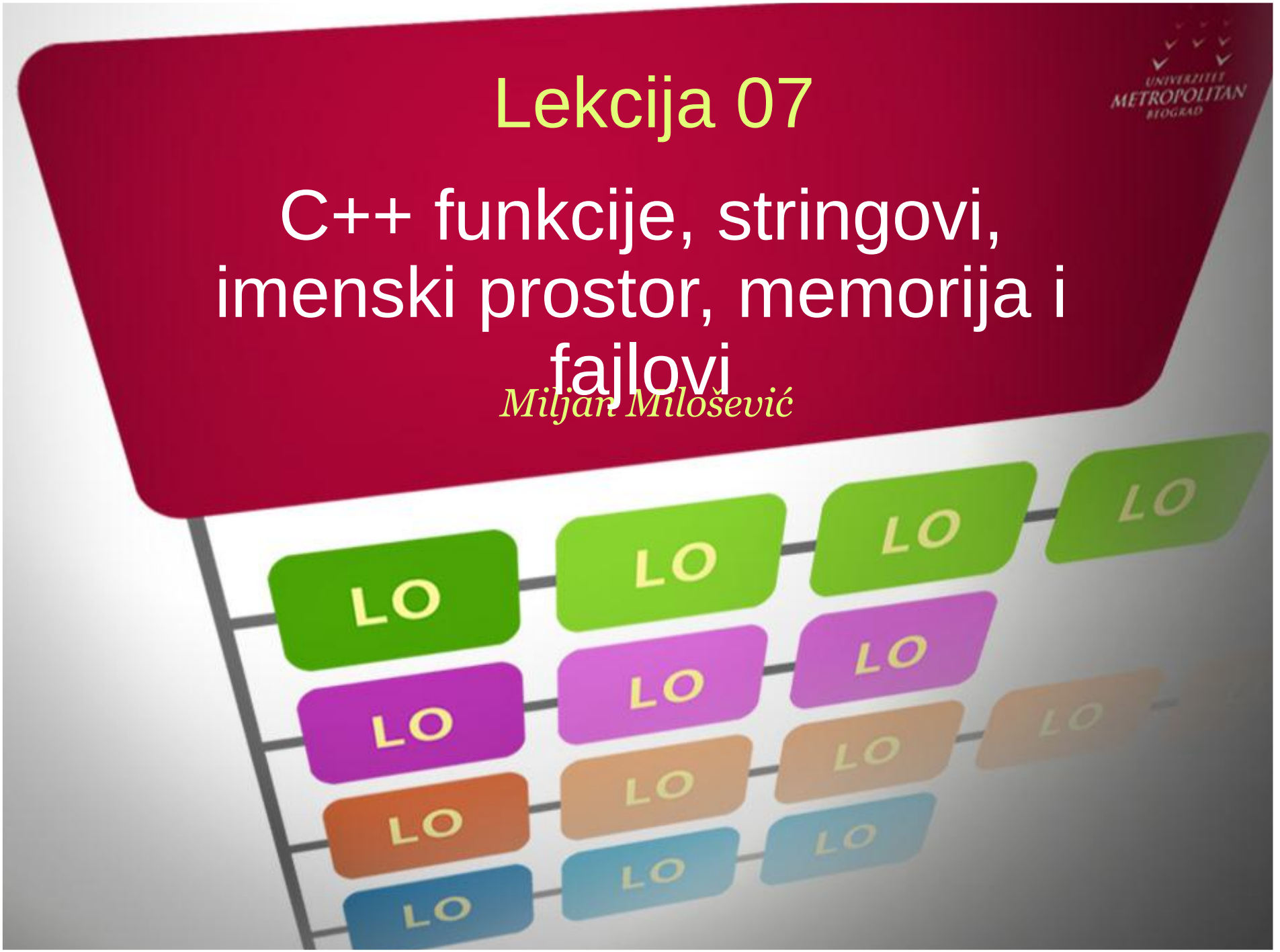
Lekcija 07

C++ funkcije, stringovi, imenski prostor, memorija i fajlovi

Miljan Milošević



C++ FUNKCIJE, STRINGOVI, IMENSKI PROSTOR, UPRAVLJANJE MEMORIJOM I FAJLOVI

	01	02	03	04
Uvod	**Reference**	**Funkcije u C++-u**	**Dodatne mogućnosti funkcija u C++-u**	**C++ string klasa**
	➢ *Uvod u reference* ➢ *Upotreba referenci u C++ programu*	➢ *Poziv funkcije po vrednosti* ➢ *Poziv funkcije po referenci* ➢ *Konstanta kao argument funkcije* ➢ *Rezultat funkcije je referenca* ➢ *Uporedni primeri pokazivača i referenci* ➢ *Poređenje pokazivača i referenci*	*1.Linijske funkcije* *2.Funkcije sa podrazumevajućim vrednostima* *3.Uvod u preklapanje funkcija*	➢ *String klasa u C++-u* ➢ *Upotreba funkcija za rad sa string klasom* ➢ *Učitavanje stringova pomoću funkcije getline(...)*

C++ FUNKCIJE, STRINGOVI, IMENSKI PROSTOR, UPRAVLJANJE MEMORIJOM I FAJLOVI

05

C++ string metode za rad sa tekstom

- *Manipulisanje tekstom (string metode)*
- *Upotreba metoda za manipulisanje tekstom*
- *Upotreba insert, erase i replace metoda za manipulisanje tekstom*

06

Upravljanje memorijom u C++-u

- *Funkcije za dinamičko upravljanje memorijom*
- *Dinamičko alociranje, nizovi i funkcije*

07

Imenski prostor

- *Problemi kod preklapanja naziva funkcija*
- *Uvod u imenski prostor*
- *Ključna reč "using"*
- *Razdvojeni delovi imenskog prostora*
- *Ugnježđeni imenski prostor*
- *Prostor imena standardne biblioteke*
- *Anonimni imenski prostor*

08

Rad sa fajlovima u C++-u

- *C++ tokovi*
- *Otvaranje i zatvaranje datoteke*
- *Upisivanje i čitanje*
- *Pozicioni pokazivač datoteke*
- *Objekti za rad sa fajlovima kao argumenti funkcije*
- *Korišćenje konstruktora objekata za rad sa fajlovima*
- *Dodatak - Izvod funkcija za rad sa C++ fajlovima*

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa raznim bitnim pojmovima programskog jezika C++:

- ❑ C++ funkcije
- ❑ C++ string klasa
- ❑ Imenski prostor
- ❑ Upravljanje memorijom
- ❑ Rad sa C++ fajlovima

C++ uvodi reference kao nov pojam u odnosu na programski jezik C. Referencna promenljiva je pseudonim, alias, što ustvari predstavlja drugi naziv za postojeću promenljivu. Osim poziva funkcija po vrednosti i adresi, C++ omogućava poziv funkcije po referenci što će biti opisano u okviru ove lekcije. Takođe, referenca može biti rezultat izvršavanja funkcije.

Programski jezik C++ ima dodatne mogućnosti kada je u pitanju rad sa funkcijama, kao što su: preklapanje funkcija, podrazumevajuće vrednosti funkcija i linijske funkcije.

Nekada je moguće da se desi, pri radu više programera na jednom projektu, da se u projektu nadje više funkcija istog imena što će dovesti do kompajlerske greške. C++ omogućava definisanje i korišćenje imenovanih oblasti u kojima se grupišu funkcije sličnih karakteristika (slično paketima u Java-i) čime se olakšava rad na složenijim projektima na kojima učestvuje veliki broj programera.

Kada je u pitanju rad sa tekstualnim tipovima podataka, programski jezik C++ uvodi novi tip podatka, tj. string, koji se često u literaturi pominje kao C++ string klasa. C++ string klasa se u većini slučajeva koristi umesto C stringova zbog jednostavnije upotrebe. U okviru dela lekcije o stringovima biće opisane funkcije koje u mnogome olakšavaju rad sa tekstualnim podacima.

Na kraju lekcije će biti opisani C++ objekti za rad sa fajlovima (datotekama). Funkcije za rad sa fajlovima su smeštene u standardnoj biblioteci fstream, tako da će biti dat njihov opis i način upotrebe, bilo da je u pitanju rad sa tekstualnim ili binarnim fajlovima.

Reference

Reference, inicijalizacija i deklaracija, reference i pokazivači

- *Uvod u reference*
- *Upotreba referenci u C++ programu*

01

UVOD U REFERENCE

Referencna promenljiva je pseudonim, alias, tj. drugi naziv za postojeću promenljivu. Jednom inicijalizovana referenca ne može biti promenjena da referencira drugu promenljivu

Referencna promenljiva je pseudonim, alias, što ustvari predstavlja drugi naziv za postojeću promenljivu. Nakon što je referenca jednom inicijalizovana, moguće je pristupiti promenljivoj ili preko njenog naziva ili korišćenjem reference.

❑ **Poređenje pokazivača i referenci:**

Reference se često mešaju sa pokazivačima, međutim postoje tri glavne razlike između pokazivača i referenci, a to su:

- U programu ne može da postoji NULL referenca. Uvek mora da važi pretpostavka da je referenca povezana sa legitimnim tj. aktivnim delom memorije odnosno skladišta (storage).
- Jednom inicijalizovana referenca ne može biti promenjena da referencira drugu promenljivu ili objekat. Suprotno referencama, pokazivači mogu da budu promenljeni i da u različitim vremenskim trenucima pokazuju na različite promenljive odnosno objekte.
- Referenca mora biti inicijalizovana prilikom kreiranja, za razliku od pokazivača koji mogu biti inicijalizovani u bilo kom trenutku.

