

**Obyektyönlü
programlaşdırma**

C++



Sabit metodu

Mündəricat

Sabit metodu	3
SƏTİR sinfinin yaradılmasına aid nümunə	7
Operatorlara yükləmə.....	12
Siniflə təyin edilən çevirmələr	16
Operatorlara yükləmə ilə SƏTİR sinfinə nümunə	20
Ev tapşırığı	27

Obyekt metodu onun icrasından sonra obyektin vəziyyəti dəyişməzsə, o dəyişməzlik (sabitlik) xüsusiyyəti qazanır. Əgər dəyişməzlik xüsusiyyətinə nəzarət edilməzsə, onda onun təmin edilməsi tamamilə proqramçının qiymətləndirməsindən asılıdır. Əgər dəyişməzlik metodu icra prosesində kənar effektlər verəcəksə, onda nəticə çox gözlənilməz olacaq, bu cür kodu sazlamaq və dəstəkləmək çətindir.

C++ dili metodu sabit kimi qeyd etməyə imkan verir. Bu zaman obyektin sabit olmayan metodlarının qeyd edilmiş metodun gövdəsində istifadəsi qadağandır və bu metodun məzmununda obyektin özünə və onun bütün sahələrinə istinad sabit olacaq. Sabitlik təyini üçün `const` açar sözündən istifadə edilir.

Qeyd: Yeri gəlmişkən!!! Həmçinin, istinad (və ya göstəricini) də sabit kimi qeyd etmə imkanı var. İstinada sabitlik xüsusiyyətinin tətbiq edilməsi, o deməkdir ki, bu istinad vasitəsilə yalnız sabit metodu çağırmaq olar. Sabit olmayan istinada sabit olanı mənimsətmək qadağan edilmişdir.

Gəlin sabit metoda aid misala baxaq:

```
#include <iostream>
#include <string.h>
```

```

using namespace
std; class Personal
{
public:
    //parametrlı konstruktör
    //biz burada yaddaş ayırırıq
    //lakin bizim misalda nə
    //destruktör, nə də konstruktör var
    //sabit metodun işini nümayiş
    //etdirmək üçün biz köçürmənin
    //yeganə məqsəd olduğunu göstəririk
    Personal(char*p,char*n,int
a){ name=new
char[strlen(n)+1]; if(!name){
        cout<<"Error!!!";
        exit(0);
    }
    picture_data=new char[strlen(n)+1];
    if(!picture_data){
        cout<<"Error!!!";
        exit(0);
    }
    strcpy(picture_data,p)
    ; strcpy(name,n);
    age=a;
}

//Sabit metodlar qrupu
//onların daxilində hansısa
//xassələrədən birini dəyişdirmək olmaz
const char*Name()const{
    return name;
}
int Age()const{
    return age;
}
}

```

```

const char*Picture()const{
    return picture_data;
}
void SetName(const char*n){
    strcpy(name,n);
}
void SetAge(int a){
    age=a;
}
void SetPicture(const char*p){
    strcpy(picture_data,p);
}

private:
    char*picture_data; //fotoya yol
    char*name; //ad
    int age; //yaş
};

void main(){
    Personal A("C:\\Image\\", "Ivan", 23);
    cout<<"Name: "<<A.Name()<<"\n\n";
    cout<<"Age: "<<A.Age()<<"\n\n";
    cout<<"Path for picture: "<<A.Picture()<<"\n\n";
    A.SetPicture("C:\\Test\\"); A.SetName("Leonid");

    A.SetAge(90);
    cout<<"Name: "<<A.Name()<<"\n\n";
    cout<<"Age: "<<A.Age()<<"\n\n";
    cout<<"Path for picture: "<<A.Picture()<<"\n\n";
}

```

Verilmiş misalda Name, Age, Picture metodları sabit kimi elan edilmişdir. Bundan başqa sabit göstəricilərin də istifadə edildiyini görmək olar:

SetName və SetPicture metodlarının parametrləri, Name və Picture metodlarının qaytarılan qiymətləri. Sabit metodların Personal sinfini reallaşdıran obyektin vəziyyətinin dəyişməsi şəklində yan effektlər vermir. Qadağan edilmiş əməliyyatı icra etməyə cəht edildiyi zaman, kompilyator səhv haqqında məlumat verəcək.

SƏTİR sinfinin yaradılmasına aid misal

İndi isə keçilmiş materialı qüvvətləndirmək üçün növbəti məsələyə baxaq:

Sətirlərlə işi həyata keçirən sinif yaratmaq: qiymətləndirmə, giriş-çıxış və çeşidləmə funksiyalarının reallaşdırılması.

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

class string_
{
private:
    //Sətir
    char* S;

    //Sətrin uzunluğu
    int len;
public:
    //parametrsiz susmaya
    //görə konstruktor
    string_();

    //Yüklənmiş parametrli konstruktor
    string_(char*
s);

    //Köçürmə konstruktoru
    string_(const string_ & s);
```

```

//Destruktor
~string_(){
    delete [] S;
}

//Sıralama metodu
void Sort(string_ s[], int n);

//sətrin məzmununu qaytaran sabit metod

const char*GetStr()const
{
    return S;
}
//Məzmunu istifadəçi tərəfindən
//dəyişdirilə bilən metod
void SetStr()
{
    //sətir boş deyilsə, silməli
    if(S!=NULL)
        delete[]S;
    //massivin yaradılması və
    //istifadəçidən verilənlərin daxil
    //edilməsinin istənməsi
    char a[256];

    cin.getline(a,256);

    //ölçünü oxuyuruq
    len=strlen(a)+1;

    //yaddaş ayırırıq
    S = new char[len];

    //daxil edilmiş sətri
    //obyektə yazırıq
    strcpy(S,a);
}

```

```

//parametr vasitəsilə məzmunun
//dəyişdirilməsinə imkan verən metod
void SetStr2(char*str)
{
    //sətir boş deyilsə, sil
    if(S!=NULL)
        delete[]S;
    //ölçünü oxuyuruq
    len=strlen(str)+1;
    //yaddaş ayırırıq
    S = new char[len];
    //daxil edilmiş sətri
    //obyektə yazırıq
    strcpy(S, str);
}

};

string_::string_()
{
    //ilkin qiymətləndirmə
    S = NULL;
    len = 0;
}

string_::string_(char* s)
{
    len = strlen(s);
    S = new char[len + 1];
    //istifadəçinin göndərdiyi
    //sətirle qiymətləndirmə
    strcpy(S, s);
}

string_::string_(const string_& s)
{
    len = s.len;
    //Təhlükəsiz köçürmə
    S = new char[len + 1];
}

```

```

        strcpy(S, s.S);
    }

void string_::Sort(string_ s[], int n)
{
    //Qabarcıqlı çeşidləmə metodu
    //ilə sətirin çeşidlənməsi
    string_ temp;
    for(int i=0;i<n-1;i++)
    {
        for(int j=n-1;j>i;j--)
        {
            //iki sətirin müqayisəsi
            if(strcmp(s[j].S,s[j-1].S)<0)
            {
                //s[j] sətirinin temp-ə yazılması
                temp.SetStr2(s[j].S);
                //s[j-1] sətirinin s[j]-yə yazılması
                s[j].SetStr2(s[j-1].S);
                //temp sətirinin s[j-1]-ə yazılması
                s[j-1].SetStr2(temp.S);
            }
        }
    }
}

void main()
{
    int n,i;
    //Sözlərin sayını daxil edirik
    cout << "Input the number of string
s:\t"; cin >> n;
    if(n < 0)
    {
        cout << "Error number:\t" << n <<
endl; return;
    }
}

```

```

//axından Enter ("\n") simvolunu qəbul edirik
char c[2];
cin.getline(c, 2);

//n sətirdən ibarət massiv yaradırıq
string_ *s = new string_[n];
//Klaviaturadan sətiri daxil edirik
for(i = 0; i < n; i++)
    s[i].SetStr();

//Sətrlərin çeşidlənməsi
//Göstərici vasitəsilə çağırma, belə ki,
//funksiya yalnız bir deyil, bir qrup
//obyektlər üçün işləyir
s->Sort(s, n);
//Çeşidlənmiş sətirlərin çıxışa verilməsi
for(i = 0; i < n; i++)
    cout<<"\n"<<s[i].GetStr()<<"\n";

//Sətirlər massivinin ləğv edilməsi

delete [] s;
}

```

Operatorların yüklənməsi

C++-da abstrakt verilənlər tipli operandlar üzərində standart əməliyyatların aparılması imkanı vardır. Abstrakt tipli operandlar üzərində standart əməliyyatlardan birini təyin etmək üçün, proqramçı operator işarə adında funksiya yazmalıdır, burada işarə bu əməliyyatın işarəsidir (məsələn, + - | += və s.).

Lakin C++-da operatorların təyini zamanı bəzi məhdudiyyətlər vardır:

1. Əməliyyatlar üçün yeni simvollar yaratmaq olmaz.
2. Əməliyyatları yenidən təyin etmək olmaz:

```

::
* (adlandırma, binar vurma deyil)
?:
sizeof
f
##
#
.

```

3. Unar simvol əməliyyatı binar əməliyyat üçün və əksinə təyin edilə bilməz. Məsələn, << simvolu yalnız binar əməliyyat üçün istifadə edilə bilər, ! – yalnız unar, & – isə həm unar, həm də binar üçün.

4. Əməliyyatların təyin edilməsi onların prioritetlərini və icra ardıcılıqlarını (soldan sağa və ya sağdan sola) dəyişdirmir.
5. Əməliyyatların yüklənməsi zamanı kompüter onların xassələri haqqında heç bir ehtimal eləmir. Bu o deməkdir ki, əgər standart += əməliyyatı + və = əməliyyatları ilə ifadə olunarsa, yəni, $a + = b$ və $a = a + b$ ekvivalent olarlarsa, onda əməliyyatların bu cür yüklənməsi mövcud deyildir, buna baxmayaraq proqramçı bunu təmin edə bilər.
6. Heç bir əməliyyat standart tipli operandlar üçün təyin edilə bilməz.
7. Unar əməliyyatlar kimi binar əməliyyatlar üçün də operator () funksiyasının argumentlərinin sayı bu əməliyyatın operandlarının sayı ilə tamamilə eyni olmalıdır. Bundan başqa binar operatora yükləmə zamanı bir argument ötürmək lazımdır, belə ki, ikincisi aşkar deyil. İstənilən funksiyada bu var – sinfin üzvü – bu elə this - metodun çağırıldığı obyekt göstəricisidir. Beləliklə, unar operatorun təyini zamanı ümumiyyətlə heç bir şey göndərmək lazım deyil.

Qeyd: operator () funksiyaya parametrlərin ötürülməsi qiymətə görə deyil, istinada görədir.

Nömunə:

```

#include <iostream>

using namespace std;

```

```

class Digit{
private
:
    int dig;
    //ədəd
public:
    Digit(){
        dig=0;
    }
    Digit(int iDig){
        dig=iDig;
    }
    void Show(){
        cout<<dig<<"\n";
    }
    //dört operator yükləyirik
    //diqqət edin ki, bütün operatorlar
    //binardırlar, buna görə də biz
    //onlara bir parametr göndəririk,
    //bu ifadədə operatorun sağında
    //duran operanddır.
    //sol operand this vasitəsilə ötürülür.
    Digit operator+(const Digit &N)
    {
        Digit temp;
        temp.dig=dig+N.dig; return temp;
    }
    Digit operator-(const Digit &N)
    {
        Digit temp;
        temp.dig=dig-N.dig;
        return temp;
    }
    Digit operator*(const Digit &N)
    {
        Digit temp;
        temp.dig=dig*N.dig
        ; return temp;
    }
}

```

```

        Digit Digit::operator%(const Digit &N)
        {
            Digit temp;
            temp.dig=dig%N.dig
            ; return temp;
        }
};

void main()
{
    //operatorların işinin yoxlanılması
    Digit A(8),B(3);
    Digit C;

    cout<<"\nDigit A:\n";
    A.Show();

    cout<<"\nDigit B:\n";
    B.Show();

    cout<<"\noperator+:\n";
    C=A+B;
    C.Show();

    cout<<"\noperator-
    :\n"; C=A-B;
    C.Show();

    cout<<"\noperator*:\n";
    C=A*B;
    C.Show();

    cout<<"\noperator%:\n";
    C=A%B;
    C.Show();
}

```


Siniflə təyin olunan çevrilmə

Şərti olaraq tiplərin bütün çevrilmələrini dörd əsas qrupa ayırmaq olar:

- ■ **Standartın standart** — bu çevrilmələrə artıq dərslərin birində ətraflı baxmışıq.
- ■ **Standartın abstrakta** — bu qrup çevrilmələr konstruktorların istifadəsinə əsaslanır.

```
#include <iostream>
using namespace
std; class Digit
{
    private:
        int
    dig; public:
        Digit(int iDig){
            dig=iDig;
        }
        void Show(){
            cout<<dig<<"\n"
            ;
        }
};
void main()
{
    //int-dən Digit-ə çevrilmə
    Digit A(5);
    A.Show();
}
```

```
//double-dan Digit-ə çevrilmə
Digit B(3.7);
B.Show();
}
```

Verilmiş nümunəyə əsasən belə nəticə çıxarmaq olar ki, tək argumentli `Class::Class(type)` konstruktoru həmişə yalnız obyektə aşkar müraciət zamanı onun yaradılması üsulunu deyil, həmçinin, `type` tipinin `Class` tipinə çevrilməsini təyin edir.

- ■ Abstraktın standart
- ■ Abstraktın abstrakta

Abstrakt tipin standart və ya abstraktın abstrakta çevrilməsi zamanı C++-da vasitələr - tiplərin çevrilməsini icra edən funksiya və ya tiplərin çevrilməsini icra edən operator funksiyalar mövcuddur. O növbəti sintaksisə malikdir:

```
Class::operator type (void);
```

Bu funksiya istifadəçi tərəfindən təyin edilmiş `Class` tipinin `type` tipinə çevrilməsini yerinə yetirir. Bu funksiya `Class` sinfinin üzvü olmalıdır və argumentləri yoxdur. Bundan başqa, onun təyininə qaytarılan qiymətin tipi göstərilir.

Bu funksiya həm aşkar, həm də qeyri-aşkar müraciət oluna bilər. Aşkar çevrilmənin yerinə yetirilməsi üçün həm ənənəvi, həm də funksional formadan istifadə etmək olar.

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

```

class Number{
    private:
        int
    num; public:
        Number(int iNum){
            num=iNum;
        }
        void Show(){
            cout<<num<<"\n"
            ;
        }
};
class Digit
{
    private:
        int
    dig; public:
        Digit(int iDig){
            dig=iDig;
        }
        void Show(){
            cout<<dig<<"\n"
            ;
        }
        //Digit-dən int-tə çevirmə
        operator int (){
            return dig;
        }
        //Digit-dən Number-ə çevirmə
        operator Number (){
            return Number(dig);
        }
};
void main()
{
    Digit A(5);
    cout<<"In Digit A:\n";

```

```

    A.Show();
    //Digit-dən int-tə çevirmə
    int a=A;
    cout<<"In int a:\n";

    cout<<a<<"\n";

    Digit B(3);
    cout<<"In Digit
    B:\n"; B.Show();

    Number b(0);
    cout<<"In Number b
    (before):\n"; b.Show();
    //Digit-dən Number-ə çevirmə
    b=B;
    cout<<"In Number b
    (after):\n"; b.Show();
}

```

Yüklənmiş operator vasitəsilə SƏTİR sinfinə aid nümunə

İndi əldə etdirlən biliklərə əsasən dərsdə şərh edilən SƏTİR sinfini tamamlayaq.

Xüsusilə də, ona + binar operatoruna yükləmədən istifadə etməklə sətirləri birləşdirmə funksiyasını əlavə edək, mənimsətmə operatorunu yükləyək və bizim obyektə sətirlərin çevrilməsi imkanını yaradaq.

```
#include <iostream>
#include <string.h>

using namespace std;

class string_
{
private:
    //Sətir

    char* S;

    //Sətrin uzunluğu
    int len;
public:
    //Susmaya görə parametrsiz konstruktör
    string_();

    //Parametrli yüklənmiş konstruktör
    string_(char* s);
```

```
//Destruktör
delete [] S;
}

//Çeşidləmə metodu
void Sort(string_ s[], int n);

//Sabit metod
}
//Məzmunun istifadəçi tərəfindən dəyişdirilməsinə
//imkan verən metod

void SetStr()
{
    //sətir boş deyilsə silmək
    if(S!=NULL)

        delete[]S;

    //massiv yaradırıq
    //və istifadəçidən verilənlərin daxil
    //etməsinə istəyirik
    char a[256];
    cin.getline(a,256);

    //ölçünü oxuyuruq
    len=strlen(a)+1;

    S = new char[len];

    //daxil edilmiş sətiri obyektə yazırıq
}
```

```

//Binar operatorun yüklənməsi
//İlk parametr this göstəricisi ilə qeyri-aşkar
//göndərilir
//Funksiya sətirlərin birləşməsinə reallaşdırır
    string_ operator+(const string_&);

//Binar operatorun yüklənməsi
//İlk parametr this göstəricisi ilə qeyri-aşkar
//göndərilir
//Funksiya obьekt1=obьekt2 olduqda obьektlərin bir-
//birinə düzgün mənimsədilməsini reallaşdırır
//Xatırladıyıq ki, bu hal bit-bit köçürmənin
//dördüncü halıdır ki, bu zaman köçürmə konstrukturu
//gücsüzdür.
    string_&operator=(const string_&);

//tipin yüklənməsi
//Funksiya obьekt sinfin char* tipinə
//çevrilməsini reallaşdırır
operator char*() { return S; }
};

string_::string_()
{
    //İlkin qiymətləndirmə
    S = NULL;
    len = 0;
}

string_::string_(char* s)
{
    len = strlen(s);
    S = new char[len + 1];
    //İstifadəçi tərəfindən ötürülən
    //sətrin qiymtləndirilməsi

    strcpy(S, s);
}

```

```

string_::string_(const string_& s)
{
    len = s.len;
    //Təhlükəsiz köçürmə
    S = new char[len + 1]; strcpy(S, s.S);
}

void string_::Sort(string_ s[], int n)
{
    //Sətirlərin qabarçıqlı çeşidlənməsi
    string_ temp;
    for(int i=0;i<n-1;i++)
    {
        for(int j=n-1;j>i;j--)
        {
            //iki sətirin müqayisəsi
            if(strcmp(s[j].S,s[j-1].S)<0)
            {
                //artıq, bizdə yüklənmiş
                //bərabərlik operatoru varsa,
                //keçən misalda mənimsətmə üçün istifadə
                //etdiyimiz əlavə SetStr2 funksiyasına
                //ehtiyac yoxdur
                // s[j] sətirini temp-ə yazılması
                temp=s[j];

                //s[j-1] sətirinin s[j]
                //sətrinə yazılması
                s[j]=s[j-1];

                //temp sətirinin s[j-1]
                //sətrinə yazılması
                s[j-1]=temp;
            }
        }
    }
}

```

```

//Sətirlərin birləşməsi funksiyası
//(yüklənmiş binar +) string_
string_::operator+(const string_ &str)
{
    //Dəyişən obyektin yaradılması
    string_ s;

    //Sətrin yeni uzunluğunun hesablanması
    s.len = len + str.len;

    //Yeni sətir üçün yaddaşın ayrılması
    s.S = new char[s.len + 1];

    //Sətrin ilk hissəsinin qiymətləndirilməsi
    strcpy(s.S, S);

    //Sətrin ikinci hissəsinin
    //qiymətləndirilməsi
    strcat(s.S, str.S);

    //Yeni obyektin qaytarılması
    return s;
}
//Təhlükəsiz mənimsətməni reallaşdıran funksiya
string_ & string_::operator=(const string_ &str)
{
    //STRING = STRING; (özünə
    //mənimsətmə)variantının aradan qaldırılması
    //Burada STRING sinfin dəyişənidir
    if(this == &str)
        return *this;
    //əgər sətirlərin ölçüləri uyğun
    //gəlmirlərsə və ya yazma baş verən sətir
    //formalaşmayıbsa
    if(len != str.len || len == 0)
    {
        //Köhnə sətirin silinməsi
        delete [] S;

```

```

        //sətrin yeni uzunluğunun hesablanması
        len = str.len;
        //Yeni sətir üçün yaddaşın ayrılması
        S = new char[len + 1];
    }

    //sətirin qiymətləndirilməsi
    strcpy(S, str.S);

    //Özünə istinadın qaytarılması
    //Bu imkana görə obyektləri bir-
    //birinə dəfələrlə mənimsətmək olar
    //məsələn, string_ a, b, c; a = b = c;
    return *this;
}

void main()
{
    int n,i;

    //Sətirlərin sayını daxil edirik
    cout << "Input the number of string
    s:\t"; cin >> n;
    if(n < 0)
    {
        cout << "Error number:\t" << n <<
        endl; return;
    }
    //Axından Enter ("\n")simvolunu alırıq
    char c[2];
    cin.getline(c, 2);

    //n sətirlik massiv yaradıırıq
    string_ *s = new string_[n];

    //Sətirlərin klaviaturadan daxil edilməsi

    for(i = 0; i < n; i++)
        s[i].SetStr();

```

Ev tapşırığı

1. Tarix (gün, ay, il) haqqında məlumatlar ehtiva edən Date sinfini yaradın. Operatorlara yükləmə mexanizmindən istifadə edərək iki tarixin fərqi (tarixlər arasındakı günlərin sayını) və tarixi müəyyən sayda gün artırmağı təyin edin.
2. Sətir sinfinə iki sətirin kəsişməsini, yəni, hər iki sətir üçün eyni simvolları ehtiva edən sətiri yaradan funksiya əlavə edin. Məsələn, "sdqcg" və "rgfas34" sətirlərinin kəsişməsinin nəticəsi "sg"dir. Funksiyanı reallaşdırmaq üçün operator * (binar vurma) yükləyin.

```
//Sətirlərin çeşidlənməsi
//Göstərici vasitəsilə
//çağırma, belə ki, funksiya
//yalnız bir deyil, bir qrup
//obyektlər üçün icra edilir.
s->Sort(s, n);
//Çeşidlənmiş sətirlərin qaytarılması
for(i = 0; i < n; i++)
    cout<<"\n"<<s[i].GetStr()<<"\n";

//Sətirlər massivinin ləğv edilməsi
delete [] s;

cout<<"\n\n+++++\n\n";
//operator + -u və çevrilməni yoxlayırıq

string_ A,B,C,RES;

A="Ivanov ";
B="Ivan ";
C="Ivanovich";
RES=A+B+C;
cout<<RES.GetStr()<<"\n\n";

}
```