում, այնպես էլ առանձին,

Օրինակ՝

```
unsigned short int Width=0 ;      //վերագրումը կատարվում է հայտարարման տողում

unsigned short int Width;
Width=0;                  //վերագրումը կատարվում է հայտարարմանը հաջորդող տողում
```

➢ **Հաստատուններ**

Հաստատունների հայտարարումը իրականացվում է 2 եղանակով

1. Հաշվի չի առնվում հայտարարված հաստատունի տիպը:

#define հաստ_անվ լիտ_հաստ_մեծություն

որտեղ ***define***–ը նախապրոցեսորի հրամանով ձևակերպման հրաման է, ***հաստ_անվ***-ը հաստատունին վերագրված անվանումն է, իսկ ***լիտ_հաստ_մեծություն***-ը՝ լիտերային հաստատունի մեծությունն է

Օրինակ՝

```
#define  studensPerClass 10;
```

2. Հաշվի է առնվում հայտարարված հաստատունի տիպը:

const տիպ անվ = լիտ_հաստ_մեծ;

որտեղ ***const***-ը պահեստավորված բառ է հաստատունի հայտարարման մասին, ***տիպ***-ը հաստատունի տիպն է, ***անվ***-ը՝ մեծության անվանումը, իսկ ***լիտ_հաստ_մեծ-ը***՝ լիտերային հաստատունի մեծության արժեքն է

Օրինակ՝

```
const unsigned short int studentsPerClass=20;
```

§ 1-4 Արտահայտություններ և օպերատորներ

➢ **Արտահայտություններ**

Արտահայտությունները փոփոխականների, հաստատունների, գործողությունների որոշակի թույլատրելի հավաքածուներ են, որոնք ավարտվում են կետ-ստորակետով (;):

Ջրոյական արտահայտությունը՝ պարզ կետ-ստորակետ (;) է: Ամենապարզ արտահայտություններից է վերագրման գործողությունը:

Օրինակ՝

```
a=b+c;
```

նշանակում է b-ին գումարել c-ն, արդյունքը վերագրել a–ին: Չնայած այս արտահայտության մեջ իրականացվում է երկու գործողություն՝ գումար և վերագրում՝ նրա վերջում դրվում է մեկ կետ-ստորակետ:

Նկատենք որ բացակը, տաբուլացիան կամ տողի ընդհատումը արտահայտությունների մեջ հաշվի չեն առնվում և կիրառվում են ծրագրի ընթեռնելիության բարձրացման համար: Օրինակ ստորև բերված արտահայտությանները համարժեք են անկախ գրելաձևերից

```
X=a+b;      կամ      x      =a
                     +      b      ;
```

➢ Բլոկներ կամ կոմպլեքս արտահայտություններ

Տրամաբանորեն փոխկապակցված արտահայտությունների հավաքածուն կարելի է միավորել *կոմպլեքսների՝ բլոկների* մեջ: Բլոկը սկսվում է բացվող ձևավոր փակագծով ({) և ավարտվում է փակվող ձևավոր փակագծով (}): Նրանում առկա կամայական արտահայտություն ավարտվում է կետ-ստորակետով (;), սակայն բլոկի վերջում կետ ստորակետ չի դրվում:

C++ լեզվում կամայական հրաման ավարտվում է *կետ ստորակետով*: Տողի ավարտը չի նշանակում հրամանային տողի ավարտ:

Օրինակ

```
{
    temp = a;
    a = b;
    b = temp;
}
```

➢ Գործողություններ և օպերատորներ

Այն ամենը, ինչի արդյունքում ստացվում են արժեքներ, C++-ում կրում է գործողություն անվանումը: Օրինակ 3+2 գործողության մասին ասում են, որ այն վերադարձնում է 5 արժեք:

Օպերատորը - լիտերալ է, որը կոմպիլյատորին ստիպում է կատարել որոշ գործողություններ: **Օպերատորները** ազդում են **օպերանդների** վրա: **Օպերանդը** կարող է լինել ինչպես առանձին լիտերալ, այնպես և մի ամբողջ արտահայտություն: Առկա են երկու տիպի օպերատորներ վերագրման և մաթեմատիկական :

Վերագրման օպերատորը (=) թույլ է տալիս հավասարման նշանից ձախ գտնվող օպերանդի արժեքը փոխել նրանից աջ գտնվող արտահայտության արժեքով:

C++ -ում առկա 5 մաթեմատիկական օպերատորները բերված են աղյուսակ 1.4.1-ում:

Աղյուսակ 1.4.1

ՀՀ	Անվ.	Բացատրություն	Օրինակ , a=15, b=10
1	+	Գումարում	X = a + b ; պատ՝ X=25
2	-	Հանում	X = a – b ; պատ՝ X=5
3	*	Բազմապատկում	X = a * b ; պատ՝ X=150
4	/	Ամբողջ թիվ բաժանում	X = a / b ; պատ՝ X=1
5	%	Բաժանում ըստ մոդուլ 2-ի	X = a % b ; պատ՝ X=5

➢ Ինքրեմենտ և Դեքրեմենտ: Պրեֆիքս և պոստֆիքս

Մեծության աճը 1-ով անվանում են **ինքրիմենտ**, իսկ նվազումը 1-ով՝ **դեքրիմենտ**: Ինքրիմենտի օպերատորն է՝ **(++), իսկ** դեքրիմենտինը՝ **(--)**:

Օրինակ՝

```
x ++ ;          // Համարժեք է x = x + 1 ; արտահայտությանը
x  -- ;         // Համարժեք է x = x - 1 ; արտահայտությանը
```

Ինքրեմենտի և դեքրեմենտի օպերատորները կարող են դրվել ինչպես փոփոխականներից առաջ /այդ դեպքում կասենք որ ունենք **պրեֆիքսի** օպերատոր, օրինակ ++myAge/, այնպես էլ նրանցից հետո /այս դեպքում կասենք, որ ունենք **պոստֆիքսի** օպերատոր, օրինակ myAge++/: Պրեֆիքսի օպերատորը հաշվարկվում է նախքան վերագրումը, պոստֆիքսինը՝ վերագրումից հետո:

Դիտարկենք ստորև բերված ծրագիրը

Ծրագրային կոդ 4-1

```
1   #include <iostream.h>
2   int main()
3   {
4       int a, b ;
5       a=15;
6       b=a;
7       cout<<"b=a    a="<<a<<"\t"<<"b="<<b<<"\n";
8       b=a++;
9       cout<<"b=a++  a="<<a<<"\t"<<"b="<<b<<"\n";
10      b=a--;
11      cout<<"b=a--  a="<<a<<"\t"<<"b="<<b<<"\n";
12      b=++a;
13      cout<<"b=++a  a="<<a<<"\t"<<"b="<<b<<"\n";
14      b=--a;
15      cout<<"b=--a  a="<<a<<"\t"<<"b="<<b<<"\n";
16      return 0;
17  }
```

Թողարկման արդյունքը՝

```
b=a     a=15    b=15
b=a++   a=16    b=15
b=a--   a=15    b=16
b=++a   a=16    b=16
b=--a   a=15    b=15
```

Օպերատորների կատարման առաջնայնության խնդիրը լուծված է ճիշտ այնպես ինչպես սովորական հանրահաշվում: Հավասարազոր

առաջնայնության օպերատորների կատարման հաջորդականությունը իրականացվում է տողում ձախից աջ հանդիպման հաջորդականությամբ, բացառությամբ վերագրման օպերատորից, որի մեջ կատարման հաջորդականությունը աջից ձախ է: Գործողությունների կատարման հաջորդականությունը կարելի է փոփոխել գործածելով կլոր փակագծեր:

- **Համեմատման օպերատորներ**

Նախատեսված են մեծությունների հավասարությունը /կամ անհավասարությունը/ ճշտելու համար: Ցանկը բերված է աղ. 1.4-2-ում:

Աղյուսակ 1.4-2

ՀՀ	Անվանումը	Օպերատոր	Օրինակ	Վեր. մեծութ.
1	Հավասար է	==	100 == 105 50 == 50	False True
2	Հավասար չէ	!=	100 !=105 50 != 50	True False
3	Մեծ է	>	100 > 150 50 > 50	False False
4	Մեծ կամ հավասար է	>=	100 >= 150 50 >= 50	False True
5	Փոքր է	<	100 < 150 50 < 50	True False
6	Փոքր կամ հավասար է	<=	100 <= 150 50 <= 50	True True

-

```
if (պայման)
   արտահայտություն;
```

Պայմանական օպերատոր if

if օպերատորի պարզագույն տեսքն է՝

(պայման)-ը կարող է լինել կամայական արտահայտություն, սակայն սովորաբար այն **համեմատման օպերատոր է**: Եթե պայմանը վերադարձնում է **false** ապա հաջորդող օպերատորը բաց է թողնվում, **true** վերադարձնելիս՝ կատարվում է:

Օրինակ՝

```
if (X > Y)
   X = Y;
```

Այստեղ ստուգվում է (X>Y) պայմանը: Եթե X>Y,ապա X-ին վերագրվում է Y-ի արժեքը, հակառակ դեպքում վերագրում չի կատարվում:

Հիշենք, որ կարելի է մի շարք արտահայտություններ խմբավորել մեկ բլոկի մեջ: Այդ դեպքում, ըստ բերված օրինակի պայմանի կատարման դեպքում կկատարվեն բլոկում բերված բոլոր արտահայտությունները՝

Օրինակ

```
if (X > Y)
{
   cout<<"X>Y\n";
   X = Y;
   cout<<"And now X=Y\n";
}
```

նախ էկրան դուրս կբերվի X>Y արտահայտությունը, այնուհետև X-ին կվերագրվի Y-ի արժեքը, և էկրան է դուրս կբերվի And now X=Y արտահայտությունը:

- **Պայմանական օպերատոր if…else**

Այն if օպերատորի մեկ այլ գրելաձև է որը գրված ծրագիրը դարձնում է առավել հասկանալի:

```
if (պայման)
   արտահայտություն_1;
else
   արտահայտություն_2;
```

Այս դեպքում կրկին ստուգվում է **(պայման)**-ը, եթե այն բավարարվում է, ապա կատարվում է արտահայտություն_1 -ը, հակառակ դեպքում՝ արտահայտություն_2 -ը:

Օրինակ

```
if (x>y)
{
   x=y;
   cout<<"x  -in veragrvel e y mecutyan arjeq\n"
}
else
   cout<<"x>y paymany chi apahovvats\n"
cout<<"Katarvum e ankax paymani chisht kam sxal lineluc \n"
```

➢ Օպերատոր switch

Բազմակի ներդրված if/else կառուցվածքների գործածումը բարդացնում է ծրագրի կառուցվածքը, ուստի նպատակահարմար է գործածել **switch** օպերատորը: Այն ունի բերված է ստորև կառուցվածքը

```
switch (արտահայտություն)
    {
            case արժեք_1 :    օպերատոր;
            break;
            case արժեք_2 :    օպերատոր;
            break;
      ……..……
            case արժեք_n :    օպերատոր;
            break;
            default :         օպերատոր;
    }
```

switch օպերատորից հետո բերված կլոր փակագծերում կարող է լինել կամակայան արտահայտություն: (օպերատոր)-ի փոխարեն թույլատրվում է կամայական օպերատոր կամ արտահայտություն, կամ օպերատորների և արտահայտությունների հաջորդականություն, որի արդյունքը ամբողջ թիվ է /արգելվում է տրամաբանական գործողությունների կամ համեմատման արտահայտությունների գործածումը:

Օպերատորը աշխատում է հետևյալ կերպ՝ **(արտահայտություն)**-ը համեմատվում է **case** –երից հետո գրված արժեքների հետ, և ո՞ր արժեքի հետ իրականացվում է համընկնում, կատարվում են այդ տողում գրված արտահայտությունները և իրականացվում է անցում պայմանից դուրս: Եթե համընկնում չկա ոչ մի տողում, ապա կատարվում է **default** տողում գրված օպերատորները և կրկին պայմանից դուրս է գալիս: Եթե արտահայտությունների մեջ բացակայում է **break** հրամանը, ապա մի **case** - ի կատարումից հետո կշարունակվեն կատարել բոլոր մնացած **case** - երը, այդ թվում նաև **default**-ը: **break** - ը թույլ է տալիս բարեհաջող ելք առանց մյուս **case** - երի կատարման:

➢ Տրամաբանական օպերատորներ

Մի քանի պայմանների միաժամանակյա ստուգման համար անհրաժեշտ է գործածել տրամաբանական օպերատորներ /տես աղ աղ. 1.4.3/, որոնց ճշմարտացիության աղյուսակները բերված են աղ. 1.4.4 – 1.4.6-ում:

Աղյուսակ 1.4-3

ՀՀ	Օպերա-տոր	Սիմ-վոլ-	Օրինակ
1	ԵՎ	&&	արտ_1&& արտ_2
2	ԿԱՄ	||	արտ_1 || արտ_2
3	ՈՉ	!	! արտ

Օպերատոր ԵՎ *Աղյուսակ 1.4-4*

Արտ_1	Արտ_2	Արդյունքը
True	True	True
True	False	False
False	True	False
False	False	False

Արդյունքը ճշմարիտ է (true), եթե ճշմարիտ են (true) թե՛ արտահայտություն_1-ը և թե՛ արտահայտություն_2-ը:

Օպերատոր ԿԱՄ *Աղյուսակ 1.4-5*

Արտ_1	Արտ_2	Արդյունքը
True	True	True
True	False	True
False	True	True
False	False	False

Արդյունքը ճշմարիտ է (true), եթե կամ արտահայտություն_1-ը կամ արտահայտություն_2-ը ճշմարիտ են (true):

Օպերատոր ՈՉ *Աղյուսակ 1.4-6*

Արտահայտություն	Արդյունքը
True	False
False	True

Արդյունքը ճշմարիտ է (true), եթե արտահայտությունը կեղծ է (false է) և հակառակը:

Հիշենք, որ C++ լեզվում 0-ն համարժեք է false արժեքին, իսկ մնացած թվային արժեքները համարժեք են true մեծությանը, քանի որ բոլոր արտահայտությունները միշտ ունեն թվային արժեք, ուստի գործածական են ստորև բերված գրելաձևերը

§ 1-5 Ֆունկցիաներ

Ֆունկցիան ենթածրագիր է, որն աշխատում է տվյալների հետ և վերադարձնում է որոշակի մեծություն: Գոյություն ունեն երկու տիպի ֆունկցիաներ՝ օգտագործողի կողմից ձևակերպված՝ ստանդարտ և, կոմպիլյատորի փաթեթի հետ տրամադրված՝ ներդրված:

Գործածումից առաջ անհրաժեշտ է ֆունկցիաները **հայտարարել** և **սահմանել**: Ֆունկցիայի հայտարարումը անվանում են ***նախատիպ և այն*** ունի հետևյալ գրելաձևը

