1. Օբյեկտ-կողմնորոշված ծրագրավորմանն հատկությունները:

Օբյեկտային կողմնորոշված ծրագրավորումը (ՕԿԾ կամ OOP)՝ ծրագրերի ստեղծման նոր մոտեցում է: Հաշվիչ տեխնի-կայի զարգացմանը զուգահեռ առաջանում էին նաև ծրագրա-վորման նոր մեթոդներ: Յուրաքանչյուր փուլում ստեղծվում էր նոր մոտեցում, որը օգնում էր ծրագրավորողներին հաղթել ծրագրերի բարդացման հետ կապված դժվարությունները: Սկզբնական փուլում ծրագրերը կազմվում էին մեկ ամ-բողջական մոդուլի տեսքով: Սակայն, երբ ծրագրի չափսը հասնում էր որոշակի երկարության, այդպիսի ոչ ստրուկտու-րային ծրագիրը դառնում էր անկարող: Անհրաժեշտություն առաջացավ նոր մեթոդներ մշակել: Այդպիսով հաջորդ փու-լում ստեղծվեց ստրուկտուրային ծրագրավորում (structured programming language): Ստրուկտուրային ծրագրավորման իմաստը կայանում է ծրագիրը բաղադրամասերի բաժանման հնարավորության մեջ: Այդ բաղադրամասերը իրենցից ներ-կայացնում են ավտոնոմ ենթածրագրեր, որոնք թույլ են տալիս օգտագործել լոկալ փոփոխականներ և ռեկուրսիա: Ծրագրավորման հաջորդ փուլը դարձավ օբյեկտային կողմնորոշված ծրագրավորումը: ՕԿԾ-ն իր մեջ ընդգրկում է ստրուկտուրային ծրագրավորման լավագույն հնարավորու-թյունները, դրանց ավելացնելով օբյեկտային ծրագրավորման հզոր միջոցների համակարգ: ՕԿԾ-ն ծրագիրը բաժանում է բա-ղադրամասերի, որոնցից յուրաքանչյուրը դառնում է ինքնուրույն օբյեկտ, որը պարունակում է միայն այդ օբյեկտին պատկանող կոդերը և տվյալները: Այդ դեպքում ամբողջ ծրագիրը դառնում է ավելի պարզ և կառավարելի:

Յուրաքանչյուր ոք, ով սկսում է օգտագործել ՕԿԾ, առա-ջինը պետք է սկսի օգտագործել օբյեկտային մտածելակերպ: ՕԿԾ-ի ամենամեծ խնդիրը օբյեկտային մտածելակերպ ձեռք բերելն է: Առանց օբյեկտային մտածելակերպի հնարավոր չէ օգտագործել ՕԿԾ-ի ամբողջ ուժը և հզորությունը: ՕԿԾ-ն հիմնված է դասի (class) և օբյեկտի (object) հասկացությունների վրա: Դասը սահմանում է այն կառուցվածքն ու պահվածքը (տվյալներ և կոդ), որն օգտագործվելու է օբյեկտների կողմից: Հենց այս պատճառով էլ, երբեմն, օբյեկտները կոչվում են դասի նմուշներ: Այսպիսով, դասը տրամաբանական կառուցվածքն է, իսկ օբյեկտը՝ նրա ֆիզիկական մարմնավորումը:
ՕԿԾ-ի երեք հիմնական հասկացություններն են՝ ինկապսուլացիա, ժառանգականություն և պոլիմորֆիզմ:
Ինկապսուլացիա: Ինկապսուլացիան այն մեխանիզմն է, որը կապում է կոդը տվյալների հետ (որոնք այն օգտագործում է)՝ պաշտպանելով այդ երկուսը արտաքին ազդեցությունից: Ինկապսուլացիան կարելի է համարել պաշտպանիչ շերտ, որը պաշտպանում է կոդը և տվյալները այլ արտաքին կոդի կող-մից շահագործումից:
Պոլիմորֆիզմ: Պոլիմորֆիզմը հնարավորություն է տա-լիս օգտագործել նույն ինտերֆեյսը դասի միանման, բայց տեխնիկապես տարբեր գործողությունների համար: Յուրա-քանչյուր գործողություն կախված էորոշակի իրադրությունից:
Ժառանգականություն: Այն գործընթացը, որի ժամանակ մի օբյեկտը ստանում է մեկ այլ օբյեկտի հատկությունները, կոչվում է ժառանգականություն: Ժառանգականության դեպ-քում մի օբյեկտը մասնակի կամ ամբողջությամբ ժառանգում է

մեկ այլ օբյեկտի հատկությունները, և կարող է ունենալ իրեն բնորոշ հատկությունները:
Երբեմն ՕԿԾ-ի հիմնական հասկացություններին են դասում նաև աբստրակցիան: Այն պաշտոնապես չի համար-վում ՕԿԾ-ի հիմնական գաղափարներից մեկը, սակայն ոչ պակաս կարևոր նշանակություն ունեցող գաղափարէ:
Աբստրակցիա: Աբստրակցիան մի միջոց է, որով առանձ-նացվում են օբյեկտի կարևոր բնութագրերը, դիտարկումից դուրս թողնելով ոչ կարևորները: Հետևաբար, աբստրակցիան բոլոր այդպիսի բնութագրերի համախումբն է:

2. Դասի հասկացությունը:

Դասը C++-ի հիմնական հասկացություններից մեկն է: Դասերը հնարավորություն են տալիս ամբողջությամբ իրա-կանացնել օբյեկտային կողմնորոշմամբ ծրագրավորման սկզբունքները: Կարելի է ասել, որ դասը կյանքում հանդիպող խնդրի ծրագրային մոդելն է: Այն հնարավորություն է տալիս խնդիրը ներկայացնել որպես հատկանիշների, այսինքն՝ բնու-թագրիչների և ֆունկցիաների համախումբ: Դա այն միջոցն է, որը հնարավորություն է տալիս ստեղծել օբյեկտ: Դասի նկա-րագրությունից հետո այն արդեն կարող է օգտագործվել տար-բեր ծրագրերում որպես բազմակի օգտագործման ծրագրային կոդ: Դրա համար անհրաժեշտ է ստեղծել դասի օբյեկտը: Դասի օբյեկտը ստեղծվում է այն հայտարարելու պահին օպե-րացիոն համակարգի կողմից նրան ռեալ հիշողություն հատ-կացնելով: Օբյեկտի ստեղծման պահին սկսում է աշխատել դասի հատուկ ֆունկցիան, որը կոչվում է կոնստրուկտոր: Որպես կանոն, կոնստրուկտորն ունենում է դասի անունը և չի վերադարձնում արժեք, այսինքն՝ չի կարող ավարտվել return օպերատորով: Մնացած բոլոր հատկանիշներով կոնստրուկ-տորը նման է մնացած ֆունկցիաներին: Իհարկե, դասը կարող է և չունենալ կոնստրուկտոր, այդ դեպքում օպերացիոն հա-մակարգն ինքն է հատկացնում օպերատիվ հիշողություն օբյեկ-տին: Բայց, եթե անհրաժեշտ է դասի օբյեկտին տալ որոշակի նախնական արժեքներ, ինչպես նաև ստուգել այդ արժեքների ճիշտ լինելը, ապա դա արվում է կամ կոնստրուկտորի, կամ որևիցե ֆունկցիայի օգնությամբ: Ընդհանուր դեպքում դասի սահմանումը իրականացվում է հետևյալ կերպ՝

class <անուն>

{private:

<անդամ տվյալներ>

<անդամ ֆունկցիաներ>

protected:

<անդամ տվյալներ>

<անդամ ֆունկցիաներ>

public:

<անդամ տվյալներ>

<անդամ ֆունկցիաներ>

} [օբյեկտների ցուցակ];

որտեղ

class-ը ծառայողական բառ է,

<անուն>-ը թույլատրված իդենտիֆիկատոր է,

[]-ը նշանակում է ոչ պարտադիր բաղադրամաս:

private, protected, public անդամ տվյալների և անդամ ֆունկցիաների հասանելիության մոդիֆիկատորներ են.

private(փակ) տարրերը հասանելի են միայն դասի մարմ-նում,

protected(պաշտպանված) տարրերը օգտագործվում են դասը ժառանգելիս,

public(բաց) տարրերը մատչելի են ցանկացած ֆունկ-ցիայում:

3. Դասի կոնստրուկտոր և դեստրուկտոր:

Սովորաբար, օբյեկտի ստեղծման պահին անհրաժեշտ է լինում սկզբնարժեքավորել փոփոխականները կամ կատարել ինիցիալացում: C++-ում օբյեկտի ինիցիալացումը պարտադիր պահանջ է: Այդ նպատակով դասում նախատեսված է հատուկ ֆունկցիա, որը հնարավորություն է տալիս օբյեկտը ինիցիա-լացնել հենց ստեղծման պահին: Ֆունկցիան կոչվում է կոնստ-

րուկտոր (constructor function) կամ կառուցիչ: Կոնստրուկտորի և դասի անունները պետք է համընկնեն: Օբյեկտին անհրաժեշտ ցանկացած ինիցիալացում կոնստրուկտորի առկայության դեպ-քում կատարվում է ավտոմատորեն, օբյեկտի ստեղծման պա-հին: Կոնստրուկտորը չունի վերադարձվող արժեքի տիպ:

```
class myclass
{private:
int a;
```

```
public:

myclass(); //լռելիությամբ կոնստրուկտոր

void show();

};

myclass :: myclass()

{cout<<"Inside of constructor\n";

a=10;

}

void myclass :: show()

{cout<<a<<"\n";

}

int main()

{myclass ob;

ob.show();

return 0;

}
```

Օրինակում a-ի արժեքը ինիցիալացվում է myclass() կոնստրուկտորի օգնությամբ, որի կանչը իրականացվում է ob օբյեկտի ստեղծման պահին:

Դասերի մեջ օգտագործվում է նաև կոնստրուկտորին հակառակ գործողություն կատարող ֆունկցիա, որը կոչվում է դեստրուկտոր (destructorfunction) կամ փլուզիչ: Դեստրուկ-

տորը հեռացնում է օբյեկտը նրա օգտագործման տիրույթից դուրս գալու պահին: Դեստրուկտորի անվանումը նույնպես համընկնում է դասի անվան հետ, սակայն նրան նախորդում է ~ (թիլդա) սիմվոլը: Դեստրուկտորը չունի վերադարձվող արժեքի տիպ և չի կարող վերաբեռնվել:

Կոնստրուկտորը և դեստրուկտորը կանչվում են ավտոմատորեն, նայած, թե դասի օբյեկտը որպես տվյալ երբ է ստեղծվում և երբ է հեռացվում օպերատիվ հիշողու-թյունից, իսկ ընդհանրապես սկզբում կանչվում է կոնստրուկ-տորը, հետո՝ դեստրուկտորը:

```
class myclass

{private:

int a;

public:

myclass(); // կոնստրուկտոր

~myclass(); // դեստրուկտոր

void show();

};
```

4. Դասերի ժառանգականություն:

Դասերի կարևոր հատկություններից է ժառանգականու-թյունը: Այն թույլ է տալիս ստեղծել մի օբյեկտ, որը հիմնված է լինելու մեկ այլ օբյեկտի վրա և որը պարունակելու է իր սեփական ու ժառանգված օբյեկտի անդամները միասին: Քանի որ ժառանգականությունը ծրագրավորման մեջ այն միջոցն է, որի օգնությամբ մեկ օբյեկտը կարող է ձեռք բերել մեկ այլ օբյեկտի հատկությունները, հնարավոր է պարզ օբյեկտներից ստանալ ավելի բարդ օբյեկտներ: Սկզբնական դասը կկոչվի բազային, իսկ նրանից ստացվածը՝ ժառանգ կամ ածանցյալ դաս: Սովորաբար, ժառանգ դասերը ավելի որոշակի են և կողմնորոշված են առանձին խնդիրներին: Դրանք կարող են պահպանել կամ փոփոխել բազային դասի անդամ տվյալները կամ անդամ ֆունկցիաները:

Ժառանգ դասը իր հերթին կարող է լինել բազային մեկ այլ դասի համար: Դրա արդյունքում ստացված ժառանգ դասը իր մեջ կընդգրկի բազային դասի տարրերը: Մեկ բազային դասից կարելի է ստանալ մի քանի ժառանգ դասեր: Հետևա-բար, հնարավոր է դասերի ժառանգականության հիերարխիկ կառուցվածք: Այն դասերը, որոնք ստեղծվում են մեկ այլ դասերից, ժառանգում են այդ դասերի տեսանելի անդամները: Դա նշանակում է, որ եթե ժառանգվող դասը պարունակում է, օրինակ, A անունով անդամ, և ժառանգող դասը պարունակում է մի ինչ-որ B անդամ, ապա վերջնական դասը կպարունակի A և B անդամները:
Եթե մեկ դասից մեկ այլ դաս է ժառանգվում, օգտագործ-վում է հետևյալ գրելաձևը.
class <անուն1> : <հասանելիության ձև><անուն2>
{private:
<անդամ տվյալներ>
<անդամ ֆունկցիաներ>
protected:
<անդամ տվյալներ>
<անդամ ֆունկցիաներ>
public:
<անդամ տվյալներ>
<անդամ ֆունկցիաներ>
};
որտեղ
<անուն1> -ը ածանցյալ դասի անունն է,
<անուն2>-ը բազային դասի անունն է,
<հասանելիության ձև>-ը կամ մատչելիության մոդիֆի-կատորը՝ private, protected կամ public հասանելիության մոդի-ֆիկատորներից մեկն է:

5. Բազային դասեր և ածանցված դասեր: Օրինակներ:

Ժառանգումը օբյեկտային կողմնորոշմամբ ծրագրավորման հիմնական և կարևորագույն բնութագրերից է: Այն արդեն ճշգրտված ու շտկված ծրագրային ապահովման կրկնակի օգտագործման մի եղանակ է, որը նոր ստեղծվող դասերին թույլ է տալիս գոյություն ունեցող դասերից վերցնել նրանց հատկություններն ու ֆունկցիաները՝ դրանք հարստացնելով նոր հնարավորություններով: Դա նշանակում է, որ կարելի է ստեղծել մի համընդհանուր դաս, որը որոշում է իրար հետ հարաբերականորեն կապված տվյալների ընդհանուր բնութագրիչ գծերը: Այդ դասը կարող է ժառանգվել այլ ավելի յուրահատուկ դասերի կողմից, ավելացնելով միայն այնպիսի բնութագրիչներ, որոնք բնորոշ են ժառանգող դասին:

Այն դասը, որը ժառանգվում է կոչվում է ***բազային դաս*** : Այն դասը, որը ժառանգում է կոչվում է ***ածանցված դաս:*** Ածանցված դասը հետագայում կարող է օգտագործվել որպես բազային դաս այլ ածանցված դասերի համար՝ առաջացնելով դասերի աստիճանակարգում:

Երբ մի դասը ժառանգում է մյուսին, օգտագործվում է հետևյալ ընդհանուր տեսքը.

```
class    ածանցված_դաս : հաս_բնութագրիչ  բազային_դաս {
   //դասի մարմին
}
```

Երբ մի դասը ժառանգում է մյուսին, բազային դասի անդամները դառնում են ածանցված դասի անդամներ: Ածանցված դասում բազային դասի անդամների կարգավիճակը որոշվում է նրա ***հասանելիության բնութագրիչի (հաս_բնութագրիչ)*** միջոցով: Բազային դասի հասանելիության բնութագրիչը կարող է լինել **public, private** կամ **protected**: Եթե հասանելիության բնութագրիչը բացակայում է, ապա այն լռելությամբ համարվում է **private**՝ եթե ածանցված դասը **class** է, և **public**՝ եթե այն **struct** կամ **union** է:

Հետևյալ օրինակում **derived** տիպի օբյեկտները կարող են անմիջականորեն դիմել **base**-ի բաց անդամներին.

```
#include <iostream>
using namespace std;
class base {
   int i,j;
public:
   void set(int a, int b) {i=a;j=b;}
   void show() {cout<<i<<" "<<j<<"\n";}
};
class derived:public base {
   int k;
public:
   derived(int x) {k=x;}
   void showk() {cout<<k<<"\n";}
};
int main() {
   derived ob(3);
   ob.set(1,2);   //դիմում է կատարվում բազայինի անդամին
   ob.show();     //դիմում է կատարվում բազայինի անդամին
   ob.showk();    //դիմում է կատարվում ածանցվածի անդամին
   return 0;
}
```

6. Օրինակ՝ ուղղանկյուն և քառակուսի դասերի ժառանգում:

```
#include <iostream>
using namespace std;
```

```
// Հիմք դաս
class Rectangle {
protected:
   int width, height;

public:
   Rectangle(int w, int h) {
      width = w;
      height = h;
   }

   int area() {
      return width * height;
   }
};

// Ժառանգ դաս
class Square : public Rectangle {
public:
   Square(int side) : Rectangle(side, side) {}
};

int main() {
   Rectangle r(5, 3);
   Square s(4);

   cout << "Ուղղանկյան մակերեսը = " << r.area() << endl;
   cout << "Քառակուսիի մակերեսը = " << s.area() << endl;

   return 0;
}
```

7. Բազային դասի հասանելիության կառավարում: Օրինակներ:

Երբ մի դասը ժառանգում է մյուսին, օգտագործվում է հետևյալ ընդհանուր տեսքը.

```
class    ածանցված_դաս : հաս_բնութագրիչ  բազային_դաս {
   //դասի մարմին
}
```

Երբ մի դասը ժառանգում է մյուսին, բազային դասի անդամները դառնում են ածանցված դասի անդամներ: Ածանցված դասում բազային դասի անդամների կարգավիճակը որոշվում է նրա ***հասանելիության բնութագրիչի*** (***հաս_բնութագրիչ***) միջոցով: Բազային դասի հասանելիության բնութագրիչը կարող է լինել **public**, **private** կամ **protected**: Եթե հասանելիության բնութագրիչը բացակայում է, ապա այն լռելությամբ համարվում է **private**՝ եթե ածանցված դասը **class** է, և **public**՝ եթե այն **struct** կամ **union** է:

Երբ բազային դասի հասանելիության բնութագրիչը **public** է, բազային դասի բոլոր բաց անդամները ածանցված դասի համար դառնում են բաց անդամներ, և բազային դասի բոլոր պաշտպանված անդամները դառնում են պաշտպանված՝ ածանցված դասի համար: Բազային դասի փակ անդամները շարունակում են մնալ հասանելի միայն բազային դասի անդամ-ֆունկցիաներին և ընկեր ֆունկցիաներին: Ածանցված դասի անդամներին նրանք անմիջականորեն հասանելի չեն: Դա շատ բնական է և իմաստալից, որովհետև հակառակ դեպքում կխախտվեր ՕԿԾ-ի թաղանթապատման սկզբունքը:

```
#include <iostream>
using namespace std;
class base {
   int i,j;
public:
   void set(int a, int b) {i=a;j=b;}
   void show() {cout<<i<<" "<<j<<"\n";}
};
class derived:public base {
   int k;
public:
   derived(int x) {k=x;}
   void showk() {cout<<k<<"\n";}
};
int main() {
   derived ob(3);
   ob.set(1,2);   //դիմում է կատարվում բազայինի անդամին
   ob.show();     //դիմում է կատարվում բազայինի անդամին
   ob.showk();    //դիմում է կատարվում ածանցվածի անդամին
   return 0;
}
```

Երբ բազային դասը ժառանգվում է՝ օգտագործելով **private** հասանելիության բնութագրիչը, այդ դասի բոլոր **public** և **protected** անդամները դառնում են **private** անդամներ ածանցված դասի համար: Դա նշանակում է, որ նրանք հասանելի են ածանցված դասի անդամներին, բայց հասանելի չեն ծրագրի այն մասերին, որոնք բազային կամ ածանցված դասի անդամներ չեն:

8. Ժառանգ դասի ստեղծում համապատասխան անդամներով ու մեթոդներով:

Կլասերի կարևոր հատկություններից է՝ ժառանգականությունը: Այն թույլ է տալիս

ստեղծել մի օբյեկտ, որը հիմնված է լինելու մեկ այլ օբյեկտի վրա, և որը պարունակելու է իր սեփական ու ժառանգված օբյեկտի անդամները միասին:
Օրինակ, դիցուք մենք ուզում ենք հայտարարել ուղղանկյուն՝ **CRectangle**, և եռանկյուն՝ **CTriangle**, կլասերը: Նրանք երկուսն էլ ունեն ընդհանուր հատկություններ, ինչպես օրինակ նրանք երկուսն էլ կարող են նկարագրվել երկու մեծություններով՝ բարձրությամբ և հիմքով:
Կլասերի աշխարհում **CRectangle** և **CTriangle** կլասերը կարող են նկարագրվել ընդհանուր՝ **CPolygon** կլասով:

CPolygon կլասը պարունակելու է այն անդամները, որոնք ընդհանուր են բոլոր բազմանկյունների համար, մեր դեպքում՝ բարձրությունը (height) և լայնությունը (width):

Այն կլասերը, որոնք ստեղծվում են մեկ այլ կլասերից, ժառանգում են այդ կլասերի տեսանելի անդամները: Դա նշանակում է, որ եթե ժառանգվող կլասը

պարունակում է **A** անունով անդամ, և ժառանգող կլասը պարունակում է մի ինչ-որ **B** անդամ, ապա վերջնական կլասը կպարունակի **A** և **B** անդամները:
Որպեսզի մի կլասը ժառանգի մեկ այլ կլասին, պետք է օգտագործենք **:** օպերատորը ժառանգող կլասի հայտարարության մեջ հետևյալ կերպ.
class կլասի անուն**: public** ժառանգվող կլասի անուն**;**
Այստեղ **public**–ը կարող է փոխարինվել հետևյալ տեսանելիության նշիչներով՝ **protected** կամ **private**:

```
#include <iostream.h>
class CPolygon {
protected:
int width, height;
public:
void set_values (int a, int b)
{ width=a; height=b;}
};
class CRectangle: public CPolygon {
public:
int area (void)
{ return (width * height); }
};
class CTriangle: public CPolygon {
public:
int area (void)
{ return (width * height / 2); }
};
int main () {
CRectangle rect;
CTriangle trgl;
rect.set_values (4,5);
trgl.set_values (4,5);
cout << rect.area() << endl;
cout << trgl.area() << endl;
```

return 0;
}

9. Կոնստրուկտորների և դեստրուկտորների ժառանգում:

Այժմ տեսնենք, թե երբ են կանչվում բազային դասի ու ածանցված դասի կոնստրուկտոր և դեստրուկտոր ֆունկցիաները, և ինչպես կարելի է պարամետրեր փոխանցել բազային դասի կոնստրուկտորին:

Ինչպես գիտենք, երբ բազային դասը ժառանգվում է **public** բնութագրիչով, ածանցված դասի օբյեկտը միաժամանակ դառնում է նաև բազային դասի օբյեկտ: Սա է դասերի միջև աստիճանակարգված հարաբերությունների էությունը: Որպեսզի հասկանալի լինի, թե ածանցված դասի օբյեկտն ստեղծվելիս և ոչնչանալիս բազային դասի ու ածանցված դասի կոնստրուկտոր և դեստրուկտոր ֆունկցիաները ինչ հերթականությամբ են կատարվում, դիտարկենք հետևյալ ծրագիրը, որը պարզապես ստեղծում և ոչնչացնում է **derived** դասի **ob** օբյեկտը.

```
#include <iostream>
using namespace std;
class base {
public:
   base() {cout<<"Բազայինի ստեղծում\n";}
   ~base() {cout<<"Բազայինի ոչնչացում \n";}
};
class derived:public base {
public:
   derived() {cout<<"Ածանցվածի ստեղծում \n";}
   ~derived() {cout<<"Ածանցվածի ոչնչացում \n";}
};
int main() {
   derived ob;
//ob օբյեկտը ստեղծելուց և ոչնչացնելուց բացի ոչինչ չանել
   return 0;
}
```

