# Obyektyönlü proqramlaşdırma

# C++

# Dərs №5

C++ dili ilə obyektyönlü proqramlaşdırma

# Mündəricat

# Bir daha yükləmə haqqında...

Bu gün biz sizinlə operatorlara yükləmə ilə bağlı mövzunu yekunlaşdırırıq. Buna görə də bu bölmədə daha bir neçə nümunənin baxılması üzərində dayanacağıq.

## () "dairəvi mötərizə" operatoru

() "dairəvi mötərizə" operatoru çox zaman bizə sinif istifadəsində sonuncu yaradılmış obyektin məzmununu dəyişdirməyə imkan verilməsini lazım gəldiyi zaman istifadə edilir. Yəni, verilmiş operator vasitəsilə biz sinifdən SetValue tipində funksiyanı çıxardırıq:

```
#include  <iostream>
using      namespace
std; class MyPoint{
    int X;
    int Y;
public:
    MyPoint(){
        X=0;
        Y=0;
    }
    //Obyekt yaradıldıqdan sonra onun sahələrinin
    //qiymətini dəyişdirmək üçün istifadə edilən
    //funksiya. Məz dairəvi mötərizələrə yükləməklə biz
    //ondan qurtulacayıq
    void SetValue(int x,int y){
       X=x;
       Y=y;
    }
    void Show(){
        cout<<X<<" "<<Y<<"\n\n";
    }
};

void main(){
    MyPoint P;
    P.Show();

    //obyektin sahələrinin qiymətinin dəyişdirilməsi
    P.SetValue(2,3);
    P.Show();
}
```





Göründüyü kimi, sinfə aid baxılan nümunə çox sadədir. Onu daha da sədələşdirək :))). Bunun üçün "dairəvi mütərizələri" növbəti sintaksisdə istifadə edək:

```
Qaytarılan_Qiymətin_Tipi operator() (arqumentlərin_siyahısı);
```

Модифицируем код:

```
#include     <iostream>
using  namespace  std;
class MyPoint{
    int X;
    int Y;
public:
    MyPoint(){
        X=0;
        Y=0;
    }

    // () operatoruna yükləmə
    void operator()(int x,int y){
        X=x;
```

```
        Y=y;
    }
    void Show(){
        cout<<X<<" "<<Y<<"\n\n";
    }
};
void main(){
    MyPoint P;
    P.Show();
    //obyektin sahələrinin qiymətlərinin
    //dətişdirilməsi
    P(2,3);
    P.Show();
}
```

Çox sadədir, elə deyilmi??? Lakin həmişə sizin yadda saxlamağınız gərkən hallar var:

1. Binar və unar operatorlarından fərqli olaraq, funksiyanı çağırma operatoru istənilən sayda (0 və daha çox) arqument qəbul edə bilər, başqa sözlə, funksiyanın parametrlərinin sayı yalnız ağıllı mənada məhdudlaşır.
2. Yüklənmiş funksiya çağırma operatoru funksiyanın çağırılması mexanizmini dəyişdirmir, o yalnız, operatorun verilmiş sinfin obyektinə necə tətbiq ediləcəyi zaman interpretasiya olunduqda dəyişdirir.

### new, new[], delete, delete[] operatorlarına yükləmə

Qeyd etmək lazımdır ki, new və delete operatorları ixtiyarı tip üçün, həmçinin, sinif mexanizmi vasitəsilə təyin edilən abstrakt tip üçün də təyin edilir.





Yəni, onların sinif daxilində yüklənməsini yaratmaq vacib deyil. Lakin, obyekt üçün yaddaşın ayrılmasında hansısa qeyri-adi üsuldan istifadə etmək lazım gəlir. Bunun üçün new və delete operatorlarına yükləmək olar. Məsələn, onun olmaması halında da yaddaş ayırmaq, informasiyanın saxlanması üçün diskdə fayl yaratmaq olar. Bu cür yüklənməni həyata keçirmək çox sadədir. Ümumi sintaksisə baxaq:

```
//Bir obyekt altında yaddaş ayırmaq və onu ləğv etmək:
void* operator new(size_t размер){
    код оператора
    return yaddaş_göstəricisi;
}

void operator delete(void* p){
    //operatorun kodu
}

//Çox sayda obyekt altında yaddaş ayırmaq və onları
//ləğv etmək
: void* operator new[](size_t ölçü){
    //operatorun kodu
    return yaddaş_göstəricisi;
}
void operator delete[](void* p){
    //operatorun kodu
}
```

Bəzi xüsusiyyətlər.

1. Ölçü parametri obyektin yerləşdirilməsi üçün lazım gələn baytların sayını ehtiva edir. Bu qiymət avtomatik olaraq formalaşır, yəni, onu aşkar göndərməyə ehtiyac yoxdur.