վերադարձի տիպ անուն (պարամ_1, պարամ_2,… պարամ_n);

Օրինակ՝

```
int myFunc(int someInt, float someFloat);
```

որը նշանակում է myFunc() ֆունկցիան որպես պարամետր ընդունում է 1 ամբողջ և 1 իրական արժեք և վերադարձնում է ամբողջ տիպի արժեք:

➢ Ֆունկցիաների սահմանումը

Իրականացվում է ստորև բերված սխեմայով

```
վերադարձի տիպ    անուն(պարամետր_1, պարամետր_2,… պարամետր_n)
    {

        // ֆունկցիայի մարմին

      return վերադարձվող մեծություն;
    }
```

Օրինակ՝

```
int Area (int length, int width)
   {
     ……
     return (length*width);
   }
```

➢ Ֆունկցիաների կանչը

Ֆունկցիայի կանչը կատարվում է հետևյալ սխեմայով՝

ֆունկցիա_անուն (արգումենտ_1, արգումենտ_2, …. արգումենտ_n);

Դիտարկենք ֆունկցիայի հայտարարման, սահմանման և կանչի մի օրինակ /տես 1.5.1 ծրագրային կոդը

Ծրագրային կոդ 1.5-1

```
1    #include <iostream.h>
2    #include "cube.h"
3    int Gumar(int, int);
4    int main()
5    {
6        int  a, b,  c;
7        c=Gunar(5,15);
8        cout<<”c=”<<c<<endl;
9        cout<<”Input a=”;
10       cin>>a;
11       cout<<” Input b=”;
12       cin>>b;
13       c=Gumar(a,b);
14       cout<<”c=”<<c<<endl;
```

```
15    return 0;
16    }
17
18    int Gumar (int x, int y )
19          {
20              nеturn x+y;
21    }
```

Թողարկման արդյունքը՝

```
c=20
Input a=30
Input b=40
c=70
```

Բերված օրինակում դիտարկված են 2 ֆունկցիաներ՝ main() գլխավոր ֆունկցիան և Gumar()ֆունկցիան:

Ինչպես տեսնում ենք, Gumar() ֆունկցիան ստանում է 2 ամբողջ տիպի պարամետրեր, վերադարձնում՝ ամբողջ տիպի մեծություն: 7 և 13-րդ տողերում իրականացվում է այդ ֆունկցիայի կանչը հետևյալ տարբերակներով: 7-րդ տողում՝ ֆունկցիայի կանչի պահին կատարվում է անցում 18-րդ տողին: Ֆունկցիայի x, y արգումենտներին վերագրվում են համապատասխանաբար 5 և 15 արժեքները և նա 20-րդ տողում հաշվարկում է և վերադարձնում է x+y ՝ 5+15 արժեքը, որն էլ վերագրվում է c փոփոխականին:

13-րդ տողում ամեն ինչ նույնն է , այն տարբերությամբ, որ այս դեպքում Gumar() ֆունկցիային վերագրած արժեքները ներկայացված են փոփոխականների տեսքով, որոնք նախապես արժեքներ ստանում են կոնսուլային ռեժիմում 10 և 12-րդ տողերում: Մնացած ամեն ինչը նույնն է , ինչ նախորդ դեպքում:

Ֆունկցիաների ներսում հայտարարված փոփոխականները, ինչպես նաև ֆունկցիաներին փոխանցված պարամետրերը անվանում են տեղային կամ լոկալ: Նրանք գործում են տվյալ ֆունկցիայի սահմաններում և նրա սահմաններից դուրս հասանելի չեն: Ֆունկցիաների ներսում, բլոկներում, հայտարարված փոփոխականների տեսանելիության տիրույթը սահմանափակվում է տվյալ բլոկի սահմաններում: Իսկ բոլոր ֆունկցիաներից /այդ թվում նաև main()-ից/ դուրս հայտարարված փոփոխականները հասանելի են կամայական ֆունկցիաներից և կոչվում են ***գլոբալ փոփոխականներ***:

➢ Ֆունկցիաների գերբեռնումը

Միևնույն անվանումով տարբեր ֆունկցիաների ստեղծումը անվանում են ***ֆունկցիաների գերբեռնում***: Գերբեռնված ֆունկցիաները

միմյանցից պետք է տարբերվեն կամ նրանց փոխանցվող պարամետրերի քանակով, կամ նրանց փոխանցվող պարամետրերի տիպերով կամ էլ վերադարձվող մեծության տիպով կամ էլ թե մեկով և թե մյուսով միաժամանակ: Ստորև բերված օրինակում myFunction() ֆունկցիան գերբեռնված է 4 անգամ: Իսկ թե որ ֆունկցիան կաշխատի նրա կանչի ժամանակ պայմանավորված է նարն փոխանցված արգու-մենտների տիպով, քանակով կամ ֆունկցիայից սպասվող մեծության տիպով:

Օրինակ

```
int myFunction(int, int);
int myFunction(long, long);
int myFunction(int, int, long);
long myFunction(int, long);
```

Սինենույն արգումենտների քանակի և տիպերի դեպքում չի կարելի գերբեռնել 2 այնպիսի ֆունկցիաներ, որոնք կվերադարձնեն տարբեր տիպի մեծություններ: Օրինակ

```
int myFunction(int, int);
void myFunction(int, int);
```

Օրինակը սխալ է: Կոմպիլյատորը չի կարող կողմնորոշվել թե որ ֆունկցիան պետք է աշխատի կանչի ժամանակ և ինչ մեծություն վերադարձնի /int թե void/, երբ նրան փոխանցվեն 2 ամբողջ տիպի մեծություններ:

ԳԼՈՒԽ 2 ՕԲՅԵԿՏԱՅԻՆ ԿՈՂՄՆՈՐՈՇՄԱՆ ԾՐԱԳՐԱՎՈՐՄԱՆ ՏԱՐԲԵՐԸ

§ 2-1 Բազային կլասներ

Կլասը փոփոխականների և նրանց հետ կապված ֆունկցիաների հավաքածու է: Կլասի փոփոխականները անվանում են **փոփոխական-անդամներ,** իսկ ֆունկցիաները՝ **ֆունկցիա-անդամներ** /կամ **մեթոդներ**/:

➢ **Կլասների հայտարարումը**

Իրականացվում է ըստ ստորև բերված սխեմայի.

```
class անվանում
{
      փոփոխական_անդամ_1;
      …
      փոփոխական_անդամ_n;

      ֆունկցիա_անդամ_1;
      …
      ֆունկցիա_անդամ_n;
};
```

Օրինակ `

```
class Man
{
    unsigned int itsAge;          // unsigned int տիպի itsAge փոփոխական-անդամ
    void Sing();                  // Sing() ֆունկիցիա-անդամ /կամ մեթոդ/
};
```

Կլասի հայտարարումը հիշողություն չի պահեստավորում:

➢ **Օբյեկտների հայտարարումը**

Օբյեկտը կլասի առանձին օրինակ է, որի հայտարարումը իրականացվում է ըստ ստորև բերված սխեմայի

կլաս_անվանում օբյեկտ_անվանում;

Օրինակ՝

```
    Man Harry;          // Man տիպի Harry օբյեկտ
```

Օբյեկտի ստեղծումից հետո կարելի է դիմել նրա փոփոխական անդամներին և մեթոդներին, որը իրականացվում է ըստ բերած սխեմայի՝

օբյեկտ.փոփոխական_անդամ

օբյեկտ.ֆունկցիա_անդամ (պարամ_1, պարամ_2 … , պարամ_n);

Օրինակ՝

```
Harry.itsAge = 50;          //Harry օբյեկտի itsAge անդամին վերագրել 50 արժեք
Harry.Sing();               //Կանչել Harry օբյեկտի համար Sing() մեթոդը
```

Չի կարելի դիմել օբյեկտի այն անդամին, որը հայտարարված չէ այդ օբյեկտի կլասի համար:
Օրինակ

```
Man Harry;                  //  Ստեղծել Harry օբյեկտը
Harry.Speak();              // Harry օբյեկտի համար աշխատացնել Speak() մեթոդը
```

Օրինակը սխալ է: Man կլասում բացակայում է Speak() մեթոդը

- **Կլասի անդամների հասանելիության ռեժիմները**

Կլասի անդամները կարող են հայտարարվել և լինել public- բաց կամ հասանելի, private- փակ կամ անհասանելի, protected- պաշտպանված: Ըստ լռելիության կլասի անդամները փակ են /private/ և նրանց դիմել ուղղակիորեն հնարավոր չէ: Սակայն ***public*** բառի շնորհիվ կարելի է կլասի անդամները հայտարարել բաց, և դիմել նրանցից յուրաքանչյուրին: Օրինակ.

Ծրագրային կոդ 2.1-1

```
1    #include <iostream.h>
2    class Man
3    {
4        public:
5            int itsAge;
6            int itsWeight;
7        };
8    int main()
9    {
10       Man Harry;
11       Harry.itsAge=5;
12       cout<<"Harry is a Man, who is  ";
13       cout<<Harry.itsAge<<"years old\n";
14       return 0;
15   }
```

Ծրագրակման արդյունքը `

```
Harry is a Man, who is 5 years old
```

Սովորաբար կլասներում **փոփոխական անդամները հայտարարվում են փակ, իսկ մեթոդները՝ բաց**:

➢ Կլասի մեթոդների սահմանումը

Կլասներում հայտարարված բոլոր մեթոդները պետք է նաև սահմանվեն: Սահմանումը նման է սովորական ֆունկցիաների սահմանմանը, այն տարբերությամբ, որ այս դեպքում ֆունկցիայի անվանման և վերադարձվող մեծության միջև դրվում է կլասի անվանումը որին հաջորդում են երկու վերջակետեր (::) /Օրինակ -1/: Սակայն մեթոդների սահմանումները կարելի է նաև գրել նրանց հայտարարման մեջ: /Օրինակ -2/:

Օրինակ- 1

```
1  class Man
2   {
3       public:
4            void Sing();
5  };
6  void Man::Sing()
7  {
8       cout<<"La la la   La la la \n";
9  }
```

/Օրինակ -2/

```
1  class Man
2  {
3       public:
4  void Sing() {cout<<"La la la   La la la \n"; };
5  };
```

➢ Կոնստրուկտոր և դեստրուկտոր

Կոնստրուկտորը կլասի մեթոդ է, որի անունը համընկնում է կլասի անվան հետ և արժեք չի վերադարձնում: Այն նախատեսված է օյեկտների ստեղծման և ինիցիալիզացման համար:

Դեստրուկտորը ևս ունի կլասի անվանում, սակայն դրանից առաջ դրվում է (~) տիլդայի նշան: Այն որևէ արգումենտ չի ընդունում, ոչ մի արժեք չի վերադարձնում և նախատեսված է հիշողությունից օբյեկտների հեռացման համար:

Եթե կլասի կոնստրուկտորը և դեստրուկտորը որևիցե արգումենտ չեն ընդունում և ոչ մի գործողություն չեն կատարում, նրանց անվանում են **ստանդարտ կոնստրուկտոր կամ դեստրուկտոր**:

Եթե բացահայտ ձևով չի հայտարարվել կլասի կոնստրուկտորը կամ դեստրուկտորը, ապա **դա անում է կոմպիլյատորը**: Օրինակ

Ծրագրային կոդ 2.1-2

```
1     #include <iostream.h>
2     class Man
3     {
4     public:
```

```
5         Man(int initialAge) { itsAge=initialAge;};

6         ~Man() {};
7         int GetAge() { return itsAge;};
8     private:
9         int itsAge;
10    };
11    int main ()
12    {
13        Man Harry(5);
14        cout<<"Harry is a Man who is   ";
15        cout<<Harry.GetAge()<<"  years old\n";
16        return 0;
17    }
```

Ծրագրակման արդյունքը՝

```
Harry is a Man who is   5 years old
```

➢ Բարեկամական ֆունկցիաներ

Հիշենք որ, կլասի փակ փոփոխական-անդամներին կարելի է դիմել միայն նրանում առկա բաց ֆունկցիա-անդամներից: Սակայն երբեմն անհրաժեշտություն է առաջանում նրանց դիմելու ծրագրի այնպիսի ֆունկցիաներից, որոնք տվյալ կլասի անդամ չեն: Ասվածը իրականացվում է բարեկամական ֆունկցիաների օգնությամբ, որոնց նախապես պետք է հայտարարել այն կլասի մեջ, որի փակ անդամներին պետք է դիմել հետագայում:

Օրինակ՝

```
class C1
{
    public:
        friend void f_funct();
}
```

այս օրինակում f_funct() ֆունկցիան C1 կլասի համար հայտարարված է բարեկամական: Բարեկամական ֆունկցիաների ներմուծման հիմնական պատճառը կայանում է 2 և ավելի տարբեր կլասների փակ անդամներին միևնույն ֆունկցիայից դիմելու անհրաժեշտության մեջ: Բնական է, որ այդ դեպքում տվյալ ֆունկցիան պետք է բարեկամական հայտարարվի բոլոր այն կլասների համար, որոնց փակ անդամներին հետագայում պետք է դիմի տվյալ ֆունկցիան: Բերենք մի օրինակ

Ծրագրային կոդ 2. 1-3

```
1   #include <iostream.h>
2   class car;
3   class lorry
4   {
5       int itsWeight;
6       int itsHorse_Force;
7   public:
8       // compare() բարեկամական ֆունկցիայի հայտարարումը
9       friend void compare(lorry l, car c);
10      void SetWeight(int weight){itsWeight=weight;};
11      int GetWeight(){return itsWeight;};
12  };
13  class car
14  {
15      int itsWeight;
16      int itsHorse_Force;
17  public:
18      // compare() բարեկամական ֆունկցիայի հայտարարումը
19      friend void  compare(lorry l, car c);
20      void SetWeight(int weight){itsWeight=weight;};
21      int GetWeight(){return itsWeight;};
22  };
23  // compare() բարեկամական ֆունկցիայի սահմանումը
24  void compare(lorry l, car c)
25  {
26      // Համեմատվում են lorry և  car կլասների itsWeight փակ անդամները
27      if (l.itsWeight > c.itsWeight)
28          cout<<"Lorry Weight is  >  Car Weight\n";
29      else
30          cout<<"Lorry Weight  ia < Car Weight\n";
31  };
32  int main()
33  {
34      lorry ob_l1, ob_l2;
35      car ob_c1,ob_c2;
36      ob_l1.SetWeight(2500);
37      ob_c1.SetWeight(1500);
38      ob_l2.SetWeight(1400);
39      ob_c2.SetWeight(1500);
40      compare(ob_l1,ob_c1);
41      compare(ob_l2,ob_c2);
42   return 0;
43  }
```

```
Lorry Weight is  >  Car Weight
Lorry Weight is  <  Car Weight
```