Ծրագիրը կարտածի հետևյալը.

```
Բազայինի ստեղծում
Ածանցվածի ստեղծում
Ածանցվածի ոչնչացում
Բազայինի ոչնչացում
```

Ընդհանրացնելով նշված օրինակի միտքը, կարելի է ասել, որ երբ ածանցված դասի օբյեկտ է ստեղծվում, սկզբից կանչվում է բազային դասի կոնստրուկտորը , որն սկզբնարժեքավորում է այդ օբյեկտի բազային դասին պատկանող մասը: Դրանից հետո կանչվում է ածանցված դասի կոնստրուկտորը: Այն ավարտում է ածանցված դասի օբյեկտի սկզբնարժեքավորումը, վերջնականապես ստեղծելով այդ օբյեկտը: Երբ ածանցված օբյեկտը ոչնչանում է, ապա սկզբից կանչվում է նրա դեստրուկտորը, որից հետո կանչվում է բազային դասի դեստրուկտորը: Յուրաքանչյուր դեստրուկտոր ոչնչացնում է օբյեկտի այն մասը, որն ստեղծվել էր ածանցված կամ բազային դասերի համապատասխան կոնստրուկտորով: Այսպիսով, կարելի է ա-

սել, որ կոնստրուկտոր ֆունկցիաները կատարվում են իրենց ածանցման կարգով, իսկ դեստրուկտոր ֆունկցիաները՝ ածանցման հակառակ կարգով:

10. Օրինակ՝ վեկտոր և մատրիցա դասերի ժառանգում:

```
#include <iostream>
using namespace std;
class vector{
protected:
int n;
double x[20];
public:
vector();
double min();
~vector();
};
class matrix:public vector{
protected:
int m;
double y[5][5];
public:
matrix();
void F(double Mn);
37
~matrix();
};
vector::vector()
{cout<<"vector's constructor"<<endl;
cout<<"n=";cin>>n;
for(int i=0;i<n;i++)cin>>x[i];
}
double vector::min()
{double Mn;
Mn=x[0];
for(int i=1;i<n;i++)
if(x[i]<Mn)Mn=x[i];
return Mn;
}
vector::~vector()
{cout<<"vector's destructor"<<endl;}
matrix::matrix()
{cout<<"matrix's constructor"<<endl;
cout<<"m=";cin>>m;
for(int i=0;i<m;i++)
for(int j=0;j<m;j++)
cin>>y[i][j];
}
void matrix::F(double Mn)
{int i,j;
for(i=0;i<m;i++)y[i][i]=Mn;
for(i=0;i<m;i++)
{
38
```

```
for(j=0;j<m;j++)
cout<<y[i][j]<<" ";
cout<<endl;
}
}
matrix::~matrix()
{cout<<"matrix's destructor"<<endl;}
int main()
{matrix ob;
double Mn;
Mn=ob.min();
ob.F(Mn);
return 0;
}
```

Պատասխանը՝

```
vector's constructor
n=10
6 8 3 1 5 4 8 6 7 3
matrix's constructor
m=3
6 4 5
3 5 4
7 8 9
1 4 5
3 1 4
7 8 1
matrix's destructor
vector's destructor
```

11. Օրինակ՝ եռանկյուն և բուրգ դասերի ժառանգում:

```
#include <iostream>

#include <cmath>

using namespace std;



// Հիմք դաս

class Triangle {
protected:
    double base;
    double height;

public:
    Triangle(double b, double h) {
        base = b;
```

```
        height = h;
    }

    double area() {
        return 0.5 * base * height;
    }
};

// Ժառանգ դաս
class Pyramid : public Triangle {
private:
    double pyramidHeight;

public:
    Pyramid(double b, double h, double ph)
        : Triangle(b, h) {
        pyramidHeight = ph;
    }

    double volume() {
        return (1.0 / 3.0) * area() * pyramidHeight;
    }
};

int main() {
    Triangle t(6, 4);
    cout << "Եռանկյան մակերեսը = " << t.area() << endl;

    Pyramid p(6, 4, 10);
    cout << "Բուրգի ծավալը = " << p.volume() << endl;

    return 0;
}
```

12. Ժառանգ դասի օբյեկտի հայտարարում:

Ժառանգ դասի օբյեկտը այն օբյեկտն է, որը ստեղծված է **ժառանգ (derived) դասից**, և որը ավտոմատ կերպով ունի **հիմք դասի բոլոր հասանելի հատկությունները և մեթոդները**:

Ժառանգ դասի օբյեկտ է կոչվում այն օբյեկտը, որը ստեղծված է մի դասից, որը **ժառանգում է (inherits)** մեկ այլ դասից:

```
class Derived : public Base {
    // ժառանգ դասի մարմին
};
```

- `Base` → բազային դաս
- `Derived` → ժառանգ դաս
- `public` → ժառանգման տիպը

Ինչպես են հայտարարում ժառանգ դասի օբյեկտը

```
Derived obj;
```

կամ կոնստրուկտորով՝

```
Derived obj(պարամետրեր);

Օրինակ
class Animal {
public:
    void speak() {
        cout << "Animal sound" << endl;
    }
};

class Dog : public Animal {
};

int main() {
    Dog d;       // ժառանգ դասի օբյեկտ
    d.speak();  // օգտագործում է հիմք դասի մեթոդը
}
```

Ժառանգ դասի օբյեկտը **կարող է օգտագործել բազային դասի public և protected անդամները**,
բայց **ոչ private**–ները:

Ժառանգ դասի օբյեկտ ստեղծելիս՝
1Կանչվում է **հիմք դասի կոնստրուկտորը**
2 Հետո՝ **ժառանգ դասի կոնստրուկտորը**

13. Օբյեկտին արժեքների փոխանցում: Օրինակներ:

Օբյեկտին արժեքների փոխանցումը նշանակում է **օբյեկտի դաշտերին (փոփոխականներին) արժեքներ տալը**:
C++–ում դա կարելի է անել մի քանի տարբեր եղանակով: Ստորև՝ **պարզ օրինակներով**:

Կոնստրուկտորի միջոցով

```
#include <iostream>
using namespace std;

class Student {
public:
  string name;
  int age;

  Student(string n, int a) {
    name = n;
    age = a;
  }
```

```
};

int main() {
   Student s("Անահիտ", 20);  // արժեքները փոխանցվում են օբյեկտին
   cout << s.name << " " << s.age;
}
```
Արժեքները փոխանցվում են օբյեկտ ստեղծելու պահին:
Set ֆունկցիայի միջոցով
```
class Rectangle {
private:
   int width, height;

public:
   void setValues(int w, int h) {
      width = w;
      height = h;
   }

   int area() {
      return width * height;
   }
};

int main() {
   Rectangle r;
   r.setValues(5, 4);  // փոխանցում ենք արժեքներ
   cout << r.area();
}
```
Օբյեկտի դաշտերին ուղիղ վերագրումով
```
class Point {
public:
   int x, y;
};

int main() {
   Point p;
   p.x = 3;
   p.y = 7;   // ուղիղ փոխանցում
}
```

15. Օբյեկտի ֆունկցիոնալության իրականացում:
Օբյեկտի ֆունկցիոնալության իրականացում նշանակում է, որ
օբյեկտը **ոչ միայն պահում է տվյալներ**, այլ նաև **կատարում է գործողություններ այդ տվյալների վրա**:
Այս գործողությունները իրականացվում են **դասի մեթոդներով (member functions)**:

Օբյեկտի ֆունկցիոնալությունը նրա կարողություններն են՝

- հաշվարկել
- փոխել վիճակ
- վերադարձնել արդյունք
- արձագանքել կանչերին

```
#include <iostream>
using namespace std;

class Rectangle {
private:
   int width, height;

public:
   void setValues(int w, int h) {
      width = w;
      height = h;
   }

   int area() {
      return width * height;
   }

   int perimeter() {
      return 2 * (width + height);
   }
};

int main() {
   Rectangle r;
   r.setValues(5, 3);

   cout << "Մակերես = " << r.area() << endl;
   cout << "Պարագիծ = " << r.perimeter() << endl;
}
```

- օբյեկտը **պահում է տվյալներ** (`width`, `height`)
- և **կատարում է գործողություններ** (`area()`, `perimeter()`)

16. get և set ֆունկցիաներ: Օրինակներ:

Քանի որ դասի անդամ տվյալները, սովորաբար, ունե-նում են հասանելիության փակ, իսկ անդամ ֆունկցիաները՝ բաց ռեժիմներ, դասի մեջ պետք է ընդգրկվեն փակ անդամ տվյալների ներմուծման/արտածման համար նախատեսված բաց անդամ ֆունկցիաներ:
Դիտարկենք օրինակներ:

```
class myclass
{private:
int a;
public:
void set(int num);
};
void myclass :: set(int num)
{a=num;}
int main()
{myclass ob;
int b;
cin>>b;
```

```
ob.set(b);
return 0;
}
```
Այստեղ սահմանված myclass դասի փակ անդամն է a փոփոխականը, որը set բաց անդամ ֆունկցիաի միջոցով ար-ժեք է ստանում main ֆունկցիայից:
```
class myclass
{private:
int a;
public:
void set(int num);
int get();
};
void myclass :: set(int num)
{a=num;}
int myclass :: get()
{return a;}
int main()
{myclass ob;
int b, c;
cout<<"b="; cin>>b;
ob.set(b);
c=ob.get();
cout<<"c="<<c;
return 0;
}
```
Այս օրինակում set անդամ ֆունկցիայի միջոցով դասի a փակ անդամ տվյալը ստանում, իսկ get անդամ ֆունկցիայի՝ վերադարձնում է արժեքը main ֆունկցիային:

17. Բազային դասի անդամների վերասահմանումը ածանցված դասերում:

Երբ բազային և ածանցված դասերում նկարագրվում են նույն անունով անդամ-ֆունկցիաներ, որոնց պարամետրերի քանակը և տիպերը համընկնում են, ապա ածանցված դասի ֆունկցիայի տարբերակը ծածկում է բազային դասի համապատասխան ֆունկցիան: Դա նշանակում է, որ եթե նման ֆունկցիան կանչվի ածանցված դասի օբյեկտի կողմից, ապա կկատարվի այդ ֆունկցիայի վերասահմանված տարբերակը: Որպեսզի ածանցված դասին հասանելի լինի բազային դասի ֆունկցիայի տարբերակը, պետք է օգտագործել պատկանելիության գործողությունը:

Սովորաբար, ածանցված դասում բազային դասի անդամ-ֆունկցիայի վերասահմանման դեպքում ընդունված է կանչել այդ ֆունկցիայի բազային դասի տարբերակը, որից հետո կատարել որոշ լրացուցիչ գործողություններ: Ընդ որում, բազային դասի անդամ-ֆունկցիան առանց պատկանելիության գործողության կանչելիս կարող է առաջանալ անվերջ ռեկուրսիա, որովհետև ածանցված դասի անդամ-ֆունկցիան իրականում կկանչի ինքն իրեն:

Օրինակ.

```
#include <iostream>
using namespace std;
class base {
int i;
public:
void set(int a)     { i=a; }
void show() {
  cout<<" Սա բազային դասի անդամն է"<<i<<"\n";
   }
};
class derived: public base {
int j;
public: derived (int x )     { j=x;}
   void show() {
  base::show();

    cout<<" Սա ածանցած դասի անդամն է"<<j<<"\n";
    }
};
int main ()     {
  derived ob(3);
  ob.set(1);
  ob.show();
  return 0;
}
```

Այս օրինակում ածանցված դասի show() անդամ-ֆունկցիան նախ դիմում է բազային դասի նույն անունով անդամ-ֆունկցիային, որից հետո կատարում է իր արտածումները:

18. Մի քանի բազային դասերի ժառանգում:

Ածանցված դասը կարող է ժառանգել երկու կամ ավելի բազային դասերի: Այս դեպքում ասում են, որ տեղի ունի *բազմակի ժառանգում*: Բազմակի ժառանգման ժամանակ օգտագործվում է հետևյալ ընդհանրացված հայտարարումը.

```
class ածանցված_դաս : հաս_բնութագրիչ բազային_դաս1,
                      հաս_բնութագրիչ բազային_դաս2,
                                . . .
                      հաս_բնութագրիչ բազային_դասN {
//... դասի մարմին
}
```