2. Ölçü parametrinin verilənlər tipi size_t-dir. Bu tip unsigned int kimi bilinən – təxəllüsdür (psevdonim).
3. new operatorunun prototipində onun çağırılması zamanı ötürüləcək əlavə parametrlər göstərmək olar. Bu zaman, əsas odur ki, funksiyanın elanı zamanı birinci yerdə size_t tipində parametr olsun.
4. Yüklənmiş new funksiyası ayrılmış yaddaşın göstəricisini qaytarmalıdır.
5. delete funksiyası boşaldılması lazım gələn yaddaş sahəsinın göstəricisini alır, bu zaman o bu yaddaşı boşaltmalıdır.
6. delete-ə, həmçinin, əlavə parametrlər də göndərmək olar.

İndi isə nümunəyə baxaq:

```
#include <iostream>
using namespace std;
//yaddaşayırma funksiyası üçün kitabxana

#include <malloc.h>

class SomeClass{

    int some;

public:
    //yüklənmiş new və delete operatorları,
    //burada, həmçinin, operatorların daxilinə
    //ötürülən parametrlər də isitifadə edilir
    void * operator new(size_t size,char* str =

    "New"); void operator delete(void* ptr);
```





```
    //yüklənmiş new [] və delete [] operatorları
    void * operator new [] (size_t fullsize,
                                char* str = "New []");
    void operator delete [] (void* ptr);
};

void * SomeClass::operator new(size_t size,char* str)
{
    cout <<"\n"<<str<<"\n";
    /*
    Burada yaddaşın ayrılması üçün standart
    void *malloc( size_t size );funksiyası istifadə
    edilir. Ona ayrılacaq baytların sayını ifadə edən
    size ötürülür. Əgər yaddaş ayrılarsa, malloc
    funksiyasından ayrılmış yaddaş blokunun ilk ünvanı
    qaytarılır.
    */
    void*ptr = malloc(size);

    if(!ptr){
        cout<<"\nError memory new!!!\n";
    }
    else{
        cout<<"\nMemory new - OK!!!\n";
    }
    return ptr;
}

void * SomeClass::operator new[]
                            ( size_t fullsize,char* str)
{
    cout <<"\n"<<str<<"\n"; /*
    Burada yaddaşın ayrılması üçün standart
    void *malloc( size_t size ); funksiyası istifadə
    edilir. Ona ayrılacaq baytların sayını ifadə edən
    size ötürülür. Əgər yaddaş ayrılarsa, malloc
    funksiyasından ayrılmış yaddaş blokunun ilk ünvanı
    qaytarılır.
    */
```

```
    void*ptr = malloc(fullsize);

    if(!ptr){
        cout<<"\nError memory new[]!!!\n";
    }
    else{
        cout<<"\nMemory new[] - OK!!!\n";
    }
    return ptr;
}

void SomeClass::operator delete( void* ptr)
{
    /*
    Yaddaşın silinməsi üçün void free( void *
    memblock ); standart funksiyası istifadə
    edilir. free funksiyası yaddaşı silir,
    memblock silinəcək sahənin başlanğıc
    ünvanıdır
    */
    free(ptr); cout<<"\nDelete
    memory\n";
}

void SomeClass::operator delete[](void* ptr)
{
    free(ptr);
    cout<<"\nDelete [] memory\n";
}

void main()
{
    /*
    new( size_t size,char* str="New")operatorunu
    çağırır, size yerinə SomeClass sinfinin ünvanı
    veriləcək, str isə susmaya görə qiymət alacaq,
    yəni "New"
    */
```





```
    SomeClass *p = new SomeClass;

    /*
    new[](size_t size,char* str="New[]") operatoru
    çağırılır
    size yerinə SomeClass sinfinin tələb olunun
    elementlərin sayına vurulmuş ölçüsü veriləcək
    str isə susmaya görə qymət alacaq, yəni, "New[]"
    */
    SomeClass *r = new SomeClass [3];

    /*
    delete(void*  ptr)  operatoru  çağırılır,  ptr
    yerinə  p  obyekti  altında   ayrılmış  yaddaş
    ünvanı veriləcək
    */
    delete p;

    /*
    delete[](void* ptr) operatoru çağırılır,
    ptr yerinə r obyekti altında  ayrılmış
    yaddaş ünvanı veriləcək.
    */

    delete[]r;
}
```

---

Proqramın icrasının nəticəsi:

```
New
Memory new - OK!!!
New []
Memory new[] - OK!!!
Delete memory
Delete [] memory
```