Այս օրինակում հայտարարված են երկու՝ lorry և car կլասները: Նրանցում հայտարարված է compare() բարեկամական ֆունկցիան, որը որոշված չէ այդ կլասներից և ոչ մեկում: Այդ ֆունկցիայի ներսում իրականացված է ուղղակի դիմում այդ 2 կլասների ՓԱԿ փոփոխական անդամներին, չնայած որ այն ՈՐԵՎԷ կլասի մեթոդ չէ: Հիմնական ծրագրում ստեղծվուծ են այդ կլասների հիմքի վրա 2-ական օբյեկտներ, և compare() ֆունկցիայի միջոցով համեմատվում են այդ կլասների itsWeight փոփոխական անդամները:

Եթե որևէ կլասի սահմանման ժամանակ, նրանում հայտարարված բոլոր ֆունկցիա անդամները բարեկամական են մեկ այլ կլասի համար, ապա կարելի է համարել, որ տվյալ կլասը բարեկամական կլաս է մյուս կլասի համար:

➢ Գերբեռնված ֆունկցիա-անդամներ

Ֆունկցիաների գերբեռնման անալոգիայով կարելի է գերբեռնել նաև կլասների ֆունկցիա-անդամները՝ այդ թվում նաև կոնստրուկտորները: Դիտարկենք 2.1-4 ծրագրային կոդում բերված օրինակը:

Ծրագրային կոդ 2. 1-4

```
1   #include<iostream.h>
2   class Man
3   {
4   public:
5       Man();
6       Man(int age, int weight);
7       ~Man(){cout<< "This is a destructor...\n";};
8   private:
9       int Age;
10      int Weight;
11  };
12  Man::Man()
13  {
14      cout<< "This is a standart constructor ...\n";
15      Age = 7;
16      Weight = 10;
17  };
18  Man::Man(int age, int weight)
```

```
19    {
20        cout << "This is not standart constructor...\n";
21        Age = age;
22        Weight = weight;
23    };
24    int main()
25    {
26        Man Harry;                          // Կանչվում է Man() կոնստրուկտրը
27        Man Murka(12,33);                   // Կանչվում է Man(int, int) կոնստրուկտրը
28    return 0;
29    }
```

Թողարկման արդյունքը՝

```
This is a standart constructor ...
This is not standart constructor...
This is a destructor...
This is a destructor...
```

§ 2 -2 Ցիկլեր

Երբ խնդիրների լուծման ժամանակ պահանջվում է մի շարք գործողությունների անհրաժեշտ քանակությամբ կրկնում, գործածվում են ռեկուրսիվ կամ իտեռատիվ ալգորիթմներ, որոնք էլ իրենց հերթին բերում են ցիկլերի կազմակերպման խնդրին:

➢ Առանց պայմանի անցման օպերատոր / օպերատոր goto/

Ծրագրի մի հատվածից մեկ այլ հատված, առանց պայմանի, անցում կարելի է իրականացնել պիտակների /label/ և **goto** օպերատորի օգնությամբ: Պիտակը դա իդենտիֆիկատոր է, որից հետո դրվում է վերջակետ (:): Պիտակը դրվում է այն օպերատորից առաջ, որին պետք է փոխանցվի ղեկավարումը, իսկ անցման համար գործածվում է **goto** օպերատորը: Օրինակ

Ծրագրային կոդ 2.2-1

```
1   #include <iostream.h>
2   int main()
3   {
4       int counter=0;
5   loop :    counter++;
6       cout<<"counter="<< counter<<"\n";
7       if(counter<5)
8       goto loop;
9       cout<<"Complete counter="<< counter<<". \n";
10      return 0;
```

```
11    }
```

Թողարկման արդյունքը՝

```
Counter=1
Counter=2
Counter=3
Counter=4
Counter=5
Complete counter=5.
```

goto օպերատորի օգնությամբ կարելի է կազմակերպել ցիկլ ինչպես վերը բերված օրինակում, սակայն ցիկլերի կազմակերպման համար նպատակահարմար են հետևյալ 3 օպերատորները՝ for, while և do…while:

➢ Ցիկլերի կազմակերպումը while օպերատորի օգնությամբ

Այդ օպերատորը ունի հետևյալ տեսքը

```
while (պայման)
{
        ցիկլի մարմին
}
```

և աշխատում է հետևյալ տրամաբանությամբ, ստուգվում է (պայմանը) և քանի դեռ այն բավարարված է, կատարվում են բոլոր այն օպերատորները, որոնք գրված են ցիկլի մարմնում: Երբ (պայմանը) դադարում է գործելուց, ծրագրի կատարումը փոխանցվում է նրա մարմնից դուրս:

Նախորդ ծրագիրը ներկայացնենք այս օպերատորի օգնությամբ: Թողարկման արդյունքը նույնն է :Տես ծրագրային կոդ 2.2-2 –ը:

Ծրագրային կոդ 2.2-2

```
1     #include <iostream.h>
2     int main()
3     {
4          int counter=0;
5           while(counter<5)
6                  {
7                    counter++;
8                    cout<<”counter=”<< counter<<”\n”;
9                  }
10         cout<<”Complete counter=”<< counter<<”. \n”;
11         return 0;
```

```
12  }
```

- **Ցիկլերի կազմակերպումը do…while օպերատորի օգնությամբ**

Այդ օպերատորը ունի հետևյալ տեսքը

```
do
{
      ցիկլի մարմին
} while (պայման)
```

Այն աշխատում է հետևյալ տրամաբանությամբ: Կատարվում են բոլոր այն օպերատորները, որոնք գրված են ցիկլի մարմնում, այնուհետև ստուգվում է (պայման)-ը: Եթե այն բավարարվում է, իրականացվում է անցում ցիկլի սկիզբ և կրկին կատարվում են **do** –ից հետո գրված բլոկի օպերատորները՝ այլ խոսքով ցիկլի մարմնին: Երբ (պայման)-ը դադարում է գործելուց. ծրագրի կատարումը փոխանցվում է նրան հաջորդող տողին:

Նուն ծրագիրը ներկայացնենք այս օպերատորի օգնությամբ: Թողարկման արդյունքը կրկին նույնն է : Տե՛ս ծրագրային կոդ 2.2-3 ը:

Ծրագրային կոդ 2.2-3

```
1     #include <iostream.h>
2     int main()
3     {
4          int counter=0;
5          do
6          {
7                counter++;
8                cout<<"counter="<< counter<<"\n";
9          } while   (counter<5);
10         cout<<"Complete counter="<< counter<<". \n";
11         return 0;
12    }
```

- **Ցիկլերի կազմակերպումը for օպերատորի օգնությամբ**

Այդ օպերատորը ունի հետևյալ տեսքը

```
for (արտ_1 ; արտ_2 ; արտ_3)
{
      ցիկլի մարմին
}
```

որտեղ (արտ_1)-ը սահմանում է ցիկլի հաշվի սկզբնական արժեքը, որը սովորաբար ամբողջ թիվ է, հայտարարվում և ինիցիլիզացվում է հենց for ցիկլի մեջ: (արտ_2)-ը՝ ցիկլի շարունակման պայմանն է, իսկ (արտ_3)-ը սահմանվում է ցիկլի քայլը /լռելիության դեպքում այն հավասար է 1-ի/: Ինչպես արտ_1-ում այնպես և արտ_3-ում կարելի է գործածել կամայական արտահայտություն: արտ_2-ը անպայման պետք է վերադարձնի տրամաբանական մեծություն: Այժմ նույն ծրագիրը փորձենք ներկայացնել for օպերատորի օգնությամբ և համոզվենք, որ արդյունքը նույնն է, բացառությամբ վերջին տողից: Այս դեպքում էկրանին է հայտնվում

Complete counter=6. արտահայտությունը Complete counter=5 –ի փոխարեն,

քանի որ այս օպերատորի գործածման դեպքում հաշվիչի աճը և պայմանի ստուգումը իրականացվում է նույն տողում և օպերատորից դուրս այն արդեն ստացել է 6 արժեք

Ծրագրային կոդ 2.2-4

```
1       #include <iostream.h>
2       int main()
3       {
4           int counter;
5           for( counter=1; counter<=5; counter++)
6           {
7                   cout<<"counter="<< counter<<"\n";
8           }
9           cout<<"Complete counter="<< counter<<". \n";
10          return 0;
11      }
```

for օպերատորում կարող է բացակայել կամայական պարամետր կամ էլ նույնիսկ բոլորը միասին: Բացակա պարամետրը անվանում են զրոյական և նրանից հետո միևնույնն է դրվում է կետ-ստորակետ:

§ 2-3 Ցուցիչներ

Ցուցիչը փոփոխական է, որի մեջ պահվում է հասցե: Մեքենայական հիշողության մեջ պահպանվող յուրաքանչյուր փոփոխական ունի հասցե, որը կարելի է իմանալ (&)-հասցեի օպերատորի օգնությամբ: Տես 2.3-1 ծրագրային կոդը

Ծրագրային կոդ 2.3-1

```
1    #include <iostream.h>
2    int main()
3    {
4        int A;
5        A=15;
6        cout<<"I am variable   A . \n";
7       cout<<"A="<<A<<"\n";
8       cout<<"My address is   " <<&A<<"\n";
9    return 0;
10   }
```

Թողարկման արդյունքը՝

```
I am variable   A .
A=15
My address is   0x0012FF7C
```

- **Ցուցիչների սահմանումը**

Իրականացվում է ստորև բերված սխեմայով՝

տիպ ցուցչի_անվանում;

որտեղ ***տիպը*** դա ցուցչի տիպն է, որը համապատասխանում է այն օբյեկտի տիպին, որի հասցեն հետագայում պետք է պահպանվի ***ցուցչի_անվանում*** անվանումով ցուցչի մեջ: Ցուցչի անվանումը կարող է լինել կամայական իդենտիֆիկատոր որին նախորդում է (*) սիմվոլը:

Ցուցիչը դա առանձին փոփոխականի տիպ է: Եթե նախապես հայտնի չէ, թե ինչպիսի հասցե պետք է պահվի տվյալ ցուցչի մեջ, ապա նրան պետք է վերագրել 0 արժեք /այդպիսի ցուցիչներին անվանում են դատարկ կամ 0-ական/, հակառակ դեպքում դա կարող է հետագայում գլխացավանք առաջացնել:

Օրինակ:

```
int *pAge = 0;
```

Այս օրինակում pAge –ը 0-ական ցուցիչ է, որին հետագայում կարելի է վերագրել որևէ ամբողջ տիպի փոփոխականի հասցե:

Օրինակ՝

```
int howOld = 50 ;   //Հայտարարվում է int տիպի howOld փոփոխականը
int *pAge = 0;      //Հայտարարվում է int տիպի pAge 0-ական ցուցիչը
pAge= &howOld;  //howOld փոփոխականի հասցեն վերագրվում է pAge ցուցիչին
```

Կարելի է ցուցիչի հայտարարումը և նրան որևէ փոփոխականի հասցեի վերագրումը միավորել մեկ տողում

Օրինակ:

```
int howOld = 50 ;          // Հայտարարվում է փոփոխական
int *pAge = &howOld;  // Հայտարարվում է ցուցիչ howOld փոփոխականի համար
```

Ցուցիչները նախատեսվախ են

- Հիշողության ազատ տարածքներում տվյալների բաշխման և գործածման համար,
- Կլասներում ֆունկցիաների և տվյալների հասանելիության համար,
- Ֆունկցիաներին պարամետրեր փոխանցելու համար:

➢ Օպերատոր new

Նախատեսված է դինամիկ բաշխվածության տիրույթում հիշողություն զբաղեցնելու համար: Այն իրականացվում է հետևյալ սխեմայով

new օբյեկտ_տիպ

Արդյունքում վերադարձվում է հատկացված բջջի հասցեն, որը պետք է վերագրել որևէ ցուցչի, պահել նրա մեջ

Օրինակ՝

```
unsigned short int *pPointer = new unsigned short int;
```

Դինամիկ հիշողության տարածքում առանձնացվում է 2 բայթ հիշողություն և վերադարձրված հասցեն վերագրվում է pPointer ցուցչին: Այժմ նրանում կարելի է թիվ գրանցել հետևյալ կերպ:

```
*pPointer = 73;
```

➢ Օպերատոր delete

Դինամիկ հիշողությունը պետք է ազատվի ծրագրավորողի կողմից, անհրաժեշտության վերացման հետ մեկտեղ: Դա իրականացվում է **delete** օպերատորի օգնությամբ, որին պարզապես հետևում է ցուցչի անվանումը: Այսպես վերևում քննարկված pPointer ցուցչի օգնությամբ զբաղեցված հիշողությունը ազատվում է

```
delete pPointer;
```

հրամանով:

Իրականում կատարվում է ոչ թե ցուցչի հեռացում, այլ դինամիկ հիշողության ազատում այն հասցեով, որը գրանցված էր տվյալ ցուցչի մեջ: Հիշողության հասցեի ազատումից հետո ցուցչի մեջ կարելի է պահել այլ հասցե: Դիտարկենք 2.3-2 ծրագրային կոդը:

Ծրագրային կոդ 2.3-2

```
1    #include<iostream.h>
2    int main()
3    {
4    int *pHeap = new int;
5    *pHeap = 7;
6    cout << “*pHeap =\t” << *pHeap << “\n”;
7    delete pHeap;
8    pHeap = new int;
9    *pHeap = 9;
10   cout << “*pHeap =\t” << *pHeap << “\n”;
11   delete pHeap;
12   return 0;
13   }
```

Թողարկման արդյունքը `

```
*pHeap =   7
*pHeap =   9
```

Հիշեք, որ չի կարելի առանց դինամիկ հիշողության ազատման նույն ցուցիչը օգտագործել այլ տարածք զբաղեցնելու համար, այլապես տեղի կունենա հիշողության կորուստ, քանի որ նախկին տարածքին դիմելու հնարավորություն չի լինի /ոչ ազատելու և ոչ էլ գործածելու իմաստով/:

§ 2-4 Հղումներ

Հղումը դա կեղծանուն է, որը հայտարարման ժամանակ սկզբնական արժեք է ստանում մեկ այլ օբյեկտի՝ հասցեատիրոջ, անվանումով: Այդ պահից սկսած հղումը տվյալ օբյեկտի այլընտրանքային անվանումն է, և այն ամենը, ինչ կատարվում է հղման հետ նույնն է, թե կատարվում է տվյալ օբյեկտի հետ:

- **Հղումների հայտարարումը**

Իրականացվում է հետևյալ կառուցվածքով

օբյեկտի_տիպը & հղման_անվանում

որին պետք է անպայման վերագրված լինի որևէ օբյեկտ և հրամանը ավարտվի կետ ստորակետով (;):

Օրինակ

```
int &rSomeRef = someInt;
```

որտեղ rSomeRef - ը someInt ամբողջ տիպի (int) օբյեկտի հղումն է:

Այստեղ հասցեատեր է համարվում someInt օբյեկտը: Որպեսզի ասվածը պարզ լինի, քննարկենք ստորև բերված օրինակը