Այստեղ *բազային_դաս1, ... , բազային_դասN*-ը իրարից տարբեր բազային դասերի անուններ են, *հաս_բնութագրիչ*-ը հասանելության բնութագրիչն է, որը յուրաքանչյուր դասի համար կարող է լինել տարբեր:

Հետևյալ օրինակում derived-ը ժառանգում է base1-ը և base2-ը.

```
#include <iostream>
using namespace std;
class base1 {
protected:
   int x;
public:
   void showx() {cout<<x<<"\n";}
};
class base2 {
protected:
   int y;
public:
   void showy() {cout<<y<<"\n";}
};
//ժառանգում է մի քանի դասերի
class derived:public base1,public base2 {
public:
   void set(int i,int j) {x=i;y=j;}
};
int main() {
   derived ob;
   ob.set(10,20);          //տրամադրվել է derived-ի կողմից
   ob.showx();             //այս մեկը՝ base1-ի կողմից
   ob.showy();             //իսկ սա՝ base2-ի կողմից
   return 0;
}
```

19. Բազմաձևության հասկացությունը: Օրինակներ:

ՕԿԾ-ում ***բազմաձևությունը*** («*մեկ ինտերֆեյս, բազմաթիվ մեթոդներ*») մի հատկություն է, երբ միևնույն անվան օգտագործմամբ կարող են հասանելի լինել ֆունկցիայի տարբեր իրականացումներ: Բազմաձևությունը C++-ում կարող է իրականանալ և՛ թարգմանության, և՛ կատարման ժամանակ: ***Թարգմանության ժամանակ իրականացվող բազմաձևությունը*** իրագործվում է վերաբեռնված ֆունկցիաների և գործողությունների միջոցով: ***Կատարման ժամանակ իրականացվող բազմաձևությունը*** կապված է ժառանգման և ***վիրտուալ ֆունկցիաների*** հետ: Դա հաճախ անվանում են նաև ***դինամիկ բազմաձևություն:***

Լինում են 2 ձևի

Compile-time polymorphism

- function overloading
- operator overloading

Run-time polymorphism

- virtual functions
- override

```
#include <iostream>
using namespace std;

class Shape {
public:
   virtual double area() {
      return 0;
   }
};

class Rectangle : public Shape {
private:
   double w, h;

public:
   Rectangle(double width, double height) {
      w = width;
      h = height;
   }

   double area() override {
      return w * h;
   }
};

class Triangle : public Shape {
private:
   double b, h;

public:
   Triangle(double base, double height) {
      b = base;
      h = height;
   }

   double area() override {
      return 0.5 * b * h;
   }
};

int main() {
   Shape* s1 = new Rectangle(5, 4);
   Shape* s2 = new Triangle(6, 3);

   cout << s1->area() << endl;
```

```
    cout << s2->area() << endl;
}
```

19.1. Թաղանթապատման հասկացությունը։

Թաղանթապատում (Encapsulation) է կոչվում տվյալների և դրանց վրա գործող ֆունկցիաների **միասնական փակումը մեկ դասի մեջ** և արտաքին միջավայրից **ուղիղ հասանելիության սահմանափակումը**:

- Պաշտպանում է տվյալները
- Կանխում է սխալ օգտագործումը
- Դասը դարձնում է վերահսկելի

```
#include <iostream>
using namespace std;

class BankAccount {
private:
   double balance;   // թաքնված տվյալ

public:
   BankAccount(double b) {
      balance = b;
   }

   void deposit(double amount) {
      if (amount > 0)
         balance += amount;
   }

   double getBalance() {
      return balance;
   }
};

int main() {
   BankAccount acc(1000);
   acc.deposit(500);
   cout << acc.getBalance();
}
```

20. Վիրտուալ բազային դասեր։ Վիրտուալ ֆունկցիաներ և բազմաձևություն։

Վիրտուալ ֆունկցիաները այն անդամ-ֆունկցիաներն են, որոնք բազային դասում հայտարարվել են որպես **virtual**: Այդպիսի ֆունկցիաները սովորաբար վերասահմանվում են ածանցված դասում: Ֆունկցիայի վերասահմանումը ածանցված դասում ծածկում է այդ ֆունկցիայի բազային դասում հայտարարված տարբերակին: Բազային դասում հայտարարված վիրտուալ ֆունկցիաները բնութագրում են գործողությունների ընդհանուր դասը, և նախատեսում են ինտերֆեյսի տեսքը: Ածանցված դասում վիրտուալ ֆունկցիաների վերասահմանումը ապահովում է այդ ֆունկցիաների կողմից կատարվող իրական գործողությունները՝ կապված ածանցված դասի յուրահատկությունների հետ:

Երբ վիրտուալ ֆունկցիաներին դիմում են «նորմալ» ձևով, ապա նրանց վարքագիծը չի տարբերվում դասի այլ տիպի անդամ-ֆունկցիաներից: Վիրտուալ ֆունկցիաների կարևորությունը և դինամիկ բազմաձևության դրսևորման նրանց հնարավորությունն ի հայտ է գալիս այն ժամանակ, երբ նրանց դիմում են ցուցիչի միջոցով:

Ինչպես հայտնի է, բազային դասի ցուցիչը կարող է օգտագործվել այդ դասից ածանցված դասի օբյեկտ մատնանշելու համար: Երբ բազային ցուցիչը մատնանշում է վիրտուալ ֆունկցիա պարունակող որևէ ածանցված օբյեկտ, C++-ը հիմնվելով այդ ցուցիչի մատնանշած օբյեկտի տիպի վրա, որոշում է, թե այդ ֆունկցիայի ո՛ր տարբերակը պետք է կանչվի:

Դիտարկենք հետևյալ օրինակը.

```
#include <iostream>
using namespace std;
class base {
public:
   virtual void vfunc() {
         cout<<"Սա base-ի vfunc()-ն է:\n";
   }
};
class derived1:public base {
public:
   void vfunc() {
         cout<<" Սա derived1-ի vfunc()-ն է:\n";
   }
};
class derived2:public base {
public:
   void vfunc() {
         cout<<" Սա derived2-ի vfunc()-ն է:\n";
   }
};
int main() {
   base *p,b;
   derived1 d1;
   derived2 d2;
   p=&b;          //մատնանշում է base-ը
   p->vfunc();    //կանչվում է base-ի vfunc()-ը
   p=&d1;         //մատնանշում է derived1-ը
   p->vfunc();    //կանչվում է derived1-ի vfunc()-ը
   p=&d2;         //մատնանշում է derived2-ը
   p->vfunc();    //կանչվում է derived2-ի vfunc()-ը
   return 0;
}
```

Այս ծրագիրը կարտածի.

```
Սա base-ի vfunc()-ն է:
Սա derived1-ի vfunc()-ն է:
Սա derived2-ի vfunc()-ն է:
```

21. Մաքուր վիրտուալ ֆունկցիաներ: Բազմաձևության կիրառումը:

Մաքուր վիրտուալ ֆունկցիաներ

Ինչպես արդեն գիտենք, երբ վիրտուալ ֆունկցիան չի վերասահմանվում ածանցված դասում, ապա կատարվում է բազային դասում որոշված ֆունկցիան: Երբեմն հնարավոր չի լինում բազային դասում կայացնել վիրտուալ ֆունկցիայի իմաստալից որոշում: Բացի այդ, որոշ դեպքերում անհրաժեշտ է լինում համոզվել, որ բոլոր ածանցված դասերը վերասահմանում են վիրտուալ ֆունկցիան: Այսպիսի դեպքերում պետք է օգտվել ***մաքուր վիրտուալ ֆունկցիայից***, որի ընդհանուր տեսքը հետևյալն է.

```
virtual type ֆունկցիայի_անուն (պարամետրերի_ցուցակ) = 0;
```

Երբ վիրտուալ ֆունկցիան մաքուր է, ապա յուրաքանչյուր ածանցված դաս պետք է ունենա այդ ֆունկցիայի իր սեփական որոշումը, որպեսզի հնարավոր լինի ստեղծել այդ դասի օբյեկտ:

Հետևյալ օրինակում **number** բազային դասը պարունակում է **show()** մաքուր վիրտուալ ֆունկցիան: Ածանցված դասերը վերասահմանում են **show()**-ն այնպես, որ **val** փոփոխականի արժեքն ամեն անգամ արտածվում է համապատասխան հիմքով (16-ական, 10-ական կամ 8-ական):

Ինչպես արդեն նշվել է, ՕԿԾ-ի հիմնական տեսակետներից մեկը «*մեկ ինտերֆեյս, բազմաթիվ մեթոդներ*» սկզբունքն է: Դա նշանակում է, որ կարելի է որոշել հաստատուն ինտերֆեյսով միասնական գործողությունների մի համընդհանուր դաս, իսկ յուրաքանչյուր ածանցված դասում տալ իրեն բնորոշ մեթոդը: Այլ կերպ ասած, բազային դասը կարող է օգտագործվել ընդհանուր ինտերֆեյսի բնույթը որոշելու համար, որից հետո յուրաքանչյուր ածանցված դաս իրագործում է իր կողմից օգտագործվող տվյալների տիպին բնորոշ յուրահատուկ գործողություններ:

22. Օբյեկտ կողմնորոշված մոդելը ծրագրային ապահովում ստեղծելիս։

Օբյեկտ կողմնորոշված մոդելը ծրագրավորման մոտեցում է,
որտեղ ծրագիրը կառուցվում է **օբյեկտների** հիման վրա,
ոչ թե առանձին ֆունկցիաների:

Ծրագիրը դիտվում է որպես իրական աշխարհի օբյեկտների հավաքածու,
յուրաքանչյուրը ունի՝

- **վիճակ (տվյալներ)**
- **վարք (գործողություններ)**

OOP-ի հիմնական բաղադրիչներն են

Դաս (Class)

Օբյեկտի նկարագրություն կամ «շաբլոն»

```
class Car {

   int speed;
```

```
    void drive();
};
```

Օբյեկտ (Object)

```
Car myCar;
```

Թաղանթապատում (Encapsulation)

Տվյալների պաշտպանություն

```
private:
    int speed;
```

Ժառանգում (Inheritance)

Նոր դաս՝ արդեն գոյություն ունեցողի հիման վրա

```
class ElectricCar : public Car { };
```

Բազմաձևություն (Polymorphism)

Նույն ֆունկցիա՝ տարբեր վարք

```
virtual void drive();
```

Կողը դառնում է

- ընթեռնելի
- կրկնակի օգտագործելի
- հեշտ փոփոխելի

✓ Թիմային աշխատանքը պարզվում է
✓ Ծրագրի ընդլայնումը՝ անվտանգ

- OOP = օբյեկտների վրա հիմնված մտածողություն
- Յուրաքանչյուր օբյեկտ ունի տվյալ + վարք
- OOP-ը հիմք է մեծ ծրագրերի համար

23. Օբյեկտ կողմնորոշված մոդելը կոնկրետ օրինակներում։

34. Օբյեկտ կողմնորոշված մոտեցման կիրառում։

Օբյեկտ	Ատրիբուտներ	Ֆունկցիաներ
Student	name, id, grade	calculateAverage()
Course	title, credits	addStudent()

Օբյեկտ	Ատրիբուտներ	Ֆունկցիաներ
Teacher	name	assignGrade()

```
class Student {
private:
   string name;
   int id;
   double grade;

public:
   Student(string n, int i) {
      name = n;
      id = i;
      grade = 0;
   }

   void setGrade(double g) {
      grade = g;
   }

   double getGrade() {
      return grade;
   }

};
```

Օրինակ 2. Բանկային համակարգ

📓 Օբյեկտներ

- BankAccount
- Client

```
class BankAccount {
private:
   double balance;

public:
   BankAccount(double b) {
      balance = b;
   }

   void deposit(double amount) {
      balance += amount;
   }

   void withdraw(double amount) {
      if (amount <= balance)
         balance -= amount;
   }
};
```