# Dost funksiyalar

Sinifin üzvü olmayan dost funksiyası onun xüsusi (private) və qorunmuş (protected) üzvlərinə müraciət edə bilər. Funksiya başqa sinifdə “onun razılığı olmadan” verilə bilməz. Dost haqqı qazanmaq üçün funksiya sinfin gövdəsində friend açar sözü ilə təyin edilməlidir. Məhz bu cür təyinin olması zamanı sinif qorunmuş və xüsusi üzvlərinə müraciət haqqı təqdim edir:

```
#include <iostream>

using namespace std;

//Düzbucaqlı sinfi

class rect{

    //genişlik və yüksəklik
    int Width, Height;
    //əks etdirmə simvolu
    char Symb;
    //Simvolu dəyişdirmək üçün dost funksiyanın
    //prototipi:
    friend void friend_put(rect*r,char s);

public:

    //Konstruktor
    rect(int wi, int hi, char si)
    {
        Width = wi;
        Height = hi;
```





```
        Symb = si;
    }
    //Fiqurun ekrana çıxarılması
    void display ()
    {
        cout<<"\n\n";
        for(int i=0;i<Height;i++){
            for(int j=0;j<Width;j++){
                cout<<Symb;
            }
            cout<<"\n\n";
        }
        cout<<"\n\n";
    }
};
//Müəyyən obyektdəki simvolun
//dəyişdirilməsi üçün dost funksiya:
void friend_put(rect*r, char s)
{
    //Bağlı üzvə burada müraciət mümkündür
    //belə ki, funksiya sinif ilə "dostluq edir"
    r->Symb = s;
}
void main ()
{
    //Obyektlərin yaradılması
    rect A(5,3,'A');
    rect B(3,5,'B');
    A. display ();
    B. display ();
    //simvolların friend-funksiya vasitəsilə
    //dəyişdirilməsi

    friend_put(&A,'a'); friend_put(&B,'b');
    A.display ();
    B.display ();
}
```

### *Nümunənin şərhi*

1. friend_put funksiyası rect sinfində dost kimi təyin edilmişdir və adi qlobal funksiya kimi (sinif xaricində, onun adı göstərilmədən, :: operatoru olmadan və friend spesifikatoru olmadan təyin edilmişdir.
2. Dost kimi o sinfin istənilən verilənlərinə müraciət haqqı əldə edir və ünvanı onun tərəfindən birinci parametrin qiyməti kimi ötürüləcək obyektin qiymətini dəyişdirir.
3. main funksiyasında fiqurların ölçülərinin və çıxış üçün simvolların təyin edildiyi iki A və B obyektləri elan edilir. Sonra isə fiqurlar ekranda göstərilir.
4. friend_put funksiyası obyektlərin simvollarını müvəffəqiyyətlə dəyişdirir ki, bu da ekrana yenidən çıxarılmasını nümayiş etdirir.

#### Dost funksiyaların bəzi xüsusiyyətləri

İndi, biz dost funksiyaların yaradılması mexanizmi ilə tanış olduqdan sonra bəzi vacib məqamlar üzərində dayanaq. İndi biz sizə söhbət açacağımız hər bir şey o hal ilə bağlıdır ki, dost funksiya sinfin üzvü deyil. Beləliklə:

1. Dost funksiya çağırılan zaman this göstəricisi almır.
2. Sinfin obyektləri dost funksiyaya yalnız parametrlər apparatı vasitəsilə aşkar ötürülməlidir.
3. Dost funksiyaları onların dostu olduqları obyekt-siniflər, həmçinin, bu obyektlərin göstəriciləri vasitəsilə,





çağırmaq olmaz, başqa sözlə növbəti şəkildə yazmaq qadağandır:

```
obyektin_adı.funksiyanın_adı
obyekt_göstəricisi -> funksiya_adı
```

4. Dost funksiyalar üçün müraciət spesifikatorları (public, protected, private) istifadə edilmir, buna görə də dost funksiyanın prototipinin müəyyən sinfin daxilində yeri əhəmiyyət kəsb eləmir.
5. Dost finksiya dost olduğu sinfin üzvü ola bilməz, lakin o sadəcə qlobal funksiya, həmçinin, əvvəl təyin edilmiş sinfin üzv funksiyası ola bilər.
6. Dost funksiya bir neçə funksiya ilə münasibətdə dost ola bilər.

## Bəzi tətbiqlər

Dost funksiya mexanizmindən istifadə edilməsi siniflər arasında interfeysi sadələşdirməyə imkan verir. Məsələn, dost funksiya eyni zamanda bir neçə sinfin xüsusi və ya qorunan üzvünə müraciət haqqı əldə etməyə imkan veriri. Həmçinin, yalnız bu "gizli" üzvlərə müraciət üçün nəzərdə tutulmuş üzv funksiyaları siniflərdən silmək olar.

Nümunə kimi, "müstəvi üzərində nöqtə" və "müstəvi üzərində düz xətt" kimi iki sinfin dost funksiyasına baxaq.

1. “müstəvi üzərində nöqtə” sinfi nöqtənin (x, y) koordinatlarını vermək üçün üzv verilənləri ehtiva edir.
2. “müstəvi üzərində düzxətt” sinfinin üzv verilənləri A*x+B*y+C = 0 düzxəttin ümumi tənliyinin A, B, C əmsallarıdır.
3. Dost funksiya verilmiş nöqtənin verilmiş düzxətdən olan məsafəsinini təyin edir. Əgər (a, b) — nöqtənin koordinatlarıdırsa, onda tənliyində A, B, C əmsalları olan düzxətt üçün məsafə A*a+B*b+C ifadəsinin qiyməti kimi hesablanır.

