Objektno Orijentisano Programiranje - C++

Prof. Dragan Janković

Реализатори

Проф. Драган Јанковић - канцеларија 320ж, e-mail: dragan.jankovic@elfak.ni.ac.rs

Доц. Петар Рајковић – канцеларија 534ц, e-mail: petar.rajkovic@elfak.ni.ac.rs

Мр Владан Михајловић - канцеларија 331ц, e-mail: vladan.mihajlovic@elfak.ni.ac.rs

Александар Миленковић – канцеларија 533ц, e-mail: aleksandar.milenkovic@elfak.ni.ac.rs

Александар Вељановски канцеларија 533ц, e-mail: aleksandar.veljanovski@elfak.ni.ac.rs

Анђелија Ђорђевић – канцеларија 533ц, e-mail: andjelija.djordjevic@elfak.ni.ac.rs

Osnovno o predmetu

- Obim: 2+2+1
- III semestar
- Način polaganja:

1.	Lab. Vežbe	: 0 – 20 (obavezne)
2.	I kolokvijum	: 0 – 20 (>49%)*
3.	II kolokvijum	: 0 – 20 (>49%)*
4.	Pismeni	: 0 – 40 (>49%)*
5.	Usmeni	: 0 – 40 (>49%)
6.	Obavezno prisustvo nastavi	: 0 – 5

1. Konačna ocena = (1+2+3+5+6)

2. Konačna ocena = (1+4+5+6)

- Poeni koji nisu ostvareni kroz lab. vežbe se ne mogu nadoknaditi !!!

*(Položeni pisani deo ispita preko kolokvijuma važi zaključno sa aprilskim ispitnim rokom. Pismeni ispit važi jedan rok)

Оцене

- 51 - 60 : 6
- 61 - 70 : 7
- 71 - 80 : 8
- 81 - 90 : 9
- 91 - 100 : 10

Циљ предмета

- Проучавање принципа објектно оријентисане парадигме програмирања.
- Проучавање програмског језика Ц++ као представника објектно оријентисаних језика.

Исход предмета

- Потпуно овладавање објектно оријентисаном парадигмом и програмским језиком Ц++
- По завршетку курса студенти ће усвојити ОО парадигму и бити способни да реализују програме у програмском језику Ц++

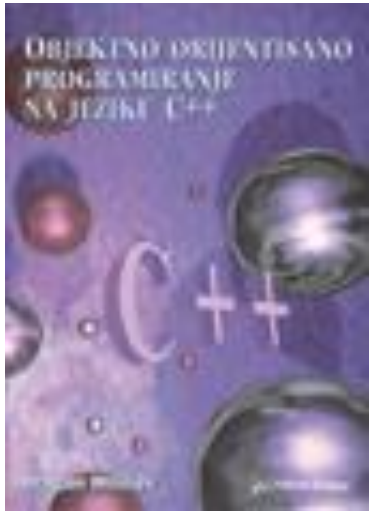
Теме

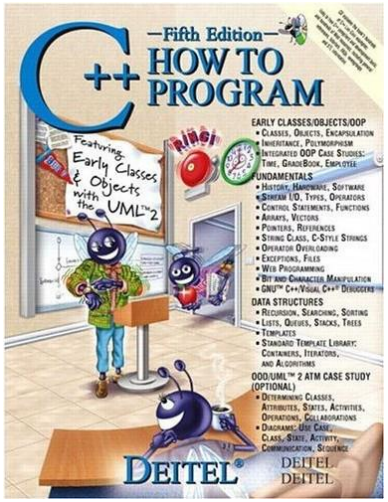
- Преглед техника програмирања. Концепт апстракције података. Модули као средство апстракције. Основни концепти ОО програмирања, класе, објекти. Преглед ОО језика. Елементи програмског језика Ц++. Статичка и динамича имплементација ОО језика. Наслеђивање. Генерализација и специјализација. Полиморфизам. Концепт апстрактних класа. Шаблонске функције и класе. Обрада изузетака. СТЛ. Развој ОО апликација. Вишеструко коришћење кода.

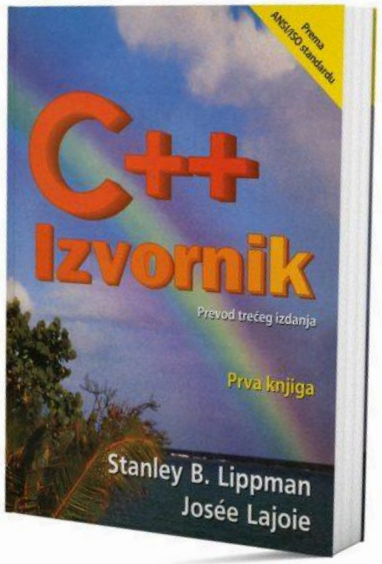
Литература

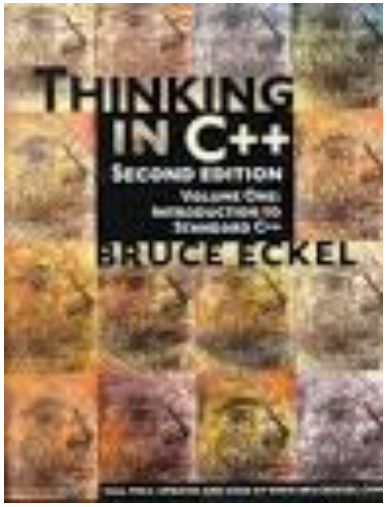
- М. Станковић, С. Стојковић, М. Радмановић и И. Петковић, *Објектно оријентисани језици Ц++ и Јава – са решеним задацима*, Електронски факултет у Нишу, Едиција Помоћни уџбеници, 2005.
- Ласло Краус, *Програмски језик Ц++ са решеним задацима*, Академска мисао, Београд, 2007.
- Драган Милићев, Објектно оријентисано програмирање на језику Ц++, 2010, Микро књига
- Paul Deitel, C++ how to program
- Електронски материјали на:
  - ЛМС-у Катедре за рачунарство

Literatura …









UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

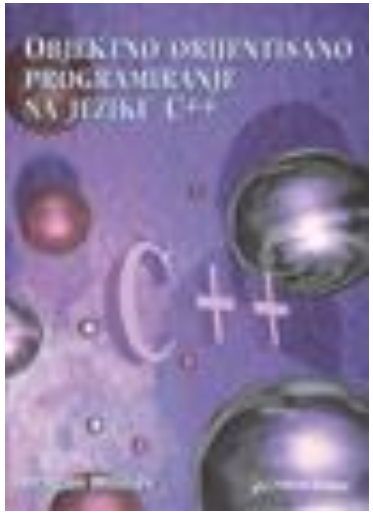
Dušan T. Malbaški

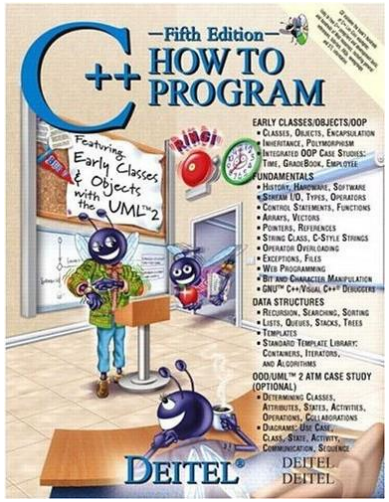
OBJEKTNO ORIJENTISANO PROGRAMIRANJE kroz programski jezik C++

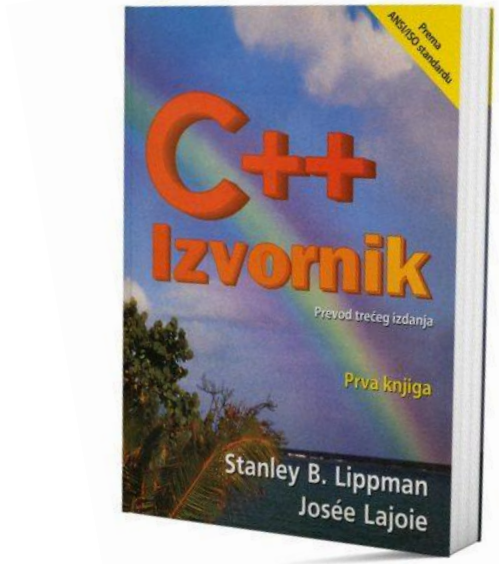
Novi Sad, 2008.

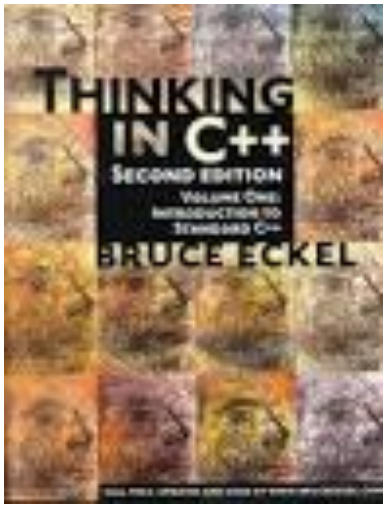
- Pitanja?
- Komentari?
- Predlozi?