Ծրագրային կոդ 2.4-1

```
1   #include <iostream.h>
2   int main()
3   {
4       int intOne;
5       int &rSomeRef = intone;
6       intOne =  8;
7       cout << "intOne \t\t = " << intOne<<endl;
8        cout << "rSomeRef\t = " <<rSomeRef << endl;
9       rSomeRef=15;
10      cout << "intOne \t\t = " << intOne<<endl;
11      cout << "rSomeRef\t = " <<rSomeRef << endl;
12      cout << "&intOne \t\t = " << &intOne<<endl;
13      cout << "&rSomeRef\t = " << &rSomeRef << endl;
14  return 0;
15  }
```

Թողարկման արդյունքը

```
intOne          = 8
rSomeRef        = 8
intOne          = 15
rSomeRef        = 15
&intOne         =  0x0012FF7C
&rSomeRef       =  0x0012FF7C
```

Հղումը չի կարելի վերաձևակել:

➢ Հղումները որպես ֆունկցիայի արգումենտներ

Դիտարկենք մի այսպիսի օրինակ

Տարբերակ 1 ***Ծրագրային կոդ 2.4-2***

```
1   #include<iostream.h>
2   void Swap(int x, int y)
3   {
4       int temp;
5       cout<<"Swap Function:  before swap\tx="<<x<<"\ty="<< y<<endl;
6       temp=x;
7       x=y;
8       y=temp;
9       cout<<"Swap Function:  after swap\tx="<<x<<"\ty="<< y<<endl;
10  }
11  int main()
12  {
13      int x=5,y=10;
14      cout<<"Main Function:  before swap\tx="<<x<<"\ty="<< y<<endl;
```

```
15        Swap(x,y);

16        cout<<"Main Function:  after swap\tx="<<x<<"\ty="<< y<<endl;
17    return 0;
18    }
```

Թողարկման արդյունքը՝

```
Main  Function:  before swap  x=5     y=10
Swap Function:  before swap  x=5     y=10
Swap Function:  after    swap  x=10   y=5
Main  Function:after     swap  x=5     y=10
```

Այստեղ Swap() ֆունկցիային փոխանցվում են x և y փոփոխականների արժեքները: Նրա ներսում այդ երկու փոփոխականների արժեքները փոխվում են տեղերով, սակայն main() ֆունկցիա վերադառնալուց հետո դրանք չեն պահպանվում: Պատճառը այն է որ Swap() ֆունկցիային փոխանցվում են ոչ թե արգումենտի իրական արժեքները, այլ նրանց պատճենները: Այս պրոբլեմը լուծվում է, երբ ֆունկցիաներին որպես արգումենտ փոխանցվում են նրանց հասցեները/ հղումների կամ ցուցիչների տեսքով: Ասվածը դիտարկենք օրինակների վրա

Տարբերակ 2 : *Ծրագրային կոդ 2.4-3*

```
1    #include<iostream.h>
2    void Swap(int *px, int *py)
3    {
4         int temp;
5         cout<<"Swap Function:  before swap\tx="<<*px<<"\ty="<< *py<<endl;
6         temp=*px;
7         *px=*py;
8         *py=temp;
9         cout<<"Swap Function:  after swap\tx="<<*px<<"\ty="<< *py<<endl;
10   }
11   int main()
12   {
13        int x=5,y=10;
14        cout<<"Main Function:  before swap\tx="<<x<<"\ty="<< y<<endl;
15        Swap(&x,&y);
16        cout<<"Main Function:  after swap\tx="<<x<<"\ty="<< y<<endl;
17   return 0;
18   }
```

Թողարկման արդյունքը՝

```
Main  Function:  before swap  x=5     y=10
Swap Function:  before swap  x=5     y=10
Swap Function:  after swap  x=10   y=5
Main  Function:  after swap  x=10    y=5
```

Այս օրինակում Swap() ֆունկցիային փոխանցվում են x և y փոփոխականների հասցեները, իսկ դա նշանակում է, որ այդ ֆունկցիայի հայտարարման մեջ պետք է հայտարարվեն երկու ամբողջ տիպի ցուցիչներ, քանզի նրան փոխանցվում են երկու ամբողջ տիպերի հասցեներ: Արդյունքում Swap() ֆունկցիայի մեջ տեղերով փոխվում են x և y փոփոխականների հասցեների պարունակությունները, իսկ դա նշանակում է, որ հենց այդ փոփոխականների արժեքները կփոխվեն տեղերով և ոչ թե նրանց պատճենները: Եվ main() ֆունկցիային վերադառնալուց կպահպանեն այդ հասցեների պարունակությունները, այսինքն՝ արժեքները կփոխվեն տեղերով

Տարբերակ 3 : *Ծրագրային կոդ 2.4-4*

```
1   #include<iostream.h>
2   void Swap(int &x, int & y)
3   {
4        int temp;
5        cout<<"Swap Function:  before swap\tx="<<x<<"\ty="<< y<<endl;
6        temp=x;
7        x=y;
8        y=temp;
9        cout<<"Swap Function:  after swap\tx="<<x<<"\ty="<< y<<endl;
10  }
11  int main()
12  {
13       int x=5,y=10;
14       cout<<"Main Function:  before swap\tx="<<x<<"\ty="<< y<<endl;
15       Swap(x,y);
16       cout<<"Main Function:  after swap\tx="<<x<<"\ty="<< y<<endl;
17  return 0;
18  }
```

Թողարկման արդյունքը՝

```
Main  Function:  before swap  x=5     y=10
Swap Function:  before swap  x=5     y=10
Swap Function:  after swap  x=10   y=5
Main  Function:  after swap  x=10     y=5
```

Այս օրինակում ցուցիչների փոխարեն գործածվում են հղումներ: Swap() ֆունկցիային փոխանցվում են x և y փոփոխականները, որոնք ստանում են &x և &y կեղծանունները: Իսկ դա նշանակում է, որ այն ամենը, ինչ կկատարվի այդ կեղծանունների հետ, նույնն է, թե կատարվում է այդ փոփոխականների հետ: Արդյունքում Swap() ֆունկցիայի մեջ տեղերով փոխվում են x և y փոփոխականների հասցեների պարունակությունները, իսկ դա նշանակում է, որ հենց այդ փոփոխականների արժեքները կփոխվեն տեղերով և ոչ թե նրանց պատճենները: Եվ ինչպես նախորդ տարբերակում, այս դեպքում նս main() ֆունկցիային վերադառնալուց կպահպանեն այդ հասցեների պարունակությունները, այսինքն՝ արժեքները կփոխվեն տեղերով: Իր բնույթով այս ծրագիրը համարժեք է նախորդ տարբերակին, սակայն այն ունի ավելի պարզ գրելաձև:

Ինչպես տեսնում ենք ֆունկցիաներին, որպես արգումենտ ցուցիչ կամ հղում փոխանցելուց, հնարավոր է ոչ միայն ելակետային տվյալների փոփոխում, այլև այդպիսի ֆունկցիաներից կարելի է ստանալ մեկից ավելի արժեքներ:

§ 2-5 Զանգվածներ

Զանգվածը համակարգչի հիշողության մեջ միևնույն տիպի տվյալների հավաքածու է: Տվյալների յուրաքանչյուր միավոր անվանում են զանգվածի էլեմենտ:

Զանգվածի հայտարարման համար անհրաժշտ է նշել նրա տիպը, անունը և չափը: Օրինակ

```
long int  LongArray[20];
```

որը նշանակում է. հայտարարել LongArray անվանումով զանգված, կազմված 20 հատ **long int** տիպի էլեմենտներից:

Զանգվածներին կարելի է վերագրել կամայական անուններ, սակայն այն պետք է տարբերվի այլ զանգվածների, փոփոխականների կամ հաստատունների անուններից, ինչպես նաև չպետք է լինի պահեստավորված բառ:

Զանգվածի էլեմենտների հասցեավորումը որոշվում է 1-ին էլեմենտի հասցեի նկատմամբ, որը համարվում է զրոյական: Բերված օրինակում LongArray-ի առաջին էլեմենտին կարելի է դիմել այսպես՝ LongArray[0]:

Զանգվածների էլեմենտներին բազային տիպի արժեքներ կարելի է վերագրել նրանց հայտարարման ժամանակ հետևյալ տեխնոլոգիայով՝,

անվանումից հետո դրվում է հավասարման նշան, որից հետո ձևավոր փակագծերի մեջ միմյանցից ստորակետով բաժանվում են այն մեծությունները, որոնք վերագրվում են վերջինիս էլեմենտներին: Օրինակ՝

```
int IntArray[5] = {10, 20, 30, 40, 50};
```

Գրառումը նշանակում է, հայտարարել int տիպի IntArray զանգվածը, և նրա 1-ին էլեմենտին վերագրել 10 արժեք, 2-րդին՝ 20 և այլն:

Չի կարելի զանգվածին վերագրել ավելի շատ էլեմենտներ, քան տրված է նրա հայտարարման մեջ, սակայն հակառակը կարելի է: Օրինակ

```
int IntegerArray[5] = {10, 20, 30, 40, 50, 60}; //Սխալ գրառում է:
int IntegerArray[5] = {10, 20};  //Սխալ չէ
```

➢ Օբյեկտների զանգվածները

Կամայական օբյեկտ կարող է պահպանվել զանգվածի մեջ: Եթե կլասը պարունակում է լռելիությամբ տրված ստանդարտ կոնստրուկտոր, ապա կլասի օբյեկտը կարող է ստեղծվել և պահպանվել զանգվածում, զանգվածի հայտարարման հետ միաժամանակ:

Զանգվածի էլեմենտների առանձին փոփոխական-անդամների կամ մեթոդների դիմելաձևը ներկայացված է ստորև

Ծրագրային կոդ 2.5-1

```
1     #include <iostream.h>
3     class Man
4     {
6         int itsAge;
8         int itsWeight;
10    public:
12        Man(){itsAge=1; itsWeight=0;};
14        ~Man(){};
16        int GetAge(){return itsAge;};
19        int GetWeight(){return itsWeight;};
21        void SetAge(int age){itsAge=age;};
22    };
24    int main()
25    {
26        //Հայտարարվում է Litter[] զանգվածը, որը պարունակում է Man կլասի օբյեկտներ
28        Man Litter[5];
29        int i;
30        for (i=0; i<5; i++)
31            // Զանգվածի i-րդ անդամի համար աշխատեցվում է SetAge() մեթոդը
32            Litter[i].SetAge(2*i+1);
```

```
33          for (i=0; i<5; i++)

34            {
35                  cout << "Man #"<< i+1<<":";
36                  // Զանգվածի i-րդ անդամի համար աշխատեցվում է GetAge()   մեթոդը
38                  cout << Litter[i].GetAge() <<  endl;
39             }
40      return 0;
41      }
```

Թողարկման արդյունքը

```
Man # 1: 1
Man # 2:  3
Man # 3:  5
Man # 4:  7
Man # 5:  9
```

➢ Բազմաչափ զանգվածներ

Կարելի է ստեղծել և գործածել մեկ չափից ավելի զանգվածներ, որտեղ կամայական էլեմենտ կարելի է գտնել համապատասխան ինդեքսավորմամբ: Օրինակ՝ երկչափանի զանգվածի համար անհրաժեշտ են երկու ինդեքսներ, եռաչափ զանգվածի համար երեք և այլն: Տեսականորեն կարելի է ստեղծել կամայական չափի զանգված, սակայն գործնականում օգտագործվում են երկչափանի զանգվածները, որոնց ուսումնասիրությամբ էլ մենք կզբաղվենք ստորև:

Ենթադրենք ունենք Square անվանումով կլասը: Board անվանումով միաչափ զանգվածի հայտարարումը, թվով 64 էլեմենտների դեպքում, այդ կլասի օբյեկտների պահպանման համար, կունենա հետևյալ տեսքը

```
Square Board[64];
```

Իսկ երկչափ զանգվածի համար կարելի է պարզապես գրել

```
Square Board[8][8]
```

Երկչափանի զանգվածների գործածումը առավել նպատակահարմար է այն դեպքերում, երբ փորձ է կատարվում մոդելավորել այնպիսի խնդիրներ, որտեղ այդպիսի ներկայացումը ավելի մոտ է իրական խնդրի դրվածքին /օրինակ շախմատի խաղի ծրագրավորման համար/:

Բազմաչափ զանգվածներին արժեքների վերագրումը իրականացվում է նույն ձևով, ինչ միաչափ զանգվածների դեպքում: Օրինակ thisArray[5][3] զանգվածին արժեքների վերագրումը, որի էլեմենտները ամբողջ թվեր են, կարելի է իրականացնել հետևյալ կերպով

```
int thisArray[5][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
```

Բազմաչափ զանգվածի մեկ այլ գրելաձև բերված է ստորև բերված ծրագրային կողում

Ծրագրային կոդ 2.5-2

```
1    #include <iostream.h>
2    int main()
3    {
4        int SomeArray[4][3]={
5        {0,0,0},
6        {1,2,4},
7        {2,4,16},
8        {3,6,36}
9        };
10       int i,j;
11       for  (i=0; i<4;i++)
12               for (j=0; j<3; j++)
13               {
14                       cout << "SomeArray ["<<i <<"]["<<j<<"]=";
15                       cout <<SomeArray[i][j] <<endl;
16               }
17       return 0;
18   }
```

Թողարկման արդյունքը՝

```
SomeArray[0][0] = 0
SomeArray[0][1] = 0
SomeArray[0][2] = 0
SomeArray[1][0] = 1
SomeArray[1][1] = 2
SomeArray[1][2] = 4
SomeArray[2][0] = 2
SomeArray[2][1] = 4
SomeArray[2][2] =16
SomeArray[3][0] = 3
SomeArray[3][1] = 6
SomeArray[3][2] = 36
```

➢ Սիմվոլների զանգվածները

Տեքստի տողը իրենից ներկայացնում է սիմվոլների հավաքածու: Մինչ այժմ մեր կողմից գործածված տողերը դրանք անանուն տողային հաստատուններ են եղել, որոնք մենք գրում էինք cout օպերատորի հետ

Օրինակ՝

```
cout << " Hello World \n";
```

Սակայն C++ -ում տողը կարելի է ներկայացնել որպես սիմվոլների զանգված, որը ավարտվում է վերջնական զրոյական տողի սիմվոլով: Այդպիսի զանգվածը կարելի է հայտարարել և ինիցիլիավորել ինչպես կամայական այլ տիպի զանգված:

Օրինակ՝

```
char Greeting[ ]={'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l', 'd', '\0'};      կամ
char Greeting[ ] = "Hello World";
```

Ծրագրային կոդ 2.5-3

```
1     #include <iostream.h>
2     int main()
3     {
5          char buffer[80];
7          cout<<"Enter the string : \t";
9          cin>>buffer;
12         cout<<"Here is the buffer: \t"<<buffer<< endl;
13    return 0;
14    }
```