A

Aşağıda şərh edilmiş proqramda ortaq dost funksiyası olan siniflər təyin edilmişdir, əsas proqramda bu siniflərin obyektləri daxil edilmişdir və nöqtənin düzxətdən olan məsafəsi hesablanmışdır:

```
#include <iostream>

using namespace std;

//Sinif haqqında ilkin xatırlatma

line_. class line_;

//”müstəvi üzərində nöqtə” sinfi:
class point_
{
    //Nöqtənin müstəvi üzərində koordinatları.
     float x, y;
public:
        //Konstruktor.
        point_(float xn = 0, float yn = 0)
        {
            x = xn;
            y = yn;
        }
        friend float uclon(point_,line_);
};
```





```
//”müstəvi üzərində düzxətt" sinfi:
class line_
{
    //Düzxəttin parametri
        float A, B, C;
public:
        //Konstruktor
        line_(float a, float b, float c)
        {
            A = a;
            B = b;
            C = c;
        }
        friend float uclon(point_,line_);
};

//Dost funksiyanın xarici təyini
float uclon(point_ p, line_ l)
{
    //düzxəttin məsafəsinin hesablanması
    return l.A * p.x + l.B * p.y + l.C;
}
void main()
{
    //P nöqtəsinin təyini

    point_ P(16.0,12.3);

    //L düzxəttinin təyini

    line_ L(10.0,-42.3,24.0);

    cout << "\n Result" << uclon(P,L) << "\n\n";
}
```

# Dost yüklənmə

Beləliklə, biz dost funksiyalara və onların bir neçə tətbiqlərinə baxdıq. Lakin bu xüsusi funksiyaların əsas xüsusiyyətlərindən biri onlar vasitəsilə operatorlara yükləmənin yerinə yetirilməsinin mümkünlüyüdür. Bu cür yükləmə dost adlanır.

Sinfin dost funksiyası kimi operator-funksiyanın tərtib edilməsinin xüsusiyyətlərini nümayiş etdirək.

```
#include <iostream>

using namespace std;

//məntiqi qiymətlərlə işi reallaşdıran sinif
class Flag
{
    bool flag;
    //dost funksiya (! Operatoruna yükləmə
    // - bayrağın qiymətinin əksinə dəyişdirilməsi
    friend Flag& operator !(Flag&f);

public:
    //Konstruktor.
    Flag(bool iF)
    {
        flag = iF;
    }
    //Bayrağın qiymətini mətn formatında göstərən
    //üzv funksiya
```





```
    void display(){
        if(flag) cout<<"\nTRUE\n";
        else cout<<"\nFALSE\n";
    }
};
//Dost operator-funksiyanın təyini.
//(this ötürülmür, buna görə də 1 parametr)
Flag& operator !(Flag & f)
{
    //qiymətin əksinə dəyişdirilməsi
    f.flag=!f.flag;
    return f;
}

void main()
{
    Flag A(true);

    //ilkin qiymətin göstərilməsi
    A.display();

    //yüklənmiş operator vasitəsilə
    //qiymətin əksinə dəyişdirilməsi
    A=!A;
    //dəyişmiş qiymətin göstərilməsi
    A.display();
}

Proqramın icrasının nəticəsi:

TRUE
FALSE
```

## Qlobal yükləmə

C++ dilində sizə məlum olan iki müxtəlif yükləmədən (sinfə yükləmə və dost yükləmə) başqa daha bir anlayış – xarici görünmə oblastı ilə yerinə yetirilən qlobal yükləmə də mövcuddur.

Tutaq ki, a və b dəyişənləri C sinfinin obyektləri kimi elan edilmişdir. C sinfində C::operator+(C) operatoru təyin edilmişdir, ona görə də

```
a+b a.operator+(b) deməkdir
```

Lakin, + operatoruna qlobal yükləmə də mümkündür:

```
C operator+(C,C) {....}
```

Yükləmənin bu variantı, a+b ifadəsinə də tətbiq edilir. Burada a və b uyğun olaraq funksiyanın birinci və ikinci parametrinə ötürülür. Bu iki formadan sinfə yükləməyə daha çox üstünlük vürilir. Belə ki, ikinci forma sinfin üzvünə aşkar müraciət etməyi tələb edir ki, bu da inkapsulyasiyanın ciddi estetikasına mənfi təsir göstərir. İkinci forma kitabxanada olan siniflərə uyğunlaşmaq üçün daha rahatdir, belə ki, bu kitabxanalarda cari mətni dəyişdirmək və kompilyasıya etmək mümkün deyil. Yəni, sinfə onun metodu kimi yükləməni daxil etmək real deyil.

Bu iki formanı proqramda birləşdirmək məqsəduyğun deyil. Əgər hər hanası bir operator üçün hər iki forma eyni tipli formal parametrlərlə təyin edilərsə, onda operatorun istifadəsi ikimənalılığa səbəb olacaq ki, bu da qəbul edilməzdir.