❑ **Kreiranje referenci u C++-u**

O imenu promenljive možemo razmišljati kao o labeli koja je se odnosi na lokaciju promenljive u memoriji. Ako krenemo korak dalje, o referencama možemo razmišljati kao o drugoj labeli koja se odnosi na tu memorijsku lokaciju. Stoga je moguće pristupiti sadržaju promenljive ili preko naziva promenljive ili preko reference. Pretpostavimo da imamo sledeće promenljive u programu:

```
int    i = 17;
double d;
```

Možemo zatim, na osnovu definisanog pravila, deklarisati referencu za promenljivu na sledeći način.

```
int&    r = i;
double& s = d;
```

Oznaka & (ampersend) se u C++-u čita kao referenca. Stoga, prvu deklaraciju možemo pročitati kao "r je celobrojna referenca promenljive i" a drugu kao "s je realna double referenca promenljive d".

UPOTREBA REFERENCI U C++ PROGRAMU

Referenca mora biti inicijalizovana prilikom kreiranja. Za deklaraciju reference se koristi operator & (ampersand) koji sledi za imenom tipa ili prethodi nazivu promenljive

U nastavku je dat ceo primer korišćenja reference na celobrojne i realne promenljive:

```
#include <iostream>

using namespace std;

int main ()
{
   // declare simple variables
   int    i;
   double d;

   // declare reference variables
   int&    r = i;
   double& s = d;

   i = 5;
   cout << "Value of i : " << i << endl;
   cout << "Value of i reference : " << r  << endl;

   d = 11.7;
   cout << "Value of d : " << d << endl;
   cout << "Value of d reference : " << s  << endl;

   return 0;
}
```

Rezultat programa je:

```
Value of i : 5
Value of i reference : 5
Value of d : 11.7
Value of d reference : 11.7
```

Reference se obično koriste kao argumenti funkcije ili kao povratne vrednosti (rezultati) funkcije, što će detaljnije biti opisano u nastavku

Funkcije u C++-u

Funkcije, referenca kao argument f-je, referenca kao rezultat f-je

- *Poziv funkcije po vrednosti*
- *Poziv funkcije po referenci*
- *Konstanta kao argument funkcije*
- *Rezultat funkcije je referenca*
- *Uporedni primeri pokazivača i referenci*
- *Poređenje pokazivača i referenci*

02

POZIV FUNKCIJE PO VREDNOSTI

U programskom jeziku C++ je neophodno da se deklaracija funkcije navede pre bloka u kome se funkcija poziva, u slučaju da je funkcija definisana ispod bloka u kome se poziva

Evo jednog primera sa prototipovima funkcija, gde imamo glavnu funkciju i dve „obične“ funkcije, i obe su date prvo preko svojih deklaracija tj. prototipova, a posle glavne funkcije date su definicije ovih funkcija:

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

float calcXc(float, float, float, float);
float calcYc(float, float, float);

int main()
{
 float gLine1, gLine2, YOLine1, YOLine2, Xc, Yc;
 char stopchar;
 cout<<"Input gradient and Y axis value for first line"<<endl;
 cin>>gLine1>>YOLine1;
 cout<<"Input gradient and Y axis value for second line"<<endl;
 cin>>gLine2>>YOLine2;
 Xc =calcXc(gLine1,YOLine1,gLine2,YOLine2);
 Yc =calcYc(Xc,gLine1,YOLine1);
 cout<< "The coordinates " << Xc<< ", " << Yc << endl << "press a key " ;
 cin >> stopchar;
 return 0;
}

float calcXc(float grad1,float Ycept1,float grad2, float Ycept2)
{
 float Xc;
 Xc=(Ycept2-Ycept1)/(grad1-grad2);
 return Xc;
}
```

POZIV FUNKCIJE PO REFERENCI

Pozivanjem po referenci vrši se kopiranje reference stvarnog argumenta u formalni parametar funkcije, a promene izvršene nad formalnim parametrom preslikavaju se i na stvarni argument

Pri pozivu funkcije u programskom jeziku C++ osim poziva po vrednosti i adresi, postoji i dodatan način prosleđivanja parametara a to je prosleđivanje po referenci. Pozivom funkcije po referenci vrši se kopiranje reference stvarnog argumenta u formalni parametar funkcije. Unutar funkcije, referenca se koristi da bi se prostupilo stvarnom parametru koji je prosleđen funkciji. Ovo znači da se promene izvršene nad fiktivnim parametrom preslikavaju i na stvarni argument. Prosleđivanje po referenci se vrši na isti način kao i kod ostalih vrednosti. Stoga je neophodno deklarisati funkciju tako da parametri funkcije budu reference. U nastavku je dat izmenjen primer funkcije swap(), koja zamenjuje dve celobrojne vrednosti korišćenjem referenci.

```
// function definition to swap the values.
void swap(int &x, int &y)
{
   int temp;
   temp = x; /* save the value at address x */
   x = y;    /* put y into x */
   y = temp; /* put x into y */
   return;
}
```

Deklaracija funkcije će sada imati izmenjen oblik:

```
// function declaration
void swap(int &x, int &y);
```

dok će poziv funkcije biti isti kao kod poziva po vrednosti:

```
/* calling a function to swap the values using variable reference.*/
   swap(a, b);
```

Rezultat izvršavanja prethodnog koda će biti isti kao kod poziva po adresi.

KONSTANTA KAO ARGUMENT FUNKCIJE

Ukoliko očekujemo da se u funkciji ne treba promeniti vrednost argumenta ali ne želimo da prosledimo argument po vrednosti, najbolji način je da koristimo konstantnu referencu

Jedna od glavnih mana poziva po vrednosti je ta što se svi argumenti kopiraju u formalne parametre funkcije. Ovo može oduzeti dosta vremena i memorije kada su argumenti složeni tipovi podataka (klase, strukture, itd; o klasama će biti više reči u narednim lekcijama). Reference obezbeđuju da se izbegne ovaj problem. Kada se neki argument prosledi po referenci, kreira se referenca za stvarni parametar (što oduzima minimalno vreme) a ne kopira se cela vrednost. Na ovaj način je moguće proslediti složene tipove funkcije bez mnogo utrošaka (vremenskih i memorijskih). Međutim, i ovo otvara neke potencijalne probleme. Reference dozvoljavaju funkciji da promeni vrednost argumenta što nije poželjno u mnogo slučajeva. Ukoliko očekujemo da se u funkciji ne treba promeniti vrednost argumenta ali ne želimo da prosledimo argument po vrednosti, najbolji način je da koristimo konstantnu referencu (const reference).

Već nam je poznato da reference označene kao const ne dozvoljavaju izmenu promenljive koju referenciraju. Stoga, ukoliko koristimo const referencu kao parametar, garantujemo da funkcija neće (i ne može) promeniti argument. Naredna funkcija će da proizvede kompajlersku grešku:

```
void foo(const int &x)
{
    x = 6;  // x is a const reference and can not be changed!
}
```

REZULTAT FUNKCIJE JE REFERENCA

Kada je promenljiva vraćena po referenci, referenca na promenljivu se prosleđuje pozivaocu. Pomoću ovog metoda ne mogu se vratiti lokalne promenljive definisane unutar funkcije

Kao i kod poziva po referenci, i u povratku po referenci povratna vrednost mora biti promenljiva (ne sme se vratiti referenca na literal ili izraz). Kada je promenljiva vraćena po referenci, referenca na promenljivu se prosleđuje pozivaocu. Pozivaoc funkcije može zatim koristiti referencu da bi nastavio sa modifikovanjem promenljive, što može biti od koristi u raznim slučajevima. Povratak po referenci je takođe brz proces i može biti od koristi pri radu sa strukturama i klasama. Međutim, vraćanje po referenci ima jedan nedostatak a to je da se pomoću nje ne mogu vratiti lokalne promenljive definisane unutar funkcije. Razmotrimo sledeći primer:

```
int& DoubleValue(int nX)
{
    int nValue = nX * 2;
    return nValue; // return a reference to nValue here
} // nValue goes out of scope here
```

Da li primećujete problem u prethodnom kodu? Funkcija pokušava da vrati referencu na vrednost nValue koja će izaći iz opsega nakon povratka iz funkcije. Ovo bi značilo da pozivaoc dobija referencu na promenljivu koja će biti u korpi za otpatke (garbage). Na sreću, kompajler će tretirati kao grešku ukoliko pokušate da uradite ovo.

UPOREDNI PRIMERI POKAZIVAČA I REFERENCI

Prosleđivanje vrednosti po referenci je često mnogo lakši način u odnosu na prosleđivanje po adresi (pokazivaču)

Prosleđivanje vrednosti po referenci je često mnogo lakši način u odnosu na prosleđivanje po pokazivaču. Da bi smo razumeli razlike u ova dva metoda posmatraćemo sledeći primer koji korišćenjem funkcije udvostručuju vrednost neke promenljive. Program gde se promenljiva prosleđuje funkciji po referenci bi imao sledeću sintaksu:

```
#include <iostream>
using namespace std;
void doubleIt(int&);

int main ()
{
   int num;
   cout << "Enter number: ";
   cin >> num;
   doubleIt(num);
   cout << "The number doubled in main is " << num
<< endl;
   return 0;
}
void doubleIt (int& x)
{
   cout << "The number to be doubled is " << x <<
endl;
   x *= 2;
   cout << "The number doubled in doubleIt is " << x
<< endl;
}
```

```
Enter number: 3
The number to be doubled is 3
The number doubled in doubleIt is 6
The number doubled in main is 6
```

Modifikujmo sada program tako da se prosleđivanje promenljive vrši po adresi umesto po referenci:

```
#include <iostream>
using namespace std;
void doubleIt(int*);

int main ()
{
   int num;
   cout << "Enter number: ";
   cin >> num;
   doubleIt(&num);
   cout << "The number doubled in main is " << num
<< endl;
   return 0;
}
void doubleIt (int* x)
{
   cout << "The number to be doubled is " << *x <<
endl;
   *x *= 2;
   cout << "The number doubled in doubleIt is " <<
*x << endl;
}
```

Na osnovu prethodna dva primera možemo da izvedemo sledeći zaključak o sličnostima i razlikama pokazivača i referenci.

POREĐENJE POKAZIVAČA I REFERENCI

Postoje četiri bitne razlike u sintaksi između korišćenja pokazivača odnosno referenci kao argumenata funkcije

Kao što se može primetiti, postoje četiri bitne razlike u sintaksi između prethodna dva programa:

1. Kod prototipa funkcije, koristi se adresni operator (ampersand - &) za prosleđivanje po referenci a indirektni operator (asterisk - *) za prosleđivanje po adresi:

```
void doubleIt(int&);  // by reference
void doubleIt(int*);  // by address
```

2. Slično prethodnom, u zaglavlju funkcije, takođe se adresni operator (&) koristi za prosleđivanje po referenci a indirektni operator (*) za prosleđivanje po adresi

```
void doubleIt (int& x);   // by reference
void doubleIt (int* x);   // by address
```

3. Kada pozivate funkcije, ne treba da navedete adresni operator (&) za prosleđivanje po referenci, ali je potrebno navesti adresni operator da bi ste prosledili adresu promenljive x:

```
doubleIt(num);   // by reference
doubleIt(&num);  // by address
```

4. U telu funkcije, ne treba da dereferencirate argument kod prosleđivanja po referenci, ali je neophodno da dereferencirate argument kada ga prosledite po adresi jer promenljiva x, koju ste prosledili po adresi, nije vrednost već pokazivač:

```
// by reference - no dereference
{
   cout << "The number to be doubled is " << x << endl;
   x *= 2;
   cout << "The number doubled in doubleIt is " << x <<
endl;
}
// by address - need to dereference
{
   cout << "The number to be doubled is " << *x << endl;
   *x *= 2;
   cout << "The number doubled in doubleIt is " << *x <<
endl;
}
```

Kao što se može primetiti iz prethodnog primera, korišćenje referenci umesto pokazivača je mnogo lakši i jednostavniji način. Međutim, postoje slučajevi gde je neophodno da prosledite pokazivače funkciji jer je to, kao kod dinamičkog alociranja memorije - jedini mogući način da upravljate memorijom.

Dodatne mogućnosti funkcija u C++-u

C++, inline funkcije, podrazumevajuća vrednost, overloading

1.Linijske funkcije

2.Funkcije sa podrazumevajućim vrednostima

3.Uvod u preklapanje funkcija

03

UVODNA RAZMATRANJA

Programski jezik C++ ima kao dodatak nekoliko mogućnosti koje nema jezik C. Neke od osnovnih dodataka koje će biti razmatrane u okviru ove sekcije su:

- Linijske funkcije
- Funkcije sa podrazumevajućim vrednostima
- Preklapanje funkcija

Linijske funkcije

Funkcija, C++, inline

- *Osnovi o linijskim (inline) funkcijama*

03

OSNOVI O LINIJSKIM (INLINE) FUNKCIJAMA

Obično se inline koristi kod kratkih funkcija (sa malo koda, ne više od nekoliko linija), koje se pozivaju unutar petlji koje se izvršavaju mnogo puta

Pozivanje funkcije zahteva dodatno vreme i dodatni memorijski prostor. Ponekad se ovo može izbeći ako se funkcija definiše kao „linijska“ funkcija. Ovo ukazuje kompajleru da zameni svaki poziv funkcije sa samim kodom funkcije, umesto da je poziva. Sa gledišta programera, jedina razlika je u tome što se koristi specifikator „inline“. Uzmimo u obzir sledeći deo koda:

```
int min(int nX, int nY)
{
    return nX > nY ? nY : nX;
}

int main()
{
    using namespace std;
    cout << min(5, 6) << endl;
    cout << min(3, 2) << endl;
    return 0;
}
```

Ovaj program poziva funkciju min() dva puta. S obzirom da je funkcija min() kratka, ona je idealan kandidat da postane inline ili linijska funkcija:

```
inline int min(int nX, int nY)
{
    return nX > nY ? nY : nX;
}
```

Sada, kada kompajler kompajira main program, on će prevođenje izvršiti na isti način kao za main program koji je napisan u nastavku:

```
int main()
{
    using namespace std;
    cout << (5 > 6 ? 6 : 5) << endl;
    cout << (3 > 2 ? 2 : 3) << endl;
    return 0;
}
```

Obično se inline koristi kod kratkih funkcija (sa malo koda, ne više od nekoliko linija), koje se pozivaju unutar petlji koje se izvršavaju mnogo puta. Takođe treba napomenuti da je korišćenje ključne reči samo preporuka – kompajler je taj koji vodi glavnu reč i koji može da ignoriše inline ukoliko naiđe na funkciju sa dugačkim kodom.

Funkcije sa podrazumevajućim vrednostima

funkcija, C++, podrazumevajuća vrednost

- *Osnovi o funkcijama sa podrazumevajućim vrednostima*
- *Upotreba funkcija sa podrazumevajućim vrednostima*

03

OSNOVI O FUNKCIJAMA SA PODRAZUMEVAJUĆIM VREDNOSTIMA

Podrazumevajuće vrednosti se specificiraju korišćenjem operatora jednako i dodeljivanjem vrednosti argumentima pri definisanju funkcije

Pri definisanju funkcija, moguće je postaviti podrazumevane vrednosti za svaki od parametara funkcije. Ove vrednosti će biti korišćene ukoliko su neki od argumenata izostavljeni prilikom poziva funkcije.

Podrazumevajuće vrednosti se specificiraju korišćenjem operatora jednako i dodeljivanjem vrednosti argumentima pri definisanju funkcije. U slučaju da vrednost argumenta nije prosleđena pri pozivu funkcije, podrazumevajuća vrednost će biti korišćena, u suportnom će podrazumevana vrednost biti ignorisana a prednost će imati vrednost prosleđenog argumenta. Ako funkcija f() treba da ima jedinstveni celobrojni parametar kome će podrazumevajuća vrednost biti 10, onda se ona može opisati na sledeći način:

```
void f(int i = 10)  {…}
```

Ovako napisana funkcija može biti pozvana na dva načina:

```
f(1); // parametar i dobija vrednost 1
f();  // parametar i dobija podrazumevajuću vrednost 10
```

Na levoj strani se nalazi primer korišćenja podrazumevajućih vrednosti.

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

int sum(int a, int b=20)
{
  int result;
  result = a + b;
  return (result);
}

int main ()
{
   int a = 100;
   int b = 200;
   int result;

   result = sum(a, b);
   cout << "Total value is :" << result << endl;

   result = sum(a);
   cout << "Total value is :" << result << endl;

   return 0;
}
```

Nakon izvršenja programa dobiće se sledeći rezultat:

```
Total value is :300
Total value is :120
```

UPOTREBA FUNKCIJA SA PODRAZUMEVAJUĆIM VREDNOSTIMA

U slučaju da vrednost argumenta nije prosleđena pri pozivu funkcije, podrazumevajuća vrednost će biti korišćena. U suportnom će prednost imati vrednost prosleđenog argumenta

Pogledajmo još jedan primer:

```
void PrintValues(int nValue1, int nValue2=10)
{
    using namespace std;
    cout << "1st value: " << nValue1 << endl;
    cout << "2nd value: " << nValue2 << endl;
}

int main()
{
    PrintValues(1); // nValue2 will use default parameter of 10
    PrintValues(3, 4); // override default value for nValue2
}
```

Program će proizvesti sledeći izlaz:

```
1st value: 1
2nd value: 10
1st value: 3
2nd value: 4
```

Pri prvom pozivu funkcije, pozivaoc nije specificirao argument za parametar nValue2, pa će funkcija koristiti podrazumevajuću vrednost 10. Kod drugog poziva, oba argumenta su prosleđena funkciji pa će se ti argumenti proslediti parametrima funkcije, dok će podrazumevajuće vrednosti biti ignorisane.

Uvod u preklapanje funkcija

funkcija, C++, preklapanje, overload

- *Osnovi o preklapanju funkcija*

03

OSNOVI O PREKLAPANJU FUNKCIJA

Preklapanje funkcija je deo jezika C++ koji omogućava kreiranje više funkcija sa istim imenom ali sa različitim parametrima.

Neka je definisana funkcija Add koja sabira dva cela broja na sledeći način:

```
int Add(int nX, int nY)
{
    return nX + nY;
}
```

U C++-u možemo da deklarišemo drugu funkciju Add() koja preuzima realne brojeve kao parametre:

```
double Add(double dX, double dY)
{
    return dX + dY;
}
```

Stoga, sada imamo dve verzije funkcije Add():

```
int Add(int nX, int nY); // integer version
double Add(double dX, double dY); // floating point version
```

Koju verziju funkcije Add() ćemo da pozovemo zavisi od argumenata koje prosledimo funkciji — ukoliko prosledimo dva cela broja, C++ će znati da želimo da pozovemo funkciju Add(int, int). Ukoliko prosledimo dva realna broja biće pozvana funkcija Add(double, double). U stvari, mi možemo definisati onoliko različitih preklopljenih funkcija Add() tako da svaka ima različite parametre. Takođe je moguće definisati funkciju Add() sa različitim brojem parametara:

```
int Add(int nX, int nY, int nZ)
{
    return nX + nY + nZ;
}
```

C++ string klasa

C++, string,deklaracija,getline

- *String klasa u C++-u*
- *Upotreba funkcija za rad sa string klasom*
- *Učitavanje stringova pomoću funkcije getline(...)*

04

STRING KLASA U C++-U

U okviru standardne C++ biblioteke postoji gotova string klasa za rad sa tekstualnim podacima

U okviru standardne C++ biblioteke postoji već gotova klasa string koja obezbeđuje potpunu funkcionalnost za rad sa tekstualnim podacima. Klase još uvek nisu uvedene u okviru ovog C++ kursa i biće detaljnije opisane u nekom od narednih predavanja, ali je prepostavka da su studenti u okviru kursa o Javi naučili osnove o klasama i objektima.
Da bi smo mogli koristiti promenljive klase string, neophodno je da uključimo standardnu C++ biblioteku korišćenjem pretprocesorske naredbe na početku programa:

```
#include <string>
```