Ծրագրակման արդյունքը՝

```
Enter the string :  Hello World
Here is the buffer  Hello
```

Այս օրինակում առկա են հետևյալ երկու պրոբլեմները. նախ՝ հայտարարված զանգվածի էլեմենտների քանակից ավելին սիմվոլների ներմուծում հնարավոր չէ, և երկրորդ՝ **cin** օպերատորը բացակի ներմուծումը համարում է որպես տողի ավարտ: Այդ է պատճառը, որ մուտքագրված **Hello World** բառակապակցությունից էկրան է դուրս բերվում միայն **Hello** բառը: Այս պրոբլեմը լուծվում է **get()** մեթոդի օգնությամբ: Տե՛ս ստորև բերված օրինակը՝

Ծրագրային կոդ 2.5-4

```
1     #include <iostream.h>
2     int main()
3     {
5          char buffer[80];
7          cout<<"Enter the string : \t";
10         cin.get(buffer,79);
13         cout<<"Here is the buffer:\t "<<buffer<< endl;
```

```
14    return 0;
15    }
```

Թողարկման արդյունքը

```
Enter the string :          Hello World
Here is the buffer          Hello World
```

ԳԼՈՒԽ 3 ԺԱՌԱՆԳԱԿԱՆՈՒԹՅՈՒՆ ԵՎ ՊՈԼԻՄՈՐՖԻԶՄ

§ 3-1 ժառանգականություն

Ժառանգականությունը գործիք է, որի օգնությամբ մի կլասի հատկությունները և մեթոդները փոխանցվում են մեկ այլ կլասի, որտեղ դրանք կարող են պահպանվել, գերբեռնվել կամ ծածկվել կախված այդ անդամների հասանելիության տիպից և ժառանգման տեխնոլոգիայից:

Ժառանգականության հիմնական սխեման բերված է ստորև

```
class բազային_կլաս_անվ. : հաս._ռեժիմ ածանցյալ_կլասի_անվ.
{
   …  // ածանցյալ կլասի մարմին
};
```

Ընդունված է ժառանգվող կլասին անվանել ***բազային կլաս*** - ***base class***, իսկ ժառանգված կլասը՝ **ածանցյալ կլաս** :

Հասանելիության _ռեժիմը կարող է լինել ***public, private*** կամ ***protected***: Կախված բազային կլասի անդամների և ժառանգման մեխանիզմում գործածված ***հասանելիության ռեժիմներից,*** ածանցյալ կլասի անդամների համար, հնարավոր հասանելիության հետևյալ տարբերակները /տես աղ.3.1-1-ը/:

Աղյուսակ 3.1-1

Հասանելիության ռեժիմը		
Բազային կլասի անդամներին	Ժառանգման մեխանիզմում	Ածանցյալ կլասի անդամների
Private	Public	Անհասանելի են
Protected		Protected
Public		Public
Private	Protected	Անհասանելի են
Protected		Protected
Public		Protected
Private		Անհասանելի են

Protected	Private	Private
Public		Private

Օրինակ Ծրագրային կոդ 3.1-1

```
1   // Բազային կլաս X
2   class X
3   {
4         int i;
5         int j;
6   public:
7         void Set_ij ( void );
8         void Get_ij ( void );
9   };
10  // Ածանցյալ կլաս Y: Ժառանգված է X բազային կլասից Հաս. ռեժիմը ՝ public
11  class Y: public X
12  {
13        int k;   /
14  public:
15        void Get_k ( void );
16        void Make_k ( void );
17  }
```

Այս օրինակում X-ը բազային կլաս է, Y-ը՝ ժառանգված կամ ածանցյալ կլաս: Y կլասի ֆունկցիա-անդամները կարող են գործածել X բազային կլասի ընդհանուր (բաց) անդամները՝ Set_ij() և Get_ij() մեթոդները, սակայն նրանք չեն կարող գործածել i կամ j փակ անդամները:

Եթե X կլասի i և j անդամները հայտարարվեն **protected,** ապա դրանք հասանելի կլինեն Y կլասից, սակայն կշարունակվեն փակ մնալ ծրագրի այլ հատվածներից:

➢ Կոնստրուկտորներ և դեստրուկտորներ

Ածանցյալ կլասների հիմքերի վրա օբյեկտների ստեղծման ժամանակ կանչվում են նախ բազային, այնուհետև՝ ածանցյալ կլասների կոնստրուկտորները, իսկ նրանց հեռացման ժամանակ դեստրուկտորները՝ կոնստրուկտորների կանչման հակառակ հաջորդականությամբ: Տես օրինակ 11-4-ը

Ծրագրային կոդ 3.1-2

```
1   #include <iostream.h>
2   // Սահմանվում է Human բազային կլասը
3   class Human
```

```
4    {
5    protected:
6         int itsAge;
7         int itsWeight;
8    public:
9         Human()  { cout<<"Human constructor\n"; };
10        ~Human()  { cout<<"Human destructor\n"; };
11   };
12   // Սահմանվում է Man ածանցյալ կլասը
13   class Man : public Human
14   {
15   public:
16        Man() { cout<<"Man constructor\n"; };
17        ~Man() { cout<<"Man destructor\n"; };
18   };
19   int main()
20   {
21        Man Harry;
22   return 0;
23   }
```

Թողարկման արդյունքը `

```
Human constructor
Man  constructor
Man  destructor
Human destructor
```

➢ Վերածածկվող մեթոդներ

Եթե բազային կլասում հայտարարված մեթոդը ածանցյալ կլասում պահպանի իր տեսքը, սակայն իրականացնի մեկ այլ գործողություն, ապա կարելի է ասել, որ այն ծածկում է բազային կլասի համանուն մեթոդին: Ածանցյալ կլասի հիմքի վրա ստեղծված օբյեկտի համար վերոհիշյալ մեթոդը կանչելուց կաշխատի վերածածկած տարբերակը, իսկ բազային կլասի հիմքի վրա ստեղծված օբյեկտների համար ` բազային տարբերակը: Դիտարկենք ասվածը ստորև բերված օրինակը

Ծրագրային կոդ 3.1-3

```
1    #include <iostream.h>
2    class Human
3    {
4    protected:
5         int itsAge;
6         int itsWeight;
7    public:
```

```
8       Human() { cout << "Human standart constructor \n"; };

9       ~Human() { cout<< "Human destructor \n";};
10      void Voice() { cout << “Human sound \n “; };
11  };
12  class Man: public Human
13  {
14  public:
15      int itsLenght;
16      Man() { cout << "Man standart constructor \n"; };
17      ~Man() {cout<< "Man destructor \n";};
18      void Voice() { cout << “Man sound \n “; };
19  };
20  int main()
21  {
22      Human Olga;
23      Man Harry;
24      Olga.Voice();
25      Harry.Voice();
26      return 0;
27  }
```

Թողարկման արդյունքը՝

```
Human standart constructor
Human standart constructor
Man standart constructor
Human Sound
Man sound
Man denstructor
Human denstructor
Human denstructor
```

➢ Վիրտուալ ֆունկցիաներ

Եթե ածանցյալ կլասի հիմքի վրա ստեղծված օբյեկտի հիշողության հասցեն վերագրվի բազային կլասի ցուցչին, որի հիմքի վրա ածանցվել է այդ օբյեկտի կլասը, ապա այդ ցուցչի օգնությամբ կարելի կլինի դիմել ինչպես բազային կլասի օբյեկտներին, այլև նրանց հիմքի վրա ստեղծված ածանցյալ կլասի օբյեկտներին: Օրինակ

Ծրագրային կոդ 3.1-4

```
1   #include <iostream.h>
2   class Human
3   {
4   protected:
```

```
5         int itsAge;
6         int itsWeight;
7     public:
8                 void Voice() { cout << “Human Sound \n “ ; };
9         virtual  void Move() { cout << “Human moves one step \n” ;};
10    };
11    class Man: public Human
12    {
13    public:
14        int itsLenght;
15        void Voice() { cout << “Man Sound  \n “; };
16        void Move() { cout << “Man  moves four  step \n” ;};
17    };
18    int main()
19    {
20        Human *pHuman_1 = new Human;
21        Man *pMan = new Man;
22        Human  *pHuman_2 = new Man;
23        pHuman_1 -> Voice();
24        pMan -> Voice();
25        pHuman_2 -> Voice();
26        pHuman_1 -> Move();
27        pMan -> Move();
28        pHuman_2 -> Move();
29    return 0;
30    }
```

Թողարկման արդյունքը

```
Human Sound
Man Sound
Human Sound
Human moves one step
Man moves four step
Man moves four step
```

Այս օրինակում, Human բազային կլասում հայտարարված են երկու մեթոդներ, որոնք վերածածկված են Man ածանցյալ կլասում: Նրանցից մեկը սովորական մեթոդ է՝ Voice(), իսկ Move()-ը՝ գրելաձևում պարունակում է նոր (virtual) բանալիային բառ: Այնուհետև դինամիկ հիշողությունում հայտարարված են երեք օբյեկտներ, որոնց կարելի է դիմել համապատասխան ցուցիչների օգնությամբ: Նրանցից մեկը Human կլասի օբյեկտ է, որի հասցեն պահվում է *pHuman_1՝ Human տիպի ցուցչի մեջ (տող 23): Երկրորդը՝ Man տիպի օբյեկտ է, որի հասցեն պահվում է *pMan՝ Man տիպի ցուցչի մեջ (տող 25), իսկ վերջինը, որը

Man կլասի օբյեկտ է, նրա հասցեն վերագրված է Human տիպի ցուցչին (*pHuman_2) (տող 27): Բոլոր երեք օբյեկտների համար հերթով կանչվում են նախ՝ Voice() – սովորական (տող 30-32), այնուհետև՝ Move() - վիրտուալ մեթոդները (տող 34-36): Եթե առաջին դեպքում աշխատեցվում է վերածածկված մեթոդի այն տարբերակը, որի հիմքի վրա հայտարարված են եղել ցուցիչները, ապա երկրորդ դեպքի համար կանչվում է համապատասխան ածանցյալ կլասի վերածածկված մեթոդը: Հենց սրանում էլ կայանում է **վիրտուալ ֆունկցիաների** առաքելությունը: Նրանց օգնությամբ բազային կլասներում ստեղծվում են անհրաժեշտ ինտերֆեյսներ, այնուհետև ածանցյալ կլասներում դրանք վերածածկվում են պահանջվող եղանակներով, որից հետո ստեղծվում են ածանցյալ կլասների օբյեկտներ, որոնց հասցեները վերագրվելով բազային կլասների հիմքի վրա սահմանված ցուցչներին, հնարավորություն է ստեղծվում կանչելու համապատասխան ածանցյալ կլասում գործող մեթոդի վերածածկված տարբերակը:

Եթե կլասում հայտարարված են վիրտուալ ֆունկցիաներ, ապա դեստրուկտորը ևս պետք է հայտարարվի վիրտուալ:

- **Մաքուր վիրտուալ ֆունկցիաներ և աբստրակտ տիպեր**

Վիրտուալ ֆունկցիաները կոչվում են մաքուր վիրտուալ, եթե դրանք հայտարարված են բազային կլասներում, սակայն նրանցում որոշված չեն և նրանք սահմանվում են ածանցյալ կյասներից յուրաքանչյուրում առանձին:

Մաքուր վիրտուալ ֆունկցիան հայտարարվում է հետևյալ սխեմայով՝

virtual վերադարձվող_տիպ ֆունկցիա_անվանում (պարամետր_1, պարամետր_2, …. պարամետր_n) = 0 ;

որտեղ **վերադարձվող_տիպ -**ը ֆունկցիայից վերադարձվող մեծության տիպն է, **ֆունկցիա_անվանում** – ը այն ֆունկցիայի անվանումն է, որը հայտարարվում է որպես մաքուր վիրտուալ ֆունկցիա, **(պարամետր_1, պարամետր_2, …. պարամետր_n)**-ը այն պարամետրերի ցանկն է, որը ստանում է տվյալ ֆունկցիան, իսկ = 0 արտահայտությունը ցույց է տալիս, որ ունենք մաքուր վիրտուալ ֆունկցիա, և այդ կլասում չի սպասվում այդ ֆունկցիայի սահմանումը:

Եթե կլասը պարունակում է գոնե մեկ մաքուր վիրտուալ ֆունկցիա, ապա նրան անվանում են **աբստրակտ տիպի կլաս**: Աբստրակտ կլասի հիմքի վրա **օբյեկտ հայտարարել հնարավոր չէ**: Սակայն թույլատրվում է

ածանցյալ կլասների հիմքի վրա ստեղծել օբյեկտներ, և նրանց հասցեները վերագրել աբստրակտ բազային կլասի հիմքի վրա հայտարարված ցուցիչներին: Դիտարկենք ստորև բերված օրինակը:

Ծրագրային կոդ 3.1-5

```
1   #include <iostream.h>
2   class Human
3   {
4   protected:
5           int itsAge;
6           int itsWeight;
7   public:
8       void Voice() { cout << "Human Sound \n" ; };
9       virtual void Move()=0;
10  };
11  class Man: public Human
12  {
13  public:
14      void Voice() { cout << "Man sound  \n"; };
15      void Move() { cout << "Man  move four  step \n" ;};
16  };
17  class Woman: public Human
18  {
19  public:
20      void Voice() { cout << "Woman sound\n"; };
21      void Move() { cout << "Woman moves six step \n" ;};
22  };
23  int main()
24  {
25      //  Human Some_Human;
26      Man *pMan = new Man;
27      pMan -> Voice();
28      pMan -> Move();
29      Woman Ann;
30      Ann.Voice();
31      Ann.Move();
32      *pHuman ցուցչի մեջ: */
33      Human *pHuman = new Woman;
34      pHuman->Voice();
35      pHuman->Move();
36  return  0;
37  }
```

Ինչպես երևում է բերված օրինակից, աբստրակտ կլասի հիմքի վրա հնարավոր չէ ստեղծել որևէ օբյեկտ: Տե՛ս հրամանային կողի 25-րդ արգելափակված տողը

Աբստրակտ կլասները կարող են լինել միայն բազային: Նրանց հիմքի վրա կարելի է ստեղծել ածանցյալ կլասներ, որտեղ մաքուր վիրտուալ ֆունկցիաները կարող են սահմանվել վերջնական տեսքով: Ածանցյալ կլասների հիմքի վրա հայտարարված օբյեկտների հասցեները կարելի է պահպանել, աբստրակտ կլասի հիմքի վրա հայտարարված ցուցչի մեջ: Տես հրամանային կողում 58-րդ տողը: Իսկ դա նշանակում է, որ նրա օգնությամբ կարելի է կանչել բազային կլասում սահմանված և համապատասխան ածանցյալ կլասում որոշված մաքուր վիրտուալ ֆունկցիայի համարժեք տարբերակը: Տես 62-րդ հրամանային տողը որը էկրան է դուրս բերում