Օրինակ 3. Գրաֆիկական պատկերներ (բազմաձևություն)

```
class Shape {
public:
   virtual double area() = 0;
};

class Rectangle : public Shape {
public:
   double w, h;

   Rectangle(double w_, double h_) {
      w = w_;
      h = h_;
   }

   double area() override {
      return w * h;
   }
};

class Circle : public Shape {
public:
   double r;

   Circle(double r_) {
      r = r_;
   }

   double area() override {
      return 3.14 * r * r;
   }
};
```

24. Ռացիոնալ թվերը նկարագրող դասը պարունակող ծրագրի ստեղծում:

40. Rational class պարունակող ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Rational {
private:
   int numerator;     // համարիչ
   int denominator;   // հայտարար

public:
   // Կոնստրուկտոր
   Rational(int n, int d) {
      numerator = n;
      denominator = d;
   }
```

```
    // Տպում
    void print() {
        cout << numerator << "/" << denominator << endl;
    }

    // Թվային արժեք
    double value() {
        return (double)numerator / denominator;
    }
};

int main() {
    Rational r1(1, 2);
    Rational r2(3, 4);

    r1.print();
    r2.print();

    cout << "r1 = " << r1.value() << endl;
    cout << "r2 = " << r2.value() << endl;

    return 0;
}
```

25. Ժամանակը նկարագրող դասը պարունակող ծրագրի ստեղծում։

```
#include <iostream>
using namespace std;

class Time {
private:
    int hours;
    int minutes;
    int seconds;

public:
    // Կոնստրուկտոր
    Time(int h, int m, int s) {
        hours = h;
        minutes = m;
        seconds = s;
    }

    // Ժամերի, րոպեների և վայրկյանների տպում
    void print() {
        cout << hours << ":" << minutes << ":" << seconds << endl;
    }

    // Ժամի ավելացում
    void addHours(int h) {
        hours += h;
        hours %= 24; // 24 ժամանոց համակարգ
    }

    // րոպեի ավելացում
```

```
    void addMinutes(int m) {
        minutes += m;
        hours += minutes / 60;
        minutes %= 60;
        hours %= 24;
    }

    // վայրկյանի ավելացում
    void addSeconds(int s) {
        seconds += s;
        minutes += seconds / 60;
        seconds %= 60;
        addMinutes(0); // ժամերի և րոպեների թարմացում
    }
};

int main() {
    Time t(12, 30, 45);
    t.print();

    t.addHours(2);
    t.addMinutes(40);
    t.addSeconds(30);

    cout << "Վերջին ժամանակը = ";
    t.print();

    return 0;
}
```

26. Դասերի և ֆունկցիաների կաղապարներ։

Եթե ֆունկցիաներում իրականացվող գործողությունները միատեսակ են՝ նպատակահարմար է դրանք վերաբեռնավորել մեկ այլ եղանակով, որն անվանում են *շաբլոնային*: Շաբլոնային վերաբեռնավորման դեպքում *ֆունկցիաների մարմինները համընկնում են*. մնում է վերնագիրը կազմավորել այնպես, որ վերաբեռնավորված ֆունկցիաներից յուրաքանչյուրի կանչի դեպքում կոմպիլյատորը միարժեքորեն իմանա, թե *պարամետրերի ինչպիսի տիպերին է* տվյալ պահին հարմարեցնելու շաբլոնը: Ընդ որում՝ կարելի է շաբլոնով տալ ինչպես ֆունկցիայից վերադարձվող արժեքի տիպը, այնպես էլ պարամետրերի տիպերը:

Ֆունկցիաների շաբլոնային վերաբեռնումն իրականացվում է ըստ վերադարձվող արժեքների և ֆորմալ պարամետրերի տիպերի:

Ֆունկցիայի *շաբլոնի (ընդհանիքի)* ընդհանուր տեսքը հետևյալն է.

template <շաբլոնային պարամետրերի ցուցակ> շաբլոնային ֆունկցիայի մարմին

որտեղ *template*-ը առանցքային բառ է, իսկ < >-ի մեջ առնված *շաբլոնային պարամետրերի ցուցակը* պարունակում է ստորակետերով անջատված մեկ կամ մի քանի տիպերի շաբլոնային անվանումներ, որոնցից յուրաքանչյուրի դիմաց պետք է գրված լինի *class* կամ *typename* առանցքային բառերից որևէ մեկը: Այստեղ ներառված պարամետրերը, վերջին հաշվով, ***ֆորմալ տիպեր*** են, որոնք արժեքներ են ստանում ֆունկցիայի կանչի պահին:

Օրինակ՝ շաբլոնային ֆունկցիայի վերնագիրը կարող է լինել հետևյալ տեսքի՝

```
template <classT, class C>
    T max(T x,C y)
```

որը կնշանակի, որ ֆունկցիան *շաբլոնացված է երկու իրարից տարբեր C և T տիպերի համար:*

Գրենք հետևյալ խնդրի լուծման ծրագիրը. *որոշել a, b իրական, c, d ամբողջ և e, f սիմվոլային տիպի մեծություններից մեծագույները՝ երկու պարամետրերից մեծագույնը որոշող շաբլոնով վերաբեռնավորված ֆունկցիայի կիրառմամբ:*

```
#include <iostream.h>
template <class T>
T max (T x,T y)
   {if (x>y) return x; else return y;}                    //1
   void main( )
        {double a,b;
        cin >> a >> b;
        cout <<max(a,b) <<endl;                           //2
        int c,d;
        cin >> c >> d;
        cout <<max(c,d) <<endl;                           //3

        char e,f;
        cin >> e >> f;
        cout <<max(e,f) <<endl;                                  //4
}
```

Երբ //2 տողում *max* ֆունկցիան կանչվում է *double* տիպի *a* և *b* փաստացի պարամետրերի համար, թարգմանիչը *max* ֆունկցիայում ամենուրեք *T* շաբլոնները ավտոմատ փոխարինում է *double* տիպով և այդպիսով ոչ միայն *x, y* փոփոխականներն են ստանում *double* տիպ, այլև ֆունկցիայից վերադարձվող արժեքը: Երբ //3 տողում երկրորդ անգամ է *max*-ը կանչվում՝ արդեն *int* տիպի *c* և *d* փաստացի պարամետրերի համար, այս դեպքում շաբլոնի *T*-ն փոխարինվում է *int* , //4-ում առկա կանչի դեպքում՝ *char* տիպերով:

Ֆունկցիաների շաբլոններից բացի, կարելի է ստեղծել նաև *դասերի շաբլոններ*. *Շաբլոն-դաս* սահմանելիս, ստեղծվող դասում որոշվում են այդ դասի բոլոր ալգորիթմները, իսկ մշակվող տվյալների փաստացի տիպերը այդ դասի օբյեկտներ ստեղծելիս տրվում են որպես պարամետրեր: Թարգմանիչը կառուցում է օբյեկտների ճշգրիտ տիպը՝ հիմնվելով օբյեկտների ստեղծման ժամանակ նրանց տիպի բնութագրման վրա:

Շաբլոն-դասի հայտարարման ընդհանուր տեսքը հետևյալն է.

```
template <class Ttype> class դասի_անուն {
:
}
```

27. Տեքստերի մշակման աշխատանքներ:

TextProcessor դաս, որը կատարում է տարբեր **տեքստերի մշակման աշխատանքներ**: Սա ուսանողական մակարդակի համար հարմար է՝ ցույց տալու **դաս, օբյեկտ, ֆունկցիոնալություն, թաղանթապատում**:

TextProcessor դասը կարող է անել՝

- տեքստի տպում
- երկարության որոշում
- փոքրատառ/Մեծատառ փոխարինում
- որոշ բառերի որոնում

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

class TextProcessor {
```

```
private:
    string text;  // թաղանթապատում՝ տեքստը private է

public:
    // Կոնստրուկտոր
    TextProcessor(string t) {
        text = t;
    }

    // Տեքստի տպում
    void print() {
        cout << text << endl;
    }

    // Տեքստի երկարություն
    int length() {
        return text.length();
    }

    // Փոփոխել ամբողջ տեքստը փոքրատառ
    void toLower() {
        transform(text.begin(), text.end(), text.begin(), ::tolower);
    }

    // Փոփոխել ամբողջ տեքստը Մեծատառ
    void toUpper() {
        transform(text.begin(), text.end(), text.begin(), ::toupper);
    }

    // Բառի որոնում (return true եթե կա)
    bool findWord(string word) {
        return text.find(word) != string::npos;
    }
};

int main() {
    TextProcessor tp("Hello World! Welcome to C++ Programming.");

    tp.print();
    cout << "Length: " << tp.length() << endl;

    tp.toLower();
    cout << "Lowercase: ";
    tp.print();

    tp.toUpper();
    cout << "Uppercase: ";
    tp.print();

    string word = "C++";
    if (tp.findWord(word))
        cout << word << " is found in the text." << endl;
    else
        cout << word << " is not found." << endl;
```

```
   return 0;
}
```

28. Տողային տվյալներ: String դաս:

29. String դասի ֆունկցիաները:

C++ É»½íáõÙ û·ï³·áñÍíáõÙ ¿ 2 ï»ë³ÏÇ ïáÕ»ñª
1.Ü»ñÏ³éáõóí³Í ïÇåÇ, áñÁ ÙÝ³ó»É ¿ C É»½íÇó:
2. C++ É»½íÇ ëï³Ý¹³ñï ý³ÛÉ³¹³ñ³ÝÇ string ¹³ëÇ ÙÇçáóáí ïñíáÕ:
²í»ÉÇ Ñ³ñÙ³ñ ¿ û·ï³·áñÍ»É »ñÏñáñ¹ Ó¨Á:
1.*Ü»ñÏ³éáõóí³Í ïáÕ³ÛÇ ïÇåÁ* Å³é³Ý·í»É ¿ C É»½íÇó: ²Ûëï»Õ ëÇÙíáÉÝ»ñÇ ïáÕÁ ÑÇßáÕáõÃÛ³Ý Ù»ç å³Ñíáõ Ù ¿ áñå»ë ½³Ý·í³Í, ÇëÏ Ýñ³Ý ¹ÇÙ»ÉÁ Çñ³·áñÍíáõÙ ¿ char* ïÇåÇ óáõó³ÝßÇãÇ û·ÝáõÃÛ³Ùµ: C É»½íÇ ëï³Ý¹³ñï ý³ÛÉ³¹³ñ³ÝÁ å³ñáõÝ³ÏáõÙ ¿ ïáÕ»ñÇ Ñ»ï ·áñ-ÍáÕáõÃÛáõÝÝ»ñÇ Ï³ï³ñÙ³Ý ýáõÝÏóÇ³Ý»ñ:
úñÇÝ³Ïª

```
   //  Ì»ñ³¹³ñÓÝáõÙ ¿ ïáÕÇ »ñÏ³ñáõÃÛáõÝÁ
           int  strlen( const  char*);
   //  Ð³Ù»Ù³ïáõÙ  ¿ »ñÏáõ ïáÕ
           int  strcmp( const  char*, const  char* );
   //  ä³ï×»ÝáõÙ  ¿  Ù»Ï ïáÕÁ ÙÛáõëÇ Ù»ç
          char*  strcpy( char, const  char* );
```

²Ûë ýáõÝÏóÇ³Ý»ñÇ û·ï³·áñÍÙ³Ý Ñ³Ù³ñ å»ïù ¿ Ý»ñ³é»É <cstring> Ëáñ³·ñ³ÛÇÝ ý³ÛÉÁ, û·ï³·áñÍ»Éáí
```
#include <cstring>
```
Õ»Ï³í³ñÙ³Ý Ññ³Ñ³Ý·Á:

char ïÇåÇ óáõó³ÝßÇãÁ, áñÇ û·ÝáõÃÛ³Ùµ Ù»Ýù ¹ÇÙáõÙ »Ýù ïá- ÕÇÝ, óáõÛó ¿ ï³ÉÇë Ñ³Ù³å³ï³Ë³Ý ïáÕÇ ïñÙ³Ý ëÇÙíáÉÝ»ñÇ ½³Ý·-í³ÍÁ: ºÃ» ÝáõÛÝÇëÏ Ù»Ýù ·ñáõÙ »Ýù ïáÕ³ÛÇÝ ÉÇï»ñ³É, ûñÇÝ³Ïª

```
const  char  *st  = “Ø»Ï ßÇß ·ÇÝáõ ³ñÅ»ùÁ\n”;
```

