ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Класи

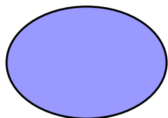
Објектно-ориентирано програмирање

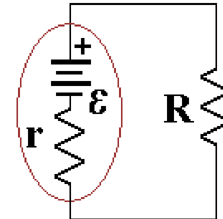
Објектно-ориентирано програмирање

- Ние го набљудуваме светот како свет од објекти. Погледнете наоколу и ќе забележите столица, маса, ..., и објекти поврзани меѓусебно на овој или оној начин.
- Објектно-ориентираното програмирање се обидува да го имитира светот, притоа дизајнирајќи објекти што меѓусебно кооперираат за да решат дадена задача.
- Објектно-ориентираното програмирање се состои во определување на објектите и нивните задолженија при решавање на даден проблем.

Што е објект?

- Работи што може да се допрат – кола, печатар, ...
- Улоги – вработен, шеф, ...
- Настани – лет, преполнување, ...
- Интеракција – потпишување договор, продажба, ...
- Спецификации – боја, изглед, ...
- Објекти, единици, релации, апстрактни или реални термини, со добро дефинирана улога во доменот на проблемот.





Што е објект?

- Секој објект се опишува со два дела

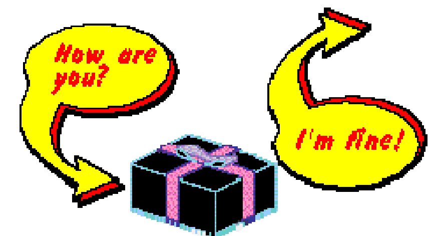
  Објект = Податоци + Методи

  Секој објект има одговорност да знае и одговорност да работи.

=

+

- Многу важен аспект на објектот е неговото однесување (или работите што тој може да ги изврши).
- Однесувањето се иницира со испраќање на порака до објектот (со повикување на метод дефиниран за самиот објект).



Објекти и класи

Објектите работат заедно и решаваат проблеми.

Објектите имаат состојба, дефинирана со вредности на атрибутите. Нивната состојба се менува кога ќе се сменат овие вредности.

Објектите комуницираат меѓусебно со испраќање на пораки. Испраќањето порака значи дека од објектот се бара да изврши една од неговите операции.

Со набљудување на објектите може да се забележи дека некои објекти имаат исти карактеристики (сите лица имаат име, ист број на хромозоми, ...) така да може да се дефинира една генерална – класа на објекти.

Класа дефинира множество на објекти што делат исти карактеристики, однесување и релации.

Креирана кориснички дефинирана структура Vreme со три члена cas, minuta, sekunda.

```
#include <iostream>
using std::cout;
using std::endl;
struct Vreme { int cas; int min; int sek; };
void printMilitary( const Vreme & );  // prototip
void printStandard( const Vreme & );  // prototip
int main(){
   Vreme vecera;
   vecera.cas = 18; vecera.min = 30; vecera.sek = 0;
   cout << “Vecerata ke bide vo ";
   printMilitary( vecera );
   cout << " voeno vreme,\nkoe e ";
   printStandard( vecera );
   cout << " standardno vreme.\n";
   vecera.cas = 29; vecera.min = 73;
   cout << "\nVreme so pogresni vrednosti: ";
   printMilitary( vecera );
   cout << endl;
   return 0;
   }
void printMilitary( const Vreme &t ){
cout << ( t.cas < 10 ? "0" : "" ) << t.cas << ":“ << ( t.min < 10 ? "0" : "" ) << t.min;
}
void printStandard( const Vreme &t ){
   cout << ((t.cas == 0 || t.cas == 12) ? 12 : t.cas % 12 ) << ":" << (t.min < 10 ? "0" : "" ) << t.min << ":"
          << (t.sek < 10 ? "0" : "") << t.sek << (t.cas < 12 ? " AM" : " PM" );
}
```

Се може и со структури

```
Vecerata ke bide vo 18:30 voeno vreme,
koe e 6:30:00 PM standardno vreme.
Vreme so pogresni vrednosti: 29:73
```

Се може и со структури во кои се дефинирани функции

```
#include <iostream>
using std::cout;
using std::endl;
struct Vreme {
      int cas; int min; int sek;
     void printMilitary( ),  printStandard( );
 };

int main(){
  Vreme vecera;
  vecera.cas = 18; vecera.min = 30; vecera.sek = 0;
  cout << "Vecerata ke bide vo ";
  vecera.printMilitary( );
  cout << " voeno vreme,\nkoe e ";
  vecera.printStandard( );
  cout << " standardno vreme.\n";
  vecera.cas = 29; vecera.min = 73;
  cout << "\nVreme so pogresni vrednosti: ";
  vecera.printMilitary( );
  cout << endl;
  return 0;
}
void Vreme::printMilitary( ){
  cout << ( cas < 10 ? "0" : "" ) << cas << ":" << ( min < 10 ? "0" : "" ) << min;
}
void Vreme::printStandard( ){
  cout << ((cas == 0 || cas == 12) ? 12 : cas % 12 ) << ":" << (min < 10 ? "0" : "" ) << min << ":"
<< (sek < 10 ? "0" : "") << sek << (cas < 12 ? " AM" : " PM" );
}
```