Literatura









UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

Dušan T. Malbaški

OBJEKTNO ORIJENTISANO PROGRAMIRANJE kroz programski jezik C++

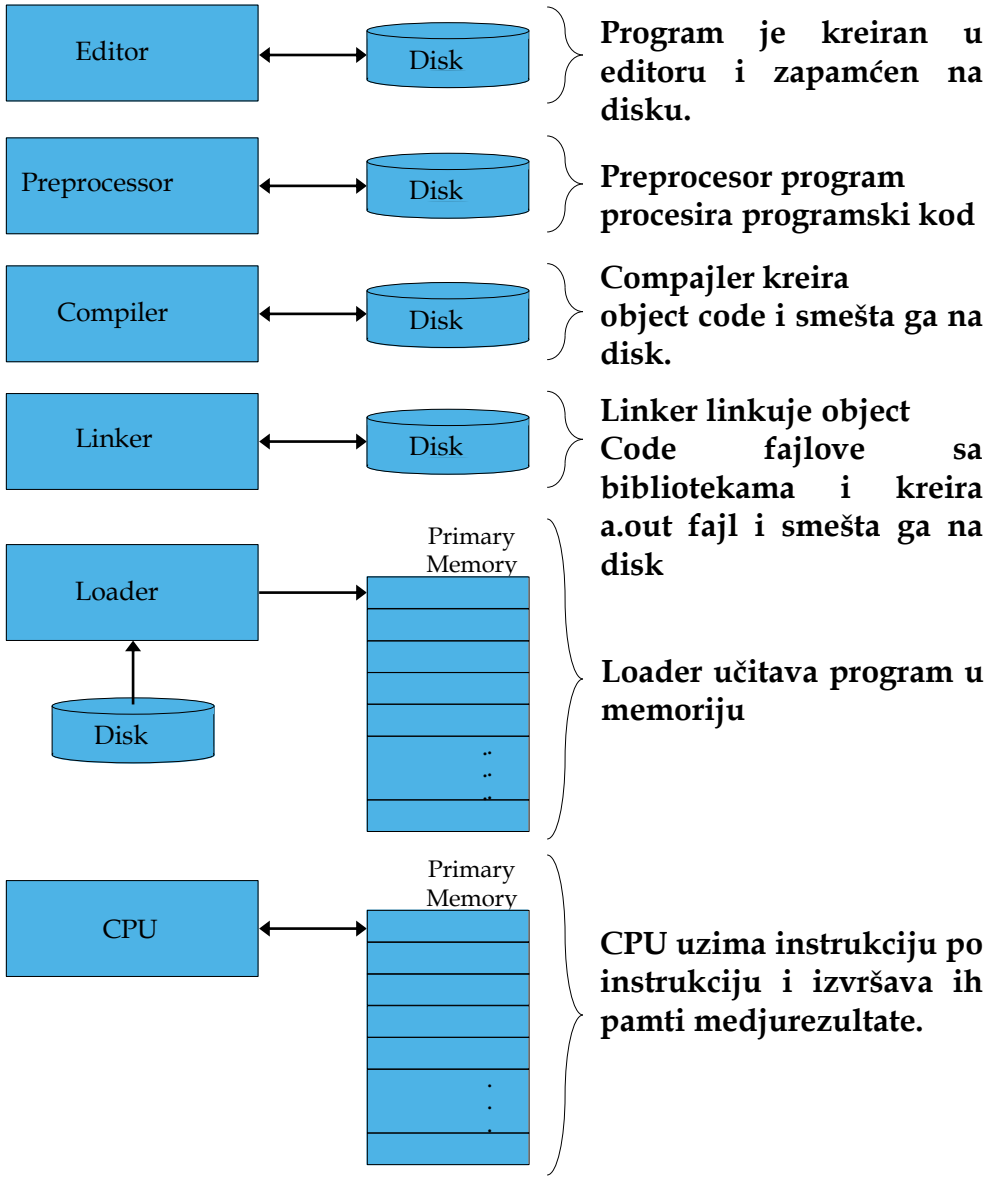
Novi Sad, 2008.

- Pitanja?
- Komentari?
- Predlozi?

Elementi standardnog C++ okruženja

Faze u razvoju C++ programa:

1. Editovanje
2. Preprocesiranje
3. Kompajliranje
4. Linkovanje
5. Učitavanje
6. Izvršavanje

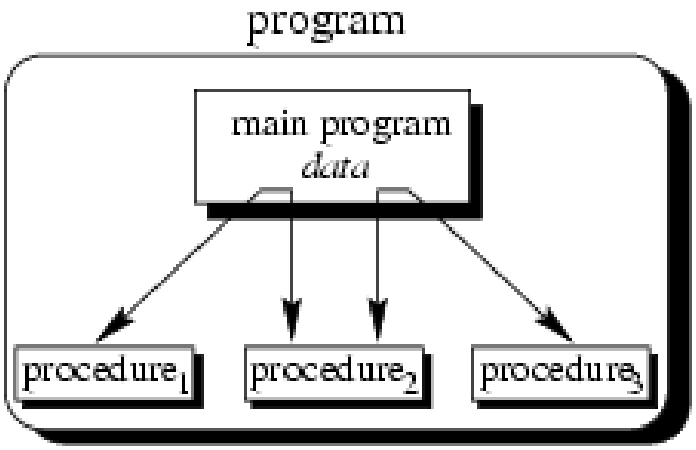


Pregled tehnika programiranja

- **Nestrukturno - proceduralno programiranje**
  - **Glavni program direktno operiše sa globalnim podacima.**
  - **Dugi i nepregledni programi**
  - **Copy paste- Kod se višestruko koristi kopiranjem delova**

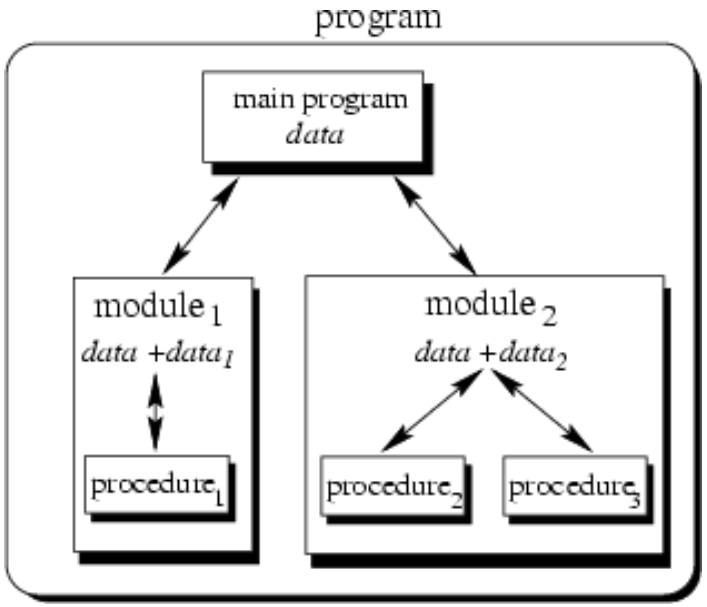
Pregled tehnika programiranja

- **Proceduralno programiranje**
  - **Program se može posmatrati kao sekvenca poziva potprograma (procedura).**
  - **Strukture podataka se modeliraju odvojeno od koda procedura koje ih obrađuju.**
  - **Višestruko korišćenje koda postiže se preko biblioteka procedura i funkcija.**



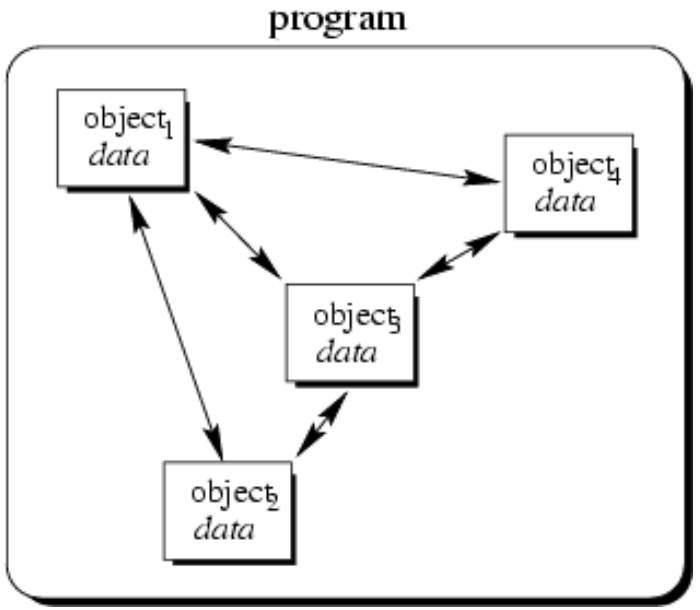
Pregled tehnika programiranja

- **Modularno programiranje**
  - **Procedure sa zajedničkom funkcionalnošću su integrisane u jedan modul**
  - **Svaki modul može da ima svoje sopstvene podatke.**
  - **Višestruko korišćenje struktura podataka i procedura**



Pregled tehnika programiranja

- **Objektno orijentisano programiranje**
  - **Strukture podataka i procedure integrisane u klase**
  - **Program može da se posmatra kao mreža objekata koji su u interakciji pri čemu svaki objekat zna svoje stanje.**
  - **Apstracija, enkapsulacija, nasleđivanje i polimorfizam**
  - **Ponovno korišćenje objekata**



Modelovanje problema

- Definisati objekte koji se sreću u opisu problema
- Apstrahovati objekte klasama
- Definisati elemente klasa
- Definisati broj objekata, trenutak njihovog nastajanja i nestajanja i način medjusobne interakcije tokom vremena
- Definisati odgovornosti