Woman moves six step արտահայտությունը, այն դեպքում երբ 60-րդ տողը էկրան է դուրս բերում

Human Sound արտահայտությունը

§ 3-2 Պոլիմորֆիզմ

C++ լեզուն, ֆունկցիաների և կլասների պոլիմորֆիզմի շնորհիվ, աջակցում է տարբեր օբյեկտներում համանուն ֆունկցիաների զանազան իրականացման հնարավորությունների ապահովմանը:

Երբ ունենք բազային կլաս, որում առկա են մաքուր վիրտուալ ֆունկցիաներ և այդ կլասից ժառանգվում ու ստացվում են մեկից ավելի ածանցյալ կլասներ, որոնցից յուրաքանչյուրում իրականացվում է մաքուր վիրտուալ մեթոդների յուրօրինակ սահմանումներ, այդ դեպքում, եթե ածանցյալ կլասներից յուրաքանչյուրի հիման վրա ստեղծվեն օբյեկտներ, իսկ նրանց հասցեները պահպանվեն բազային կլասի ցուցչի մեջ, ապա հնարավոր կլինի վերջինիս օգնությամբ կանչել ածանցյալ կլասի վերասահմանված մեթոդի համապատասխան տարբերակը: Ասվածը դիտարկենք 3.2-1-ում բերված օրինակի մեջ

Ծրագրային կոդ 3.2-1

```
1       #include <iostream.h>
3       class Animal
4       {
5               int itsAge;
```

```
6       public:
8               virtual void Eat()=0;
9       };
11      class Dog:public Animal
12      {
13      public:
15              void Eat(){cout<<"I am Dog. I like meat....."<<endl;}
16      };
18      class Cat:public Animal
19      {
20      public:
22              void Eat(){cout<<"I am Cat. I like milk....."<<endl;}
23      };
25      class Cow:public Animal
26      {
27      public:
29              void Eat(){cout<<"I am Cow. I like green....."<<endl;}
30      };
32      int main()
33      {
35              int choice;
37              Animal *p;
38              cout<<"Insert the number drom diapazone 1...3";
41              cin>>choice;
42      switch (choice)
43      {
45      case 1: {p=new Dog;
46              break;}
48      case 2: {p=new Cat;
49              break;}
51      case 3: {p=new Cow;
52              break;}
53      default: break;
54      };
56      p->Eat();
58      delete p;
59      return 0;
60      }
```

Ծրագրի արդյունքը՝

```
Insert the number from diapazone 1...3    2
        I am Cat. I like milk.....

Insert the number from diapazone 1...3    1
        I am Dog. I like meat.....
```

```
Insert the number from diapazone 1...3  3

    I am Cow. I like green.....
```

➢ Բազմակի ժառանգման մեխանիզմները

Գործնականում հնարավոր է այնպիսի տարբերակ, երբ ածանցյալ կլասը ժառանգվում է 1-ից ավելի բազային կլասներից: Այսպիսի պրոցեսը կոչվում է բազմակի ժառանգում: Այս դեպքում ածանցյալ կլասի հայտարարման ժամանակ բազային կլասները թվարկվում և միմյանցից բաժանվում են ստորակետներով՝ յուրաքանչյուրի համար ժառանգման հասանելիության ռեժիմի շեշտմամբ: Սխեմատիկ այն կարելի է ներկայացնել հետևյալ կերպ՝

```
class ածանցյալ_կլաս : հաս_ռեժիմ_1 բազային_կլաս_1, հաս_ ռեժիմ_2 բազային_կլաս_2…… հաս_ռեժիմ_N բազային_կլաս_N
    {
        հաս_ռեժիմ:
                փոփոխական անդամ_1;
                փոփոխական անդամ_2;
                ……..
        հաս_ռեժիմ:
                ֆունկցիա անդամ_1
                ֆունկցիա անդամ_2;
                ……
    };
```

հաս_ռեժիմ_i –դրդր դա i- րդ բազային կլասի հասանելիության ռեժիմն է:

Նման ժառանգման դեպքում ածանցյալ կլասի հիմքի վրա օբյեկտների ստեղծումը բերում է բազային կլասների կոնստրուկտորների կանչ՝ հայտարարման տողում դրանց թվարկման հաջորդականությամբ՝ ձախից աջ: /Դեստրուկտորները կանչվում են կոնստրուկտորների հակառակ հաջորդականությամբ/: Ասվածը դիտարկենք ստորև բերված օրինակի վրա:

Ծրագրային կոդ 3.2-2

```
1       #include <iostream.h>
2       class Base1
3       {
4       public:
5           Base1(){cout<<"Constructor for  class Base_1\n";};
6           ~Base1(){cout<<" Destructor for  class Base_1\n";};
7       };
8       class Base2
```

```
9       {
10      public:
11          Base2(){cout<<"Constructor for  class Base_2\n";};
12         ~Base2(){cout<<" Destructor for  class Base_2\n";};
13      };
14      class Derive: public Base1, public Base2
15      {
16      public:
17          Derive(){cout<<"Constructor for  class Derive\n";};
18          ~Derive(){cout<<" Destructor for  class Derive\n";};
19      };
20      int main()
21      {
22          Derive Derive_object;
23      return 0;
24      }
```

Թողարկման արդյունքը

```
Constructor for  class Base_1
Constructor for  class Base_2
Constructor for  class Derive
Destructor for  class Derive
Destructor for  class Base_2
 Destructor for  class Base_1
```

➢ Վիրտուալ բազային կլասներ

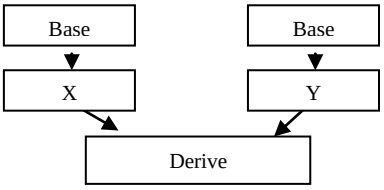
Բազմակի ժառանգման ժամանակ բազային կլասը ածանցյալ կլասում չի կարող հանդիպել 1-ից ավելի անգամ:

```
class Der: Base, Base{ . . . };                    // սխալ է
```

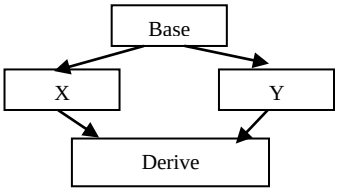
Սակայն բազային կլասը կարող է փոխանցվել ածանցյալ կլասին անուղղակիորեն: Այսպես օրինակ

```
class X: public Base { . . . };
class Y: public Base { . . . };
class Derive: public X, public Y { . . . };
```

Ասվածը համապատասխանում է ժառանգման հետևյալ նկ 1 սխեմային



նկ 1



նկ2

Այս դեպքում Derive ածանցյալ կլասում Base բազային կլասը անուղղակիորեն առկա է 2 անգամ: Բազային կլասի անդամներին դիմելուց առաջանում է բազմանշանակ իրավիճակ, որից խուսափելու համար անհրաժեշտ է բազային կլասը հայտարարել վիրտուալ: Այդ դեպքում կունենանք հետևյալ նշանակումը

```
class X: virtual  public Base { . . . };
class Y: virtual  public Base { . . . };
class Derive: virtual  public X, public Y { . . . };
```

Այժմ Derive /ածանցյալ կլասը/ պարունակում է միայն մեկ Base կլաս: Այդ օրինակը համապատասխանում է ժառանգման նկ 2-ի սխեմային

Հավելված 1

Օրագրերի կազմումը, կոմպիլացիան, կատարողական ֆայլերի ստացումը և թողարկումը **Microsoft Visual C++ 6.0** միջավայրում:

Պարզագույն նախագծի ստեղծման համար անհրաժեշտ է

1. Անցնել **Start=>Microsoft Visual Studio 6.0=>Microsoft Visual C++6.0** ճամապարհով և թողարկել այն /*կամ որևէ եղանակով թողարկել **MSDEV.EXE** ֆայլը/*
2. Հրամանային մենյուի **File** կետում ընտրել **New** ենթակետը և թողարկել այն:
3. Բացված պատուհանում ընտրել **Win32 Console Application** կետը, Project name դաշտում մուտքագրել նախագծի անվանումը և սեղմել **OK** հրամանային կոճակը:
4. Բացված պատուհանում ընտրել **An Empty Project** կետը և սեղմել **Finish** կոճակը:
5. Բացված նոր պատուհանում սեղմել **OK** կոճակը

Ելակետային ֆայլերի ստեղծման և կատարողական ֆայլերի ձևափոխման և դրանց թողարկման համար անհրաժեշտ է՝

6. Հրամանային մենյուի **File** կետում ընտրել **New** կետը
7. Բացված պատուհանիկում ընտրել **C++ Source File-ը կետը: Add to Project** տեքստային դաշտում մուտքագրել նախագծի անվանումը, **File name** պատուհանիկում մուտքագրել ֆայլի անունը և սեղմել OK հրամանային կոճակը:
8. Մուտքագրել որևէ ծրագրային կոդ՝ : Օրինակ

```
#include <iostream.h>
int main()
{
        cout<<"Hello world! \n";
return 0;
}
```

9. Հրամանային մենյուի **Build** կետում ընտրել **Build Hello.exe** հրամանը և թողարկել այն /կամ սեղմել **F7** ֆունկ. կոճակը/
10. Համոզվել, որ կոմպիլացիայի արդյունքում կոմպիլյատորը ոչ մի սխալ չի հայտնաբերել:
11. Հրամանային մենյուի Build կետում ընտրել ! Execute Hello.exe կետը և թողարկել այդ կատարողական ֆայլը /կամ սեղմել **<Ctrl + F5>** ֆունկ. կոճակները:
12. Էկրանին ստանալ թողարկման արդյունքը

```
Hello word
```

13. Աշխատանքը ավարտելու համար սեղմել բացակ կոճակը:

Ա. Տողերի հետ աշխատող ֆունկցիաներ

ՀՀ	Անվանումը	Նշանակությունը
1	strcpy(s1, s2)	s1-ը նպատակային զանգվածի մեջ արտագրվում է s2 աղբյուր զանգվածի պարունակությունը
2	strncpy(s1, s2, M_L)	s1-ը նպատակային զանգվածի մեջ արտագրվում է s2 աղբյուր զանգվածի պարունակությունը M_L-ից ոչ ավել
3	strcat(s1,s2)	s2 տողի պարունակությունը միացվում է s1-ին և արդյունքը տեղավորում է s1-ի մեջ
4	strlen(s1)	վերադարձվում է s1 նպատակային զանգվածի էլեմենտների քանակը
5	strcmp(s1, s2)	Իրականացնում է s1 ու s2 տողերի համեմատում և արդյունքում վերադարձնում ամբողջ թիվ: Ընդ որում՝ 0, եթե s1 և s2 տողերը բառարանային իմաստով համընկնում են 1, եթե s1 տողը բառարանային իմաստով մեծ է s2-ից -1, եթե s1 տողը բառարանային իմաստով փոքր է s2-ից

Բ. Մաթեմատիկական ֆունկցիաներ

ՀՀ	Անվանումը	Նշանակությունը
1	sin(x)	հաշվարկում է x արգումենտի sin-ը
2	cos(x)	հաշվարկում է x արգումենտի cos-ը
3	tn(x)	հաշվարկում է x արգումենտի tg-ը
4	pow(x,y)	հաշվարկում է x^y մեծության արժեքը
5	exp(x)	հաշվարկում է e^x մեծության արժեքը
6	lg(x)	հաշվարկում է բնական հիմքով լոգարիթմ x-ը
7	log10(x)	հաշվարկում է 10 հիմքով լոգարիթմ x-ը

Հավելված 3

Ստուգողական աշխատանքների տարբերակների լուծված օրինակներ, *

Ստուգողական աշխատանք 1.

1. Գտնել տրված a,b,c թվերից մեծագույնը:
2. Տրանսպոնացնել n x n չափանի իրական մատրիցը:
3. Կազմել ծրագիր, որը նշված միջակայքում, □x քայլով հաշվի տրված ֆունկցիայի արժեքները: Տպել x և y փոփոխականների արժեքների աղյուսակը:

$y = \sqrt[3]{x + \sqrt{\sin(x+1)+3}}$, $x \in [8;13,7]$, $\Delta x = 0,4$:

4. Գերբեռնեք կլասի կոնստրուկտորը 2 անգամ այնպես, որ մի դեպքում նա չստանա պարամետրեր, իսկ մյուս դեպքում նրան փոխանցված արժեքները վերագրվեն կլասի փոփոխական անդամներին:

Ստուգողական աշխատանք 2.

1. Տրված են y=ax+b և y=cx+d ուղիղների a,b,c,d իրական գործակիցները: Տպել y-ի արժեքը, եթե

$$y = \begin{cases} 0 & \text{երբ ուղիղներն համընկնում են} \\ 1 & \text{երբ ուղիղները զուգահեռ են} \\ 2 & \text{երբ ուղիղներն հատվում են:} \end{cases}$$

2. Տրված է n բնական թվի համար հաշվել արտահայտության արժեքը $\sin 1 + \sin 1 \cdot \sin 2 + \ldots + \sin 1 \cdot \ldots \cdot \sin n$,

3. Գերբեռնեք թվի խորանարդը հաշվող ֆունկցիաները int, float, double և long տիպերի համար:
4. Սահմանեք Dog և Cat կլասներները: Նրանց համար սահմանեք Age() բարեկամական ֆունկցիան, որը կհամեմատի այդ կլասների հիմքի վրա ստեղծված օբյեկտների տարիքները:

Ստուգողական աշխատանք 3.

1. Տրված է m x m չափանի իրական մատրից: Ստանալ b վեկտորը, որի b(k) (k=1…m) տարրը հավասար է տրված մատրիցի k -րդ տողի առաջին տարրին, եթե այդ տողում կա գոնե մեկ բացասական տարր, հակառակ դեպքում՝ k-րդ տողի վերջին տարրին:
2. Հաշվել արտահայտության արժեքը

*) Ստուգողական առաջադրանքները վերցրված են [4]-ից և [5]-ից:

$$\sum_{k=1}^{8} \frac{k!+6}{\frac{1}{3}+\frac{1}{5}+\ldots+\frac{1}{2k+1}}$$

3. Տրված է Machine բազային կլասը՝ ներառում Move() մաքուր վիրտուալ ֆունկցիան: Աշակեք ծրագիր, որը կներկայացնի այդ ֆունկցիայի իրականացում 4 տարբեր ձևերով:
4. *Գրեք ծրագիր որը 10-ական համակարգում ներմուծված թիվը կվնատիոխի 2-ականի:*

Լուծումներ

Ստուգողական աշխատանք 1.