Класи во C++

- Класите овозможуваат моделирање на објекти за една C++ програма, моделирајќи ги нивните:
  - атрибути (репрезентирани со податочни членови).
  - однесување или операции (репрезентирани со методи - функции).
- Дефиницијата на една класа започнува со клучниот збор `class`.
- Телото на класата е опишано меѓу пар од големи загради, `{};`.
- Во телото на класата, клучните зборови `private:` и `public:` определуваат кој има пристап до членовите на класата.
- Иницијално се претпоставува дека сите членови на класата се од типот `private`.
- Вообичаено, податочните членови на класата се декларираат во секцијата `private:` на класата и методите во секцијата `public:`.
- Членовите означени како приватни се недостапни надвор од класата, односно информацијата од оваа секција е затворена за сите надворешни ,,клиенти” на класата.

Дефинирање на класата време

```
class Vreme {
public:
    Vreme();
    void setVreme(int, int, int);
    void pokaziVreme();
    void pokaziVoenoVreme();
private:
    int cas; //0-23
    int min; //0-59
    int sek; //0-59
};
```

public: и **private:** го одредуваат пристапот до членовите на класата

setVreme, pokaziVreme, и **pokaziVoenoVreme** се методи на класата или функции членови на класата. **Vreme** е конструктор.

cas, **min**, и **sek** се податочни членови

Не е можно податочните членови во класата да бидат иницијализирани при нивната декларација во класата.

Класата не може да содржи член класа од истиот тип, но може да содржи покажувач кон објект од истата класа.

Креирање на објекти

Името на класата станува нов идентификатор (податочен тип).

```
Vreme sunset,                    // објект од класата Vreme
      arrayOfTimes[ 5 ],         // поле од објекти од класата Vreme
      *pointerToTime,            // покажувач кон објект од кл. Vreme
      &dinnerTime = sunset;      // референца на објект од кл. Vreme
```

Имплементација на методи

- Во декларацијата на класата најчесто се наведуваат само прототиповите на функциите членови.
- Методите за секоја класа се дефинираат на ист начин како и која и да е функција во Ц, но?
- Дефиницијата на метод член од една класа се врзува со истата со користење на scope операторот (::) за да се означи на која класа припаѓа
  - различни класи може да имаат методи со исто име
- Формат

```
TipVrednost ImeKlasa::ImeClenFunkcija(listaParametri){
    naredbi;
  }
```

Имплементација на методи

- Ако методот се дефинира во рамките на класата scope операторот и името на класата не се неопходни во дефиницијата

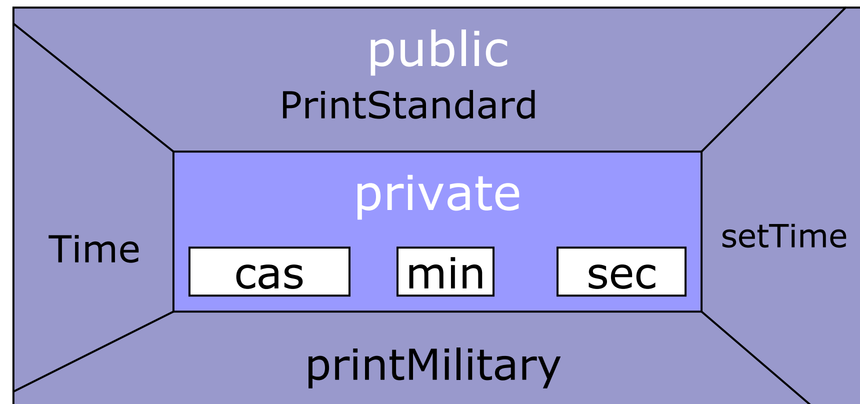
- Дефинирањето на класните методи надвор од класата немаат врска со тоа дали методот е дефиниран како **`public`** или **`private`**