- Dva ključna termina:
  - **Klasa**
  - **Objekat**

Modelovanje problema klasama

- **Klase**
  - Modeluju (apstrahuju) objekte
    - Atribute (podaci članovi)
    - Ponašanja (funkcije članice)
  - Definišu se upotrebom ključne reči class
  - Funkcije članice
    - Metode
    - Aktiviraju se kao odgovori na poruke

```
class ImeKlase
{


};
```

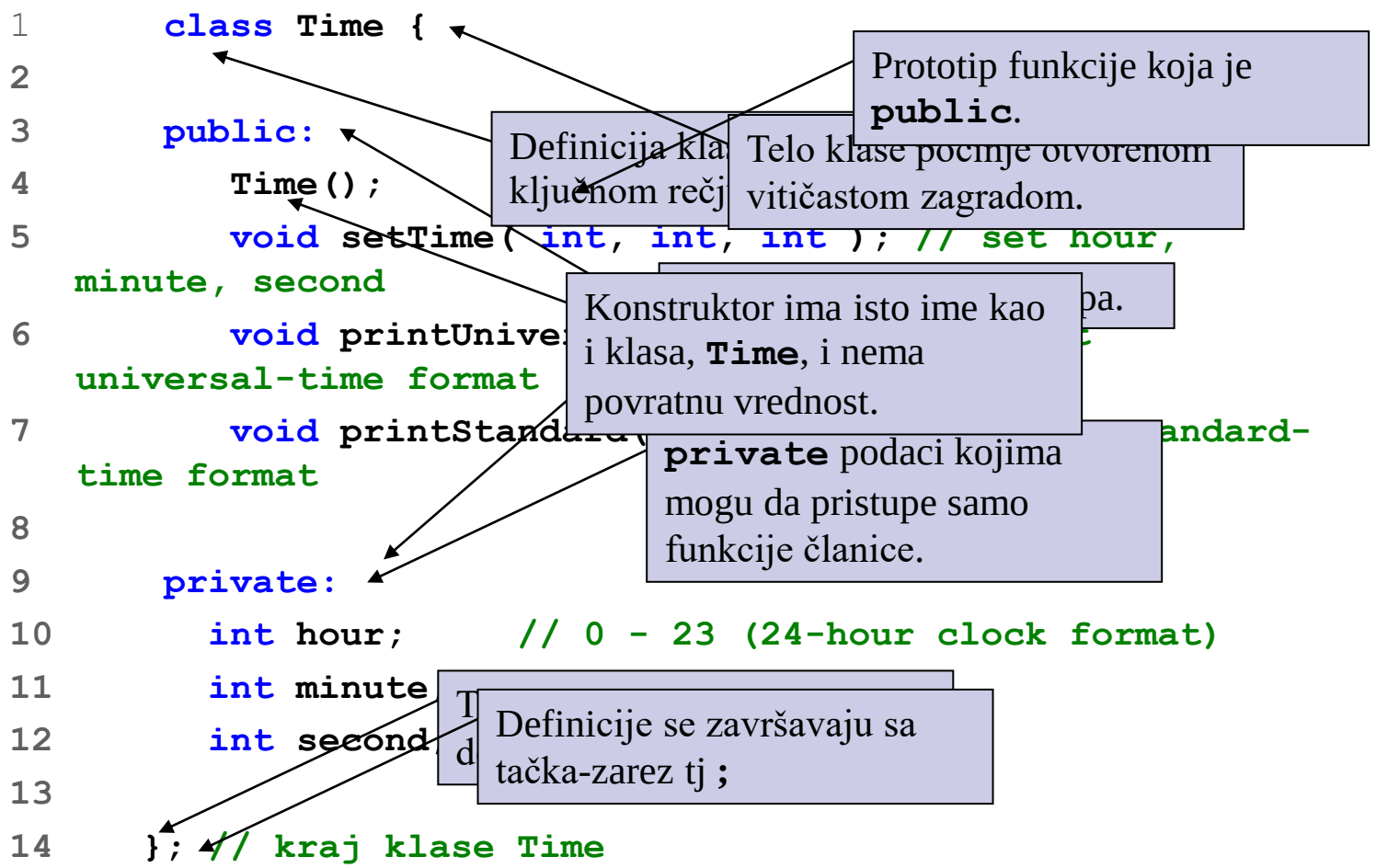
- **Specifikatori prava pristupa članovima klase**
  - **public:**
    - Pristup omogućen svuda gde objekat klase ima scope
  - **private:**
    - Pristup omogućen samo u funkcijama članicama
  - **protected:**
    - U izvedenim klasama

Konstruktori

- Specijalne funkcije članice
  - Inicijalizuju podatke članove klase
  - Isto ime kao i ime klase
- Pozivaju se kada se kreira (instancira) objekat klase
- Više konstruktora
  - Overloading funkcija
- Nemaju povratni tip

```
class ImeKlase
{


};
```

```
ImeKlase( parametri)
{

};
```



Realizacija funkcija članica:

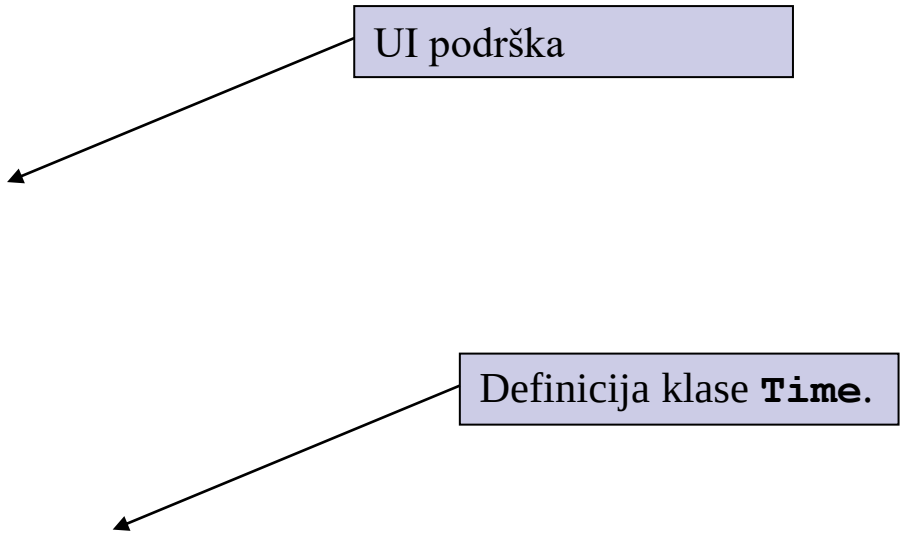
- u samoj klasi
- van klase (u istom fajlu, u drugom fajlu)

- **Funkcije članice definisane van klase**
  - **Binarni rezolucioni operator (::)**
    - "Vezuje" ime funkcije sa imenom klase
    - Jedinstveno identifikuje funkcije konkretne klase
    - Različite klase mogu imati funkcije članice sa istim imenima
  - **Format za definisanje funkcija članica**

```
PovratniTip ImeKlase::ImeFunkcijeČlanice( )
{
   …
}
```

  - **Ne zavisi da li je funkcija public ili private**
- **Funkcije članice definisane unutar klase**
  - **Nije potrebno navoditi operator :: niti ime klase**

```
1
2   // klasa Time
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   #include <iomanip>
9
10  using std::setfill;
11  using std::setw;
12
13  // Definicija apstraktnog tipa Time
14  class Time {
15
16  public:
17     Time();                          // konstruktor
18     void setTime( int, int, int ); // set hour, minute, second
19     void printUniversal();           // print universal-time format
20     void printStandard();            // print standard-time format
21
```



```
22    private:
23       int hour;      // 0 - 23 (24-hour clock format)
24       int minute;    // 0 - 59
25       int second;    // 0 - 59
26
27    }; // end class Time
28
29    // Konstruktor Time inicijalizuje sve podatke članove na nulu
30    // obezbeđujući da svi objekti klase Time imaju isti oblik
31    Time::Time()
32    {
33       hour = minute = second = 0;
34
35    } // end Time konstruktor
36
37    // set new Time value using universal time, p
38    // checks on the data values and set invalid
39    void Time::setTime( int h, int m, int s )
40    {
41       hour = ( h >= 0 && h < 24 ) ? h : 0;
42       minute = ( m >= 0 && m < 60 ) ? m : 0;
43       second = ( s >= 0 && s < 60 ) ? s : 0;
44    } // end function setTime
45
```

Konstruktor inicijalizuje **`private`** podatak na **0**.

`public` funkcija članica proverava vrednost parametara pre setovanja **`private`** podataka.

```
47    // print Time u universalnom formatu
48    void Time::printUniversal()
49    {
50       cout << setfill( '0' ) << setw( 2 ) << hour << ":"
51            << setw( 2 ) << minute << ":"
52            << setw( 2 ) << second;
53
54    } // end funkcije printUniversal
55
56    // print Time u standardnom formatu
57    void Time::printStandard()
58    {
59       cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
60            << ":" << setfill( '0' ) << setw( 2 ) << minute
61            << ":" << setw( 2 ) << second
62            << ( hour < 12 ? " AM" : " PM" );
63
64    } // end funkcije printStandard
65
66    int main()
67    {
68       Time t;  // instanciranje (kreiranje) objekta t klase Time
69
```

Nema argumente.

Deklaracija promenljive **t** kao objekta klase **Time**.

```
70      // output Time object t's initial values
71      cout << "The initial universal time is ";
72      t.printUniversal();    // 00:00:00
73
74      cout << "\nThe initial standard time is ";
75      t.printStandard();     // 12:00:00 AM
76
77      t.setTime( 13, 27, 6 );   // change time
78
79      // output Time object
80      cout << "\n\nUniversal
81      t.printUniversal();
82
83      cout << "\nStandard time after setTi
84      t.printStandard();     // 1:27:06 PM
85
86      t.setTime( 99, 99, 99 );  // attempt
87
88      // output t's values after specifying invalid values
89      cout << "\n\nAfter attempting invalid settings:"
90           << "\nUniversal time: ";
91      t.printUniversal();    // 00:00:00
92
```

poziv **public** metode

Postavljanje vrednosti člana klase korišćenjem **public** funckije članice.

Pokušaj postavljanja vrednosti člana klase na nevalidnu vrednost korišćenjem **public** funkcije članice.

```
93          cout << "\nStandard time: ";
94          t.printStandard();     // 12:00:00 AM
95          cout << endl;
96
97          return 0;
98
```

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

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

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

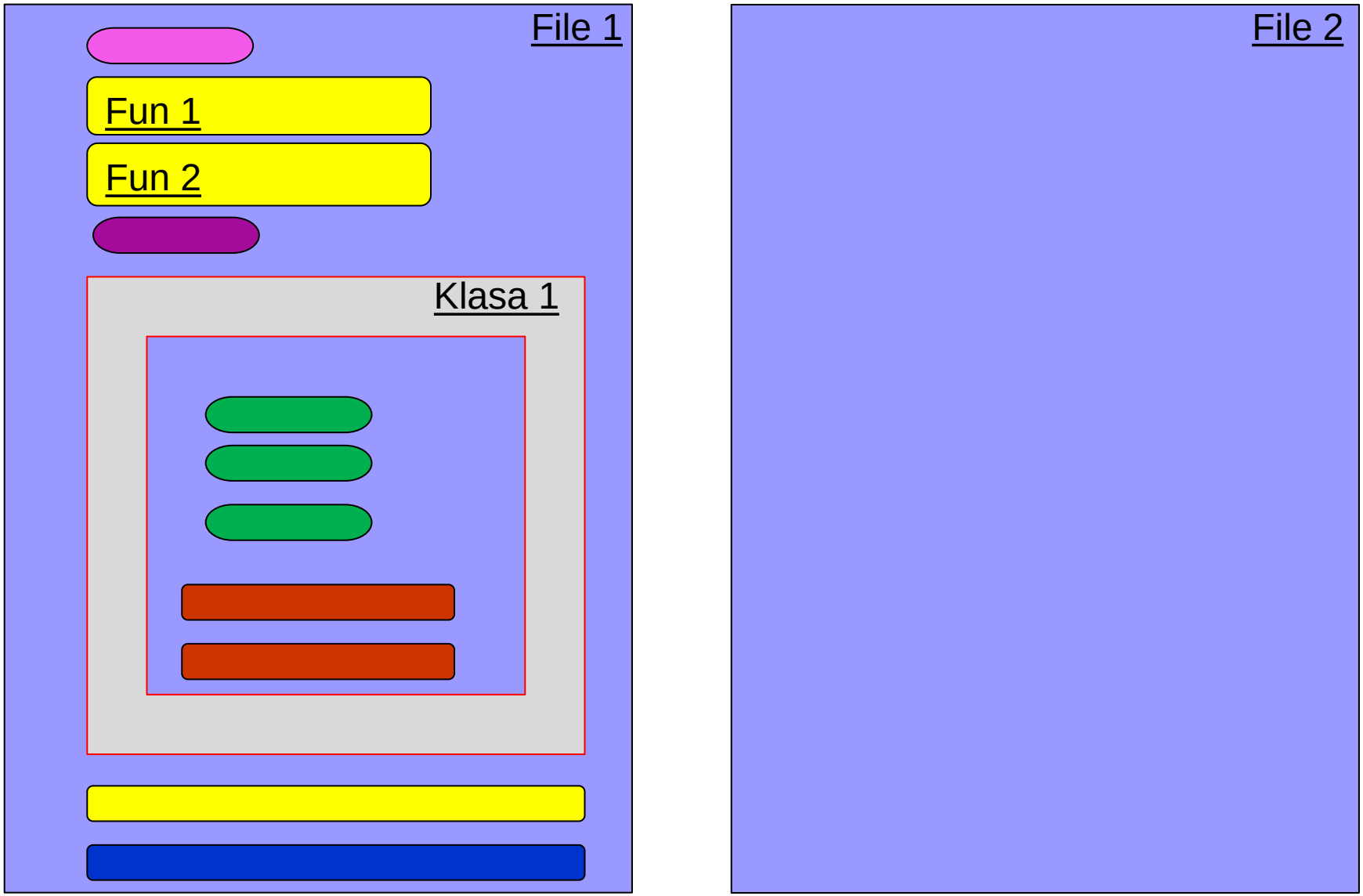
Podatak postavljen na **0** nakon pokušaja postavljanja nevalidne vrednosti.

- **Destruktori**
  - **Isto ime kao i klasa**
    - Imenu prethodi simbol tilda (~)
  - **Nemaju argumente ni povratnu vrednost**
  - **Ne mogu biti overloadovani (preklopljeni)**
  - **Izvršavaju radnje kod uništavanja objekata**
- **Prednosti korišćenja klasa**
  - **Jednostavnije programiranje**
  - **Interfejs**
    - Skrivanje implementacije
  - **Ponovno korišćenje softvera**
    - Kompozicija (agregacija)
      - **Objekti klase mogu da uključuju objekte drugih klasa**
    - Izvodjenje (nasledjivanje)
      - **Nove klase se izvode iz postojećih**

```
class ImeKlase
{


};
```

```
~ImeKlase( )
{

};
```

Vidljivost (scope)



Class Scope i pristup članovima klase

- **Class scope**
  - **U klasi se nalaze: podaci članovi, funkcije članice**
  - **U okviru class scope**
    - Članovi klase
      - Direktno dostupni svim funkcijama članicama
      - Referenciranje navodjenjem imena
  - **Van class scope**
    - Referenciranje na osnovu:
      - imena objekta,
      - reference objekta,
      - pointera na objekat
- **File scope**
  - **Funkcije koje nisu članice ni jedne klase**

Scope funkcije

- **Promenljive deklarisane u funkciji članici poznate samo u funkciji**
- **Promenljive sa istim imenom kao promenljive u class-scope**
  - Class-scope promenljive se “sakrivaju - maskiraju”
    - Pristup je moguć korišćenjem scope resolution operatora (::)

      ImeKlase::imePromenljiveKlase
- **Promenljive su poznate samo u funkcijama gde su definisane**
- **Promenljive se uništavaju nakon završetka funkcije tj. izlaska iz nje**

- **Operatori za pristup članovima klase**
  - **Identični onima kod struktura (struct)**
  - **Operator tačka (.)**
    - Objekat
    - Referenca objekta
  - **Operator strelica (->)**
    - Pointeri

- **Structure - „klase sa svim javnim članovima“**

Slanje poruka objektima

- Preko identifikatora ili reference na objekat (&) objektu šaljemo poruke (.) :

```
OsnovnaKlasa o2(30);
o2.SetPom(Pomocna());
OsnovnaKlasa &ref = o2;
ref.SetPom(Pomocna());
```

- Preko pokazivaca objektu šaljemo poruke koristeći operator (->). Objekat klase je kreiran pomoću operatora new

```
OsnovnaKlasa* ptr = new OsnovnaKlasa;
ptr->SetPom(Pomocna(40));
OsnovnaKlasa* ptr1 = new OsnovnaKlasa(30);
ptr1->SetPom(Pomocna(40)); delete ptr1;
```

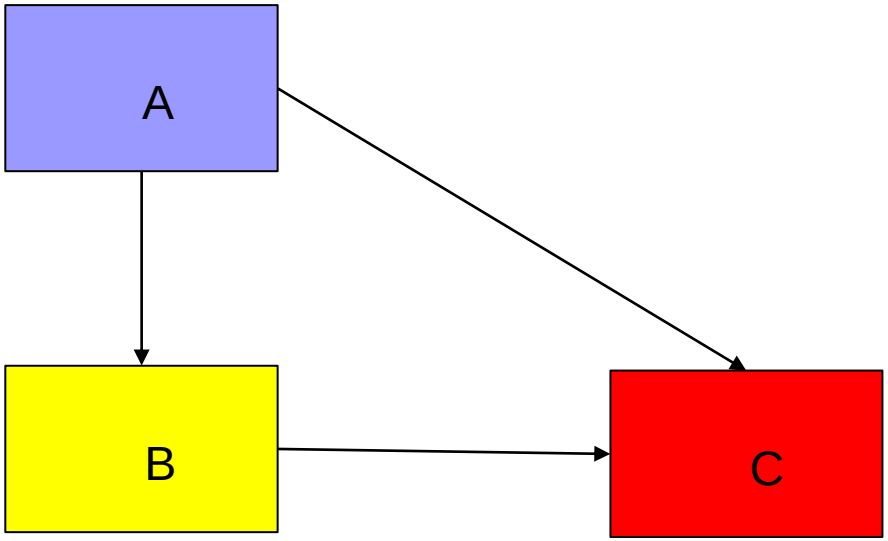
Razdvajanje Interfejsa od Implementacije

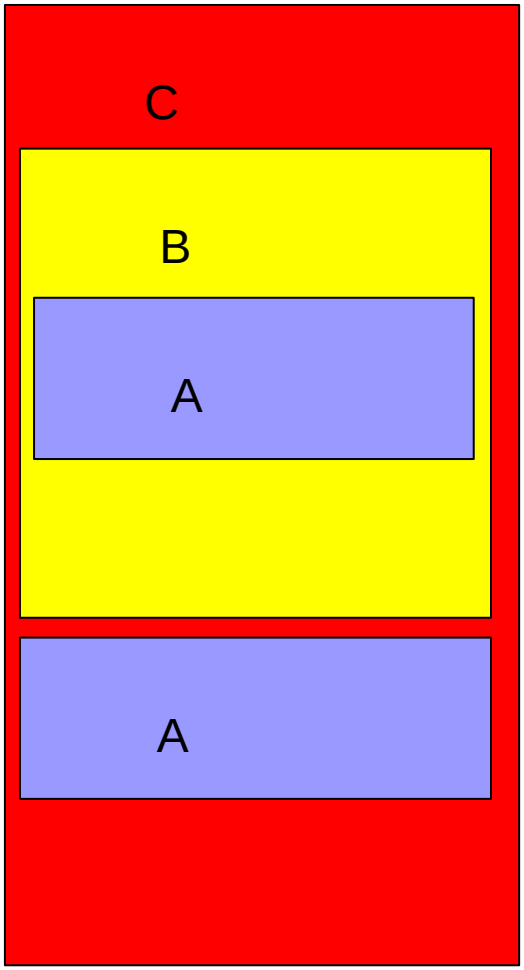
- **Razdvajanje interfejsa od implementacije**
  - **Prednosti**
    - Lakše se modifikuju programi
  - **Nedostaci**
    - Header fajlovi
      - Delovi implementacije
        - Inline funkcije članice
      - Napomene o ostalim implementacijama
        - private članovi
    - Viši stepen skrivanja korišćenjem proxy klase

- **Header fajlovi**
  - Definicija klasa i prototipova funkcija
  - Uključuju se u svaki fajl koji koristi klasu
    - #include
  - Ekstenzija fajla .h
- **Fajlovi sa kodom**
  - Definicije funkcija članica
  - Isto osnovno ime
    - konvencija
  - Kompajliraju se i posle se linkuju
  - Ekstenzija fajla .cpp

Unix: .C, .cc, .cxx, .c
GNU C++: .C, .cc, .cxx, .cpp, .c++
Digital Mars: .cpp, .cxx
Borland C++: .cpp
Watcom : .cpp
Microsoft Visual C++: .cpp, .cxx, .cc
Metrowerks CodeWarrior: .cpp, .cp, .cc, .cxx, .c++

Višestruko uključivanje fajlova





```
1
2     // Declaration of class Time.
3     // Member functions are d
4
5     // prevent multiple inclu
6     #ifndef TIME1_H
7     #define TIME1_H
8
9     // Time abstr
10    class Time {
11
12    public:
13       Time();                              // constructor
14       void setTime( int, int, int );       // set hour, minute, second
15       void printUniversal();               // print universal-time format
16       void printStandard();                // print standard-time format
17
18    private:
19       int hour;      // 0 - 23 (24-hour clock format)
20       int minute;    // 0 - 59
21       int second;    // 0 - 59
22
23    }; // end class Time
24
25    #endif
```

Preprocesorska direktiva kao zaštita od višestukog uključivanja

"If not d

Preproce
definiše

konvencija:
ime header fajla sa underscore.

```
1
2     // Member-function definitions for class Time.
3     #include <iostream>
4
5     using std::cout;
6
7     #include <iomanip>
8
9     using std::setfill;
10    using std::setw;
11
12    // include definition of class Time from time1.h
13    #include "time1.h"
14
15    // Time constructor initializes each data member to
   zero.
16    // Ensures all Time objects start in consistent
   state.
17    Time::Time()
18    {
19       hour = minute = second = 0;
20
21    } // end Time constructor
22
```

Uključivanje header fajla **`time1.h`**.

Ime header fajla u duplim navodnicima; uglasti navodnici se koriste za header fajlove iz C++ Standard Library.

```
23    // Set new Time value using universal time. Perform validity
24    // checks on the data values. Set invalid values to zero.
25    void Time::setTime( int h, int m, int s )
26    {
27       hour = ( h >= 0 && h < 24 ) ? h : 0;
28       minute = ( m >= 0 && m < 60 ) ? m : 0;
29       second = ( s >= 0 && s < 60 ) ? s : 0;
30
31    } // end function setTime
32
33    // print Time in universal format
34    void Time::printUniversal()
35    {
36       cout << setfill( '0' ) << setw( 2 ) << hour << ":"
37            << setw( 2 ) << minute << ":"
38            << setw( 2 ) << second;
39
40    } // end function printUniversal
41
```

```
42    // print Time in standard format
43    void Time::printStandard()
44    {
45       cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
46            << ":" << setfill( '0' ) << setw( 2 ) << minute
47            << ":" << setw( 2 ) << second
48            << ( hour < 12 ? " AM" : " PM" );
49
50    } // end function printStandard
```

Inline Funkcije

- Inline funkcije u C++
  - Ključna reč **inline** ispred funkcije
  - Nalog kompajleru da kopira kod u program umesto da se generiše poziv funkcije
    - Umanjuje broj poziva funkcija – ubrzanje izvršenja
    - Kompajler može da ignoriše **inline**
  - Dobro za male funkcije koje se često koriste
- **Primer:**

```
inline double cube( const double s )
    { return s * s * s; }
```

  - **const** kazuje kompajleru da funkcija ne menja **s**
  - **Funkcije članice klase:**
    - **inline ključna reč**
    - **Definicija u okviru klase**

```
inline int Max(int x, int y)
{
  return (x > y)? x : y;
}

// main funkcija
int main( )
{

  cout << "Max (20,10): " << Max(20,10) << endl;
  cout << "Max (0,200): " << Max(0,200) << endl;
  cout << "Max (100,1010): " << Max(100,1010) << endl;
  return 0;
}
```

Na izlazu:

```
Max (20,10): 20
Max (0,200): 200
Max (100,1010): 1010
```

Inline funkcije i klase

```
class S
{
  public:
     inline int square(int s) // redundantno korišćenje inline
     {
              // funkcija je automatski inline
              // telo funkcije
     }
};

// Dobar stil programiranja
class S
{
   public:
         int square(int s); // deklaracija funkcije
};

inline int S::square(int s) // definicija funkcije sa prefiksom inline
{
}
```

Ignorisanje inline zahteva

- Neke situacije kada kompajler ignoriše inline zahtev
  - Ako funkcija sadrži petlju (**for**, **while**, **do-while**)
  - Ako funkcija sadrži statičke promenljive
  - Ako je funkcija rekurzivna
  - Ako funkcija nije tipa **void** a u telu funkcije nije navedena **return** naredba
  - Ako funkcija sadrži **switch** ili **goto** naredbu
  - Ako je telo funkcije suviše veliko

Prednosti

- Ubrzanje programa za pozive funkcija koje su inline
- Smanjuje zauzeće steka koji se povećava svakim pozivom funkcije (parametri, povratna adresa, …)
- Ako se definicija nalazi u header fajlu onda koji je uključen u više unita onda se smanjuje vreme linkovanja

Ima i nedostataka

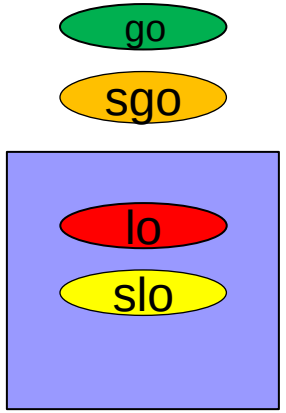
- Povećanje izvršnog fajla
- Smanjen instruction cache hit rate
- Moguće povećanje vremena kompajliranja
- Potencijalno povećanje registarskih promenljivih
- Virtuelne funkcije (u runtime se rešavaju) ne mogu biti inline (u compile tie se rešavaju)
- Nisu korisne za većinu embedded sistema jer je tu kritičan prostor a ne vreme izvršenja
- …

Preporuka za korišćenje inline fun.

- Ako je brzina izvršenja programa jako bitna
- Koristiti inline funkcije a ne macro-e (praktično suvišni u C++-u, ne mogu da pristupe privatnim članovima klase, itd.)
- Koristiti inline ključnu reč samo u definiciji fukcije članice klase
- I/O funkcije troše dosta vremena i nisu dobri kandidati za inline

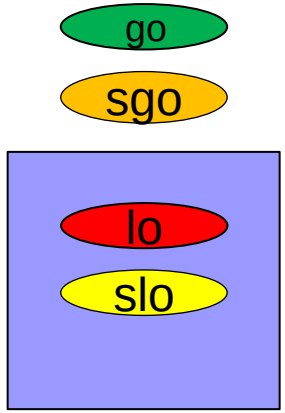
```
inline void show()
{
    cout << "value of S = " << S << endl;
}
```

Poziv Konstruktora i Destruktora



- Konstruktori i destruktori
  - Implicitno se pozivaju - kompajler
- Redosled poziva
  - Zavisi od redosleda izvršavanja
    - Kada u toku izvršavanja objekti dobijaju odnosno gube scope (oblast važenja, doseg)
  - Generalno, destruktori se pozivaju u suprotnom redosledu od redosleda poziva konstruktora

Poziv Konstruktora i Destruktora



- Redosled poziva konstruktora i destruktora
  - Globalni objekti
    - Konstruktori
      - Pre bilo koje druge funkcije (uključujući i main)
    - Destruktori
      - Kada se main završava (ili je pozvana exit funkcija)
      - Ne poziva se ako se program završi sa abort
  - Automatski lokalni objekti
    - Konstruktori
      - Kada se objekti definišu
        - Svaki put kada u toku izvršenja dobiju scope
    - Destruktori
      - Kada objekti prestanu da važe tj. gube scope
        - Kada se završava blok u kome je objekat definisan
      - Ne poziva se ako se program završi sa exit ili abort

- **Statički lokalni objekti**
  - **Konstruktori**
    - Tačno jedanput
    - Kada se dodje do mesta gde je objekat definisan
  - **Destruktori**
    - Kada se main završava ili je pozvana exit funkcija
    - Ne poziva se ako se program završava sa abort

```
1
2       // Definition of class CreateAndDestroy.
3       // Member functions defined in create.cpp.
4       #ifndef CREATE_H
5       #define CREATE_H
6
7       class CreateAndDestroy {
8
9       public:
10        CreateAndDestroy( int, char * );  // constructor
11        ~CreateAndDestroy();              // destructor
12
13    private:
14        int objectID;
15        char *message;
16
17    }; // end class CreateAndDestroy
18
19    #endif
```

```
1
2       // Member-function definitions for class CreateAndDestroy
3       #include <iostream>
4
5       using std::cout;
6       using std::endl;
7
8       // include CreateAndDestroy class definition from create.h
9       #include "create.h"
10
11    // constructor
12    CreateAndDestroy::CreateAndDestroy(
13       int objectNumber, char *messagePtr )
14    {
15       objectID = objectNumber;
16       message = messagePtr;
17
18       cout << "Object " << objectID << "  constructor runs   "
19            << message << endl;
20
21    } // end CreateAndDestroy constructor
22
```

```
23    // destructor
24    CreateAndDestroy::~CreateAndDestroy()
25    {
26
27       cout << ( objectID == 1 || objectID == 6 ? "\n" : "" );
28
29       cout << "Object " << objectID << "  destructor runs    "
30            << message << endl;
31
32    } // end ~CreateAndDestroy destructor
```

```
// Demonstrating the order in which constructors and
// destructors are called.
#include <iostream>

using std::cout;
using std::endl;

// include CreateAndDestroy class definition from create.h
#include "create.h"

void create( void );    // prototype

// global object
CreateAndDestroy first( 1, "(global before main)" );

int main()
{
   cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;

   CreateAndDestroy second( 2,"(local automa

   static CreateAndDestroy third(3,"(local static in main)" );
```

Kreiranje globalnog objekta.

Automatsi lokalni objekat

Kreiranje **static** lokalnog objekta.

```
   create();  // call function to create objects

   cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;

   CreateAndDestroy fourth( 4, "(local automatic in main)" );

   cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;

   return 0;
} // end main

// function to create objects
void create( void )
{
   cout << "\nCREATE FUNCTION: EXECUTION

   CreateAndDestroy fifth( 5, "(local au                    );

   static CreateAndDestroy sixth(6, "(lo                   )" );

   CreateAndDestroy seventh(7, "(local automatic in create)" );

   cout << "\nCREATE FUNCTION: EXECUTION ENDS\" << endl;
} // end function create
```

Kreiranje lokalnih automatskih objekata.

Kreiranje lokalnog automatskog objekta.

Kreiranje lokalnog automatskog objekta u

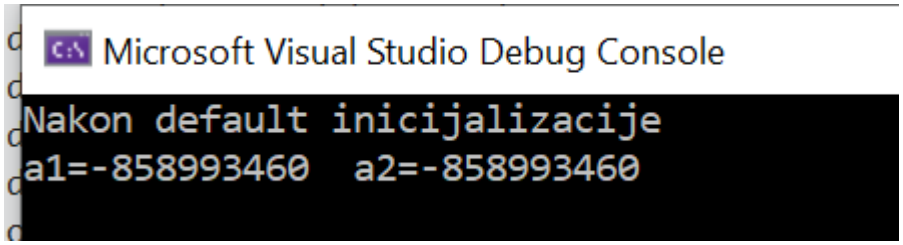
Kreiranje **static** lokalnog objekta u funkciji.

Kreiranje lokalnog automatskog objekta u funkciji.

```
Object 1   constructor runs   (global before main)

MAIN FUNCTION: EXECUTION BEGINS
Object 2   constructor runs   (local automatic in main)
Object 3   constructor runs   (local static in main)

CREATE FUNCTION: EXECUTION BEGINS
Object 5   constructor runs   (local automatic in create)
Object 6   constructor runs   (local static in create)
Object 7   constructor runs   (local automatic in create)

CREATE FUNCTION: EXECUTION ENDS
Object 7   destructor runs    (local automatic in create)
Object 5   destructor runs    (local automatic in create)

MAIN FUNCTION: EXECUTION RESUMES
Object 4   constructor runs   (local automatic in main)

MAIN FUNCTION: EXECUTION ENDS
Object 4   destructor runs    (local automatic in main)
Object 2   destructor runs    (local automatic in main)
Object 6   destructor runs    (local static in create)
Object 3   destructor runs    (local static in main)

Object 1   destructor runs    (global before main)
```

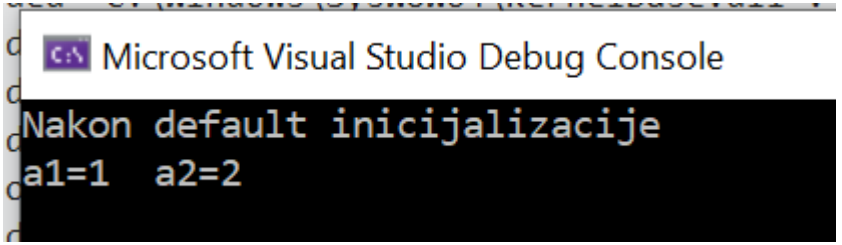
Kreiranje objekata i konstruktori

```
4   #include <iostream>
5   using namespace std;
6
7   class KreirajObj {
8       int a1;
9       int a2;
10
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16  };
17
18  int main()
19  {
20      KreirajObj kObj1;
21      cout << "Nakon default inicijalizacije" << endl;
22      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
23      kObj1.setA1(10);
24      kObj1.setA2(20);
25
26  }
27
28  void KreirajObj::setA1(int x) { a1 = x; }
29  void KreirajObj::setA2(int y) { a2 = y; }
30  int KreirajObj::getA1() { return a1; }
31  int KreirajObj::getA2() { return a2; }
```



Kreiranje objekata i konstruktori

```
4   #include <iostream>
5   using namespace std;
6
7   class KreirajObj {
8       int a1;
9       int a2;
10
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      KreirajObj() { a1 = 1;  a2 = 2; }
17  };
18
19  int main()
20  {
21      KreirajObj kObj1;
22      cout << "Nakon default inicijalizacije" << endl;
23      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
24      kObj1.setA1(10);
25      kObj1.setA2(20);
26
27  }
28
29  void KreirajObj::setA1(int x) { a1 = x; }
30  void KreirajObj::setA2(int y) { a2 = y; }
31  int KreirajObj::getA1() { return a1; }
32  int KreirajObj::getA2() { return a2; }
```



```
20  int main()
21  {
22      KreirajObj kObj1;
23      cout << "Obj1 Nakon default inicijalizacije" << endl;
24      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
25      kObj1.setA1(10);
26      kObj1.setA2(20);
27      cout << "Nakon izmene vrednosti atributa" << endl;
28      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
29
30      KreirajObj kObj2(5, 10);
31      cout << "Obj2 Nakon default inicijalizacije" << endl;
32      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << endl;
33
34      kObj2 = kObj1;
35      cout << "Obj2 Nakon dodele Obj1" << endl;
36      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << endl;
37  }
```

```
Obj1 Nakon default inicijalizacije
a1=1  a2=2
Nakon izmene vrednosti atributa
a1=10  a2=20
Obj2 Nakon default inicijalizacije
a1=5  a2=10
Obj2 Nakon dodele Obj1
a1=10  a2=20
```

```
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      //KreirajObj() { a1 = 1;  a2 = 2; }
17      KreirajObj(int x, int y) { a1 = x; a2 = y; }
18  };
19
20  int main()
21  {
22      KreirajObj kObj1;
23      cout << "Obj1 Nakon default inicijalizacije" << endl;
24      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
25      kObj1.setA1(10);
26      kObj1.setA2(20);
```

Error List

Entire Solution | 2 Errors | 0 Warnings | 1 Message | Build + IntelliSense

	Code	Description
abc	E0291	no default constructor exists for class "KreirajObj"
⊗	C2512	'KreirajObj': no appropriate default constructor available
ⓘ		see declaration of 'KreirajObj'

```
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      //KreirajObj() { a1 = 1;  a2 = 2; }
17      KreirajObj(int x = 3, int y = 4) { a1 = x; a2 = y; }
18  };
19
20  int main()
21  {
22      KreirajObj kObj1;
23      cout << "Obj1 Nakon default inicijalizacije" << endl;
24      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
25      kObj1.setA1(10);
26      kObj1.setA2(20);
```

Microsoft Visual Studio Debug Console

```
Obj1 Nakon default inicijalizacije
a1=3  a2=4
Nakon izmene vrednosti atributa
a1=10  a2=20
Obj2 Nakon default inicijalizacije
a1=5  a2=10
Obj2 Nakon dodele Obj1
a1=10  a2=20
```

```
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      KreirajObj() { a1 = 1;  a2 = 2; }
17      KreirajObj(int x = 3, int y = 4) { a1 = x; a2 = y; }
18  };
19
20  int main()
21  {
22      KreirajObj kObj1;
23      cout << "Obj1 Nakon default inicijalizacije" << endl;
24      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
25      kObj1.setA1(10);
26      kObj1.setA2(20);
27      cout << "Nakon izmene vrednosti atributa" << endl;
28                                                          1;
```

Error List

Entire Solution | 2 Errors | 0 Warnings | 3 Messages | Buil

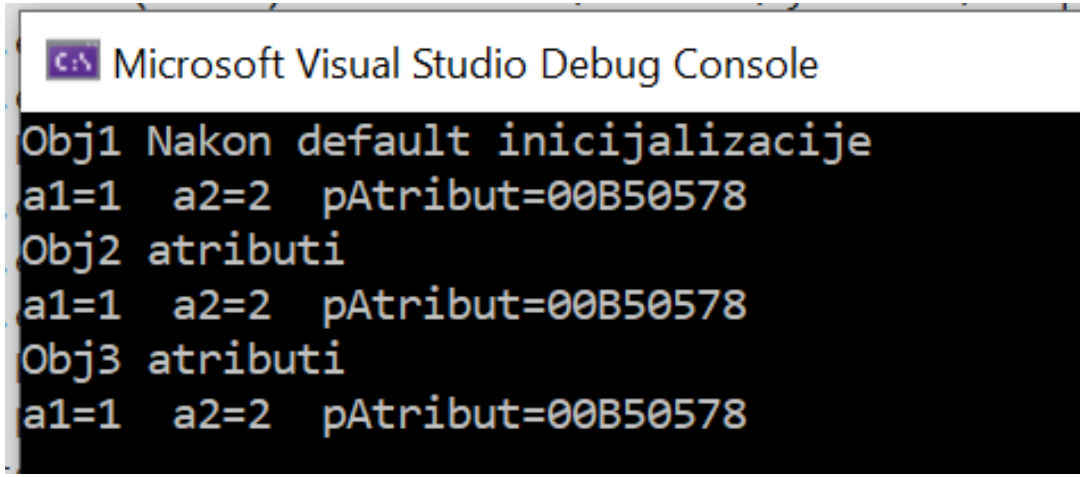
	Code	Description
abc	E0339	class "KreirajObj" has more than one default constructor
✖	C2668	'KreirajObj::KreirajObj': ambiguous call to overloaded function
ℹ		could be 'KreirajObj::KreirajObj(int,int)'
ℹ		or 'KreirajObj::KreirajObj(void)'
ℹ		while trying to match the argument list '()'

```
20  int main()
21  {
22      KreirajObj kObj1;
23      cout << "Obj1 Nakon
24      cout << "a1=" << kO
25      kObj1.setA1(10);
26      kObj1.setA2(20);
27      cout << "Nakon izmene vrednosti atributa" << endl;
28      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << endl;
29
30      KreirajObj kObj2(5, 10);
31      cout << "Obj2 Nakon default inicijalizacije" << endl;
32      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << endl;
33
34      kObj2 = kObj1;
35      cout << "Obj2 Nakon dodele Obj1" << endl;
36      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << endl;
37
38      KreirajObj kObj3(kObj2);
39      cout << "Obj3 Nakon inicijalizacije objektom Obj2" << endl;
40      cout << "a1=" << kObj3.getA1() << "  a2=" << kObj3.getA2() << endl;
41  }
```

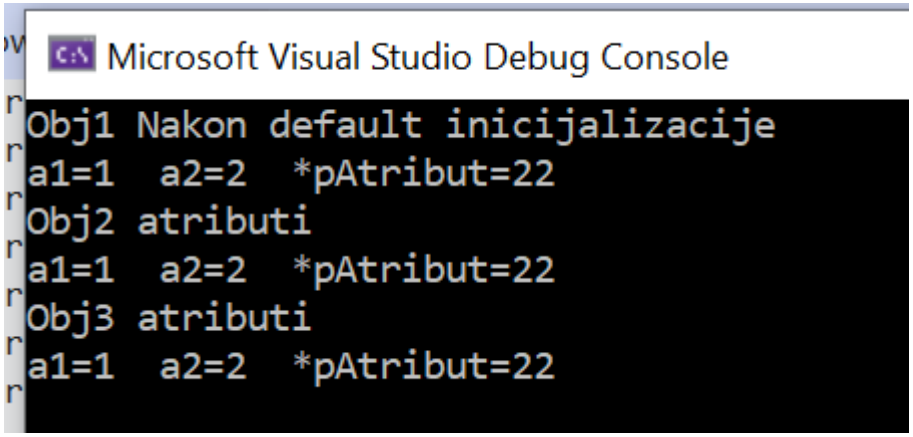
Microsoft Visual Studio Debug Console

```
Obj1 Nakon default inicijalizacije
a1=-858993460  a2=-858993460
Nakon izmene vrednosti atributa
a1=10  a2=20
Obj2 Nakon default inicijalizacije
a1=5  a2=10
Obj2 Nakon dodele Obj1
a1=10  a2=20
Obj3 Nakon inicijalizacije objektom Obj2
a1=10  a2=20
```

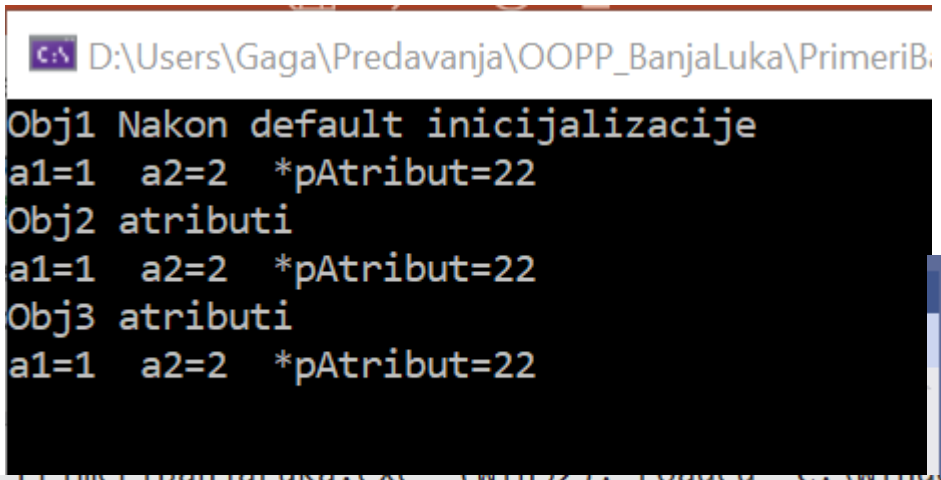
```
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      int* pAtribut;
17      KreirajObj() { a1 = 1;  a2 = 2; pAtribut = new int; }
18      KreirajObj(int x, int y) { a1 = x; a2 = y; pAtribut = new int; }
19  };
20
21  int main()
22  {
23      KreirajObj kObj1;
24      cout << "Obj1 Nakon default inicijalizacije" << endl;
25      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << "pAtribut=" << kObj1.pAtribut << endl;
26      KreirajObj kObj2(5, 10);
27      kObj2 = kObj1;
28      cout << "Obj2 atributi" << endl;
29      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << "pAtribut=" << kObj2.pAtribut << endl;
30
31      KreirajObj kObj3(kObj2);
32      cout << "Obj3 atributi" << endl;
33      cout << "a1=" << kObj3.getA1() << "  a2=" << kObj3.getA2() << "pAtribut=" << kObj3.pAtribut << endl;
34
```

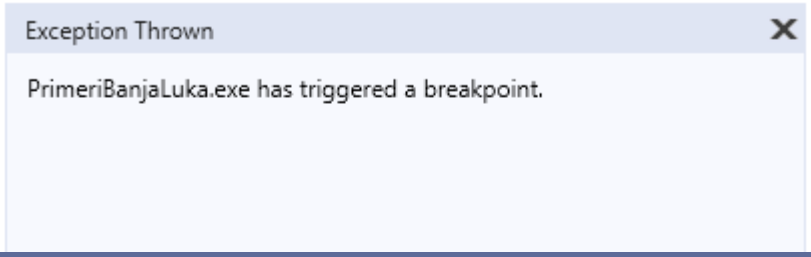


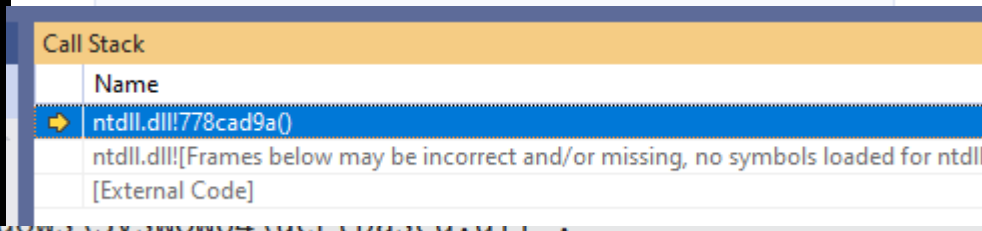
```
11  public:
12      void setA1(int);
13      void setA2(int);
14      int getA1();
15      int getA2();
16      int* pAtribut;
17      KreirajObj() { a1 = 1;  a2 = 2; pAtribut = new int; *pAtribut = 22; }
18      KreirajObj(int x, int y) { a1 = x; a2 = y; pAtribut = new int; *pAtribut = 33; }
19
20  };
21
22  int main()
23  {
24      KreirajObj kObj1;
25      cout << "Obj1 Nakon default inicijalizacije" << endl;
26      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << "  *pAtribut=" << *kObj1.pAtribut << endl;
27      KreirajObj kObj2(5, 10);
28      kObj2 = kObj1;
29      cout << "Obj2 atributi" << endl;
30      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << "  pAtribut=" << *kObj2.pAtribut << endl;
31
32      KreirajObj kObj3(kObj2);
33      cout << "Obj3 atributi" << endl;
34      cout << "a1=" << kObj3.getA1() << "  a2=" << kObj3.getA2() << "  pAtribut=" << *kObj3.pAtribut << endl;
35
```



```
16      int* pAtribut;
17      KreirajObj() { a1 = 1;  a2 = 2; pAtribut = new int; *pAtribut = 22; }
18      KreirajObj(int x, int y) { a1 = x; a2 = y; pAtribut = new int; *pAtribut = 33; }
19      ~KreirajObj() { delete pAtribut; }
20  };
21
22  int main()
23  {
24      KreirajObj kObj1;
25      cout << "Obj1 Nakon default inicijalizacije" << endl;
26      cout << "a1=" << kObj1.getA1() << "  a2=" << kObj1.getA2() << "  *pAtribut=" << *kObj1.pAtribut << endl;
27      KreirajObj kObj2(5, 10);
28      kObj2 = kObj1;
29      cout << "Obj2 atributi" << endl;
30      cout << "a1=" << kObj2.getA1() << "  a2=" << kObj2.getA2() << "  *pAtribut=" << *kObj2.pAtribut << endl;
31
32      KreirajObj kObj3(kObj2);
33      cout << "Obj3 atributi" << endl;
34      cout << "a1=" << kObj3.getA1() << "  a2=" << kObj3.getA2() << "  *pAtribut=" << *kObj3.pAtribut << endl;
```








```
The thread 0x25ac has exited with code 0 (0x0).
HEAP[PrimeriBanjaLuka.exe]: Invalid address specified to RtlValidateHeap( 00A60000, 00A60558 )
PrimeriBanjaLuka.exe has triggered a breakpoint.
```

Podrazumevana pokomponentna dodela

- int x, z; x=z; KlasaC x, z; x=z; ????
- **Dodela objekata**
  - **Operator dodele (=)**
    - **Objekat se može dodeliti objektu istog tipa**
    - **Default: pokomponentna dodela**
      - **Svaki desni član se pojedinačno dodeljuje levom**
- **Prosledjivanje, vraćanje objekata**
  - **Objekti kao argumenti funkcija**
  - **Objekti kao vraćene vrednosti iz funkcija**
  - **Default: prosledjivanje po vrednosti (pass-by-value)**
    - **Kopija objekta se prosledjuje, vraća**
      - **Copy konstruktor**
        - **Kopira originalne vrednosti u novi objekat**

Konstruktor kopije i kopiranje

- Sličan normalnom konstruktoru, ali kreira kopiju postojećeg objekta.
- Kao i kod standardnih konstruktora, kompajler će kreirati konstruktor kopije, ukoliko je potreban, a programer je zaboravio da ga definiše. Na žalost, ovakav konstruktor često ima neželjeno dejstvo i takav pristup treba izbegavati.
- Čak i kada programer nije definisao konstruktor kopije, sledeće linije koda su validne

```
Osnovna o = 20;
Osnovna kopija = o; // inicijalizacija objekta kopijom
```

- Podrazumevano ponašanje je da se kreira novi objekat čije će promenljive sadržati kopije vrednosti objekta koji se kopira.

Konstruktor kopije i kopiranje

```
class Osnovna {
  int x;
  Pomocna objPom;
  Pomocna *ptrPom;
public:
  Osnovna(int initX):x(initX), objPom(),ptrPom(new Pomocna){}

// Konstruktor kopije koji "kodira" kompajler i koji je uzrok
problema
  Osnovna(const Osnovna &obj):
       x(obj.x), objPom(obj.objPom),
            ptrPom(obj.ptrPom){}


// Mesto gde se problem manifestuje: dva pokazivaca ukazuju na
isti objekat. Pokusaj ponovnog brisanja već obrisanog objekta
ima za posledicu "runtime error".
  ~Osnovna() { delete ptrPom;}
};
```

Konstruktor kopije i kopiranje

```
class Osnovna {
  int x;
  Pomocna objPom;
  Pomocna *ptrPom;
public:
  Osnovna(int initX):x(initX), objPom(),ptrPom(new Pomocna){}

// Pravilno "kodiran" konstruktor kopije

  Osnovna(const Osnovna &obj):
       x(obj.x), objPom(obj.objPom),
              ptrPom(new Pomocna(*obj.ptrPom)){}




  ~Osnovna() { delete ptrPom;}
};
```

Konstruktor kopije

- Kada se objekat `x1` klase `XX` inicijalizuje drugim objektom `x2` iste klase, C++ će podrazumevano izvršiti prostu inicijalizaciju redom članova objekta `x1` članovima objekta `x2`.
- To ponekad nije zadovoljavajuće (često ako objekti sadrže članove koji su pokazivači ili reference), pa programer treba da ima potpunu kontrolu nad inicijalizacijom objekta drugim objektom iste klase.
- Za ovu svrhu služi tzv. konstruktor kopije (engl. *copy constructor*).
- Konstruktor kopije klase `XX` se može pozvati sa samo jednim stvarnim argumentom tipa `XX`.
- Konstruktor kopije se poziva kada se objekat inicijalizuje objektom iste klase, a to je:
  1. prilikom inicijalizacije objekta (pomoću znaka = ili sa zagradama);
  2. prilikom prenosa argumenata u funkciju (kreira se lokalni automatski objekat);
  3. prilikom vraćanj vrednosti iz funkcije (kreira se privremeni objekat).
- Konstruktor ne sme imati formalni argument tipa `XX`.
- Konstruktor kopije ima argument tipa `XX&` ili (najčešće) `const XX&`.
- Ostali eventualni argumenti kopirajućeg konstruktora moraju imati podrazumevane vrednosti.

```
3   class KK {
4       public:
5           KK (int);
6           KK (const KK&); // konstruktor kopije
7           ...
8   };
9
10  KK f(KK x1) {
11      KK x2=x1;      // poziva se konstruktor kopije XX(XX&) za x2
12      //...
13      return x2;     // poziva se konstruktor kopije za
14                     // privremeni objekat u koji se smešta rezultat
15  }
16
17  void g() {
18      KK objA=5, objB=10;
19      ...
20      objA=f(objB);  // poziva se konstruktor kopije samo za
21                     // formalni argument x1,
22                     // a u xa se samo prepisuje privremeni objekat,
23                     // ili se poziva XX::operator= ako je definisan
24  }
```

move (pomerajući) - konstruktor

- Pravimo kopiju objekta (koji će biti uništen) koji u sebi sadrži dinamički objekat tako da kopija preuzima dinamički objekat originala i nakon toga original nema veze sa dinamičkim objektom

```
61  class GenerickiNiz {
62  private:
63      int* pvrednosti;
64      int duzinaNiza;
65  public:
66      // podrazumevani  konstruktor
67      GenerickiNiz() : pvrednosti(new int[10]), duzinaNiza(10) {  }
68      // konstruktor za n elemenata
69      GenerickiNiz(int n) : pvrednosti(new int[n]), duzinaNiza(n) {  }
70      // move konstruktor
71      GenerickiNiz(GenerickiNiz&& original)
72          : pvrednosti(original.pvrednosti), duzinaNiza(original.duzinaNiza) {
73          original.pvrednosti = nullptr;
74          original.duzinaNiza = 0;
75      }
76      // konstruktor kopije
77      GenerickiNiz(const GenerickiNiz& original)
78          : pvrednosti(new int[original.duzinaNiza]), duzinaNiza(original.duzinaNiza) {
79          for (int i = 0; i < duzinaNiza; ++i)
80              pvrednosti[i] = original.pvrednosti[i];
81      }
82
83      ~GenerickiNiz() { delete[] pvrednosti; }
84  };
```

Konverzije primenom konstruktora

- Konstruktor se može iskoristiti za implicitnu konverziju:

```
class Osnovna{
   int i;
public:
   Osnovna(int i) { this->i=i;}
   void Dodaj(Osnovna); //implementira sabiranje
};
void Osnovna::Dodaj(Osnovna drugi)
{
   i+=drugi.i;
}
...
Osnovna o(30);
o.Dodaj(40);
 /* Funkcija Dodaj ocekuje argument tipa Osnovna;
   Konstruktor ove klase bice automatski pozvan da
   izvrsi implicitnu konverziju int u Osnovna */
```

Konverzije primenom konstruktora

- Ukoliko želimo da zabranimo implicitnu (automatsku) konverziju ispred konstruktora navodimo ključnu reč *explicit* :

```
class Osnovna{
   int i;
public:
   explicit Osnovna(int i) { this->i=i;}
   void Dodaj(Osnovna);
};
void Osnovna::Dodaj(Osnovna drugi)
{
   i+=drugi.i;
}
...
Osnovna o(30);
o.Dodaj(40); // Ovo kompajler nece dozvoliti
o.Dodaj(Osnovna(40));  // Ovo prolazi bez problema
```