Ññ³Ñ³Ý·Ç Ï³ï³ñÙ³Ý Å³Ù³Ý³Ï Ã³ñ·Ù³ÝÇãÁ ïáÕÇ µáÉáñ ëÇÙíáÉÝ»- ñÁ ï»Õ³íáñáõÙ ¿ ½³Ý·í³ÍÇ Ù»ç ¨ Ñ»ïá st ÷á÷áË³Ï³ÝÇÝ í»ñ³·ñáõÙ ¿ ½³Ý·í³ÍÇ ³é³çÇÝ ï³ñññ Ñ³ëó»Ý:

îáÕ»ñÇ Ñ»ï ³Ûë Ó¨áí ³ßË³ï»ÉÁ ëË³ÉÝ»ñÇ ³é³ç³óÙ³Ý Ù»Í Ñ³í³ÏÝáõÃÛáõÝ ¿ ëï»ÕÍáõÙ: ²Û¹ Ã»ñáõÃÛáõÝÝ»ñÇó ½»ñÍ ¿ C++ É»½íÇ string ëï³Ý¹³ñï ¹³ëÁ:

àñå»ë½Ç ÑÝ³ñ³íáñ ÉÇÝÇ û·ïí»É C++ É»½íÇ string ¹³ëÇ ûµÛ»ÏïÝ»ñÇó, ³ÝÑñ³Å»ßï ¿ ÁÝ¹·ñÏ»É <string> Ëáñ³·ñ³ÛÇÝ ý³ÛÉÁ, û·ï³·áñÍ»Éáí

```
#include <string>
```

Õ»Ï³í³ñÙ³Ý Ññ³Ñ³Ý·Á:

st ÷á÷áË³Ï³ÝÇÝ ³ñÅ»ù í»ñ³·ñ»Éáõ Ñ³Ù³ñ å»ïù ¿ ·ñ»Éª

```
#include <string>
string   st ( “Ø»Ï ßÇß ·ÇÝáõ ³ñÅ»ùÁ\n”);
```

Ññ³Ñ³Ý·Ý»ñÁ:

îáÕÇ »ñÏ³ñáõÃÛáõÝÁ í»ñ³¹³ñÓÝáõÙ ¿ size() ³Ý¹³Ù-ýáõÝÏóÇ³Ý (»ñÏ³ñáõÃÛ³Ý Ù»ç ãÇ ÁÝ¹·ñÏíáõÙ í»ñçÇÝ ½ñáÛ³Ï³Ý ëÇÙíáÉÁ):

```
cout  << st  << “ïáÕÁ å³ñáõÝ³ÏáõÙ ¿ª  ” << st. size()
     << “  ëÇÙíáÉÝ»ñ, Ý»ñ³éÛ³É Ý³¨ Ýáñ ïáÕÇ \n ëÇÙíáÉÁ ”
```

¸³ï³ñÏ ïáÕÇ ïñÙ³Ý Ñ³Ù³ñ ·ñíáõÙ ¿ª

```
string   st2;   //  ¸³ï³ñÏ ïáÕ
```

²Ûë ¹»åùáõÙ st.2.size() ÏáõÝ»Ý³ 0 ³ñÅ»ù, ÇëÏ st.empty() ³Ý¹³Ù-ýáõÝÏ-óÇ³Ý Ïí»ñ³¹³ñÓÝÇ true ³ñÅ»ù:

ºÃ» áõ½áõÙ »Ýù string ïÇåÇ st3 ïáÕÇÝ í»ñ³·ñ»É ÝáõÛÝ ïÇåÇ st ïáÕÇ ³ñÅ»ùÁ û·ï³·áñÍáõÙ »Ýù ÏáÝëïñáõÏïáñÇ 3-ñ¹ Ó¨Áª

```
string   st3( st );
```

ú·ï³·áñÍ»Éáí í»ñ³·ñÙ³Ý Ññ³Ñ³Ý·Á Ï³ñáÕ »Ýù ÙÇ ïáÕÁ å³ï×»- Ý»É ÙÛáõëÇ Ù»ç, ³Ûëå»ëª

```
st2 = st3   // st3  ïáÕÁ å³ï×»ÝíáõÙ ¿  st2 ïáÕÇ Ù»ç
```

îáÕ»ñÇ ÏóÙ³Ý Ñ³Ù³ñ û·ï³·áñÍíáõÙ »Ý ·áõÙ³ñÙ³Ý (+) ¨ í»ñ³·ñáõ- Ùáí ·áõÙ³ñÙ³Ý (+=) ·áñÍáÕáõÃÛáõÝÝ»ñÁ: úñÇÝ³Ïª

```
string   s1( “Ø»Ï ßÇß ”);
string   s2 ( “ ·ÇÝáõ ³ñÅ»ùÁ\n”);
string   s3 = s1 + s2;
```
Ññ³Ñ³Ý·Ý»ñÇ Ï³ï³ñáõÙÇó Ñ»ïá s3 ïáÕÁ ÏÉÇÝÇ “Ø»Ï ßÇß ·ÇÝáõ ³ñÅ»ùÁ\n” :

s1 ïáÕÇ í»ñçáõÙ s2 ïáÕÁ ³í»É³óÝ»Éáõ Ñ³Ù³ñ å»ïù ¿ ·ñ»É³

```
s1  +=  s2;
```

¶áõÙ³ñÙ³Ý (+) ·áñÍáÕáõÃÛáõÝÁ Ï³ñáÕ ¿ ÙÇ³óÝ»É Ý³¨ string ¨ Ý»ñÏ³éáõóí³Í ïÇå»ñÇ ïáÕ»ñÁ: úñÇÝ³Ï, »Ã» ïñí³Í »Ý Ñ»ï¨Û³É »ñÏáõ ï³ñµ»ñ ïÇå»ñÇ ïáÕ»ñÁ³

```
const  char  *pc = “, ” ;
string   s1( “´³ñ¨ ”);
string   s2 ( “ÁÝÏ»ñ”);
```
³å³ Ï³ñ»ÉÇ ¿ ·ñ»É

```
string   s3 = s1 + pc + s2 + “\n”;
```
ÐÝ³ñ³íáñ ¿ Ý³¨ Ñ»ï¨Û³É í»ñ³·ñáõÙÁ³

```
const  char  *pc = “, ” ;
string   s1;
s1 = pc;
```
Ð³Ï³é³Ï ·áñÍáÕáõÃÛáõÝÁ ãÇ Ï³ï³ñáõÙ: úñÇÝ³Ï, Ñ»ï¨Û³É Ññ³Ñ³Ý·Á ÏÉÇÝÇ ëË³É

```
char  *str = s1;
```

30. vector դաս: Օրինակներ:

vector ¹³ëÁ Ý»ñÏ³éáõóí³Í ½³Ý·í³ÍÝ»ñÇÝ Ñ³Ù³ñÅ»ù ïíÛ³ÉÇ ï»ë³Ï ¿, ë³Ï³ÛÝ ³ÛÝ ûÅïí³Í ¿ ³í»ÉÇ É³ÛÝ ÑÝ³ñ³íáñáõÃÛáõÝÝ»ñáí ¨ Ýñ³ û·ï³·áñÍáõÙÁ Ý³ËÁÝïñ»ÉÇ ¿:

ì»ÏïáñÇ û·ï³·áñÍÙ³Ý Ñ³Ù³ñ ³ÝÑñ³Å»ßï ¿ ÁÝ¹·ñÏ»É <vector> Ëáñ³·ñ³ÛÇÝ ý³ÛÉÁ, û·ï³·áñÍ»Éáí

```
#include <vector>
```

Õ»Ï³í³ñÙ³Ý Ññ³Ñ³Ý·Á:

Ü³Ë³ï»ëí³Í ¿ í»ÏïáñÇ û·ï³·áñÍÙ³Ý »ñÏáõ ï³ñµ»ñ Ó¨»ñ, áñáÝó ³Ýí³ÝáõÙ »Ý *½³Ý·í³ÍÇ Ó¨ ¨ STL Ó¨:*

²é³çÇÝ ¹»åùáõÙ í»ÏïáñÁ û·ï³·áñÍáõÙ »Ý Ý»ñÏ³éáõóí³Í ½³Ý·í³ÍÝ»ñÇ Ó¨áí: ê³ÑÙ³ÝíáõÙ ¿ ïñí³Í ã³÷áí í»Ïïáñ, ûñÇÝ³Ï³

```
vector <int> ai(10);
```

áñÁ Ñ³Ù³ñÅ»ù ¿ Ý»ñÏ³éáõóí³Í ½³Ý·í³ÍÇ Ñ»ï¨Û³É Ó¨ÇÝ³

```
int  ai[10];
```

ì»ÏïáñÇ ³é³ÝÓÇÝ ï³ñ»ñÇÝ ¹ÇÙ»Éáõ Ñ³Ù³ñ Ï³ñ»ÉÇ ¿ û·ï³-·áñÍ»É ÇÝ¹»ùëÝ»ñ, ûñÇÝ³Ï³ ai[10]: ì»ÏïáñÇ ã³÷Á Ï³ñ»ÉÇ ¿ áñáß»É û·ï³·áñÍ»Éáí size() ýáõÝÏóÇ³Ý, ÇëÏ í»ÏïáñÇ ¹³ï³ñÏ ÉÇÝ»ÉÁ Ï³ñ»ÉÇ ¿ ëïáõ·»É û·ï³·áñÍ»Éáí empty() ýáõÝÏóÇ³Ý: úñÇÝ³Ï³

```
void ptint_vector (vector<int> ac)
{
   if (ac. empty())
   return;
   for  (int  ix=0;  ix< ivac,size(); ++ix)
   cout  <<  ac[ ix ]  << ‘ ‘ ;
}
```

ì»ÏïáñÇ ï³ñ»ñÇÝ Éé»ÉÛ³Ý í»ñ³·ñíáõÙ »Ý 0 ³ñÅ»ùÝ»ñ, »Ã» Ýñ³Ýù Ãí³ÛÇÝ Ï³Ù óáõó³ÝßÇã³ÛÇÝ ïÇå»ñÇ »Ý: ì»ÏïáñÇ ï³ñ»ñÇÝ µ³ó³Ñ³Ûïáñ»Ý ³ñÅ»ùÝ»ñ Ï³ñ»ÉÇ ¿ í»ñ³·ñ»É û·ï³·áñÍ»Éáí Ñ»ï¨Û³É ·ñ»É³Ó¨Á³

```
vector <int> ac(10, -1);
```

áñÇ Å³Ù³Ý³Ï ac í»ÏïáñÇ µáÉáñ 10-Á ï³ññ»ñÁ ÏáõÝ»Ý³Ý –1 ³ñÅ»ùÁ:

¼³Ý·í³ÍÇ ï³ññ»ñÇÝ ³ñÅ»ùÝ»ñ Ï³ñ»ÉÇ ¿ í»ñ³·ñ»É óáõó³ÏáíªA

```
int ia[ 6 ] = {-2, -1, 0, 1, 2, 1024 } ;
```

vector ¹³ëÇ ûµÛ»ÏïÇ Ñ³Ù³ñ ãÇ ÃáõÛÉ³ïñíáõÙ ÝÙ³Ý ·áñÍáÕáõÃÛ³Ý Ï³ï³ñáõÙÁ: ´³Ûó í»ÏïáñA ï»ë³ÏÇ ûµÛ»ÏïÇ ï³ññ»ñÇÝ Ï³ñ»ÉÇ ¿ ëÏ½µÝ³Ï³Ý ³ñÅ»ùÝ»ñ í»ñ³·ñ»É Ý»ñÏ³éáõóí³Í ½³Ý·í³ÍÇ ÙÇçáóáí³A

vector <int> ac(ia, ia+6); // ia-Ç 6 ï³ññ»Á å³ï×»ÝíáõÙ »Ý ac-Ç Ù»ç

Î³ñ»ÉÇ ¿ áñå»ë ëÏ½µÝ³Ï³Ý ³ñÅ»ùÝ»ñÇ óáõó³Ï í»ñóÝ»É ½³Ý·í³ÍÇ áñáß ÙÇç³Ï³Ûù³

vector <int> ac(&ia[2],&ia[5]); //å³ï×»ÝíáõÙ »Ý 3 ï³ññ`ia[2],ia[3], ia[4]:

ú·ï³·áñÍ»ÉáíA í»ñ³·ñÙ³Ý Ññ³Ñ³Ý·Á í»ÏïáñÇÝ Ï³ñ»ÉÇ ¿ ï³É ëÏ½µÝ³Ï³Ý ³ñÅ»ùÝ»ñ ¨ Ï³ñ»ÉÇ ¿ å³ï×»Ý»É ûµÛ»ÏïÝ»ñÁ: úñÇÝ³Ï³A

```
vector <string> svac;

void int_and_assign()
{
```

// svac-Ç ÙÇç³óáí user_names–ÇÝ ïñíáõÙ »Ý ëÏ½µÝ³Ï³Ý ³ñÅ»ùÝ»ñ

```
vector <string> user_names(svac);

svac  = user_names; // user_names–Á å³ï×»ÝíáõÙ ¿ svac-Ç Ù»ç

}
```

STL Ó¨Ç ¹»åùáõÙ Ý³Ë³å»ë ë³ÑÙ³ÝíáõÙ ¿ ¹³ï³ñÏ í»ÏïáñªA

```
vector <string>  text;
```

³ÛÝáõÑ»ï¨ ï³ñµ»ñ ýáõÝÏóÇ³Ý»ñÇ ÙÇçáóáí Ýñ³ÝáõÙ ³í»É³óíáõÙ »Ý ï³ññ»ñÁ: úñÇÝ³Ï, puch_back() ýáõÝÏóÇ³Ý í»ÏïáñÇ í»ñçáõÙ ³í»-É³óÝáõÙ ¿ Ýáñ ï³ññ: ²Ûë Íñ³·ñÇ Ñ³ïí³ÍÇ ÙÇçáóáí ëï³Ý¹³ñï Ý»-ñ³ÍÙ³Ý ÑáëùáíA Ý»ñ³Íí³Í ëÇÙíáÉÝ»ñÁ ³í»É³óÝáõÙ »Ý í»ÏïáñÇÝªA

```
string  word;

while  ( cin  >>  word  )

{

text.puch_back( word );

}
```

ì»ÏïáñÇ ï³ññ»ñÇ ÁÝïñÙ³Ý Ñ³Ù³ñ Ï³ñáÕ ¿ û·ï³·áñÍí»É Ý³¨ ÇÝ¹»ùë³íáñÙ³Ý ·áñÍáÕáõÃÛáõÝÁªA

```
cout  << “Ï³ñ¹áõÙ »Ýù µ³éÁ \n”;

for ( int ix = 0; ix < text.size(); ++ix )

cout  << text[ ix ] << ‘ ‘;

cout  <<  endl;
```

STL Ó¨ÇÝ Ñ³ïáõÏ ¿ Çï»ñ³ïáñÝ»ñÇ û·ï³·áñÍáõÙÁªA

```
cout  << “Ï³ñ¹áõÙ »Ýù µ³éÁ \n”;
```

```
for (vector <string>::iterator  it=text.begin();

if  != text.end();  ++it; )

cout  <<  *it  << ‘ ‘;

cout  <<  endl;
```

Æï»ñ³ïáñÁ, ¹³ óáõó³ÝßÇã ¿ ½³Ý·í³ÍÇ ï³ññÇ íñ³: ´»ñí³Í ûñÇÝ³-ÏáõÙ it Çï»ñ³ïáñáí (³ÛÝ ÏÉÇÝÇ *it óáõó³ÝßÇãÁ) óáõÛó ¿ ïñí³Í text í»ÏïáñÇ ï³ññ»ñÁ: ++it; Ññ³Ñ³Ý·Á ï»Õ³ß³ñÅáõÙ ¿ ³Û¹ óáõó³ÝßÇãÁ Ñ³çáñ¹ ï³ññÇ íñ³:

32. Կետ և շրջանագիծը նկարագրող դասեր պարունակող ծրագրի ստեղծում։

```
#include <iostream>
#include <cmath>
using namespace std;

// Հիմնադրիչ դաս – կետ
class Point {
protected:
   double x;
   double y;

public:
   Point(double x_coord = 0, double y_coord = 0) {
      x = x_coord;
      y = y_coord;
   }

   void print() {
      cout << "Point(" << x << ", " << y << ")" << endl;
   }
};

// Ժառանգ դաս – շրջանագիծ
class Circle : public Point {
private:
   double radius;

public:
   Circle(double x_coord, double y_coord, double r)
      : Point(x_coord, y_coord) {
      radius = r;
   }

   double area() {
      return 3.14159 * radius * radius;
   }
```

```
    double circumference() {
        return 2 * 3.14159 * radius;
    }

    void print() {
        cout << "Circle centered at (" << x << ", " << y << ") with radius " << radius << endl;
        cout << "Area = " << area() << ", Circumference = " << circumference() << endl;
    }
};

int main() {
    Point p(2, 3);
    p.print();

    Circle c(5, 5, 4);
    c.print();

    return 0;
}
```

33. Person, employee դասերը նկարագրող ծրագրի ստեղծում

```
#include <iostream>
#include <string>
using namespace std;

// Հիմնադիրը՝ Person դաս
class Person {
protected:
    string name;
    int age;

public:
    Person(string n, int a) {
        name = n;
        age = a;
    }

    void print() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

// Ժառանգ դաս՝ Employee
class Employee : public Person {
private:
    string position;
    double salary;
```

```
public:
   Employee(string n, int a, string pos, double sal)
      : Person(n, a) {
      position = pos;
      salary = sal;
   }

   void print() {
      cout << "Name: " << name << ", Age: " << age
         << ", Position: " << position
         << ", Salary: $" << salary << endl;
   }
};

int main() {
   Person p1("Alice", 30);
   p1.print();

   Employee e1("Bob", 40, "Manager", 5000);
   e1.print();

   return 0;
}
```

35. Stack class պարունակող ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Stack {
private:
   int top;
   int arr[100];   // մեռնել չափով
   int capacity;

public:
   // Կոնստրուկտոր
   Stack(int size = 100) {
      capacity = size;
      top = -1;
   }

   // Ստեկը դատարկ է
   bool isEmpty() {
      return top == -1;
   }

   // Ստեկը լի՞ր է
   bool isFull() {
```

```
        return top == capacity - 1;
    }

    // Push – ավելացնում է տարր
    void push(int value) {
        if (isFull()) {
            cout << "Stack overflow!" << endl;
            return;
        }
        arr[++top] = value;
    }

    // Pop – հեռացնում է վերջին տարրը
    int pop() {
        if (isEmpty()) {
            cout << "Stack underflow!" << endl;
            return -1;
        }
        return arr[top--];
    }

    // Peek – վերադարձնում է վերջին տարրը առանց հեռացնելու
    int peek() {
        if (isEmpty()) {
            cout << "Stack is empty!" << endl;
            return -1;
        }
        return arr[top];
    }

    // Ստեկի պարունակությունը տպել
    void print() {
        if (isEmpty()) {
            cout << "Stack is empty!" << endl;
            return;
        }
        cout << "Stack: ";
        for (int i = 0; i <= top; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};

int main() {
    Stack s(5);
```

```
    s.push(10);
    s.push(20);
    s.push(30);

    s.print();

    cout << "Top element: " << s.peek() << endl;

    cout << "Popped: " << s.pop() << endl;

    s.print();

    return 0;
}
```

36. Time class և DateTime class պարունակող ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

// Հիմնադիրը՝ Time դաս
class Time {
protected:
    int hours;
    int minutes;
    int seconds;

public:
    Time(int h = 0, int m = 0, int s = 0) {
        hours = h;
        minutes = m;
        seconds = s;
    }

    void printTime() {
        cout << hours << ":" << minutes << ":" << seconds;
    }

    void addSeconds(int s) {
        seconds += s;
        minutes += seconds / 60;

        seconds %= 60;
        hours += minutes / 60;
        minutes %= 60;
        hours %= 24; // 24 ժամանոց համակարգ
```

```
    }
};

// Ժառանգ դաս՝ DateTime
class DateTime : public Time {
private:
    int day;
    int month;
    int year;

public:
    DateTime(int d, int mo, int y, int h = 0, int m = 0, int s = 0)
        : Time(h, m, s) {
        day = d;
        month = mo;
        year = y;
    }

    void printDateTime() {
        cout << day << "/" << month << "/" << year << " ";
        printTime();
        cout << endl;
    }
};

int main() {
    Time t1(14, 30, 45);
    cout << "Time: ";
    t1.printTime();
    cout << endl;

    DateTime dt1(25, 12, 2025, 14, 30, 45);
    cout << "DateTime: ";
    dt1.printDateTime();

    dt1.addSeconds(3600); // 1 ժամ ավելացնել
    cout << "DateTime հետո 1 ժամ ավելացնելուց: ";
    dt1.printDateTime();

    return 0;
}
```

37. TicTacToe խաղի ստեղծում

```
#include <iostream>

using namespace std;
```

```
class TicTacToe {
private:
    char board[3][3];
    char currentPlayer;

public:
    TicTacToe() {
        currentPlayer = 'X';
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                board[i][j] = ' ';
    }

    void printBoard() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++)
                cout << board[i][j] << " ";
            cout << endl;
        }
    }

    void play() {
        int row, col;
        for (int move = 0; move < 9; move++) {
            printBoard();
            cout << "Player " << currentPlayer << ", enter row and column (0-2): ";
            cin >> row >> col;
            if (board[row][col] == ' ') {
                board[row][col] = currentPlayer;
                currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';
```

```
            } else {
                cout << "Cell already taken. Try again.\n";
                move--; // կրկնել քայլը
            }
        }
        printBoard();
        cout << "Game over!" << endl;
    }
};

int main() {
    TicTacToe game;
    game.play();
    return 0;
}
```

38. Բառի տառերի գուշակում խաղի ստեղծում

```
#include <iostream>
#include <string>
using namespace std;

class WordGuess {
private:
    string word;      // բառը
    string guessed;   // ստուգված տառերը

public:
    WordGuess(string w) {
        word = w;
        guessed = string(word.length(), '_'); // սկիզբը բոլոր տառերը՝ "_"
    }

    void play() {
        char letter;
        int attempts = 6;

        while (guessed != word && attempts > 0) {
            cout << "Word: " << guessed << endl;
```

```
            cout << "Attempts left: " << attempts << endl;
            cout << "Guess a letter: ";
            cin >> letter;

            bool found = false;
            for (size_t i = 0; i < word.length(); i++) {
                if (word[i] == letter && guessed[i] == '_') {
                    guessed[i] = letter;
                    found = true;
                }
            }

            if (!found) {
                cout << "Wrong guess!" << endl;
                attempts--;
            }
        }

        if (guessed == word)
            cout << "Congratulations! You guessed the word: " << word << endl;
        else
            cout << "Game over! The word was: " << word << endl;
    }
};

int main() {
    WordGuess game("HELLO");
    game.play();
    return 0;
}
```

39. Complex class պարունակող ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Complex {
private:
    double real;
    double imag;

public:
    // Կոնստրուկտոր
    Complex(double r = 0, double i = 0) {
        real = r;
        imag = i;
    }
```

```
    // Գումար
    Complex add(const Complex& c) {
        return Complex(real + c.real, imag + c.imag);
    }

    // Հանում
    Complex subtract(const Complex& c) {
        return Complex(real - c.real, imag - c.imag);
    }

    // Տպում
    void print() {
        cout << real << " + " << imag << "i" << endl;
    }
};

int main() {
    Complex c1(3, 4);
    Complex c2(1, 2);

    cout << "c1 = "; c1.print();
    cout << "c2 = "; c2.print();

    Complex sum = c1.add(c2);
    cout << "Sum = "; sum.print();