Конструктори

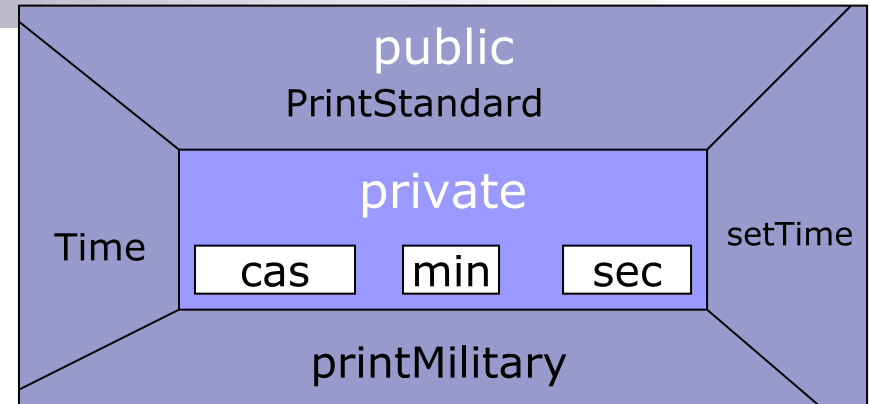
- Во примерот – Vreme()
- Специјална функција член што ги иницијализира членовите на класата
- Конструкторите не враќаат вредност
- Имаат исто име како и класата

Дефинирање на класа и имплементација на методи

```
#include <iostream>
using std::cout;
using std::endl;
class Time {
 public:
   Time();
   void setTime( int, int, int );
   void printMilitary();
   void printStandard();
private:
   int cas;    // 0 – 23
   int min;    // 0 – 59
   int sec;    // 0 – 59
};
Time::Time() { cas = min = sec = 0; }
```



Дефинирање на класа и имплементација на методи




```
void Time::setTime( int h, int m, int s ){
 cas = ( h >= 0 && h < 24 ) ? h : 0;
 min = ( m >= 0 && m < 60 ) ? m : 0;
 sec = ( s >= 0 && s < 60 ) ? s : 0;
}
void Time::printMilitary(){
 cout << ( cas < 10 ? "0" : "" ) << cas << ":"  << ( min < 10 ? "0"
: "" ) << min << endl;
}
void Time::printStandard(){
        cout  << ((cas==0 || cas==12)?12:cas%12)
                << ":" << (min< 10 ? "0" : "" ) << min
                << ":" << ( sec < 10 ? "0" : "" ) << sec
                << ( cas < 12 ? " AM" : " PM" ) << endl;
}
```

Креирање на објекти, праќање порака на објект

```
int main(){
 Time t;
 cout << "The initial military time is ";  t.printMilitary();
 cout << "\nThe initial standard time is ";  t.printStandard();
 t.setTime( 13, 27, 6 );
 cout <<"\n\nMilitary time after setTime is ";
   t.printMilitary();
 cout << "\nStandard time after setTime is ";
 t.printStandard();
 t.setTime( 99, 99, 99 );
 cout  << "\n\nAfter attempting invalid settings:"  <<
   "\nMilitary time: ";
 t.printMilitary();
 cout << "\nStandard time: ";
 t.printStandard();
 cout << endl;
 return 0;
}
```

```
The initial military time is 00:00
The initial standard time is 12:00:00 AM

Military time after setTime is 13:27
Standard time after setTime is 1:27:06 PM

After attempting invalid settings:
Military time: 00:00
Standard time: 12:00:00 AM
```

Иницијализација на податочните членови - конструктори

- Елементите на класата **Time**, не може да се иницијализираат на следниот начин

```
Time t;
t.cas=0;
t.min=0;
t.sec=0;
```

- Функцијата конструктор е член на класата и има исто име како и класата
- Конструкторот не враќа вредност, за истиот не се определува типот на вредноста што ја враќа
  - пример

```
Time::Time() { cas = min = sec = 0; }
```

Default конструктор

- Еден за секоја класа, ако програмерот не дефинирал друг конструктор, во C++ се креира конструктор што би бил дефиниран вака

```
Time::Time(){}
```