U listingu na desnoj strani je dat jedan prost primer korišćenja promenljivih klase string. Kao što možemo videti, inicijalizacija promenljive str1 i dodeljivanje vrednosti „Hello“ se obavlja na sledeći način:

```
string str1 = "Hello";
```

Klasa string podržava operator sabiranja + u cilju spajanja dva stringa. U cilju određivanja dužine stringa moguće je koristiti metodu size(). Kao rezultat programa dobija se:

```
str3 : Hello
str1 + str2 : HelloWorld
str3.size() :  10
```

```
#include <iostream>
#include <string>

using namespace std;

int main ()
{
   string str1 = "Hello";
   string str2 = "World";
   string str3;
   int  len ;

   // copy str1 into str3
   str3 = str1;
   cout << "str3 : " << str3 << endl;

   // concatenates str1 and str2
   str3 = str1 + str2;
   cout << "str1 + str2 : " << str3 << endl;

   // total lenghth of str3 after concatenation
   len = str3.size();
   cout << "str3.size() :  " << len << endl;

   return 0;
}
```

UPOTREBA FUNKCIJA ZA RAD SA STRING KLASOM

Da bi smo koristiti promenljive klase string neophodno je da uključimo standardnu C++ biblioteku korišćenjem pretprocesorske direktive: #include <string>

Sada ćemo detaljno opisati osnovne funkcije i mehanizam koji se koristi pri radu sa string tipom podataka. Jedna tekstualna promenljiva može se inicijalizovati korišćenjem operatora dodele = (ili kod deklaracije, ili kasnije), a takođe je moguće i kod deklaracije pomoću zagrada posle imena tekstualne promenljive. Primer:

```
#include <string>
#include <iostream>
using namespace std;
int main()
{
          string str1;                                  //declare str1 by using the string keyword
          string str2("inicijalizacija 1");             //str2 deklarisan i inicijalizovan
          string str3 = "inicijalizacija 2";            //str3 deklarisan i inicijalizovan
          str1 = "inicijalizacija 3";                             //str1 inicijalizovan
          cout << str1<<endl;
          cout << str2<<endl;
          cout << str3<< endl;

          return 0;
}
```

C++ klasa string obuhvata niz metoda za rad sa promeljivama tipa string tj. string objektima, npr. metodu size() ili compare(), a koje se koriste pomoću tačka operatora, npr. stringname.size() ili str1.compare(str2), itd. Klasa string tretira string promenljive kao string objekte i takođe omogućuje upotrebu operatora =, +, <, >, == i !=.

UČITAVANJE STRINGOVA POMOĆU FUNKCIJE GETLINE(...)

Funkcija getline() obezbeđuje pouzdano učitavanje stringova sa korisničkog ulaza

Iako se operator ">>" u programskom jeziku C++ može koristiti za unos stringova sa standardnog ulaza (tastature), njegovo korišćenje je limitirano zbog načina na koji radi sa prazninama (space karakterima). Analizirajmo sledeći segment koda koji vrši učitavanje stringova:

```
...
cout << "Enter name: ";
cin >> a_string;
...
```

Pretpostavimo da je prilikom unosa ukucan sledeći podatak (Rob Miller):

```
...
Enter name: Rob Miller
...
```

Nakon unosa, string promenljiva a_string će imati vrednost "Rob", jer operator ">>" radi pod pretpostavkom da uneta praznina označava kraj stringa. Stoga je često bolje koristiti funkciju "getline(...)" koja ima dva argumenta. Na primer, ako umesto prethodne linije koda (cin >> a_string;) napišemo sledeći izraz:

```
cin.getline(a_string,80);
```

korisniku će biti omogućeno da unese reč od 79 karaktera uključujući praznine, i cela ta rečenica će biti dodeljena promenljivoj a_string.

C++ string metode za rad sa tekstom

C++ string klasa, funkcije C++ string klase

- *Manipulisanje tekstom (string metode)*
- *Upotreba metoda za manipulisanje tekstom*
- *Upotreba insert, erase i replace metoda za manipulisanje tekstom*

05

MANIPULISANJE TEKSTOM (STRING METODE)

C++ klasa sadrži veliki broj metoda koje olakšavaju rad sa tekstualnim podacima

C++ klasa string obuhvata metode koje puno olakšavaju rad sa tekstovima. Da bi se koristile ove metode treba samo dodati ime metode posle imena *string* promenljive, kao i tačku. Npr. metoda size() vraća kao rezultat ukupan broj znakova i razmaka u okviru neke tekstualne promenljive *string*. Neki *string* može biti ispražnjen ako mu se zada prazan *string* sa samo dva dupla navoda, i pri tome ne sme biti nikakav prostor između ova dva znaka navoda. U nastavku su date metode koje se mogu koristiti za manipulisanje tekstom:

- name.size() - Nalaženje dužine tekstualne promenljive
- name.empty() - Test na prazne tekstualne promenljive : vraća TRUE ili FALSE
- name.compare() - Poređenje tekstualnih promenljivih
- name2.assign(name1) - Kopiranje jednog stringa u drugi (ili jednostavno operator =)
- name.find() - Nalaženje dela tekstualne promenljive
- name.erase, name.replace() - Brisanje /zamena
- name1.append(name2) - Dodavanje tj. “sabiranje” stringova
- name1.insert(position, name2) - Insertovanje jednog stringa u drugi od zadate pozicije

UPOTREBA METODA ZA MANIPULISANJE TEKSTOM

Da bi se koristile metode C++ string klase treba samo dodati ime metode posle imena string promenljive i operatora tačka (npr. string.size())

U sledećem programu je dat primer korišćenja nekih od metoda:

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

int main()
{
            string str = "C++ je interesantan
jezik";
            cout<< str<< endl;
            cout << str.size() << endl;
            str = " ";
            cout << str.size() << endl;
            string name1;
            while (name1.empty())
            {
                        getline(cin, name1);
            }
            cout <<name1<<endl;
            return 0;
}
```

Dve tekstualne promenljive se mogu porediti pomoću metode compare(), ali isto tako i pomoću operatora == i operatora !=. Takođe, neka string promenljiva može da se kopira u neku drugu string promenljivu pomoću metode assign(), ali isto tako i pomoću operatora =. U nastavku je dat primer poređenja stringova, i kopiranja jednog stringa u drugi:

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

int main()
{
            string str1, str2, str3, sum =
"strings";
            cout << "unesi string";
            getline(cin, str1);
            cout << "unesi jos jedan string";
            getline(cin, str2);
            sum += (str1 == str2) ? "identicni" :
"razliciti";
            if (str1.compare(str2) == 0)
            {
                        cout << "the same" << endl;
            }
            str3.assign(str1);
            return 0;
}
```

UPOTREBA INSERT, ERASE I REPLACE METODA ZA MANIPULISANJE TEKSTOM

Metoda insert vrši umetanje jednog stringa u drugi na određeno mesto, dok metode erase i delete brišu odnosno zamenjuju tekst između unapred zadatih indeksa u stringu

String može da se insertuje u drugi string pomoću metode *insert()*. Ova metoda traži poziciju gde će se insertovati drugi string, kao i string koji treba insertovati:

```
#include <string>
#include <iostream>
using namespace std;
int main()
{
 string str = "hello Tom";
 string sub = "dear";
 cout<<str<<endl;
 str.insert(6,sub);
 cout<<str<<endl;
 return 0;
}
```

U nastavku je dat primer korišćenja funkcije *erase()* i *replace()*. Prvi argument je startna pozicija brisanja/insertovanja, drugi argument je broj karaktera koji se brišu:

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

int main()
{
            string str = "hello";
            cout<<str<<endl;
            str.erase(1,5);
            cout<<str<<endl;
            str.replace(1,5,"hi");
            cout<<str<<endl;
            return 0;
}
```

Upravljanje memorijom u C++-u

Upravljanje memorijom, new, delete

- *Funkcije za dinamičko upravljanje memorijom*
- *Dinamičko alociranje, nizovi i funkcije*

06

FUNKCIJE ZA DINAMIČKO UPRAVLJANJE MEMORIJOM

Dinamičko alociranje memorije u C++-u se ostvaruje korišćenjem operatora new, dok se brisanje tako alocirane memorije ostvaruje korišćenjem operatora delete

Dinamičko alociranje memorije u C++-+u se obavlja korišćenjem operatora new na sledeći način:

```
new data-type;
```

gde je **data-type** bilo koji C++ tip podatka a može biti i niz ili bilo koji korisnički definisan tip podatka kao što je klasa, struktura, itd. Počnimo prvo sa standardnim tipovima podataka u C++-u. U C++-u imamo mogućnost da definišemo pokazivač na realni tip double a da zatim u toku izvršavanja programa uputimo zahtev za alokacijom memorije. To možemo uraditi korišćenjem operatora new na sledeći način:

```
double* pvalue  = NULL; // Pointer initialized with null
pvalue  = new double;   // Request memory for the variable
```

Treba napomenuti da alokacija može biti i neuspešno izvršena, ukoliko ne postoji slobodnog mesta u hip memoriji. Stoga je dobra praksa da se ispita da li operator new vraća NULL kao rezultat, odnosno da se preduzmu odgovarajući koraci u tom slučaju, što je i navedeno u nastavku:

```
double* pvalue  = NULL;
if( !(pvalue  = new double ))
{
   cout << "Error: out of memory." <<endl;
   exit(1);
}
```

Funkcija **malloc()** koja je deo C standarda je dostupna i u C++-u ali je preporučljivo izbegavati funkciju malloc. Osnovna svrha operatora new nije samo alociranje memorije, što radi i malloc, već kreiranje objekta što je i glavna svrha programa u C++-u.

U bilo kom trenutku izvršavanja programa, kada ste sigurni da vam alocirana promenljiva više ne treba, možete da oslobodite zauzeti prostor korišćenjem operatora delete, kao što je navedeno u nastavku.