Bundan başqa, operatorlara qlobal yükləmə estetik məna kəsb edən simmetrikliyi təmin edir. Nümunəyə baxaq:

```
#include <iostream>

using namespace std;

//"nöqtə" sinfi
class Point
{
    //nöqtənin koordinatları
    int X;
    int Y;
public:
    //konstruktor
    Point(int iX,int iY){
        X=iX;
        Y=iY;
    }

    //ekranda göstərmə
    void Show(){
        cout<<"\n++++++++++++++++++++++\n"
        ; cout<<"X = "<<X<<"\tY = "<<Y;
        cout<<"\n++++++++++++++++++++++\n"
        ;
    }
    //yüklənmiş + operatoru
    //Point+int halı üçün sinif metodu
    Point&operator+(int d){
        Point P(0,0);
        P.X=X+d;
        P.Y=Y+d;
        return P;
    }
    //private-üzvlərə müraciət funksiyaları

    //onlarsız qlobal yükləmə mümkün deyi
```

```
    int GetX() const{
        return X;
    }
    int GetY() const{
        return Y;
    }
    void SetX(int
        iX){ X=iX;
    }
    void SetY(int
      iY){ Y=iY;
    }
};
    //private-üzvlərə müraciət funksiyaları
    //onlarsız qlobal yükləmə mümkün deyi
Point&operator+(int d,Point&Z){
        Point P(0,0);
        P.SetX(d+Z.GetX());
        P.SetY(d+Z.GetY())
        ; return P;
}

void main()
{
    //obyektin yaradılması
    Point A(3,2);
    A.Show();

    //operator-metod
    + Point B=A+5;
    B.Show();

    //qlobal operator
    Point C=2+A;
    C.Show();
}
```





Qlobal yükləmə olmadan int + Point məsələsi həll edilmir. Belə ki, biz "döğma" tam tipə (yəni, int tipinə) müraciət edə bilmirik və onun əməliyyatını təyin etmək, sinfin operatorlarının sadə təyin edilməsinin simmetrikliyini təmin etmək mümkün olmur. Qlobal funksiyalarla həll tələb olunur.

**Qeyd:** Burada biz dost yükləməni tətbiq edə və bu şəkildə "private-üzv funksıyalarına müraciətdən" qurtulmuş olardıq. Lakin, Point sinfinin gövdəsi bizim üçün bağlı olsa idi, onda ona dost-funksiya yazmaq real olmazdı.

## Verilənlərin giriş/çıxışına yükləmə

Yükləmə haqqında yeni əldə edilən informasiyanı möhkəmləndirmək üçün << və >> operatorlarına yükləməyin mümkünlüyünə baxaq. Başlanğıc üçün bəzi məlumatlar:

- İstənilən C++ proqramının icrası iostream fayl-kitabxansında siniflərin obyektləri kimi elan edilmiş təyin edilmiş açıq axınların dəsti ilə başlayır. Onlar arasında bizim ən çox istifadə etdiyimiz iki obyekt — cin və cout da var
- **cin** — istream sinfinin obyektidir (digər giriş axınları üçün baza sinfi olan giriş üçün ümumi təyinatlı axın sinfidir)
- **cout** — ostream sinfinin obyektidir (digər çıxış axınları üçün baza sinfi olan çıxış üçün ümumi təyinatlı axın sinfidir)
- Axına çıxış yüklənmiş << sola sürüşdürmə operatoru olan əməliyyat vasitəsilə icra edilir

Onun solundakı operand çıxış axını obyektidir. Sağdakı operator kimi axına çıxış kimi qəbul edilən istənilən dəyişən istifadə edilə bilər. Məsələn, cout << "Hello!\n"; operatoru çıxışa təyin edilmiş cout axınının "Hello!" setrini verir.

- İnformasiyanın axından daxil edilməsi üçün axına giriş yüklənmiş >> sağa sürüşdürmə operatoru olan əməliyyat vasitəsilə icra edilir. >> əməliyyatının sol operandı istream sinfinin obyektidir.

Gözlənilməz hallardan qaçmaq üçün abstrakt tipli verilənlər üçün giriş-çıxış standar tiplər üçün giriş-çıxış əməliyyatları üçün təyin edilmiş qaydalara uyğun olmalıdır:

1. Giriş və çıxış əməliyyatları üçün qaytarılan qiymət bir neçə əməliyyatın eyni ifadədə icra edilməsi üçün axına istinad olmalıdır.
2. Funksiyanın birinci parametri verilənlərin çıxarılacağı axın olmalıdır, ikinci parametr isə istifadəçi tərəfindən təyin edilmiş obyektə istinad və ya göstərici olmalıdır.
3. Sinfin bağlı verilənlərinə müraciət etməyə icazə vermək üçün giriş və çıxış əməliyyatları sinfin dost funksiyası kimi təyin edilməlidir.
4. Çıxış əməliyyatına sinfin obyektinə sabit (konstant) istinad ötürmək lazımdır. Belə ki, bu əməliyyat daxil edilən obyekti dəyişdirməməlidir.

İndi isə bu cür yükləməyə aid nümunəyə baxaq:





```
#include <iostream>

using namespace std;

//"nöqtə" sinfi
class Point
{
    //nöqtənin koordinatları
    int X;
    int Y;

public:

    //konstruktor
    Point(int iX,int iY){
        X=iX;
        Y=iY;
    }
    //verilənlərin giriş və çıxışına dost yükləmə
    //funksiyaları
    friend istream& operator>>(istream& is, Point& P); friend
    ostream& operator<<(ostream& os, const Point& P);
};

//axından verilənlərin girişi
istream& operator>>(istream&is,
    Point&P){ cout<<"Set X\t";
    is >> P.X;
    cout<<"Set
    Y\t"; is >>
    P.Y; return is;
}
//Verilənlərin axına çıxışı
ostream& operator<<(ostream&os, const
    Point&P){ os << "X = " << P.X << '\t';
    os << "Y = " << P.Y <<
    '\n'; return os;
}
```

```
void main()
{
    //obyektin yaradılması
    Point A(0,0);

    //adi giriş və çıxış
    cin>>A;
    cout<<A;

    //mürəkkəəb ifadə
    Point B(0,0);
    cin>>A>>B;
    cout<<A<<B;
}
```

**Qeyd:** Yeri gəlmişkən!!! Nümunələrdən birində biz sinfin sahələrini dəyişmə haqqı olmayan sabit (konstant) metodundan istifadə etdik. Lakin, hansısa sahə mutable spesifikatoru vasitəsilə təyin edilmişdirsə, onun qiymətini const tipində metodda dəyişdirmək olar.





# Dost siniflər

Artıq bilmək lazımdır ki, yalnız funksiyalar “dostluq” etmirlər. Sinif başqa bir siniflə dost ola bilər.

### Siniflər arasında “dostluğun” xüsusiyyətləri

1. Dost sinif “dostluq təklif edən” sinfin xaricində təyin edilməlidir.
2. Sinif-dostun bütün üzv funksiyaları friend spesifikatorunu göstərmədən başqa sinif üçün dost olacaqdır.
3. Sinfin bütün üzvləri dost sinifdə əlyetərlidir, əksi isə belə deyil.
4. Dost sinif dost kimi verildikdən sonra təyin edilə bilər.

Yuxarıda deyilənləri nümayış etdirən sadə nümunəyə baxaq.

```
#include <iostream>

using namespace std;

//sonra aşağıda veriləcək sinif
//haqqında xatırlatma
class Banana;

//dost olacaq sinif
```

```
class Apple{
public:
    void F_apple(Banana ob);
};
//özü ilə dostluğa icazə verən sinif
class Banana{
    int x,
y; public:
    Banana(){
        x=y=777;
    }
    //dostluğun reallaşdırılması
    friend Apple;
};

//avtomatik "dost" olan funksiya
void Apple::F_apple(Banana ob)
{
    //private-üzvlərə müraciət
    cout<<ob.x<<"\n";
    cout<<ob.y<<"\n";
}

void main(){
    Banana b;
    Apple a;
    a.F_apple(b)

}
```





# Sinfin statik üzvləri

Eyni bir sinfin hər bir obyektinin sinfin verilənlərinin məxsusi nüsxəsi vardır. Demək olar ki, sinfin verilənləri bu sinfin obyektinin hər təyinində çoxaldılır. Onlar bir-birindən bu və ya digər obyektə bağlılıqları ilə fərqlənirlər. Bu həll edilən məsələnin tələblərinə heç də həmişə cavab vermir. Məsələn, sinfin obyektlərinin formalaşdırılması zamanı obyektlərin sayğacı tələb oluna bilər. Sözsüz ki, belə sayğacı sinfin üzvü etmək olar, lakin onu tək sayda etmək olar. Sinfin üzvünün yalnız bir nüsxə olması və sinfin yeni obyektinin yaradılması zamanı çoxaldılmaması üçün o sinif daxilində statik təyin edilməlidir və static atributu ehtiva etməlidir.

## Sinfin statik sahələri

Beləliklə, sinfin sahəsini static açar sözü ilə elan etmək olar. Bu cür sahə altında yaddaş, proqramın icrası zamanı yəni, proqramcı verilmiş abstrakt tipdə ilk obyekti aşkar yaradana qədər ayrılır. Bu zaman bütün obyektlər, onların sayından asılı olmayaraq əvvəlcədən yaradılmış özünün statik üzvünün yeganə nüsxəsini istifadə edir.

Sinfin statik üzvü sinfin təyinindən sonra və bu sinfin obyektinin ilkin təyinindən əvvəl qiymətləndirilməlidir (inisiallaşdırılmalıdır). Bu əməliyyat növbəti formada olan

Statik üzvün tam və ya seçilmiş adı ilə aparılır:

```
Sinfin_adı::statik_üzvün_adı
```

***Nümunəyə baxaq.*** Statik üzvdə, object_ tipində obyektlərin zamanın hər anında mövcud olan mövcud sayının saxlandığı object_ sinfini reallaşdırırıq.

```
#include   <iostream>
#include   <string.h>
using namespace std;
class object_{
    char *str;
public:
    //sinfin statik sahəsi
    static int num_obj;
    //konstruktor
    object_ (char *s){
        str = new char [strlen (s) +
        1]; strcpy ( str, s );
        cout <<"Create " << str <<'\n';

        //sayğacın qiymətini artırırıq
        num_obj ++ ;
    }
    //destruktor
    ~object_ (){
        cout <<"Destroy " << str <<
        '\n'; delete str;
        //sayğacın qiymətini azaldırıq
        num_obj --;
    }
};
```





```
//Qiymətləndirmə (inisiallaşdırma)
//int açar sözü bunu göstərir!
int object_::num_obj = 0;

//qlobal obyektlərin yaradılması
object_ s1 ("First global object.");
object_ s2 ("Second global object.");

//köməkçi funksiyalar
void f (char *str) {
    //Lokal obyekt
    object_ s(str);
    //statik sahəyə obyekt göstəricisi olmadan aşkar
    //müraciət
    cout <<"Count of objects - "
                            <<object_::num_obj<<".\n";
    cout <<"Worked function f()" <<".\n";
}

void main () {
    //Statik sahəyə aşkar müraciət
    cout <<"Now, count of objects"
                            <<object_::num_obj<<".\n";

    object_ M ("Object in main ().");

    //obyekt vasitəsilə statik sahəyə müraciət

    cout <<"Now, count of objects" << M.num_obj

    <<".\n";

    f ("Local object.");
    f ("Another local object.");
    cout << "Before finish main() count of objects - "
                            <<object_::num_obj<<".\n";
}

Proqramın icrasının nəticəsi:
Create First global object.
```

```
Create Second global object.
Now, count of objects 2.
Create Object in main ().
Now, count of objects3.
Create Local object.
Count of objects - 4.
Worked function f().
Destroy Local object.
Create Another local object.
Count of objects - 4.
Worked function f().
Destroy Another local object.
Before finish main() count of objects - 3.
Destroy Object in main ().
Destroy Second global object.
```

**Qeyd:** Diqqət edin ki, qlobal obyektlər üçün konstruktorlar main() funksiyasına qədər destruktorlar isə main() funksiyasından sonra çağırılır.

Sinfin statik verilənləri üçün müraciətin statusu qaydaları tətbiq edilir. Əgər statik verilənlər private statusunda olarlarsa, onlara xaricdən üzv funksiyalar vasitəsilə müraciət etmək olar. Başqa sözlə, private tipində statik sahəyə müraciətin tələb edildiyi anda sinfin obyekti təyin edilməmişdirsə, müraciət mümkün deyildir. Razılaşın ki, sinfin statik verilənlərinə müraciət zamanı müəyyən obyektin adı olmadan keçinmək imkanının olmasını istərdik. Bu məsələ statik üzv funksiya-metod vasitəsilə həll edilir.





## Sinfin statik metodları

Sinfin üzv-funksiyasının təyinindən əvvəl static açar sözü qoymaq olar və o statik üzv-funksiya olacaqdır. Belə funksiya sinfin adı (statik olmayan) funksiyasının bütün xüsusiyyətlərini saxlayır. Ona sinfin artıq mövcud olan obyektinin adı ilə və ya bu obyektin göstəricisi ilə müraciət etmək olar.

Bundan başqa, statik üzv funksiyanı növbəti şəkildə çağırmaq olar:

```
Sinfin_adı::statik_funksiyanın_adı
```

Statik üzv-funksiyalar sinfin private-statik verilənlərinə proqramda verilmiş tipdə heçbir obyektin olmadığı halda belə müraciət etməyə imkan verir.

Lakin diqqətli olun!!!

**Statik üzv funksiya üçün this göstəricisi təyin edilməmişdir. Bu lazım olanda statik üzv-funksiyanın çağırıldığı obyektin ünvanı funksiyaya arqument kimi aşkar ötürülməlidir.**

*Nümunəyə baxaq:*

```
#include <iostream>

using namespace std;

class prim{
    int numb;
    //statik sahə

    static int stat_;

public:
    prim (int i)
        { numb=i;
    }
```

```
    /*
    Statik funksiya. this göstəricisi təyin
    edilməmişdir, obyektin seçilməsi aşkar ötürülən
    göstərici ilə həyata keçirilir.
    stat_ sahəsi obyekt göstəricisi tələb etmir,
    belə ki, o prim sinfinin bütün obyektləri üçün
    eynidir.
    */
    static void func (int i, prim *p = 0) {
        //əgər heç olmazsa bir obyekt varsa
        if(p)
            p->numb = i;
        //sinfin obyektləri yoxdursa
        else
            stat_ = i;
    }
    /*
    Statik funksiya sinfin yalnız statik üzvlərinə
    müraciət edir, heç bir göstərici tələb edilmir.
    */
    static void show(){
        cout<<"stat_="<<stat_<<"\n\n;
    }

    //statik olmayan üzvün göstərilməsi
    void show2(){
        cout<<"numb="<<numb<<"\n\n";
    }
};
//Sinfin statik üzvünün qiymətləndirilməsi

//(inisiallaşdırılması)

int prim::stat_ = 8;

void main(){
    /*
    prim tipində obyektin yaradılmasına qədər
```





```
    statik üzv-funksiyaya yeganə müraciət
    üsulu ola bilər
    */
    prim::show ();
    //Sinfin statik üzvünün qiymətiini dəyişdirmək
    //olar
    prim::func(10);

    /*
    prim tipində obyektin yaradılmasından
    sonra statik üzv-funksiyaya adi abstrakt
    tiplər üçün olan üsullarla müraciət etmək
    olar
    */
    //obj obyekti və onun numb sahəsi
    //yaradılır, qiyməti 23 olur
    prim obj(23);

    obj.show2();

    //Yaradılmış obyektin qiyməti dəyişdirilə

    //bilər

    prim::func(20, &obj); //obj.numb 20.

    obj.show2();

    obj.func(27, &obj); //obj.numb
    27. obj.show2();
}
```