- Може да се повика без аргументи
- Во истиот не се дефинираат аргументи со подразбирливи вредности
- Ако програмерот обезбеди барем еден конструктор, тогаш истиот мора да го обезбеди default конструкторот

```
#include <iostream>
using std::cout;
using std::endl;
class Time {
public:
  Time( int = 0, int = 0, int = 0 );
  void setTime( int, int, int );
  void printMilitary();
  void printStandard();
private:
 int cas;    // 0 – 23
 int min;    // 0 – 59
 int sec;    // 0 – 59
};
Time::Time(int c, int m, int s) {     cas = c; min = m; sec = s; }
void Time::setTime( int h, int m, int s ){
 cas = ( h >= 0 && h < 24 ) ? h : 0;
 min = ( m >= 0 && m < 60 ) ? m : 0;
 sec = ( s >= 0 && s < 60 ) ? s : 0;
}
void Time::printMilitary() {
 cout << '\t' << ( cas < 10 ? "0" : "" ) << cas << ":"  << ( min < 10 ? "0" :
    "" ) << min << endl;
}
void Time::printStandard(){
 cout << '\t' << ((cas==0 || cas==12)?12:cas%12)
 << ":" << ( min < 10 ? "0" : "" ) << min << ":"
 << ( sec < 10 ? "0" : "" ) << sec  << ( cas < 12 ? " AM" : " PM" ) << endl;
}
```

```
int main(){
 Time t1, t2(2), t3(21, 34), t4(12, 25, 42),
          t5(27, 74, 99);
 cout << "Povikan konstruktor\n";
 cout << "Bez argumenti\n"; t1.printMilitary();  t1.printStandard();
 cout << "So eden argument\n"; t2.printMilitary();
   t2.printStandard();
 cout << "So dva argumenti\n"; t3.printMilitary();
   t3.printStandard();
 cout << "So tri argimenti\n"; t4.printMilitary();
   t4.printStandard();
 cout << "so pogresni vrednosti\n";
   t5.printMilitary(); t5.printStandard();
 cout << endl;
 return 0;
}
```

```
Povikan konstruktor
bez argumenti
   00:00
   12:00:00 AM
so eden argument
   02:00
   2:00:00 AM
so dva argumenti
   21:34
   9:34:00 PM
so tri argumenti
   12:25
   12:25:42 PM
so pogresni vrednosti
   27:74
   3:74:99 PM
```

```
Time t1, t2(2), t3(21, 34), t4(12, 25, 42),
t5(27, 74, 99);
```

се креираат пет објекти, за сите се резервира посебен простор за променливите членови, но сите ги користат истите функции

t1	t2	t3	t4	t5
cas=0	cas=2	cas=21	cas=12	cas=27
min=0	min=0	min=34	min=25	min=74
sec=0	sec=0	sec=0	sec=42	sec=99

податоци

```
void Time::printMilitary(){
 cout << ( cas < 10 ? "0" : "" ) << cas << ":"
 << ( min < 10 ? "0" : "" ) << min;
}
```

методи

```
t1.printMilitary();
```

```
t5.printMilitary();
```

Деструктори

- Методи членови на една класа
- Го извршуваат крајното чистење пред да се врати меморијата резервирана за објектот на системот
- Претставуваат комплемент на конструкторот
- Името се формира со користење на операторот *tilde* (~) на кое следи името на класата
  - **~Vreme**
- Не содржи формални параметри и не враќа вредности
- Секогаш се дефинира еден деструктор по класа – не е дозволено преоптоварување на истиот
  - пример

```
class Time {
public:
  Time( int = 0, int = 0, int = 0 );
  ~Time();
  void setTime( int, int, int );
  void printMilitary();
  void printStandard();
private:
  int cas;    // 0 - 23
  int min;    // 0 - 59
  int sec;    // 0 - 59
};
```

- Се извршува автоматски при уништувањето на објектот

Како се имплементираат класите

- Вообичаено една класа се дефинира во две датотеки:
  - .h: header/interface/declaration
  - .cpp: implementation
- Header датотеката ја содржи дефиницијата на класата

```
using namespace std;
class Numbers                // Class definition
{
   public:                   // Can be accessed by a "client".
      Numbers ( );           // Class "constructor"
      void display ( );
      void update ( );
   private:                  // Cannot be accessed by "client"
      char name[30];
      int a;
      float b;
} ;
```

Како се имплементираат класите

- .cpp ги содржи имплементациите на функциите

```
#include <iostream>
#include <cstring>      // This is the same as string.h in C
using namespace std;
#include "number.h"
Numbers::Numbers ( ) {  // Constructor member function
   strcpy (name, "Unknown") ;  a = 0;   b = 0.0;
}
void Numbers::display ( ) {     // Member function
   cout  <<  "\nThe name is "  << name  <<  "\n" ;
   cout  <<  "The numbers are "  <<  a  <<  " and "  << b <<
   endl ;
}
void Numbers::update ( )       {      // Member function
   cout  <<  "Enter name"  <<  endl ;
   cin.getline (name, 30) ;
   cout  <<  "Enter a and b"  <<  endl ;
   cin  >>  a  >>  b;
}
```

Датотека што ја користи класата

- koristi.cpp

```
#include <iostream>
#include <cstring>       // This is the same as string.h in C
using namespace std;
#include "number.h"
int main ( ){            // Main program
   Numbers no1, no2 ;    // Create two objects of the class
                         // "Numbers"
   no1.update ( ) ;      // Update the values of
                         // the data members
   no1.display ( ) ;     // Display the current
   no2.display ( ) ;     // values of the objects
}
```