```
delete pvalue; // Release memory pointed to by pvalue
```

U sledećem primeru je opisano korišćenje operatora new i delete u cilju dinamičkog alociranja i oslobađanja memorije:

```
#include <iostream>
using namespace std;
int main ()
{
   double* pvalue  = NULL; // Pointer initialized
with null
   pvalue  = new double;   // Request memory for the
variable

   *pvalue = 29494.99;     // Store value at
allocated address
   cout << "Value of pvalue : " << *pvalue << endl;

   delete pvalue;         // free up the memory.
   return 0;
}
```

DINAMIČKO ALOCIRANJE, NIZOVI I FUNKCIJE

Operatori new/delete se koriste i za alociranje/oslobađanje memorije pri radu sa jednodimenzionalnim i višedimenzionalnim nizovima

Dinamičko alociranje memorije i nizovi

Pretpostavimo da želimo da alociramo memorijski prostor za niz karaktera, tj. string koji se sastoji iz 20 karaktera. Korišćenjem slične sintakse kao u prethodnim primerima, možemo izvršiti dinamičku alokaciju na sledeći način.

```
char* pvalue  = NULL;   // Pointer initialized with null
pvalue  = new char[20]; // Request memory for the variable
```

Da bi smo oslobodili memorijski prostor koji smo rezervisali za niz pvalue koristimo sledeću liniju koda:

```
delete [] pvalue; // Delete array pointed to by pvalue
```

Korišćenjem operatora new moguće je alocirati prostor i za višedimenzionalne nizove. Sintaksa je data u nastavku:

```
double** pvalue  = NULL;     // Pointer initialized with null
pvalue  = new double [3][4]; // Allocate memory for a 3x4 array
```

Međutim u slučaju alociranja memorije višedimenzionalnog niza koristi se takođe ista sinataksa kao i kod jednodimenzionalnih nizova:

```
delete [] pvalue;        // Delete array pointed to by pvalue
```

Rezultat funkcije je pokazivač na dinamički alociranu memoriju

U okviru lekcije o dinamičkom alociranju memorije u C-u imali smo primer kako se kao rezultat funkcije pozivaocu vraća pokazivač na dinamički alociran prostor. Taj primer se u C++-u korišćenjem operatora new i delete može napisati na sledeći način:

```
#include <iostream>
using namespace std;
char * setName();
int main (void)
{
          char* str;
          str = setName();
          cout << str;
          delete [] str;
          return 0;
}

char* setName (void)
{
          char* name;
          name = new char[80];
          cout << "Enter your name: ";
          cin.getline (name, 80);
          return name;
}
```

Imenski prostor

Imenski prostor, namespace, direktiva using

- *Problemi kod preklapanja naziva funkcija*
- *Uvod u imenski prostor*
- *Ključna reč “using”*
- *Razdvojeni delovi imenskog prostora*
- *Ugnježđeni imenski prostor*

07

PROBLEMI KOD PREKLAPANJA NAZIVA FUNKCIJA

U ogromnim projektima, gde postoji više programera koji rade na realizaciji projekta, postoji velika verovatnoća da oba programera koriste ista imena za različite elemente programa

Posmatrajmo sledeći primer gde se javljaju sudari u nazivu što može da izazove nepravilno izvršavanje programa. U narednom primeru, koristimo dva fajla zaglavlja (header files) foo.h i goo.h, koja sadrže deklaracije funkcija koje obavljaju različite operacije ali imaju isti naziv i isti broj parametara.

foo.h:

```
// This DoSomething() adds it's parameters
int DoSomething(int nX, int nY)
{
    return nX + nY;
}
```

goo.h:

```
// This DoSomething() subtracts it's parameters
int DoSomething(int nX, int nY)
{
    return nX - nY;
}
```

main.cpp:

```
#include <foo.h>
#include <goo.h>
#include <iostream>

int main()
{
    using namespace std;
    cout << DoSomething(4, 3); // which DoSomething
will we get?
    return 0;
}
```

U slučaju da se fajlovi foo.h i goo.h kompajliraju posebno, tj. u dva različita projekata neće biti nikakvih problema. Međutim, ukoliko, kao u prethodnom primeru, uključimo oba fajla u istom programu, kompajler će prepoznati dve deklaracije iste funkcije sa istim brojem parametara što će izazvati sudar imena, i prijaviće sledeću grešku:

```
c:\\VCProjects\\goo.h(4) : error C2084: function 'int __cdecl DoSomething(int,int)'
already has a body
```

U projektima velike složenosti, gde postoji više programera koji rade na realizaciji projekta, postoji velika verovatnoća da oba programera koriste ista imena za različite elemente programa. Jedan od načina da programeri izbegnu ove probleme je da kucaju dugačka i komplikovana imena koja se verovatno neće ponoviti na drugim mestima projekta. U nastavku je dato bolje rešenje.

UVOD U IMENSKI PROSTOR

Korišćenjem imenskog prostora vrši se formiranje imenovanih oblasti tako da dve različite imenovane oblasti mogu da sadrže funkcije i promenljive istog naziva

Mnogo bolji način rešavanja problema je formiranje imenovane oblasti u kojoj će se grupisati srodni elementi programa (definicije i deklaracije promenljivih, funkcija i klasa). Ovakve imenovane oblasti se u programskom jeziku C++ nazivaju imenski prostor i imaju sledeći format

```
namespace namespace_name {
   // code declarations
}
```

Da bi se pozvala funkcija ili promenljiva koja je definisana u okviru imenskog prostora koristi se sledeća sintaksa:

```
name::code;  // code could be variable or function.
```

U nastavku je dat izmenjen kod fajlova foo.h i goo.h korišćenjem imenskog prostora:

foo.h:

```
namespace Foo
{
    // This DoSomething() belongs to namespace Foo
    int DoSomething(int nX, int nY)
    {
        return nX + nY;
    }
}
```

goo.h:

```
namespace Goo
{
    // This DoSomething() belongs to namespace Goo
    int DoSomething(int nX, int nY)
    {
        return nX - nY;
    }
}
```

Glavni program će biti takođe izmenjen, jer je neophodno koristi operator opsega (scope resolution operator) da bi se kompajleru eksplicitno navelo koju verziju funkcije DoSomething želimo da koristimo (da li onu koja je definisana u Foo ili u Goo imenskom prostoru):

```
int main(void)
{
    using namespace std;
    cout << Foo::DoSomething(4, 3) << endl;
    cout << Goo::DoSomething(4, 3) << endl;
    return 0;
}
```

Rezultat prethodnog programa je:

```
7
1
```

KLJUČNA REČ “USING”

Ključna reč using govori kompajleru da, ukoliko ne nađe definiciju neke funkcije, pogleda unutar imenskog prostora čiji naziv sledi nakon rezervisane reči using

Drugi način da se ukaže kompajleru da treba da baci pogled na neki imenski prostor u cilju pronalaženja funkcije ili promenljive je korišćenjem ključne reči **using**. Ključna reč using govori kompajleru da ukoliko ne može da pronađe definiciju funkcije ili promenljive, treba da pogleda unutar odgovarajućeg imenskog prostora i da potraži da li definicija slučajno postoji tamo. Na primer:

```
int main(void)
{
    using namespace std;
    using namespace Foo; // look in namespace Foo
    cout << DoSomething(4, 3) << endl;
    return 0;
}
```

Korišćenjem linije using namespace Foo obezbeđujemo da se pozivom DoSomething(4, 3) ustvari poziva funkcija Foo::DoSomething(4, 3). Stoga će rezultat programa biti: 7. Posmatrajmo još jedan primer:

```
int main(void)
{
    using namespace std;
    using namespace Foo; // look in namespace Foo
    using namespace Goo; // look in namespace Goo
    cout << DoSomething(4, 3) << endl;
    return 0;
}
```

Kao što se može pretpostaviti, prilikom kompajliranja pojaviće se sledeća greška:

C:\\VCProjects\\Test.cpp(15) : error C2668: 'DoSomething' : ambiguous call to overloaded function

U prethodnom primeru, tokom kopajliranja, kompajler ne može da pronađe definicijiu funkcije u okviru globalnog imenskog prostora tako da traži definiciju u oba imenska prostora Foo i Goo (kao i u imenskom prostoru std). Pošto je definicija funkcije DoSomething() pronađena u više od jednog imenskog prostora, kompajler je u zabuni i ne može da se odluči koju definiciju funkcije da koristi.

Oblast korišćenja ključne reči using odgovara opsegu normalne promenljive – ukoliko je ključna reč using navedena unutar funkcije ona će biti aktivna samo unutar funkcije. Ukoliko je using deklarisano van funkcije, ono će imati efekta od tog trenutka pa do kraja tekućeg fajla. Ključna reč using može da uštedi mnogo vremena u kucanju kada je neophodno koristiti veliki broj identifikatora definisanih u okviru imenskog prostora (kao na primer kada se vrše ulazno izlazne operacije korišćenjem funkcija std imenskog prostora).

RAZDVOJENI DELOVI IMENSKOG PROSTORA

Jedan imenski prostor može biti definisan iz nekoliko različitih delova. Razdvojeni delovi imenskog prostora mogu čak biti definisani i u različitim fajlovima

Jedan imenski prostor može biti definisan iz nekoliko različitih delova tako da se konačan imenski prostor dobija kao zbir svih razdvojeno definisanih delova. Razdvojeni delovi imenskog prostora mogu čak biti definisani i u različitim fajlovima.

Tako, ako jedan deo imenskog prostora zahteva ime koje je definisano u drugom fajlu, to ime mora biti deklarisano. Pisanje sledeće definicije imenskog prostora ili definiše novi imenski prostor ili na postojeći imenski prostor dodaje nove elemente:

```
namespace namespace_name {
   // code declarations
}
```

UGNJEŽDENI IMENSKI PROSTOR

Jedan imenski prostor može biti definisan unutar drugog imenskog prostora. Pristupanje unutrašnjem imenskom prostoru se ostvaruje pomoću operatora pripadnosti (::)

Imenski prostori mogu biti ugnježdeni u smislu da jedan imenski prostor može biti definisan unutar drugog imenskog prostora, kao što je navedeno u nastavku:

```
namespace namespace_name1 {
   // code declarations
   namespace namespace_name2 {
      // code declarations
   }
}
```

Pristupanje članu ugnježdenog prostora je moguće izvršiti operatorom pripadnosti, odnosno scope resolution operatorom, na sledeći način:

```
// to access members of namespace_name2
using namespace namespace_name1::namespace_name2;

// to access members of namespace_name1
using namespace namespace_name1;
```

U nastavku je dat primer gde u dva imenska prostora, spoljašnjem i unutrašnjem, imamo definicije funkcija sa istim imenom. Da bi omogućili da u programu bude vidljiva funkcija unutrašnjeg imenskog prostora, koristimo iskaz koji je naveden u sledećem primeru:

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

// first name space
namespace first_space{
   void func(){
      cout << "Inside first_space" << endl;
   }
   // second name space
   namespace second_space{
      void func(){
         cout << "Inside second_space" << endl;
      }
   }
}
using namespace first_space::second_space;
int main ()
{

   // This calls function from second name space.
   func();

   return 0;
}
```

Rezultat prethodnog programa biće:

```
Inside second_space
```

PROSTOR IMENA STANDARDNE BIBLIOTEKE

Objekti standardne biblioteke definisani su u imenskom prostoru std. Uključivanje definicija ovog imenskog prostora se ostvaruje naredbom: using namespace std

Objekti standardne biblioteke definisani su u imenskom prostoru std. Tako, na primer, deklaracije standardnih funkcija za ulazno/izlazne operacije u okviru fajla zaglavlja stdio.h smeštene su u imenskom prostoru std na sledeći način

```
//stdio.h
namespace std
{
int feof (FILE *file);
…
}
using namespace std;
```

Ono što ovde treba napomenuti je da uključivanja fajla zaglavlja stdio.h u programski kod omogućava korišćenje svih programskih elemenata definisanih i deklarisanih u okviru njega, jer je u fajl uključen operator using namespace std koji to omogućava.
Osim implicitno deklarisanih imena, u C++-u po ugledu na klase su kreirani novi fajlovi zaglavlja, tako da je umesto stdio.h moguće koristiti fajl cstdio, koji je definisan na sledeći način:

```
//cstdio
namespace std
{
int feof (FILE *file);
…
}
```

i ne sadrži operator using namespace std, tako da se obraćanje objektima imenskog prostora std može izvršiti korišćenjem operatora pripadnosti (scope resolution operatora), što je opisano u nastavku:

```
std:: feof(file);
```

ANONIMNI IMENSKI PROSTOR

U C++-u je moguće definisati anonimni imenski prostor, ali on će biti vidljiv samo u okviru fajla u kome je deklarisan

U C++-u je moguće je kreirati anonimni imenski prostor. Međutim, ono što treba znati je da će biti vidljiv samo u fajlu u kome ga kreirate**.** Pogledajmo sledeći primer:

file1.cpp

```
/*This is file1.cpp*/
#include<iostream>
using namespace std;

namespace
{
          int local;
}

void func();

int main()
{
          local = 1;
          cout << "Local=" << local << endl;
          func();
          cout << "Local=" << local << endl;
          return 0;
}
```

file2.cpp

```
/* This is file2.cpp */

namespace
{
          // Should not collide with other files
          int local;
}

void func()
{
          local = 2;
}
```

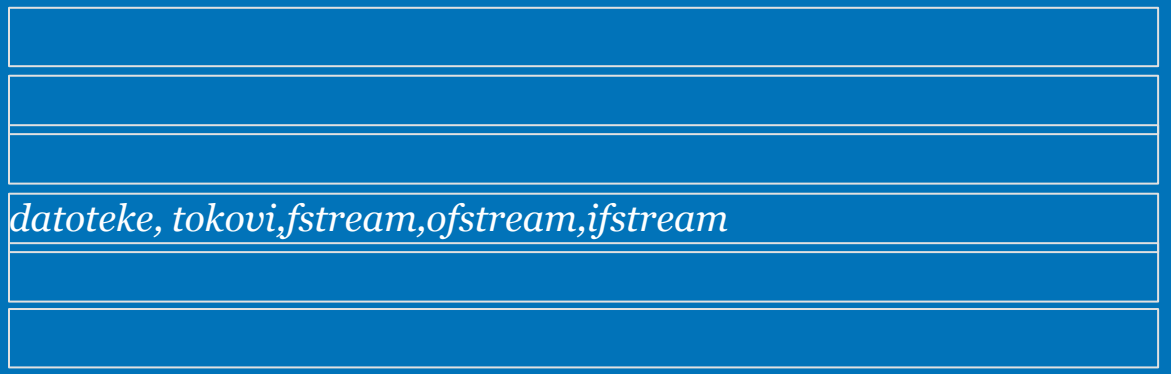
U prethodnom primeru imamo kod koji je smešten u dva različita fajla, i oba moraju biti uključena u projekat. Rezultat programa će biti:

```
Local = 1
Local = 1
```

ali neće biti (što ste mozda i očekivali):

```
Local = 1
Local = 2
```

Rad sa fajlovima u C++-u



- *C++ tokovi*
- *Otvaranje i zatvaranje datoteke*
- *Upisivanje i čitanje*
- *Pozicioni pokazivač datoteke*
- *Objekti za rad sa fajlovima kao argumenti funkcije*
- *Korišćenje konstruktora objekata za rad sa fajlovima*

08

C++ TOKOVI

Za rad sa fajlovima u C++-u se koristi standardna biblioteka fstream, u kojoj su definisana tri nova tipa podatka za rad sa fajlovima: ofstream, ifstream i fstream

Kada je u pitanju programski jezik C++, do sada smo se upoznali sa standardnom bibliotekom **iostream** u okviru koje su definisane metode **cin** i **cout** za čitanje odnosno štampanje pri radu sa standardnim ulazom.

U nastavku će biti opisane funkcije koje se koriste za rad sa fajlovima u C++-u. Pre nego se upoznamo sa organizacijom i osnovnim funkcijama za rad sa fajlovima dobro je podsetiti se nekih pojmova koji su bitni za nastavak lekcije.

Pre svega da se podsetimo šta je tok. Tok (stream) je apstraktni kanal veze koji se kreira u programu radi razmene podataka između operativne memorije i fajlova. On ne zavisi od uređaja s kojim se realizuje razmena. To znači da se ista sredstva (operacije i funkcije) mogu primenjivati sa različitim tokovima.

Pri radu sa tokovima razlikuju se nebaferizovani i baferizovani ulaz i izlaz. Bafer je oblast memorije koju koriste ulazno/izlazna sredstva za privremeno čuvanje podataka. Podaci se šalju u fajl tek pošto se napuni bafer. Kod nebaferizovane razmene slanje podataka u nebaferizovani tok se realizuje upisom podataka u fajl bez zadržavanja. Korišćenjem bafera ubrzava se razmena podataka sa fajlovima ali se zahteva rezervisanje određenog memorijskog prostora za bafer i sredstva za rad sa njim.

Prema pravcu kretanja tokovi mogu biti

- Ulazni (input stream), iz kojih se podaci učitavaju u promenljive programa.
- Izlazni (output stream), kod kojih se podaci šalju u tok iz promenljivih programa.
- Dvosmerni(input-output stream), koji omogućavaju ulaz-izlaz podataka.

Za rad sa fajlovima u C++-u se koristi standardna biblioteka **fstream**, u kojoj su definisana tri nova tipa podatka:

Tip podatka	Opis
ofstream	Tip podatka koji predstavlja izlazni tok fajla i koristi se za kreiranje fajla i upisivanje informacija u njega.
ifstream	Tip podatka koji predstavlja ulazni tok fajla i koristi se za čitanje fajla.
fstream	Tip podatka koji predstavlja opšti tok fajla i istovremeno ima sposobnosti i ofstream-a i ifstream-a, što znači da može da kreira fajl, upisuje informacije u njega, i naravno da čita informacije iz njega.

Slika-1 Tipovi podataka za rad sa C++ fajlovima

Da bi ste mogli raditi sa fajlovima u C++-u neophodno je da uključite fajlove zaglavlja <iostream> i <fstream> u vaš C++ izvorni kod.

OTVARANJE I ZATVARANJE DATOTEKE

Otvaranje fajlova se vrši pomoću funkcije open() koja je članica objekta za rad sa datotekom dok se zatvaranje datoteke vrši pozivom funkcije članice close()

❑ Otvaranje datoteke

Kao što nam je već poznato, datoteka mora biti otvorena pre nego što se pristupi operacijama čitanja i upisa.
Bilo ofstream ili fstream objekti mogu biti korišćeni za otvaranje i upis, dok je ifstream objekat moguće koristiti samo za čitanje iz fajla. U nastavku je navedena sintaksa osnovne funkcije open(), koja je definiisana u okviru fstream, ifstream, i ofstream klasa

```
void open(const char *fname, ios::openmode mode);
```

Prvi argument se odnosi na ime i lokaciju fajla koji će biti otvoren, dok se drugi argument funkcije open() odnosi na režim pristupa prilikom otvaranja fajla. Mogući režimi pristupa su navedeni u sledećoj tabeli.

Tipovi Režima	Opis
ios::app	Režim dodavanja. Podaci se upisuju na kraj postojećeg fajla.
ios::ate	Marker pozicije fajla postaviti na kraj fajla
ios::in	Otvoriti fajl za čitanje
ios::out	Otvoriti fajl za upis.
ios::trunc	Ako fajl postoji briše se a marker se postavlja na početak fajla.
ios::text	Otvara tekstualni fajl
ios::binary	Otvara binarni fajl

Slika-2 Tipovi režima pristupa pri radu sa C++ fajlovima

Vrednosti režima se mogu kombinovati pomoću operatora „ili“: |. Na primer, ako želite da otvorite fajl u režimu za upis, a želite da obrišete sadržaja ako fajl postoji, onda možete koristiti sledeću sintaksu:

```
ofstream outfile;
outfile.open("file.dat", ios::out | ios::trunc );
```

Na sličan način je moguće otvoriti fajl u cilju upisa i čitanja istovremeno:

```
fstream  afile;
afile.open("file.dat", ios::out | ios::in );
```

❑ Zatvaranje datoteke

Kada C++ program završi sa radom on automatski zatvara sve tokove, oslobađa dinamički alociranu memoriju i zatvara sve fajlove. Međutim, uvek je dobra praksa da programeri zatvore sve otvorene fajlove pre nego što program prekine sa radom. U nastavku je data standardna sintaksa funkcije close(), koja je standardna funkcija fstream, ifstream, i ofstream objekata.

```
void close();
```

UPISIVANJE I ČITANJE

Upisivanje u datoteku se može izvršiti korišćenjem operatora ubacivanja u tok (<<) dok se čitanje može izvršiti korišćenjem operatora izvlačenja iz toka (>>), koji slede nakon naziva objekta fajla

❑ **Upisivanje u datoteku**

Upisivanje u datoteku se vrši korišćenjem operatora ubacivanja u tok (<<) na isti način kako se taj operator koristi za upisivanje podatka na ekran. Jedina razlika je u tome što se umesto objekta cout koriste objekti **ofstream** ili **fstream**.

❑ **Čitanje iz datoteke**

Čitanje podataka iz fajla se vrši na isti način kao kod čitanja sa tastature: koristimo operator izvlačenja iz toka (>>), samo što se umesto objekta cin koriste objekti **ifstream** ili **fstream**.

❑ **Primer: Čitanje i upisivanje**

U nastavku je dat C++ primer koji otvara fajl u režimu za čitanje i upis. Nakon upisivanja informacija u fajl afile.dat, koje su prethodno unete od strane korisnika, vrši se učitavanje podataka iz tog istog fajla i štampanje na ekran:

U narednom primeru se koriste i dodatne funkcije cin objekta, kao što su getline() u cilju učitavanja linije teksta odnosno funkcije ignore() koja zanemaruje dodatni karakter koji je ostao iz prethodnog iskaza učitavanja.

```
#include <fstream>
#include <iostream>
using namespace std;
void main ()
{
   char data[100];
   // open a file in write mode.
   ofstream outfile;
   outfile.open("afile.dat");

   cout << "Writing to the file" << endl;
   cout << "Enter your name: ";
   cin.getline(data, 100);
   // write inputted data into the file.
   outfile << data << endl;

   cout << "Enter your age: ";
   cin >> data;
   cin.ignore();
   // again write inputted data into the file.
   outfile << data << endl;

   // close the opened file.
   outfile.close();
```

UPISIVANJE I ČITANJE

Upisivanje u datoteku se može izvršiti korišćenjem operatora ubacivanja u tok (<<) dok se čitanje može izvršiti korišćenjem operatora izvlačenja iz toka (>>), koji slede nakon naziva objekta fajla

❑ **Upisivanje u datoteku**

Upisivanje u datoteku se vrši korišćenjem operatora ubacivanja u tok (<<) na isti način kako se taj operator koristi za upisivanje podatka na ekran. Jedina razlika je u tome što se umesto objekta cout koriste objekti **ofstream** ili **fstream**.

❑ **Čitanje iz datoteke**

Čitanje podataka iz fajla se vrši na isti način kao kod čitanja sa tastature: koristimo operator izvlačenja iz toka (>>), samo što se umesto objekta cin koriste objekti **ifstream** ili **fstream**.

❑ **Primer: Čitanje i upisivanje**

U nastavku je dat C++ primer koji otvara fajl u režimu za čitanje i upis. Nakon upisivanja informacija u fajl afile.dat, koje su prethodno unete od strane korisnika, vrši se učitavanje podataka iz tog istog fajla i štampanje na ekran:

U narednom primeru se koriste i dodatne funkcije cin objekta, kao što su getline() u cilju učitavanja linije teksta odnosno funkcije ignore() koja zanemaruje dodatni karakter koji je ostao iz prethodnog iskaza učitavanja.

```
   // open a file in read mode.
   ifstream infile;
   infile.open("afile.dat");

   cout << "Reading from the file" << endl;
   infile >> data;
   // write the data at the screen.
   cout << data << endl;

   // again read the data from the file and
display it.
   infile >> data;
   cout << data << endl;

   // close the opened file.
   infile.close();
}
```

POZICIONI POKAZIVAČ DATOTEKE

Pozicioni pokazivač datoteke je neka celobrojna vrednost koja specificira lokaciju u fajlu, u obliku broja bajtova, u odnosu na početnu poziciju fajla

Obe klase **istream** i **ostream** sadrže nekoliko funkcija koje omogućavaju nasumičan pristup datoteci. U ovu grupu funkcija spadaju **seekg** ("seek get") za istream odnosno **seekp** ("seek put") za ostream objekat.

Argument funkcija seekg i seekp je obično long int. Drugi argument može biti takav da ukazuje na pravac potrage. Pravac pretrage može biti **ios::beg** (podrazumevana vrednost) za pozicioniranje linije relativno u odnosu na početak toka, **ios::cur** za pozicioniranje u odnosu na trenutnu poziciju u toku, ili **ios::end** za pozicioniranje u odnosu na kraj toka.

Pozicioni pokazivač datoteke je neka celobrojna vrednost koja specificira lokaciju u fajlu, u obliku broja bajtova, u odnosu na početnu poziciju fajla. Neki od primera korišćenja pozicionog pokazivača su dati u nastavku:

```
// position to the nth byte of fileObject (assumes ios::beg)
fileObject.seekg( n );

// position n bytes forward in fileObject
fileObject.seekg( n, ios::cur );

// position n bytes back from end of fileObject
fileObject.seekg( n, ios::end );

// position at end of fileObject
fileObject.seekg( 0, ios::end );
```

OBJEKTI ZA RAD SA FAJLOVIMA KAO ARGUMENTI FUNKCIJE

Objekat toka fajla se prosleđuje po referenci jer je u funkciji moguće promeniti unutrašnje stanje objekta (kao što je režim pristupa) iako možda nemamo nameru da menjamo sadržaj fajla

U nastavku je dat primer gde se iz glavnog programa pozivaju dve funkcije: *writeFile* koja otvara fajl za upis i kao argument ima objekat tipa *ofstream*, i *readFile* koja otvara fajl za čitanje korišćenjem objekta tipa *ifstream*. Obe funkcije sadrže deo koda koji ispituje da li su fajlovi uspešno otvoreni i kao rezultat vraćaju odgovaraju logičku vrednost u zavisnosti od ishoda:

```
bool writeFile (ofstream& file, char* strFile)
{
   file.open(strFile);
   if (file.fail())
         return false;
   else
         return true;
}
bool readFile (ifstream& ifile, char* strFile)
{
   ifile.open(strFile);
   if (ifile.fail())
         return false;
   else
         return true;
}
```

U obe funkcije, objekat toka fajla je prosleđen po referenci a ne po vrednosti bez obzira što u funkciji ne postoji namera da se promeni sadržaj fajlova. Razlog je taj što unutrašnje stanje objekta toka fajla može biti promenjeno korišćenjem operatora režima čak iako ne menjamo sadržaj fajla. Glavni program izgleda ovako:

```
#include <fstream>
#include <iostream>
#include <string>
using namespace std;
bool writeFile (ofstream&, char*);
bool readFile (ifstream&, char*);
int main ()
{
   string data;
   bool status;
   ofstream outfile;
   status = writeFile(outfile, "students.dat");
   if (!status)
   {
      cout << "File could not be opened for writing\n";
      cout << "Program terminating\n";
      return 0;
   }
   else
   {
      cout << "Writing to the file" << endl;
      cout << "==================" << endl;
      cout << "Enter class name: ";
      getline(cin, data);
      outfile << data<< endl;
      cout << "Enter number of students: ";
      cin >> data;
      cin.ignore();
      outfile << data<< endl;
      outfile.close();
   }
   ifstream infile;
   status = readFile(infile, "students.dat");
   if (!status)
   {
      cout << "File could not be opened for reading\n";
      cout << "Program terminating\n";
      return 0;
   }
   else
   {
      cout << "Reading from the file" << endl;
      cout << "====================" << endl;
      getline(infile, data);
      while(!infile.fail())
      {
         cout << data << endl;
         getline(infile, data);
      }
      infile.close();
   }
   return 0;
}
```

KORIŠĆENJE KONSTRUKTORA OBJEKATA ZA RAD SA FAJLOVIMA

Osim korišćenjem funkcije članice open() moguće je koristiti konstruktor objekata fstream, ofstream i if stream u cilju otvaranja fajlova za čitanje, upisivanje ili istovremeno za oba

Moguće je koristiti konstruktore objekata *fstream* ili *ofstream* da bi ste otvorili fajl za upisivanje. Konstruktor je funkcija koja se automatski poziva kada pokušate da kreirate instancu nekog objekta. Instanca objekta ima slično značenje kao i promenjiva primitivnog tipa podatka, npr int. Tako, na primer, sledeći iskaz se može okarakterisati kao kreiranje jedne instance, pod imenom *age,* koja je celobrojnog tipa:

```
int age;
```

Slično tome, sledeći iskaz kreira instancu afile objekta *fstream:*

```
fstream afile;
```

Konstruktori objekata mogu biti predefinisani (preopterećeni, engl. overloaded) tako da za jedan isti objekat može postojati više konstruktora, s tim što je neki bez argumenata, a ostali sa jednim ili više argumenata. Tako, predhodni iskaz *fstream afile,* se naziva konstruktor bez argumenata objekta *fstream*. Sledeći iskaz poziva konstruktor sa jednim argumentom objekta *ofstream*, koji istovremeno kreira instancu objekta *ofstream* i otvora fajl *students.dat* za upis podataka:

```
ofstream outFile(“students.dat");
```

Sledeći iskaz se naziva konstruktor sa dva arguementa objekta fstream koji istovremeno kreira instancu objekta *fstream* i otvora fajl *students.dat* za upis podataka:

```
fstream aFile(“students.dat", ios:out);
```

Deklaraciju promenljive *ofstream* (ili *fstream*) u jednom iskazu, a zatim otvaranje fajla korišćenjem funkcije članice *open* u sledećem iskazu možemo posmatrati kao skup operacija koji je analogan deklarisanju primitivne promenljive u jednom iskazu a zatim dodeli vrednosti promenljive u sledećem iskazu. Suprotno prethodnom, korišćenje konstruktora sa jednim ili dva argumenta objekta *ofstream* (ili *fstream*) je analogno inicijalizaciji primitivne promenljive, tj analogno sa:

```
int age = 39;
```

Otvaranje fajla za čitanje korišćenjem konstruktora objekta *ifstream* se može uraditi na sledeći način:

```
ifstream infile ("students.dat");
```

Otvaranje fajla za čitanje se može ostvariti korišćenjem konstruktora objekta *fstream*, s tim što će drugi argument konstruktora biti vrednost režima koja će ukazati da će fajl biti otvoren za čitanje:

```
fstream afile ("students.dat", ios::in);
```

Otvaranje fajla istovremeno za čitanje i štampu je moguće ostvariti korišćenjem konstruktora objekta fstream na sledeći način:

```
fstream afile ("students.dat", ios::in | ios::out);
```

DODATAK - IZVOD FUNKCIJA ZA RAD SA C++ FAJLOVIMA

Jezik C++ sadrži veliki broj metoda i operatora koji olakšavaju rad kako sa tekstualnim tako i sa binarnim fajlovima

U nastavku je na sledećim slikama sumirano sve što je bitno u vezi rada sa C++ fajlovima, kako binarnim tako i tekstualnim.

Za upisivanje podataka u fajl	
Tekstualni fajl ofstream out ("myfile.txt"); ili ofstream out; out.open("myfile.txt");	**Binarni fajl** ofstream out ("myfile.txt",ios::binary); ili ofstream out; out.open("myfile.txt", ios::binary);
Za dodavanje podataka na kraj postojećeg fajla	

Tekstualni fajl
ofstream out("myfile.txt",ios::app);
ili
ofstream out;
out.open("myfile.txt", ios::app);
	Binarni fajl
	ofstream out
	("myfile.txt",ios::app|ios::binary);
	ili
	ofstream out;
	out.open("myfile.txt", ios::app | ios::binary);

Za čitanje podataka	
Tekstualni fajl ifstream in ("myfile.txt"); ili ifstream in ; in.open("myfile.txt");	**Binarni fajl** ifstream in ("myfile.txt", ios::binary); ili ifstream in ; in.open("myfile.txt", ios::binary);

Slika-3 Kreiranje i otvaranje fajla

ofstream object "out"	ifstream object "in"
out.close();	in.close();

Slika-4 Zatvaranje fajla (nakon čitanja ili pisanja)

Podatak	Funkcija za čitanje fajla	Funkcija za upis u fajl
char	get();	put();
1 word	>> (extraction operator)	<< (insertion operator)
>=1 word	getline();	<< (insertion operator)
objekti	read()	write()
Binarni podatak	isto kao prethodno	isto kao prethodno

Slika-5 Čitanje/Upis podatka sa/na kraj fajla

Operacija	Funkcija	Opis
Proverava kraj fajla	eof()	Koristi se da se proveri da li dostignut kraj fajla tokom čitanja (eof)
Provera uspešnosti	bad()	Vraća tačno (true) ukoliko je čitanje/upisivanje neuspečno
Provera uspešnosti	fail()	Vraća tačno u istom slučaju kao bad(), ali i u slučaju da nastane greška u formatu.
Da li je fajl otvoren	is_open()	Ispituje da li je fajl otvoren ili ne, vraća true ako je otvoren, u suprotnom vraća false
Broj očitanih bajtova	gcount()	Vraća broj bajtova očitanih od početka fajla
Ignorisanje karaktera	ignore()	Ignoriše n bajtova iz fajla (get pokazivač je pozicioniran posle n-tog karaktera)
Ispitivanje sledećeg karaktera.	peek()	Ispituje sledeći dostupan karakter, ali neće uvećati pozicioni pokač na sledeći karakter.
Nasumični pristup(samo kod binarnih fajlova)	seekg() seekp() tellg() tellp()	Služe za nasumičan pristup pri radu sa binarnim fajlovima se. One ili daju ili postavljaju poziciju za get i put pokazivač na odgovarajuću lokaciju u fajlu

Slika-6 Funkcije koje obavljaju specijalne operacije

Zaključak

11

O C++ FUNKCIJAMA I STRINGOVIMA

Možemo zaključiti sledeće:

Programski jezik C++ u odnosu na C uvodi novi način prosleđivanja stvarnih parametara funkciji, a to je prosleđivanje po referenci. Prosleđivanje po referenci je dosta slično kao prosleđivanje po adresi, ali naravno postoje bitne razlike. Razlika u sintaksi između prosleđivanja po referenci i adresi je u tome što u prototipu i zaglavlju funkcije, koristi se adresni operator (&) za prosleđivanje po referenci a asterisk, ili indirektni operator (*) za prosleđivanje po adresi. Osim toga, ukoliko je pokazivač argument funkcije s tim što pokazuje na pojedinačnu promenljivu (ne niz) onda postoje još dve dodatne razlike između korišćenja referenci i pokazivača. Prvo, kada se vrši poziv funkcije, ne treba koristiti adresi operator (&) za prosleđivanje po referenci, ali je neophodno koristiti ga kod prosleđivanja po adresi. Drugo, u telu funkcije ne treba vršiti dereferenciranje kada se prosleđuje po referenci ali je neophodno izvršiti dereferenciranje kod prosleđivanja po adresi (pokazivaču).

Dodatno, referenca može biti i povratna vrednost funkcije.

Standardna biblioteka string obezbeđuje veliki broj funkcija korisnih za rad sa C++ string klasom. Tako je moguće koristiti: funkciju članice *length* ili *size* da bi odredili dužinu stringa, operator dodele = da bi ste dodelili vrednost C++ stringu, kombinovani operator sabiranja i dodele += da bi ste na kraj jednog stringa dodali drugi i logičke operatore da bi ste poredili dve promenljive C++ string klase.

O C++ FAJLOVIMA

Na osnovu pređenog gradiva možemo izvesti sledeći zaključak:

Da bi C++ program imao mogućnost da učitava odnosno upisuje podatke u datoteke, neophodno je da u njega uključite *fstream* standardnu biblioteku. Standardna C++ biblioteka definiše tri tipa podatka za rad sa fajlovima. Tip *ofstream* predstavlja izlazni tok fajla, gde se smer protoka informacija prostire od programa do izlaznog fajla. Tip *ifstream* predstavlja ulazni tok fajla, tj. put informacija od fajla do korisničkog programa. Konačno, tip fstream predstavlja opšti tok fajla, koji ima sposobnosti i *ofstream-a* i *ifstream*-a, u smislu da može da služi i za čitanje iz fajla i za upisivanje podataka u fajl.

Proces pristupa datoteci se sastoji iz nekoliko koraka. Prvi korak je otvaranje fajla pri čemu se uspostavlja putanja između fajla i odgovarajućeg objekta koji se odnosi na tok fajl —*fstream, ofstream,* ili ifstream. Drugi korak je, naravno, čitanje iz fajla ili upisivanje u fajl. Treći i poslednji korak je zatvaranje fajla, korišćenjem funkcije close, čiji je cilj takođe oslobašanje sistemskih resursa koji su bili neophodni za uspostavljanje komunikacione putanje između fajla i objekta toka preko koga smo pristupili fajlu. Svrha zatvaranje fajla je i da se izbegne takvozvani problem „deljenja resursa“ izazvan pokušajem da se u jednom delu koda otvori fajl koji se već koristi u nekom drugom delu programa, tj ranije je otvoren ali nije još uvek zatvoren.

Otvaranje fajla se može izvršiti ili korišćenjem funkcije članice open, ili korišćenjem konstruktora. Konstruktor je funkcija koje se automatski poziva kada pokušate da kreirate instancu objekata kao što su *fstream, ofstream,* ili *ifstream* objekat. I funkcija o*pen* i konstruktor mogu da imaju dva argumenta. Prvi argument je relativna ili apsolutna putanja fajla. Drugi argument, koji može biti opcioni, definiše režim pod kojim će fajl biti odvoren (režim za čitanje, upis, dodavanje, itd). Nije moguće unapred znati da je neki fajl uspešno otvoren za čitanje ili upisivanje. Stoga je neophodno koristiti funkciju *fail* koja proverava dali je fajl uspešno otvoren.

Informacije se upisuju u fajl korišćenjem operatora ubacivanja u tok (<<) kao kod štampanja na ekran, osim što se umesto cout objekta koristi *ofstream* ili *fstream* objekat. Slično upisu, učitavanje se vrši korišćenjem operatora izvlaćenja iz toka (>>), s tim što se umesto objekta cin koriste *ifstream* (ili *fstream*) objekat. Linija fajla se može čitati korišćenjem getline funkcije objekta kada se radi sa C-stringovima, odnosno korišćenjem funkcije *getline* pri radu sa C++ stringovima. Korišćenjem funkcije članice *fail* moguće je proveriti da li ste stigli do kraja fajla pri učitavanju podataka iz fajla liniju po liniju.