    Complex diff = c1.subtract(c2);
    cout << "Difference = "; diff.print();

    return 0;
}
```

40. Rectangle class պարունակող ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Rectangle {
private:
    double length;
    double width;

public:
    // Կոնստրուկտոր
    Rectangle(double l = 0, double w = 0) {
        length = l;
```

```
        width = w;
    }

    // Մակերես
    double area() {
        return length * width;
    }

    // Պերիմետր
    double perimeter() {
        return 2 * (length + width);
    }

    // Տպել չափերը
    void print() {
        cout << "Rectangle: length = " << length << ", width = " << width << endl;
    }
};

int main() {
    Rectangle r1(5, 3);

    r1.print();
    cout << "Area = " << r1.area() << endl;
    cout << "Perimeter = " << r1.perimeter() << endl;

    return 0;
}
```

41. Triangle class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <cmath>
using namespace std;

class Triangle {
private:
    double a, b, c;

public:
    // Կոնստրուկտոր
    Triangle(double side1 = 0, double side2 = 0, double side3 = 0) {
        a = side1;
        b = side2;
        c = side3;
    }
```

```
    // Պերիմետր
    double perimeter() {
        return a + b + c;
    }

    // Մակերես (Heron's formula)
    double area() {
        double s = perimeter() / 2;
        return sqrt(s * (s - a) * (s - b) * (s - c));
    }

    // Տպել կողմերը
    void print() {
        cout << "Triangle sides: a = " << a << ", b = " << b << ", c = " << c << endl;
    }
};

int main() {
    Triangle t1(3, 4, 5);

    t1.print();
    cout << "Perimeter = " << t1.perimeter() << endl;
    cout << "Area = " << t1.area() << endl;

    return 0;
}
```

42. Shape class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <cmath>
using namespace std;

// Հիմնական դաս՝ Shape
class Shape {
public:
    virtual void print() {
        cout << "This is a generic shape." << endl;
    }
};

// Rectangle դաս՝ ժառանգվում է Shape
class Rectangle : public Shape {
private:
    double length, width;

public:
```

```
    Rectangle(double l, double w) {
        length = l;
        width = w;
    }

    double area() {
        return length * width;
    }

    double perimeter() {
        return 2 * (length + width);
    }

    void print() override {
        cout << "Rectangle: length = " << length << ", width = " << width
             << ", Area = " << area() << ", Perimeter = " << perimeter() << endl;
    }
};

// Triangle դաս՝ ժառանգվում է Shape
class Triangle : public Shape {
private:
    double a, b, c;

public:
    Triangle(double s1, double s2, double s3) {
        a = s1;
        b = s2;
        c = s3;
    }

    double perimeter() {
        return a + b + c;
    }

    double area() {
        double s = perimeter() / 2;
        return sqrt(s * (s - a) * (s - b) * (s - c));
    }

    void print() override {
        cout << "Triangle: sides = " << a << ", " << b << ", " << c
             << ", Area = " << area() << ", Perimeter = " << perimeter() << endl;
    }
};
```

```
int main() {
    Shape* s1 = new Rectangle(5, 3);
    Shape* s2 = new Triangle(3, 4, 5);

    s1->print();
    s2->print();

    delete s1;
    delete s2;

    return 0;
}
```

43. Circle class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <cmath>
using namespace std;

class Circle {
private:
    double radius;

public:
    // Կոնստրուկտոր
    Circle(double r = 0) {
        radius = r;
    }

    // Մակերես
    double area() {
        return 3.14159 * radius * radius;
    }

    // Պարագիծ
    double circumference() {
        return 2 * 3.14159 * radius;
    }

    // Տպել շառավիղը
    void print() {
        cout << "Circle: radius = " << radius
             << ", Area = " << area()
             << ", Circumference = " << circumference() << endl;
    }
};
```

```
int main() {
    Circle c1(5);

    c1.print();

    return 0;
}
```

44. Employee class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <string>
using namespace std;

class Employee {
private:
    string name;
    int age;
    double salary;

public:
    // Կոնստրուկտոր
    Employee(string n, int a, double s) {
        name = n;
        age = a;
        salary = s;
    }

    // Տպել տվյալները
    void print() {
        cout << "Name: " << name << ", Age: " << age
             << ", Salary: $" << salary << endl;
    }

    // Աշխատավարձի փոփոխություն
    void setSalary(double s) {
        salary = s;
    }

    double getSalary() {
        return salary;
    }
};

int main() {
    Employee e1("Alice", 30, 5000);
    Employee e2("Bob", 40, 6000);
```

```
    e1.print();
    e2.print();

    cout << "Updating Alice's salary..." << endl;
    e1.setSalary(5500);
    cout << "Alice's new salary: $" << e1.getSalary() << endl;

    return 0;
}
```

45. Student class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <string>
using namespace std;

class Student {
private:
    string name;
    int age;
    double grade;

public:
    // Կոնստրուկտոր
    Student(string n, int a, double g) {
        name = n;
        age = a;
        grade = g;
    }

    // Տպել տվյալները
    void print() {
        cout << "Name: " << name << ", Age: " << age
             << ", Grade: " << grade << endl;
    }

    // Գրեյդի փոփոխություն
    void setGrade(double g) {
        grade = g;
    }

    double getGrade() {
        return grade;
    }
};

int main() {
    Student s1("Alice", 20, 85);
```

```
    Student s2("Bob", 21, 90);

    s1.print();
    s2.print();

    cout << "Updating Alice's grade..." << endl;
    s1.setGrade(95);
    cout << "Alice's new grade: " << s1.getGrade() << endl;

    return 0;
}
```

46. Point class պարունակող ծրագրի ստեղծում

```
#include <iostream>
#include <cmath>
using namespace std;

class Point {
private:
    double x;
    double y;

public:
    // Կոնստրուկտոր
    Point(double xCoord = 0, double yCoord = 0) {
        x = xCoord;
        y = yCoord;
    }

    // Տպել կետը
    void print() {
        cout << "(" << x << ", " << y << ")" << endl;
    }

    // Հեռավորություն հաշվել մեկ այլ կետից
    double distance(const Point& p) {
        return sqrt((x - p.x)*(x - p.x) + (y - p.y)*(y - p.y));
    }
};

int main() {
    Point p1(2, 3);
    Point p2(5, 7);

    cout << "Point 1: "; p1.print();
    cout << "Point 2: "; p2.print();
```

```
    cout << "Distance between p1 and p2: " << p1.distance(p2) << endl;

    return 0;
}
```

47. Լաբիրինթ ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Labyrinth {
private:
    char grid[5][5] = {
        {'P',' ',' ',' ',' '},
        {' ','#','#',' ',' '},
        {' ',' ',' ',' ','#',' '},
        {'#',' ','#',' ',' ',' '},
        {' ',' ',' ',' ',' ','E'}
    };
    int playerX = 0;
    int playerY = 0;

public:
    void printGrid() {
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++)
                cout << grid[i][j] << " ";
            cout << endl;
        }
    }

    void move(char dir) {
        int newX = playerX;
        int newY = playerY;

        if (dir == 'w') newX--;        // վերև
        else if (dir == 's') newX++;   // ներքև
        else if (dir == 'a') newY--;   // ձախ
        else if (dir == 'd') newY++;   // աջ
        else return;

        if (newX >= 0 && newX < 5 && newY >= 0 && newY < 5 && grid[newX][newY] !=
'#') {
            grid[playerX][playerY] = ' ';
            playerX = newX;
            playerY = newY;
            grid[playerX][playerY] = 'P';
        }
```

```
    }

    bool isFinished() {
        return grid[playerX][playerY] == 'E';
    }
};

int main() {
    Labyrinth game;
    char move;

    cout << "Controls: w(up), s(down), a(left), d(right)" << endl;

    while (!game.isFinished()) {
        game.printGrid();
        cout << "Enter move: ";
        cin >> move;
        game.move(move);
        system("cls"); // Windows, կոնսոլում մաքրում է էկրանը
    }

    cout << "Congratulations! You reached the exit!" << endl;

    return 0;
}
```

48. Շախմատ (թագուհի) ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Queen {
private:
    int x, y; // թագուհու դիրքը

public:
    Queen(int row = 0, int col = 0) {
        x = row;
        y = col;
    }

    void printPosition() {
        cout << "Queen position: (" << x << ", " << y << ")" << endl;
    }

    void possibleMoves() {
        cout << "Possible moves:" << endl;
        for (int i = 0; i < 8; i++) {
```

```
            for (int j = 0; j < 8; j++) {
                if (i == x || j == y || (i - j == x - y) || (i + j == x + y)) {
                    cout << "(" << i << "," << j << ") ";
                }
            }
        }
        cout << endl;
    }

    void moveTo(int row, int col) {
        x = row;
        y = col;
    }
};

int main() {
    Queen q(0, 0);

    q.printPosition();
    q.possibleMoves();

    cout << "Move queen to (3,4)" << endl;
    q.moveTo(3, 4);
    q.printPosition();
    q.possibleMoves();

    return 0;
}
```

49. Վերելակ ծրագրի ստեղծում

```
#include <iostream>
using namespace std;

class Elevator {
private:
    int currentFloor;

public:
    Elevator(int startFloor = 1) {
        currentFloor = startFloor;
    }

    void printFloor() {
        cout << "Current floor: " << currentFloor << endl;
    }

    void moveUp() {
```

```
        if (currentFloor < 5) {
            currentFloor++;
            cout << "Moved up to floor " << currentFloor << endl;
        } else {
            cout << "Already at the top floor!" << endl;
        }
    }

    void moveDown() {
        if (currentFloor > 1) {
            currentFloor--;
            cout << "Moved down to floor " << currentFloor << endl;
        } else {
            cout << "Already at the ground floor!" << endl;
        }
    }
};

int main() {
    Elevator e;

    e.printFloor();
    e.moveUp();
    e.moveUp();
    e.moveDown();
    e.moveDown();
    e.moveDown(); // ցույց է տալիս, որ ներքև այլևս հնարավոր չէ

    return 0;
}
```

50. quickSort կարգաբերման ալգորիթմը

```
#include <iostream>
using namespace std;

class QuickSort {
private:
    int arr[5];  // պարզ օրինակ՝ 5 էլեմենտով
    int size;

    int partition(int low, int high) {
        int pivot = arr[high];
        int i = low - 1;
        for (int j = low; j < high; j++) {
            if (arr[j] <= pivot) {
                i++;
                int temp = arr[i];
```

```
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1;
  }

  void quickSortRecursive(int low, int high) {
    if (low < high) {
      int pi = partition(low, high);
      quickSortRecursive(low, pi - 1);
      quickSortRecursive(pi + 1, high);
    }
  }

public:
  QuickSort(int a[], int n) {
    size = n;
    for (int i = 0; i < n; i++)
      arr[i] = a[i];  // բուն մասիվի պատճենավորում
  }

  void sort() {
    quickSortRecursive(0, size - 1);
  }

  void printArray() {
    for (int i = 0; i < size; i++)
      cout << arr[i] << " ";
    cout << endl;
  }
};

int main() {
  int array[5] = {9, 3, 7, 1, 5};

  QuickSort qs(array, 5);

  cout << "Original array: ";
  qs.printArray();

  qs.sort();
```

```
    cout << "Sorted array: ";
    qs.printArray();

    return 0;
}
```

51. bubbleSort կարգաբերման ալգորիթմը

```
#include <iostream>
using namespace std;

class BubbleSort {
private:
    int arr[5];  // պարզ օրինակ՝ 5 էլեմենտով
    int size;

public:
    // Կոնստրուկտոր՝ մասիվի պատճենավորում
    BubbleSort(int a[], int n) {
        size = n;
        for (int i = 0; i < n; i++)
            arr[i] = a[i];
    }

    // BubbleSort ալգորիթմ
    void sort() {
        for (int i = 0; i < size - 1; i++) {
            for (int j = 0; j < size - i - 1; j++) {
                if (arr[j] > arr[j + 1]) {
                    int temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    // Տպել մասիվը
    void printArray() {
        for (int i = 0; i < size; i++)
            cout << arr[i] << " ";
        cout << endl;
    }
};

int main() {
    int array[5] = {9, 3, 7, 1, 5};
```

```
    BubbleSort bs(array, 5);

    cout << "Original array: ";
    bs.printArray();

    bs.sort();

    cout << "Sorted array: ";
    bs.printArray();

    return 0;
}
```