Преоптоварување на конструктори

```
#include <iostream>
#include <cmath>
using namespace std;
const float PI = 3.14159;

class Sphere {          // Klasa sfera
public:
    float r;            // radius
    float x, y, z;      // centar
    Sphere() { /* Ne pravi nisto */ }
    Sphere(float xcoord, float ycoord, float zcoord, float radius)
            { x = xcoord; y = ycoord; z = zcoord; r = radius; }
    ~Sphere()
    {cout << "Sphere (" << x << ", " << y << ", " << z << ", " << r << ") destroyed\n";}
    float volume() { return (r * r * r * 4 * PI / 3); }
    float surface_area() { return (r * r * 4 * PI); }
};

int main() {
Sphere s(1.0, 2.0, 3.0, 4.0);
Sphere t; // bez parametri

cout << "X = " << s.x << ", Y = " << s.y << ", Z = " << s.z << ", R = " << s.r << "\n";
t = s;
cout << "The volume of t is " << t.volume() << "\n";
cout << "The surface area of t is " << t.surface_area() << "\n";
return 0;
}
```

```
X = 1, Y = 2, Z = 3, R = 4
The volume of t is 268.082
The surface area of t is 201.062
Sphere (1, 2, 3, 4) destroyed
Sphere (1, 2, 3, 4) destroyed
```

Конструктори со еден аргумент

```
#include <iostream>
using namespace std;
class X {
        int a;
   public:
        X(int j) { a = j; }
        int geta() { return a; }
   };
int main() {
   X ob = 99;              // пренесува 99 за j
   cout << ob.geta();      // прикажува 99
   return 0;
   }
```

- Во овој пример, бројот 99 автоматски се пренесува во формалниот параметар **j** на конструкторот **X()**. Оваа наредба компајлерот ја третира како да е напишана на следниот начин:

  X ob = X(99);

- Генерално, секогаш кога е потребен само еден аргумент може да се користат следните формати за проследување на вредноста до конструкторот

  ob(*i*)

  ob = *i*

Кога се извршуваат конструкторите и деструкторите

```
#include <iostream>
using namespace std;
class myclass {
   public:
        int who;
        myclass(int id);
        ~myclass();
} glob_ob1(1), glob_ob2(2);

myclass::myclass(int id) {
   cout << "Inicijalizacija " << id << "\n";
   who = id;
}

myclass::~myclass(){
   cout << "Unistuvanje " << who << "\n";
}

int main(){
   myclass local_ob1(3);
   cout << "Ova ne e prvata poraka na ekran.\n";
   myclass local_ob2(4);
   return 0;
}
```

Излез од програмата:

```
Inicijalizacija 1
Inicijalizacija 2
Inicijalizacija 3
Ova ne e prvata poraka na ekran.
Inicijalizacija 4
Unistuvanje 4
Unistuvanje 3
Unistuvanje 2
Unistuvanje 1
```

Редослед на извршување на конструктори и деструктори

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

class Test
{
public:
   // konstruktor so eden argument
  Test(const char *name);
  ~Test() { cout << "Unisten Test objekt " << ime << endl;}
private:
  char ime[20];
};
Test::Test(const char *name) {
  cout << "Kreiran Test objekt " << name << endl;
  strncpy(ime,name,19); ime[19]=0;
}
Test globaltest("global");
void func() {
  static Test statictest("static");
  Test functest("func");
}
```

```
int main() {
  Test first("main first");
  func();
  {
    Test first("main inner");
  }
  Test second("main second");
  return 0;
}
```

```
Kreiran Test objekt global
Kreiran Test objekt main first
Kreiran Test objekt static
Kreiran Test objekt func
Unisten Test objekt func
Kreiran Test objekt main inner
Unisten Test objekt main inner
Kreiran Test objekt main second
Unisten Test objekt main second
Unisten Test objekt main first
Unisten Test objekt static
Unisten Test objekt global
```

Објекти како аргументи

```
#include <iostream>
using namespace std;
class myclass {
   int i;
public:
   myclass(int n);
   ~myclass();
   void set_i(int n) { i=n; }
   int get_i() { return i; }
};
myclass::myclass(int n) { i = n; cout << "Constructing " << i <<
   "\n"; }
myclass::~myclass() { cout << "Destroying " << i << "\n"; }
void f(myclass ob);
int main() {
   myclass o(1);
   f(o);
   cout << "This is i in main: ";
   cout << o.get_i() << "\n";
   return 0;
}
void f(myclass ob) {
   ob.set_i(2); cout << "This is local i: " << ob.get_i() << "\n";
}
```

Излез од програмата:
Constructing 1
This is local i: 2
Destroying 2
This is i in main: 1
Destroying 1

Објекти како аргументи

- Декструкторот се повикува два пати, но само еднаш е повикан конструкторот.
- Како што е илустрирано во излезот од програмата, конструкторот не се повикува кога копија на **o** (во **main()**) се пренесува во **ob** (во повикот на **f()**).
- Кога се пренесува објект во функцијата, тогаш се пренесува негова копија (се креира нов објект кој е копија на постоечкиот), но ова не го прави конструкторот!
- Значи кога објект се пренесува во функцијата, не се повикува конструкоторот, меѓутоа мора да се повика деструкторот за да се уништи копијата на објектот во функцијата (оваа копија е локална за функцијата и неопходно е истата да биде уништена при напуштање на функцијата).

Copy конструктор

се извршува пред извршувањето на наредбите од телото на конструкторот