1. **Գտնել տրված a,b,c թվերից մեծագույնը:**

Լուծում

*Այս օրինակում գործածված են միաժամանակ 2 տիպի if և if…else կոնստրուկցիաներ: Ծրագրային կոդի բացատրությանը հետևեք ըստ բերված մեկնաբանությունների */*

```
# include <iostream.h>
int main()
{
//Հայտարարվում են a,b,c և max ամբողջ տիպի փոփոխականները
    int a,b,c,max;
//Ներմուծվում են a,b,c փոփոխականների արժեքները
    cout<<"Input a="; cin>>a;
    cout<<"Input b="; cin>>b;
    cout<<"Input c="; cin>>c;
/* Համեմատվում են a և b, փոփոխականների մեծությունները, նրանցից մեծի մեծությունը վերագրվում է max փոփոխականին */
    if(a>b)
        max=a;
    else
        max=b;
/* Այժմ max փոփոխականի հետ համեմատվում է c փոփոխականը, եթե վերջինս ավելի մեծ թվային արժեք ունի, ապա max –ին վերագրվում է նրա արժեքը հակառակ դեպքում ոչինչ տեղի չի ունենաում */
    if (c>max)
        max=c;
```

```
/* էկրան է դուրս բերվում max փոփոխականի արժեքը, որը առավելագույնն է այդ 3
թվերի մեջ */;
    cout<<"max from a,b & c is a  "<<max;
    return 0;
}
```

Ծրագրի արդյունքի օրինակ

```
Input a=5
Input b=15
Input c=7
max from a,b & C is a  15
```

2. Տրանսպոնացնել n x n չափանի իրական մատրիցը:

Լուծում

Տրանսպոնացնել մատրիցը, նշանակում է նրա տողերը և սյուները տեղափոխել :

```
# include <iostream.h>
/* Որպեսզի հնարավոր լինի զանգվածի չափը փոխել 1 անգամ , հայտարարված է n
հաստատուն փոփոխականը, ամբողջ տիպի, որին անմիջականորեն վերագրված է 3
արժեքը: Ծրագրի այլ հատվածներում մենք գործածել ենք n լիտերային սիմվոլը */
const int n=3;
int main()
{
/* Հայտարարվում են i j փոփոխականները / ինդեքսների համար /, a[n][n],
զանգվածը և  c փոփոխականը, որը նախատեսվում է տեղափոխման ժամանակ
միջանկյալ արժեքների  պահպանման համար */
    int i,j, a[n][n], c;
// Ներմուծվում են a[3][3] զնգվածի էլեմենտները
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
          cout<<"a["<<i<<"]["<<j<<"]=";
          cin>>a[i][j];
        }
/* Էկրան են դուրս  բերվում a[3][3] զնգվածի էլեմենտները ըստ տողերի և սյուների,
նախքան տրանսպոնացումը */
cout<<"Array A before transportation"<<endl;
        for (i=0; i<n; i++)
        {
          for (j=0; j<n; j++)
          cout<<a[i][j]<<"\t";
          cout<<endl;
        }
// Կատարվում է a[3][3] զնգվածի էլեմենտների տրանսպոնացում
    for (i=0; i<n; i++)
        for (j=0; j<i; j++)
```

```
        {

// a[i][j] զանգվածի i-րդ տողի j-րդ սյան էլեմենտը վերագրվում է c փոփոխականին
            c=a[i][j];
/* a[i][j] զանգվածի j-րդ տողի i-րդ սյան էլեմենտը վերագրվում է i-րդ տողի j-րդ սյանը */
            a[i][j]=a[j][i];
/* c փոփոխականի արժեքը, որը a[i][j] զանգվածի i-րդ տողի j-րդ սյան էլեմենտն էր վերագրվում է զանգվածի j-րդ տողի i-րդ սյանը*/
            a[j][i]=c;
//Այսքանով ավարտվում է տեղափոխությունը
        }
/* Կրկին էկրան եմ դուրս  բերվում a[3][3] զնգվածի էլեմենտները ըստ տողերի և սյուների, տրանսպոնացումից հետո */
cout<<endl<<"Array A after transportation"<<endl;
cout<<endl<<endl;
        for (i=0; i<n; i++)
        {
          for (j=0; j<n; j++)
          cout<<a[i][j]<<"\t";
          cout<<endl;
}
    return 0;
}
```

Թողարկման արդյունքի օրինակ

```
a[0][0]=15
a[0][1]=22
a[0][2]=5
a[1][0]=8
a[1][1]=33
a[1][2]=2
a[2][0]=18
a[2][1]=7
a[2][2]=89
```

Array A before transportation

```
15      22      5
8       33      2
18      7       89
```

Array A after transportation

```
15      8       18
22      33      7
5       2       89
```

3. Կազմել ծրագիր, որը նշված միջակայքում, □x քայլով հաշվի տրված ֆունկցիայի արժեքները: Տալել x և y փոփոխականների արժեքների աղյուսակը:

$$y=\sqrt[3]{x+\sqrt{\sin(x+1)+3}}\ ,\quad x\in[8;13,7]\ ,\ \Delta x=0,4$$:

Լուծում

Այս ծրագրի առանձնահատկությունը կայանում է նրանում, որ նախ ցուցադրված են առավել հաճախ հանդիպող մաթեմատիկական ֆունկցիաների գործածման առանձնահատկությունները, իսկ այնուհետև ներկայացված են ցիկլի կազմման օրինակ for օպերատորի օգնությամբ

```
# include <iostream.h>
# include <math.h>
int main()
{
    double x,y,delta_x;
    delta_x=0.4;
    for (x=8; x<=13.7; x+=delta_x)
    {
        y=pow(x+sqrt(sin(x+1)+3), 1.3);
        cout<<"x="<<x<<"\t"<<"y="<<y<<"\n";
    }
    return 0;
}
```

Թողարկման արդյունք

```
x=8         y=19.5572
x=8.4       y=20.3144
x=8.8       y=21.0562
x=9.2       y=21.8311
x=9.6       y=22.6947
x=10        y=23.6963
x=10.4      y=24.8585
x=10.8      y=26.1658
x=11.2      y=27.5734
x=11.6      y=29.0252
x=12        y=30.4691
x=12.4      y=31.8624
x=12.8      y=33.1742
x=13.2      y=34.3857
x=13.6      y=35.4897
```

3. Գերբեռնեք կլասի կոնստրուկտորը 2 անգամ այնպես, որ մի դեպքում նա չստանա պարամետրեր, իսկ մյուս դեպքում նրան փոխանցված արժեքները վերագրվեն կլասի փոփոխական անդամներին:

Լուծում

```
#include <iostream.h>
// Այստեղ հայտարարվում և սահմանվում է myClass անվանումով կլասը
class myClass
{
// x-ը y-ը կլասի փակ փոփոխական անդամներն են
    int x,y;
public:
// Հասանելիության բաց ռեժիմ
//Սահմանվում է myClass կլասի  առանց պարամետրերի myClass() կոնստրուկտորը
    myClass()
    {
        cout<<"Here myClass Constructor\n";
    };
//Սահմանվում է myClass կլասի  պարամետրերով myClass(int int) կոնստրուկտորը
    myClass(int k, int t)
    {
        x=k;
        y=t;
        cout<<"Here myClass Constructor with 2 parameters\n";
    };
//Սահմանվում է myClass կլասի  myClass() դեստրուկտորը
    ~myClass()
    {
        cout<<"Here myClass Denstructor\n";
    };
/* Getx() և Gety() ֆունկցիա անդամները վերադարձնում են myClass կլասի փակ
տիպի փոփոխական անդամների արժեքները */
    int  Getx()
    {
        return x;
    };
    int  Gety()
    {
        return y;
    };
};
// Հիմնական ծրագիր
int main()
{
/* Հայտարարվում են ob1 և ob2 օբյեկտները: ob1 օբյեկտի համար կանչվում է
myClass() առանց պարամետրերի, իսկ ob2-ի համար՝ myClass(int,int) արամետրերով
կոնստրուկտորները */
    myClass ob1, ob2(5,6);
```

```
// ob2 օբյեկտի համար էկրան է դուրս բերվում x փոփոխականի արժեքը
    cout<<"For ob2 object we have x="<<ob2.Getx()<<endl;
// ob2 օբյեկտի համար էկրան է դուրս բերվում y  փոփոխականի արժեքը
    cout<<"For ob2 object we have y="<<ob2.Getx()<<endl;
    return 0;
}
```

Ծրագրի կատարման արդյունքը

```
Here myClass Constructor
Here myClass Constructor with 2 parameters
For ob2 object we have x=5
For ob2 object we have y=5
Here myClass Destructor
Here myClass Destructor
```

Ստուգողական աշխատանք 2

1. **Տրված են y=ax+b և y=cx+d ուղիղների a,b,c,d իրական գործակիցները: Տալ y-ի արժեքը, եթե**

$$y=\begin{cases}0 & \text{երբ ուղիղները համընկնում են}\\ 1 & \text{երբ ուղիղները զուգահեռ են}\\ 2 & \text{երբ ուղիղները հատվում են:}\end{cases}$$

Լուծում

y=ax+b և y=cx+d ուղիղների կարող են

- համընկնել, եթե նրանց գործակիցների համար ճիշտ են հետևյալ պայմանները a=c և d=b :
- լինել զուգահեռ, եթե նրանց գործակիցների համար ճիշտ են հետևյալ պայմանները a=c և d≠b :
- հատվել մնացած դեպքերում

Ուստի ծրագրում պետք է ստուգել հենց այդ պայմանները

```
#include <iostream.h>
#include <math.h>
int main()
{
    double a,b,c,d;
    cout<<"a=";   cin>>a;
    cout<<"b=";   cin>>b;
    cout<<"c=";   cin>>c;
    cout<<"d=";   cin>>d;
    if(a==c && d==b)
        cout<<"Y=0"<<endl;
    else
        if(a==c && b!=d)
          cout<<"Y=1"<<endl;
        else
          cout<<"Y=2"<<endl;
```

```
    return 0;

}
```

Թողարկման արդյունք 1	Թողարկման արդյունք 2	Թողարկման արդյունք 3
a=5.2	a=12	a=7
b=1.3	b=5	b=5
c=5.2	c=12	c=12
d=1.3	d=3	d=3
Y=0	Y=1	Y=2

2. **Տրված է n բնական թվի համար հաշվել արտահայտության արժեքը**

$$\sin 1+\sin 1\cdot\sin 2+\ldots+\sin 1\cdot\ldots\cdot\sin n,$$

Լուծում

Այս ծրագրում ներմուծված են p և s փոփոխականները, որոնցից առաջինը նախատեսված է արտադրյալի հաշվման համար, իսկ 2-րդը՝ գումարի

```
#include <iostream.h>
#include <math.h>
int main()
{
    int n,i;
    double p,s;
    cout<<"Input n="; cin>>n;
    for (p=1, s=0, i=1; i<=n; i++)
    {
        p*=sin(i);   // Նույնն է ինչ գրենք p=p*sin(i)
        s+=p;        // Նույնն է ինչ գրենք s=s+p
    }
    cout<<"s="<<s<<endl;
    return 0;
}
```

Թողարկման արդյունքի օրինակ

```
Input n=15
s=1.65847
```

3. Գերբեռնեք թվի խորանարդը հաշվող ֆունկցիաները int, float, double և long տիպերի համար:

Լուծում

```
// Բացատրությունները շարադրված են մեկնաբանությունների ձևով
#include <iostream.h>
/* Ստորև տրված են myFunction ֆունկցիայի նախատիպերը int, long, float և double
տիպերի համար */
int myFunction(int x);
```

```
long myFunction(long l);
float myFunction(float f);
double myFunction(double d);
int main()
{
// Պարզության համար փոփոխականներին արժեքներ վերագրված են ծրագրի մեջ
    int variable_x=5;
    long variable_l=1222222;
    float variable_f=12.5;
    double variable_d=3333.145;
/*Հաջորդ 4 տողերում հերթով կանչվում է myFunction ֆունկցիան, որը ինչպես ցույց է տալիս թողարկման արդյունքը յուրաքանչյուր դեպքի համար կանչում է գերբեռնման օրինակի իր տարբերակը */
    cout<<"Cube of x ="<<myFunction(variable_x)<<endl;
    cout<<"Cube of l ="<<myFunction(variable_l)<<endl;
    cout<<"Cube of f ="<<myFunction(variable_f)<<endl;
    cout<<"Cube of d ="<<myFunction(variable_d)<<endl;
    return 0;
}
 // myFunction() ֆունկցիայի սահմանում երբ նա ստանում և վերադարձրում է int տիպի մեծություն
int myFunction(int x)
{
    cout<<"myFunction for int\t";
    return x*x*x;
};
// myFunction() ֆունկցիայի սահմանում երբ նա ստանում և վերադարձրում է long տիպի մեծություն
long myFunction(long x)
{
    cout<<"myFunction for long\t";
    return x*x*x;
};
// myFunction() ֆունկցիայի սահմանում երբ նա ստանում և վերադարձրում է  float տիպի մեծություն
float myFunction(float x)
{
    cout<<"myFunction for float\t";
    return x*x*x;
}
/* myFunction() ֆունկցիայի սահմանում երբ նա ստանում և վերադարձրում է double տիպի մեծություն  */
double myFunction(double x)
{
    cout<<"myFunction for double\t";
    return x*x*x;
```

```
}
```

Թողարկման արդյունք

```
myFunction for int          Cube of x =125
myFunction for long         Cube of l =1568835000
myFunction for float        Cube of f =1953.13
myFunction for double       Cube of d =3.70308e+010
```

4. Սահմանեք Dog և Cat կլասներները: Նրանց համար սահմանեք Age() բարեկամական ֆունկցիան, որը կհամեմատի այդ կլասների հիմքի վրա ստեղծված օբյեկտների տարիքները:

Լուծում

Բացատրությունները շարադրված են մեկնաբանությունների ձևով

```
#include <iostream.h>
/* Անհրաժեշտ է հայտարարել Dog կլասի մասին, որպեսզի այն հնարավոր լինի
գործածել Cat կլասի մեջ */
class Dog;
// Այստեղ սահմանվում է Cat կլասը
class Cat
{
// itsAge փակ փոփոխական անդամ Cat կլասի համար
    int itsAge;
public:
// Այստեղ հայտարարվում է բարեկամական տիպի Age ֆունկցիան
    friend void Age(Dog d, Cat c);
// SetAge() ֆունկցիայով itsAge փոփոխականին արժեք է վերագրվում
    void SetAge(int age){itsAge=age;};
//GetAge() ֆունկցիայով itsAge փոփոխականի արժեքը վերադարձվում է
    int GetAge(){return itsAge;};
};
//Սահմանվում է Dog կլասը
class Dog
{
// itsAge փակ փոփոխական անդամ Dog կլասի համար
    int itsAge;
public:
    friend void Age(Dog d, Cat c);
    void SetAge(int age){itsAge=age;};
    int GetAge(){return itsAge;};
};
//Այստեղ արդեն տրվում է Age() բարեկամական ֆունկցիայի սահմանումը
void Age(Dog d, Cat c)
{
```

```
/* Համեմատվում է Dog & Cat կլասների հիմքերի վրա ստեղծված օբյեկտների itsAge
փակ փոփոխական անդամների մեծությունները */
      if (d.itsAge > c.itsAge)
          cout<<"Dog age  >  Cat age\n";
      else
          cout<<"Dog age  <  Cat age\n";
};
// Գլխավոր ծրագիր
int main()
{
// Հայտարարված է Dog կլասի հիմքի վրա ob_dog օբյեկտը
      Dog ob_dog;
// Հայտարարված է Cat կլասի հիմքի վրա ob_cat օբյեկտը
      Cat ob_cat;
      cout<< "Dog's Age is 10\n";
/* SetAge() մեթոդի միջոցով ob_dog օբյեկտի փոփոխական անդամին վերագրվում է
10 արժեք */
      ob_dog.SetAge(10);
      cout<< "Cat's Age is 15\n";
/* SetAge() մեթոդի միջոցով ob_cat օբյեկտի փոփոխական անդամին վերագրվում է
15 արժեք */
      ob_cat.SetAge(15);
// Կանչվում է Age() բարեկամական ֆունկցիան, համեմատվում են արդյունքները
Age(ob_dog, ob_cat);
//Ծրագրի մնացած մասը արդեն կարծում ենք բացատրության կարիք չունի
      cout<<"But at now "<<endl;
      cout<< "Dog's Age is 15\n";
      ob_dog.SetAge(15);
      cout<< "Cat's Age is 7\n";
      ob_cat.SetAge(7);
      Age(ob_dog, ob_cat);
    return 0;
}
```

Ծրագրի արդյունք

```
Dog's Age is 10
Cat's Age is 15
Dog age  <  Cat age
But at now
Dog's Age is 15
Cat's Age is 7
Dog age  >  Cat age
```

Ստուգողական աշխատանք 3.