## Singleton şablonu (Singleton pattern)

Statik üzvlərin tətbiqinə aid nümunələrdən biri Singleton pattern adlı konstruksiyadır. Bu konstruksiya yalnız bir sinif nümunəsi yaratmağa imkan verir və bu ekzemplyara qlobal müraciət etməyə imkan verir.

Bəzən sinfin yalnız bir nüsxə (obyekt) yaratması lazım gəlir. Məsələn, sistemin bir neçə printeri ola bilər, lakin

Yalnız bir yığan (spuler) printer olmalıdır. Biz necə zəmanət verə bilərik ki, yalnız bir nüsxə sinif var və bu nüsxə əlyetərlidir?! Məsələn, qlobal dəyişən təyin etmək olar. Lakin, bu üsul yaxşı deyil, ona görə ki, biz birincisi adlar sahəsini korlayırıq, ikincisi isə sinfin bir neçə nüsxəsinin yaradılmasını aradan qaldıra bilmirik.

Ən yaxşı həll nüsxələrin yaradılmasını və yeganə obyektə müraciəti özü nizamlayan sinifdir. Biz elə edə bilərik ki, başqa heç bir nüsxənin yaradılmamasına sinif zəmanət versin.

Yalnız bir nüsxə yaradılması mümkün olan sinif yazmaq lazımdır, lakin bu nüsxəni digər siniflərin obyektləri də isitfadə edə bilməlidir. Bu tapşırığı reallaşdıran çox sadə sxem budur:

```
#include <iostream>

using namespace std;

class Singleton{

private:
    //sinfin yeganə nüsxəsinin göstəricisi
    static Singleton*s;
    int k;

    //bağlı konstruktor
    Singleton(int i){ k = i;
    }
public:
    //sinfin yeganə nüsxəsinin göstəricisinin əldə
    //edilməsi üçün funksiya
```





```
    static Singleton*getReference(){
        return s;
    }

    //sinfin statik olmayan üzvünün qiymətinin
    alınması
    int getValue(){
        return k;
    }
    //sinfin statik olmayan üzvünün qiymətinin
    //yazılması
    void setValue(int i){
        k = i;
    }
};

//Sinfin statik üzvünün qiymətləndirilməsi
(inisiallaşdırılması)
Singleton* Singleton::s=new Singleton(3);

void main(){
    //sinfin yeganə nüsxəsinin göstəricisinin əldə
    //edilməsi
    Singleton*p=Singleton::getReference();

    //obyektin üzvü ilə iş
    cout<<p->getValue()<<"\n\n"; p-
    >setValue(p->getValue()+5);
    cout<<p->getValue()<<"\n\n";
}
```

### *Nümunənin şərhi*

- Класс Singleton tamdır — onu genişlətmək olmaz.
- Bu konstruktor bağlıdır — heçbir metod bu sinfin nüsxəsini yarada bilməz.
- Singleton-un yeganə s nüsxəsi — statikdir, o sinfin daxilində yaradılır, lakin bu nüsxənin göstəricisini

getReference() metodu ilə əldə etmək, s nüsxəsinin vəziyyətini setValue() metodu ilə dəyişdirmək və ya onun cari vəziyyətinə getValue() metodu ilə baxmaq mümkündür.

Sözsüz ki, bu yalnız sxemdir — Singleton sinfini hələ faydalı məzmunla doldurmaq lazımdır, lakin ideya aydın və tam ifadə edilmişdir.





# Ev tapşırığı

1. Toplama, çıxma, müqayisə, giriş və çıxış əməliyyatlarının reallaşdırıldığı Zaman sinfini yaradın. Zamanın amerikan formatında çevrilməsi imkanı olmalıdır: am (pm): 10:00 pm = 22:00, 12:00 pm =00:00
2. Matrislərlə işləyən sinif yaradın. Ən azından matrislərin toplanması, vurulması, tərs çevrilməsi, matrislərin bir-birinə mənimsədilməsi, matrisin ixtiyari elementinin qurulması və əldə edilməsi funksiyaları olmalıdır. Uyğun operatorlara yükləmə olmalıdır.