```
class Rational {
    public:
        Rational(int n=0, int d=1) : num(n), den(d) { reduce(); }
        Rational(const Rational& r) : num(r.num), den(r.den) { }
        void print() { cout << num << '/' << den; }
    private:
        int num, den;
        int gcd(int m, int n) { if (n==0) return m; return gcd(n,m%n); }
        void reduce() { int g = gcd(num, den); num /= g; den /= g; }
} ;
main(){
    Rational x(100,360);
    Rational y(x); //Rational y=x;
    cout << "X = "; x.print(); tout << ", y = "; y.print();
}
```

Излез од програмата:
x=5/18, y =5/18

Copy конструкторот мора да има еден параметар објект од истата класа објекти што се декларира, и кој мора да биде дефиниран како константна референца: const &XL

copy конструктор (2)

```
class Rational {
   public:
        Rational(int n, int d) : num(n), den(d) { }
        Rational(const Rational& r) : num(r.num), den(r.den)
                { cout << "COPY CONSTRUCTOR CALLED\n"; }
private:
        int num, den;
} ;

Rational f(Rational r) {   // се повикува copy конструктор за копирање на ? во r
   Rational s = r;         // се повикува copy конструктор за копирање на r во s
   return s;               // се повикува copy конструктор за копирање на s во ?
}
main() {
   Rational x(22,7);
   Rational y(x);          // се повикува copy конструктор за копирање на x во y
   f(y) ;
}
```

```
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
COPY CONSTRUCTOR CALLED
```

copy конструкторот се повикува автоматски секогаш
- при иницијализација на податочните членови на еден објект со вредностите на податочните членови на друг објект;
- кога се проследува објект како вредност во функција;
- кога функција како вредност враќа објект.

Објекти како аргументи (б)

```
class myclass {
    int i;
public:
    myclass(int n);
    myclass(const myclass &n);
    ~myclass();
    void set_i(int n) { i=n; }
    int get_i() { return i; }
};
myclass::myclass(const myclass &n) {
    i = n.i;
    cout << "Constructing+copying " << i << "\n";
}
. . .
void f(myclass ob);
int main() {
    myclass o(1);
    f(o);
    cout << "This is i in main: ";
    cout << o.get_i() << "\n";
    return 0;
}
```

```
void f(myclass ob) {
        ob.set_i(2);
        cout << "This is local i: " << ob.get_i() << "\n";
}
```

се повикува деструктор за сите објекти од блокот (на крајот на f)

се повикува конструкторот (myclass o(1);)

се повикува COPY конструкторот (f(o);)

се повикува деструктор за сите објекти од блокот (на крајот на main)

Излез од програмата:

```
Constructing 1
Constructing+copying 1
This is local i: 2
Destroying 2
This is i in main: 1
Destroying 1
```

Функции што враќаат објекти

```
#include <iostream>
using namespace std;
class myclass {
    int i;
public:
    void set_i(int n) { i=n; }
    int get_i() { return i; }
};

// функцијата враќа објект од класата myclass
myclass f();

int main() {
    myclass o;
    o = f();
    cout << o.get_i() << "\n";
    return 0;
}

myclass f() {
    myclass x;
    x.set_i(1);
    return x;
}
```

Кога една функција враќа објект, тогаш се креира привремен објект автоматски за да се смести вредноста што се враќа (се враќа објект).
Овој објект се уништува по враќање на вредноста.

Придружување на објекти

```
#include <iostream>
using namespace std;
class myclass {
   int i;
public:
   void set_i(int n) { i=n; }
   int get_i() { return i; }
};

int main() {
   myclass ob1, ob2;
   ob1.set_i(99);
   ob2 = ob1; //
   cout << "This is ob2's i: " << ob2.get_i();
   return 0;
}
```

Вгнездување на објекти - концепт

- Вгнездувањето на објекти овозможува криење на имиња и локално резервирање на ресурси.