1. **Տրված է m x m չափանի իրական մատրից: Ստանալ b վեկտորը, որի b(k) (k=1…m) տարրը հավասար է տրված մատրիցի k -րդ տողի առաջին**

տարրին, եթե այդ տողում կա գոնե մեկ բացասական տարր, հակառակ դեպքում՝ k-րդ տողի վերջին տարրին:

Լուծում

```
#include <iostream.h>
const int m=4;
int main()
{
   bool tarr;
   int a[m][m], i, j, k, b[m];
   for (i=0; i<=m-1; i++)
     for (j=0; j<=m-1; j++)
      {
                cout<<"a["<<i<<"]["<<j<<"]=";
                cin>>a[i][j];
      }
cout<<endl;
   for (i=0; i<=m-1; i++)
   {
     for (j=0; j<=m-1; j++)
                cout<<a[i][j]<<"\t";
   cout<<endl;
   }
     for (k=0; k<=m-1; k++)
   {
     tarr=false;
     for (j=0; j<=m-1; j++)
                if (a[k][j]<0)
                {
                          tarr=true;
                          break;
                }
                if (tarr==true)
                          b[k]=a[k][0];
                else
                          b[k]=a[k][m-1];
   }
   cout<<"Your result"<<endl;
   for(k=0; k<m; k++)
     cout<<"b["<<k<<"]="<<b[k]<<"\t";
   cout<<endl;
   return 0;
}
```

Ծրագրավման արդյունքի օրինակ

a[0][0]=2
a[0][1]=3
a[0][2]=5
a[0][3]=9
a[1][0]=3
a[1][1]=-5
a[1][2]=-8
a[1][3]=3
a[2][0]=12
a[2][1]=-8
a[2][2]=3
a[2][3]=7
a[3][0]=3
a[3][1]=6
a[3][2]=5
a[3][3]=9

2 3 5 9
3 -5 -8 3
12 -8 3 7
3 6 5 9
Your result
b[0]=9 b[1]=3 b[2]=12 b[3]=9

2. Հաշվել արտահայտության արժեքը

$$\sum_{k=1}^{8} \frac{k!+6}{\frac{1}{3}+\frac{1}{5}+\ldots+\frac{1}{2k+1}}$$

Լուծում

```
#include <iostream.h>
int main()
{
/* k փոփոխականը և նրա ֆակտորյալը, որը կպահվի fact փոփոխականի մեջ հայտարարում ենք առանց նշանի կարճ ամբողջ տիպի: Լրիվ s գումարը, և կոտորակի հայտարարի 1/3+1/5+...+1/(2k+1) գումարը (որը կպահենք gumar փոփոխականի մեջ) կհայտարարենք double տիպի: */
   unsigned short int k, fact;
   double s, gumar;;
   for (s=0, fact=1, gumar=0, k=1; k<=8; k++)
     {
                fact=fact*k;
/* Տիպերի համապատասխանության նպատակով 1/(2*k+1) –ից առաջ գրում ենք է (double) */
                gumar=gumar+(double)1/(2*k+1);
```

```
/* Տիպերի համապատասխանության նպատակով (fact+6)/gumar –ից առաջ
գրում ենք է (double) */
                s=s+(double)(fact+6)/gumar;
    }
    cout<<"s="<<s<<endl;
    return 0;
}
```

Ծրագրկման արդյունքը

```
s=43251.1
```

3. **Տրված է Machine բազային կլասը՝ նրանում Move() մաքուր վիրտուալ ֆունկցիան: Մշակեք ծրագիր, որը կներկայացնի այդ ֆունկցիայի իրականացում 4 տարբեր ձևերով:**

Լուծում

Նախ կծանոթենք Machine կլասը, կստանանք 4 տարբեր ժառանգ կլասներ և նրանց հիմքի վրա կսահմանենք օբյեկտներ, որոնց հասցեները կպահենք բազային կլասի ցուցչի մեջ: Վերջինիս օգնությամբ կկանչենք ածանցյալ կլասի վերասահմանված մեթոդի համապատասխան տարբերակը:

```
#include <iostream.h>
//Սահմանում ենք Machine կլասը
class Machine
{
public:
// Հայտարարում ենք Move() մաքուր վիրտուալ ֆունկցիան
    virtual void Move()=0;
};
// Ածանցում ենք Machine կլասը ստանում ենք Bmw ժառանգ կլասը
class Bmw:public Machine
{
public:
// Վերասահմանում ենք Move() մեթոդը
     void Move(){cout<<"This is a BMW machine"<<endl;}
};
// Ածանցում ենք Machine կլասը ստանում ենք Cadilack ժառանգ կլասը
class Cadilack:public Machine
{
public:
// Վերասահմանում ենք Move() մեթոդը
        void Move(){cout<<"This is a Cadilack machine"<<endl;}
};
// Ածանցում ենք Machine կլասը ստանում ենք Mersedes ժառանգ կլասը
class Mersedes:public Machine
{
```

```
public:
// Վերասահմանում ենք Move() մեթոդը
        void Move(){cout<<"This is a Mersedes machine"<<endl;}
};
// Ածանցում ենք Machine կլասը ստանում ենք Lexus ժառանգ կլասը
class Lexus:public Machine
{
public:
// Վերասահմանում ենք Move() մեթոդը
        void Move(){cout<<"This is a Lexus machine"<<endl;}
};
//Հիմնական ծրագիր
int main()
{
    unsigned short int choice;
// Հայտարարում ենք Machine տիպի p ցուցիչը
    Machine *p;
    cout<<"Insert the number 1, 2,3 or 4          =";
    cin>>choice;
switch (choice)
{
/* Հայտարարում ենք դինամիկ հիշողության մեջ Bmw տիպի օբյեկտ և նրա հասցեն վերագրում են p ցուցչին */
case 1: {p=new Bmw;
    break;}
/* Հայտարարում ենք դինամիկ հիշողության մեջ  Cadilack տիպի օբյեկտ և նրա հասցեն վերագրում են p ցուցչին */
case 2: {p=new Cadilack;
    break;}
/* Հայտարարում ենք դինամիկ հիշողության մեջ  Mersedes տիպի օբյեկտ և նրա հասցեն վերագրում են p ցուցչին */
case 3: {p=new Mersedes;
    break;}
/* Հայտարարում ենք դինամիկ հիշողության մեջ  Lexus տիպի օբյեկտ և նրա հասցեն վերագրում են p ցուցչին */
case 4: {p=new Lexus;
    break;}
default: break;
};
// Կանչում ենք Move()  մեթոդը
p->Move();
//Ազատում ենք դինամիկ հիշողությունը
delete p;
return 0;
}
```

Թողարկման արդյունքի օրինակներ

1. Insert the number 1, 2,3 or 4 =2
This is a Cadilack machine

2. Insert the number 1, 2,3 or 4 =1
This is a BMW machine

3. Insert the number 1, 2,3 or 4 =3
This is a Mersedes machine

4. Insert the number 1, 2,3 or 4 =4
This is a Lexus machine

4. Գրեք ծրագիր որը 10-ական համակարգում ներմուծված թիվը կվևափոխի 2-ականի:

Լուծում

```
#include <iostream.h>
#include <math.h>
int main()
{
   int result,i,some_value;
   result=0;
   i=1;
   cout<<"Input some_value=";
   cin>>some_value;

   for(i=1; i<=int(log10(some_value)/log10(2)+1); i++)
      result=result+int(pow(10,i-1))*(int(some_value/pow(2,i-1))%2);
   cout<<"Your result is ="<<result<<endl;
return 0;
}
```

Թողարկման արդյունքի օրինակներ

```
Input some_value=11
Your result is =1011

Input some_value=123
Your result is =1111011
```

Գրականություն

1. Б.И.Березин, С.Б.Березин, “Начальный курс С и С++”, -М ДИАЛОГ МИФИ, 2001,-288 стр
2. К.Грегори под редакцией Г.Петриковца “Использование Visual С++ 6”, -М Специальное издание /Издательский дом “Вильямс”, 1999-864 стр
3. Х.М.Дейтел, П.Дж.Дейтел. Пер с англ. Под ред. А.Архенгельского “Как програ ммировать на С++ “ ЗАО, Издательство БИНОМ, Москва 2000
4. Ա.Հ.Նավասարդյան «C++ սկսնակների համար»: Ուսումնական ձեռնարկ: Երևան 2008թ.: Լիմուշ հրատարկչություն: 208 էջ:
5. Մ.Ղ.Սահրադյան, Գ.Վ.Դալլաքյան, Ա.Հ.Մուրադյան «Ծրագրավորման խնդիրներ»: Եր. Արևիկ 2003 - 56 էջ

ԲՈՎԱՆԴԱԿՈՒԹՅՈՒՆ

Նախաբան 2

ԳԼՈՒԽ 1 ՀԻՄՆԱԿԱՆ ՀԱՍԿԱՑՈՒԹՅՈՒՆՆԵՐ 3

§ 1-1 C++ լեզվում ծրագրերի մշակման էտապները 3

§ 1-2 Դուք արդեն ծրագիր եք գրում 3

§ 1-3 Փոփոխականներ և հաստատուններ 5

Փոփոխականների բազային տիպերը 5

Փոփոխականների հայտարարումը 6

Փոփոխականներին արժեքների վերագրումը 6

Հաստատուններ 7

§ 1-4 Արտահայտություններ և օպերատորներ 7

Բլոկներ կամ կոմպլեքս արտահայտություններ 8

Գործողություններ և օպերատորներ 8

Ինքրեմենտ և Դեքրեմենտ: Պրեֆիքս և պոստֆիքս 9

Համեմատման օպերատորներ 10

Պայմանական պարամետր if 10

Պայմանական պարամետր if/else 11

Օպերատոր switch 12

Տրամաբանական օպերատորներ 12

§ 1-5 Ֆունկցիաներ 13

Ֆունկցիաների սահմանումը 14

Ֆունկցիաների կանչը 14

Ֆունկցիաների գերբեռնումը 15

ԳԼՈՒԽ 2 ՕԲՅԵԿՏԱՅԻՆ ԿՈՂՄՆՈՐՈՇՄԱՆ ԾՐԱԳՐԱՎՈՐՄԱՆ ՏԱՐՐԵՐԸ 17

§ 2-1 Բազային կլասներ 17

Կլասների հայտարարումը 17

Օբյեկտների հայտարարումը 17

Կլասի անդամների հասանելիության ռեժիմները 18

Կլասի մեթոդների սահմանումը 19

Կոնստրուկտոր և դեստրուկտոր 19

Բարեկամական ֆունկցիաներ 20
Գերբեռնված ֆունկցիա-անդամներ 22
§ 2-2 Ցիկլեր 23
Առանց պայմանի անցման օպերատոր /Օպերատոր goto/ 23
Ցիկլերի կազմակերպումը while օպերատորի օգնությամբ 24
Ցիկլերի կազմակերպումը do…while օպերատորի օգնությամբ 24
Ցիկլերի կազմակերպումը for օպերատորի օգնությամբ 25
§ 2-3 Ցուցիչներ 26
Ցուցիչների սահմանումը 27
Օպերատոր new 28
Օպերատոր delete 28
§ 2-4 Հղումներ 29
Հղումների հայտարարումը 29
Հղումները որպես ֆունկցիայի արգումենտներ 30
§ 2-5 Զանգվածներ 33
Օբյեկտների զանգվածները 34
Բազմաչափ զանգվածներ 35
Սիմվոլների զանգվածները 36
ԳԼՈՒԽ 3 ԺԱՌԱՆԳԱԿԱՆՈՒԹՅՈՒՆ ԵՎ ՊՈԼԻՄՈՐՖԻԶՄ 38
§ 3-1 Ժառանգականություն 38
Կոնստրուկտորներ և դեստրուկտրներ 39
Վերաածակվող մեթոդներ 40
Վիրտուալ ֆունկցիաներ 41
Մաքուր վիրտուալ ֆունկցիաներ և աբստրակտ տիպեր 43
§ 3-2 Պոլիմորֆիզմ 45
Բազմակի ժառանգման մեխանիզմները 46
Վիրտուալ բազային կլասներ 48
Հավելված 1
Ծրագրերի կազմումը, կոմպիլացիան, կատարողական ֆայլերի ստացումը և թողարկումը Microsoft Visual C++ 6.0 միջավայրում: 49
Հավելված 2
Ա . Տողերի հետ աշխատող ֆունկցիաներ 50
Բ . Մաթեմատիկական ֆունկցիաներ
Հավելված 3

Ստուգողական աշխատանքների օրինակներ 51
Օգտակար հղումներ 67

Նշումների համար

ԱՐՄԻՆԵ ՀՐԱՉԻԿԻ ՆԱՎԱՍԱՐԴՅԱՆ

C++ ԾՐԱԳՐԱՎՈՐՄԱՆ ԼԵԶՈՒ

ուսումնամեթոդական ձեռնարկ
հեռակա ուսուցմամբ ուսանողների համար

Խմբագրումը և համակարգչային ձևավորումը
Ա.Նավասարդյանի

Երաշխավորված է տպագրության 14.07.2009թ

Տպագրության եղանակը
Ֆորմատ 60x84 1/16, թուղթը օֆսեթ , N1
Ծավալը 4,75 տպ մամուլ:
Տպաքանակը 300

Տպագրված է «ՄՀՄ գրատուն» տպագրատանը: