

OSNOVNA STRUKTURA PROGRAMA



FUNKCIJE

- Da bi se **olakšala izrada programa**, glavni se zadatak razlaže **na manje, smislene cjeline**.
- Svaka se cjelina **rješava zasebno**, a dobivena rješenja svake od cjelina nazivamo **funkcijama**.
- Funkciju čini **niz naredaba** koje **obavljaju određeni zadatak**.

Sanda, 2015.

2

FUNKCIJE

- Funkcije mogu biti:
 - **unaprijed pripremljene** (pohranjene u bibliotekama), korisnik ih **poziva** u svoj program.
 - **vlastite** funkcije, korisnik ih **sam stvara**.
- Način na koji se funkcije slažu i povezuju u cjelovit i smislen program nazivamo **struktura programa**.

Sanda, 2015.

3

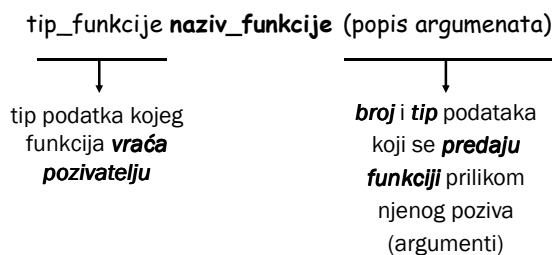
FUNKCIJE

- Funkcije treba znati:
 - **deklarirati**,
 - **definirati**,
 - **pozvati**.

Sanda, 2015.

4

DEKLARACIJA FUNKCIJE



Sanda, 2015.

5

DEFINICIJA FUNKCIJE

- **Naredbe** koje čine funkciju zapisuju se **unutar para vitičastih zagrada**, a tvore **tijelo funkcije**.
- Tijelo funkcije se **zbog preglednosti piše uvučeno** u odnosu na ostali izvorni kod.
- Svaka naredba završava **znakom točka-zarez (;)**.

```
#include <iostream>
using namespace std;
int main()
{
    int broj, pom1, rez;
    cout << "Upiši troznamenkasti broj: ";
    cin >> broj;
    pom1 = broj / 10;
    rez = pom1 % 10;
    cout << "U troznamenkastom broju " << broj
    << " na mjestu desetice je znamenka " << rez;
    return 0;
}
```

```
int broj, pom1, rez;
cout << "Upiši troznamenkasti broj: ";
cin >> broj;
```

Sanda, 2015.

6

NAREDBE

- Više se naredaba može zapisati u jednome retku, ali svaka od njih mora završiti znakom točka-zarez (;).

```
float a;  
int b;  
a=123.456;  
b=123456;  
cout<<a<<b<<endl;
```

```
float a;int b;a=123.456;b=123456;  
cout<<a<<b<<endl;
```

- Naredba se može protezati i kroz nekoliko redaka:

```
    pom1=broj/10;  
    rez=pom1%10;  
    cout<<"U troznamenkastom broju "<<broj  
    <<" na mjestu desetice je znamenka "<<rez;  
    return 0;  
}
```

Sanda, 2015.

7

TIJELO FUNKCIJE

- Tijelo funkcije završava naredbom **return**, a uz **return** se zapisuje **podatak kojeg funkcija** (kao svoj rezultat) **vraća** pozivatelju.

```
#include<iostream>  
using namespace std;  
int main()  
{  
    int broj, pom1, rez;  
    cout<<"Upisi troznamenkasti broj:";  
    cin>>broj;  
    pom1=broj/10;  
    rez=pom1%10;  
    cout<<"U troznamenkastom broju "<<broj  
    <<" na mjestu desetice je znamenka "<<rez;  
    return 0;  
}
```

Sanda, 2015.

8

POZIV FUNKCIJE

- Ako se funkcija želi rabiti u programu treba je **pozvati**.
- Poziva se **navođenjem njenog naziva i argumenata** zapisanih u redosljedu koji je zadan deklaracijom.
- Na sljedećem slajdu je primjer u kome se u glavnoj funkciji poziva funkcija **volumen_kvadra**.

Sanda, 2015.

9

```
#include<iostream>  
using namespace std;  
float volumen_kvadra (float a, float b, float c)  
{  
    float V;  
    V= a*b*c;  
    return V;  
}  
int main()  
{  
    float x,y,z,v1;  
    cout<<"Unesi vrijednosti bridova kvadra:"<<endl;  
    cin>>x>>y>>z;  
    v1=volumen_kvadra(x,y,z);  
    cout<<"Volumen kvadra iznosi"<<v1<<endl;  
    return 0;  
}
```

funkcija

poziv funkcije

Sanda, 2015.

10

FUNKCIJA MAIN

- Program može sadržavati **proizvoljno mnogo funkcija**, no kako će se "znati" na kojem mjestu započinje **glavna funkcija**?
- U tu se svrhu uvodi **funkcija main** ("ulazna točka").
- Ako program ne sadrži **funkciju main** prevoditelj javlja pogrešku.

```
Configuration: s - Win32 Debug  
-----  
Compiling...  
s.cpp  
C:\Documents and Settings\Ostali\Desktop\o\o.cpp(4) error C2447: missing function header (old-style formal list?)  
Error executing cl.exe.  
s.exe - 1 error(s), 0 warning(s)
```

Sanda, 2015.

11

DEKLARACIJA FUNKCIJE MAIN

- Postoji više oblika funkcije **main**. U ovim će se vježbama rabiti oblik:

```
int main ( )
```

- Funkcija **main** završava naredbom **return 0**.

Sanda, 2015.

12

MAIN () – RETURN 0

- Ako neka od naredbi **funkcije main** izazove prekid programa, pri izvršenju se neće stići do naredbe **return 0** i ona se neće izvršiti.
- Dakle, **0** je **poruka pozivatelju** (a to je obično operacijski sustav) da su sve naredbe funkcije **main** **izvršene ispravno**.

Sanda, 2015.

13

FUNKCIJA main

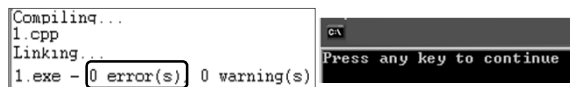
```
int main()
{
    return 0;
}
```

Sanda, 2015.

14

FUNKCIJA main

- Ako se pokrene prevođenje i povezivanje, sve će proći bez pogreške.



- Kada se program pokuša izvršiti, neće se dogoditi ništa jer u tijelu funkcije nema naredbi (osim **return 0**).

Sanda, 2015.

15

PREPROCESSORSKE NAREDBE

- Navode se **prije main funkcije**.
- Pretprocesorske naredbe **dopunjuju izvorni kôd** a zatim se takav dopunjeni (modificirani) kôd prevodi.
- **Opći oblik** pretprocesorskih naredbi je:

#naredba parametri

Ne završavaju znakom točka-zarez (;)!

Sanda, 2015.

16

INCLUDE

- Jedna od pretprocesorskih naredbi:

#include <naziv_biblioteke>

- U **izvorni kod** programa **uključuje sadržaj biblioteke** koja je navedena unutar znakova < >. Korisniku su na raspolaganju **sve funkcije te biblioteke**.

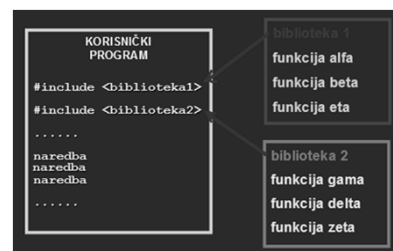
(Isto kao da je prepisan cjelokupni sadržaj odabrane biblioteke.)

Sanda, 2015.

17

INCLUDE

- U programu se može rabiti proizvoljan broj naredaba **include**.



Sanda, 2015.

18

ULAZNI I IZLAZNI TOKOVI

- Da bi program mogao **primati podatke** i **prikazivati rezultate** mora **komunicirati s ulazno/izlaznim uređajima** (tipkovnicom, monitorom, diskom i sl.).
- Ta se komunikacija u C++ ostvaruje pomoću **ulaznih i izlaznih tokova** (engl. *input and output streams*).

Sanda, 2015.

19

#include <iostream>

- **Ulazni** tok prima podatke **od ulaznog uređaja** (s konzole) i **predaje ih programu**, **izlazni** tok prima podatke **od programa** i prosljeđuje ih **izlaznom uređaju** (konzoli).
- Ulazno-izlaznim tokovima rukuje se **funkcijama biblioteke iostream**.
- Dvije najčešće rabljene funkcije su **cout** i **cin**.

Sanda, 2015.

20

ULAZNI I IZLAZNI TOKOVI

- S gledišta programa, sasvim je svejedno koji se ulazni i izlazni uređaji rabe jer se svi podaci u program unose i iz programa predaju okolini na isti način.
- Izlazna konzola je po osnovnoj postavci monitor.
- Ulazna konzola je po osnovnoj postavci tipkovnica.
- Što će biti izlazna i ulazna konzola određuje se naredbama operacijskog sustava.

Sanda, 2015.

21

cout

- Funkcija **cout** zajedno s operatorom ispisa << omogućava **ispis podataka na konzolu** koja je u većini slučajeva zaslon monitora.
- Npr.:

cout<<13;

Posljedica: na zaslonu monitora ispisat će se 13.

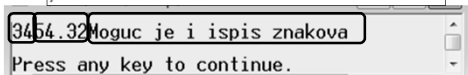
Sanda, 2015.

22

cout

- S desne strane operatora ispisa može biti **varijabla**, **brojeva konstanta** ili **znakovni niz**.

```
int main()
{
    konstanta
    float a;
    a=54.32;
    cout<<34;
    cout<<a;
    cout<<"Moguc je i ispis znakova";
    return 0;
}
```



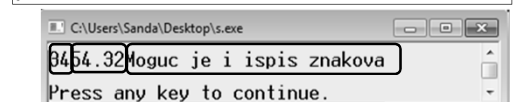
Sanda, 2015.

23

cout

- Moguće je **ulančavanje operatora ispisa** koje se izvodi s lijeva na desno.

```
int main()
{
    float a;
    a=54.32;
    cout<<34<<a<<"Moguc je i ispis znakova";
    return 0;
}
```



Sanda, 2015.

24

ENDL

- Da svi podaci ne bi bili ispisani u istome retku, treba rabiti funkciju (konstantu) **endl** iz biblioteke **iostream**.
- Zadatak joj je **pokazivač** ispisa **pomaknuti na početak sljedećeg retka** na zaslonu.

Sanda, 2015.

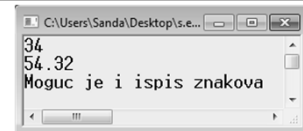
25

ENDL

- Prethodni primjer dopunjen funkcijom (konstantom)

endl:

```
int main()
{
    float a;
    a=54.32;
    cout<<34<<endl;
    cout<<a<<endl;
    cout<<"Moguc je i ispis znakova";
    return 0;
}
```



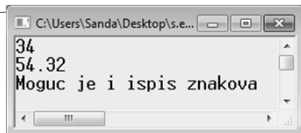
Sanda, 2015.

26

ENDL

- **endl** je moguće dodati i u slučaju **ulančavanja operatora ispisa**. Rezultat je isti:

```
int main()
{
    float a;
    a=54.32;
    cout<<34<<endl<<a<<endl<<"Moguc je i ispis znakova";
    return 0;
}
```



Sanda, 2015.

27

ULANČAVANJE OPERATORA <<

- Ako se pri ulančavanju operatora ispisa **predugi redak želi prelomiti**, to je potrebno učiniti **ispred** operatora ispisa. Npr:

cout<<34<<a<<" Moguc je i ispis znakova";

- može se prelomiti i zapisati ovako:

cout<<34<<a

<<" Moguc je i ispis znakova";

Sanda, 2015.

28

CIN

- Funkcija **cin** zajedno s operatorom unosa **>>** omogućava **unos podataka s konzole** što je u većini slučajeva tipkovnica.

Npr.:

cin>>a;

Posljedica: očekuje se unos podatka s tipkovnice, podatak će biti pohranjen u varijablu **a**.

Sanda, 2015.

29

CIN

```
int main()
{
    int a;
    float b;
    cout<<"Upisi cijeli broj:";
    cin>>a;
    cout<<"Upisi realni broj:";
    cin>>b;
    return 0;
}
```



- **Izvršenje nastavka programa** se prekida sve **dok se ne unese podatak** i pritisne tipka **Enter!**

Sanda, 2015.

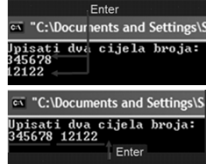
30

ULANČAVANJE OPERATORA >>

- Moгуće je **ulančavanje operatora unosa** koje se izvodi s lijeva na desno.

```
cout<<"Upisi dva cijela broja:"<<endl;
cin>>a>>b;
cout<<endl<<a<<endl;
```

- Ulazni se podaci mogu unijeti:
 - jedan ispod drugog**, a **nakon svakog** unosa **Enter** ili
 - u istome retku**, odvojeni **prazninama**, a **nakon posljednjeg** unosa **Enter**.



Sanda, 2015.

31

GREŠKE PRI UNOSU POGREŠNOG TIP PODATKA

- U primjeru se unose pa ispisuju **dva cijela broja**.

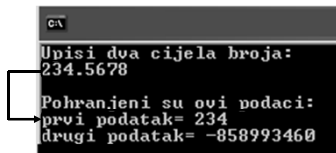
```
int main()
{
    int a,b;
    cout<<"Upisi dva cijela broja:"<<endl;
    cin>>a;
    cin>>b;
    cout<<endl<<"Pohranjeni su ovi podaci:"<<endl;
    cout<<"prvi podatak= "<<a<<endl;
    cout<<"drugi podatak= "<<b<<endl;
    return 0;
}
```

Sanda, 2015.

32

GREŠKE PRI UNOSU POGREŠNOG TIP PODATKA

- Umjesto **prvog** cijelog broja unesen je **realan broj**.
- To **prekida program** (daljnji unos).
- Cijeli dio** realnog broja je **pohranjen kao prvi broj**, a zbog decimalnog dijela broja nastaje **pogreška**.

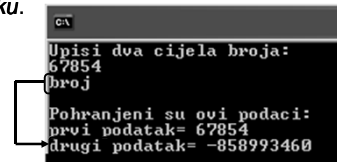


Sanda, 2015.

33

GREŠKE PRI UNOSU POGREŠNOG TIP PODATKA

- Umjesto **drugog** cijelog broja unesen je **znakovni niz "broj"**.
- Prvi** je podatak pohranjen **ispravno**, a **drugi** je izazvao **pogrešku**.

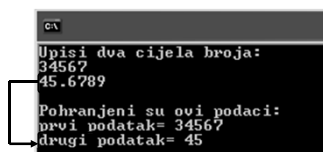


Sanda, 2015.

34

GREŠKE PRI UNOSU POGREŠNOG TIP PODATKA

- Umjesto **drugog** cijelog broja unesen je **realni broj**.
- Prvi** je podatak pohranjen **ispravno**, a kod **drugog** je **cijeli dio realnog broja pohranjen** kao cijeli broj (decimalni dio broja je odbačen).

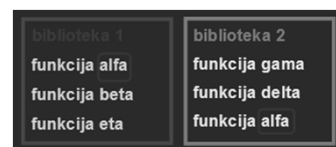


Sanda, 2015.

35

NAMESPACE

- Korisnik može u istom programu rabiti **više biblioteka**.
- Autori tih biblioteka mogu biti različiti i može se dogoditi da npr. koriste **ista imena za funkcije** koje imaju **različitu namjenu**.



Sanda, 2015.

36

NAMESPACE

- Što će se dogoditi ako korisnik **u istom programu** rabi **obje biblioteke** iz primjera, a pozove funkciju **alfa**?
- Koja će funkcija **alfa** biti izvršena?
Ona iz **biblioteke1**?
Ona iz **biblioteke2**?



Sanda, 2015.

37

NAMESPACE

- Da bi se izbjegla zabuna, funkcije pojedine biblioteke **dodatno se označavaju** ključnom riječju **namespace** uz zadavanje **dodatnog imena**:

namespace dodatno_ime

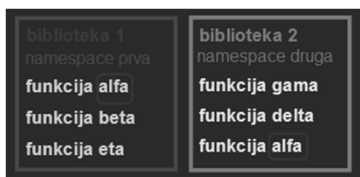
- Zadatak označavanja je **jednoznačno odrediti funkcije neke biblioteke**.

Sanda, 2015.

38

NAMESPACE

- U primjeru se funkcije **biblioteke1** označavaju dodatnim imenom **prva**, a funkcije **biblioteke2** dodatnim imenom **druga**.



Sanda, 2015.

39

NAMESPACE

- Da bi se funkcije označene dodatnim imenom **učinile aktivnima** koristi se:

- operator dodjele područja** (dvije dvotočke ::)

ili

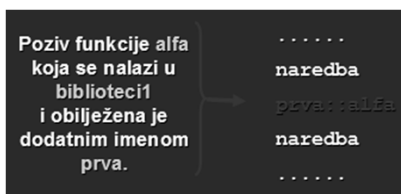
- naredba **using**.

Sanda, 2015.

40

OPERATOR DODJELE PODRUČJA

- Pri pozivanju** funkcije se **uz ime funkcije** navodi i njeno **dodatno ime**. Npr.:



Sanda, 2015.

41

OPERATOR DODJELE PODRUČJA

- Kada se u program želi uključiti **više funkcija**, a koristi se aktivacija operatorom dodjele područja, **pri pozivu svake** od funkcija **uz njeno ime** treba navesti i **njeno dodatno ime**.



Sanda, 2015.

42

KLJUČNA RIJEČ USING

- Ako se želi **sve funkcije** označene dodatnim imenom **odjednom** učiniti **aktivnima** koristi se naredba:

using namespace dodatno_ime;

Sanda, 2015.

43

KLJUČNA RIJEČ USING

- Ako se uporabi naredba **using namespace prva**, funkcije biblioteke označene dodatnim imenom **prva** pozivaju se **samo njihovim imenom**.

```
KORISNIČKI
PROGRAM

#include <biblioteka1>
#include <biblioteka2>
using namespace prva;
.....
naredba
alfa
eta
druga::alfa
naredba
beta
.....
```

Sanda, 2015.

44

STANDARDNE FUNKCIJE - STD

- U programu C++ postoji **skupina standardnih funkcija** naziva: **standardna biblioteka** (engl. standard library).
- Funkcije standardne biblioteke pohranjene su u **različitim bibliotekama**, ali su **sve** po dogovoru označene istim **dodatnim imenom std**.
- Da bi se standardne funkcije mogle pozivati samo njihovim imenima, na početku programa valja napisati:
using namespace std;

Sanda, 2015.

45

STANDARD LIBRARY

<algorithm>	<bitset>	<cassert>	<ctype>	<cerrno>	<cfloat>
<ciso646>	<climits>	<locale>	<cmath>	<complex>	<ctime>
<csignal>	<csignal>	<csignal>	<csignal>	<csignal>	<csignal>
<ctime>	<cwchar>	<cwctype>	<deque>	<exception>	<fstream>
<functional>	<hash_map>	<hash_set>	<iomanip>	<iostream>	<iostream>
<iostream>	<iostream>	<iostream>	<iostream>	<iostream>	<iostream>
<map>	<memory>	<new>	<numeric>	<ostream>	<queue>
<set>	<sstream>	<stack>	<stdexcept>	<streambuf>	<string>
<stringstream>	<utility>	<valarray>	<vector>		

Sanda, 2015.

46

STANDARDNE FUNKCIJE - STD

- Moguće ih je pozvati na jedan od dva prethodno navedena načina:

```
1 #include <iostream>
2 int main()
3 {
4     std::cout<<"PROBA"<<std::endl;
5     return 0;
6 }

#include <iostream>
using namespace std;
int main()
{
    cout<<"PROBA"<<endl;
    return 0;
}
```

Sanda, 2015.

47

OSNOVNA STRUKTURA PROGRAMA

```
#include <iostream>
using namespace std;
int main()
{
    return 0;
}
```

Sanda, 2015.

48