```
char c;                 // external scope ::c
class X {               // outer class declaration X::
    public:
        char c;         // X::c
        class Y {       // inner class declaration X::Y::
                        public:
                                void foo(char e) { X t; ::c = t.c = c = e; }
                        private:
                                char c; // X::Y::c
                        };
    };
```

- Во класата Y, методот foo(), поради ::c, се однесува на глобалната променлива c;
- Во класата Y во истиот метод користењето на X::c, овозможува пристап до класната променлива

```
void foo() {
        class local { ····· } x;
}
local y; // неточно:local важи само во функцијата foo()
```

Објекти во објекти – композиција на објекти, пример

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

class Charr {
   enum {CharrSize=40};
   private:
      char arr[CharrSize+1];
   public:
      Charr(const char *); // Charr constructor
      ~Charr();                    // Charr destructor
      void Show(void) const { cout << arr; }
};


Charr::Charr(const char * a) {
   cout << "\t\t*Constructing Charr: " << a << endl;
   strncpy(arr, a, CharrSize);
   arr[CharrSize] = '\0';
}

Charr::~Charr() {
   cout << "\t\t#Destructing Charr: " << arr << endl;
}
```

Објекти во објекти – композиција на објекти, пример

```
class Person {
   private:
      Charr ime;
      Charr prezime;
      int level;
   public:
      Person(int lv, char *name, char *sname);
      ~Person();
      void Show(void) const;
};

Person::Person(int lv, char *name, char *sname):ime(name),prezime(sname) {
   cout << "\t\t*Constructing Person: " << name << ' ' << sname << ' ` << lv <<
endl;
   level=lv;
}

Person::~Person() {
   cout << "\t\t#Destructing Person: ";
   ime.Show(); cout << ' '; prezime.Show(); cout << ' ';
   cout << level << endl;
}
void Person::Show(void) const {
   ime.Show(); cout << ' '; prezime.Show(); cout << ' ';
   cout << level;
}
```

Објекти во објекти – композиција на објекти, пример

```
int main()
{
  Charr a("Hello");
  const Charr b("World");
// Charr c; << ke javi greska
  Person y(1,"Al","Capone");

  a.Show();
  cout << endl;
  {
    Person x(1,"Tony","Luchiano");
    x.Show(); cout << endl;
    y.Show(); cout << endl;
  }
  b.Show();
  cout << endl;
}
```

```
                *Constructing Charr: Hello
                *Constructing Charr: World
                *Constructing Charr: Al
                *Constructing Charr: Capone
                *Constructing Person: Al Capone 1
Hello
                *Constructing Charr: Tony
                *Constructing Charr: Luchiano
                *Constructing Person: Tony Luchiano 1
Tony Luchiano 1
Al Capone 1
                #Destructing Person: Tony Luchiano 1
                #Destructing Charr: Luchiano
                #Destructing Charr: Tony
World
                #Destructing Person: Al Capone 1
                #Destructing Charr: Capone
                #Destructing Charr: Al
                #Destructing Charr: World
                #Destructing Charr: Hello
```

Константни објекти - а

```
#include <iostream>
#include <cstring>
using namespace std;
class Person {
enum {NameLen=20, AddrLen=30,PhoneLen=10};
public: // interface functions
  Person(const char *n = "", const char *a="-nepoznata-", const char *p="-nepoznato-");
  void setName(const char *n);
  void setAddress(const char *a);
  void setPhone(const char *p);
  const char *getName();
  const char *getAddress();
  const char *getPhone();
  void Show();
private:
  char name[NameLen+1]; // name of person
  char addr[AddrLen+1]; // address field
  char phone[PhoneLen+1]; // telephone number
};

Person::Person(const char *n, const char *a, const char *p) {
  strncpy(name,n,NameLen); name[NameLen]=0;
  strncpy(addr,a,AddrLen); addr[AddrLen]=0;
  strncpy(phone,n,PhoneLen); phone[PhoneLen]=0;
}

void Person::setName(const char *n) {  strncpy(name,n,NameLen); name[NameLen]=0;}
void Person::setAddress(const char *a) { strncpy(addr,a,AddrLen); addr[AddrLen]=0;}
void Person::setPhone(const char *p) {  strncpy(phone,p,PhoneLen); phone[PhoneLen]=0;}
```

Константни објекти - 6

```
const char *Person::getName() {    return(name);     }
const char *Person::getAddress() {         return(addr);     }
const char *Person::getPhone() {   return(phone);    }

void Person::Show() {cout << name << " " << addr << " " <<    phone << endl;}

int main() {
  Person P1("Nekoj","somebody@network.net");
  const Person P2("Drug", "other@else.com", "5551234");
  Person P3;
  char *p;
  cout << P1.getName() << endl; //OK
  P1.setName("Ovoj"); //OK
  P1.Show(); //OK

  P2.Show(); //Ne moze!!!
  cout << P2.getName() << endl; //Ne moze!!
  P3=P2;
  p=P2.getPhone(); //Ne Moze!!!
}
```

Кај константните објекти, C++ компајлерот го ограничува пристапот до методите на објектот. Единствени методи што може да бидат повикани кај овие објекти се конструкторите и деструкторите. За да се надмине ова ограничување, мора секој метод што сакаме да го користиме кај константните објекти да се декларира како const.

const Func() **const** { /* do something */ }

Константни објекти (2-а)

```
#include <iostream>
#include <cstring>
using namespace std;
class Person {
enum {NameLen=20, AddrLen=30, PhoneLen=10};
public: // interface functions
  Person(const char *n = "", const char *a="-nepoznata-",  const char *p="-nepoznato-");
  void setName(const char *n);
  void setAddress(const char *a);
  void setPhone(const char *p);
  const char *getName() const;
  const char *getAddress() const;
  const char *getPhone() const;
  void Show() const;
private:
  char name[NameLen+1]; // name of person
  char addr[AddrLen+1]; // address field
  char phone[PhoneLen+1]; // telephone number
};

Person::Person(const char *n, const char *a, const char *p) {   /* isto so prethodno */ }

void Person::setName(const char *n) {   /* isto so prethodno */}
void Person::setAddress(const char *a) {   /* isto so prethodno */}
void Person::setPhone(const char *p) {   /* isto so prethodno */}
```

Константни објекти (2-б)

```
const char *Person::getName() const {  return(name);}
const char *Person::getAddress() const {  return(addr);}
const char *Person::getPhone() const {  return(phone);}

void Person::Show() const {
  cout << name << " " << addr << " " << phone << endl;
}

int main() {
  Person P1("Nekoj","somebody@network.net");
  const Person P2("Drug", "other@else.com", "5551234");
  Person P3;
  char *p;

  cout << P1.getName() << endl; //OK
  P1.setName("Ovoj"); //OK
  P1.Show(); //OK

  P2.Show(); //OK
  cout << P2.getName() << endl; //OK
  P3=P2;
  p=P2.getPhone(); //Ne moze!!!
}
```

mutable

Доколку треба да се овозможи промена на некој од податочните членови дури и кога објектот е константен од страна на константните функции членови, можно е декларирање на тој податочен член како **mutable.**

```
class Z
{
  int i;
  mutable int j;
public:
  Z() : i(0), j(0) {}
  void f() const;
  void show() const { cout << i << ‘ ‘ << j << endl; }
};

void Z::f() const {
//! i++; // Greska, f() e konstantna funkcija clen
    j++; // OK: j e mutable
}
void main()
{
  const Z zz;
  zz.show();
  zz.f();
  zz.show();
}
```

```
0 0
0 1
```

Пријателски функции и класи

Со прогласувањето на дадена функција или класа за пријателска и се дава право да пристапува до приватните членови на класата.

```
class F1 {
  ….
  friend float soberi(const F1 &, const F2 &);
  friend class F3;
};

class F2 {
   …
   friend float soberi(const F1 &, const F2 &);
   friend void F1::namali(const F2 &);
};
```

Пријателството не мора да биде обострано и не се наследува! Не е важно дали пријателството ќе се дефинира во private или public делот.

Пријателски функции и класи

```
class F2;
class F1
{
  int i;
  float f;
public:
  F1(int a=0, float b=0.0): i(a), f(b) {}
  void show() const { cout << i << ' ' << f << endl; }
  void namali(const F2 &x);
private:
  friend float soberi(const F1 &, const F2 &);
  friend class F3;
};

class F2
{
  int i;
  float a;
  char c;
public:
  F2(int a=0, float b=0.0, char cc=' '): i(a), a(b), c(cc) {}
  void show() const { cout << i << ' ' << a << " '"
       << c << "'" << endl; }
  friend float soberi(const F1 &, const F2 &);
  friend void F1::namali(const F2 &);
};
```

пријателска глобална функција

пријателска класа

пријателска функција членка на друга класа

```
class F3
{
  int i;
  float f;
public:
  F3(int a=0, float b=0.0): i(a), f(b) {}
  void show() const { cout << i << " " << f << endl; }
  int odzemi(const F1 &x) { f-=x.f; return(i-=x.i); }
};
void F1::namali(const F2 &x) {
  i-=x.i;
  f-=x.a;
}
float soberi(const F1 &x, const F2 &y) {
   return(x.f+y.a);
}

int main()
{
  F2 y(1,2,'y');
  F1 x(3,4),z;
  F3 v(5,4.5);
  x.namali(y); x.show();
  cout << soberi(x,y) << endl;
  v.odzemi(x); v.show();
}
```

може да пристапува до приватните членови на класата F1 бидејќи целата класа F1 е пријател на класата F3

може да пристапува до приватните членови на класата F2

пријателска е за обете класи и може да пристапува до приватните членови и на класата F1 и на класата F2

```
2  2
4
3  2. 5
```