

# Lekcija 06

## Rad sa fajlovima, Otklanjanje grešaka, Uvod u C++

*Miljan Milošević*



# RAD SA FAJLOVIMA, OTKLANJANJE GREŠAKA, UVOD U C++

## 01

## 02

## 03

## 04

Uvod

**Datoteke i tokovi u C-u**

**Otvaranje i zatvaranje datoteka**

**Funkcije za rad sa datotekama**

**Dijagnostika i korigovanje grešaka**

- *Uvod u datoteke i tokove*
- *Standardne datoteke i osnovne funkcije za rad sa fajlovima*

- *Otvaranje datoteke*
- *Zatvaranje datoteka*

- ❑ *Funkcije za rad sa katekterima i datotekama*
- ❑ *Funkcije za rad sa tekstom i datotekama*
- ❑ *Formatirani ulaz/izlaz i datoteke*
- ❑ *Sistemske funkcije za rad sa datotekama*
- ❑ *Funkcije za rad sa binarnim datotekama*

- *Osnovne metode za dijagnostiku*
- *Karakteristike i prednosti procesa dijagnostike*
- *Krajnje mere i sugestije za dobro obavljenu dijagnozu grešaka*

# RAD SA FAJLOVIMA, OTKLANJANJE GREŠAKA, UVOD U C++

## 05

### Alati za dijagnostiku grešaka

- *Osnovne opcije alata za dijagnostiku*
- *Prednosti i mane alata za dijagnostiku*

## 06

### Uvod u C++

- *Rezervisane reči; Logički tip podatka*
- *Prvi C++ program*
- *Pregled tipova podataka; Lokalne i globalne promenljive*
- *Modifikator const, razmeštanje opisa promenljivih, konverzija podataka*
- *Komentari u C++-u*

## 07

### Standardni ulaz i izlaz u C++-u

- *Ulaz/ izlaz u C++-u (tokovi – streams)*
- *Ulaz/ izlaz u C++-u – osnovni primer*
- *Korišćenje cin i cout objekta kod nizova*
- *Ulazne funkcije get() i getline()*
- *Funkcija ignore()*
- *Pravila kod učitavanja karaktera i problema sa karakterom nove linije*

## 08

### Formatiranje izlaznih podataka u C++-u

- *Formatiranje pri štampanju korišćenjem ostream i ios*
- *Primer formatiranja korišćenjem ostream-a i ios-a*
- *Podešavanje širine polja, praznih polja i poravnavanje*
- *Primer podešavanja širine polja, ispune praznih polja i poravnavanja*

## 09

### Vežbe – Rad sa fajlovima u C-u

- *Primer: Otvaranje tekstualnog fajla i upisivanje*
- *Primer. Upisivanje podataka o učenicima u fajl*
- *Primer. Upisivanje podataka o učenicima u fajl*
- *Primer. Upisivanje u fajl i učitavanja iz fajla*
- *Primeri. Rad sa komandnom linijom; Dodavanje podataka u postojeći fajl*

## 10

### Vežba – Alati za dijagnostiku grešaka

- ❑ *Dijagnostika grešaka - Visual Studio 2010*
- ❑ *Dijagnostika grešaka - Netbeans IDE*

## 11

### Vežba – Uvod u C++

- *Izgled prvog C++ programa*
- *Primer. Štampanje podataka na ekran*
- *Primer. Korišćenje petlji u C++-u*
- *Primer. Formatiranje pri štampanju podataka - precision*
- *Primer. Učitavanja stringova u C++ program*
- *Primer. Formatirani izlaz - korišćenje specifikatora za*

# UVOD

*U okviru ove lekcije studenti se upoznaju sa sledećim pojmovima:*

- ❑ Rad sa fajlovima u C-u
- ❑ Alati za dijagnostiku i korigovanje grešaka u programu
- ❑ Uvod u C++: pisanje prvog programa i korišćenje ulazno/izlaznih funkcija

Od svakog programa se uglavnom očekuje da ima mogućnost da štampa podatke u datoteke ili fizičke medijume kao što su monitori ili štampači, kao i da učitava podatke iz fajlova odnosno sa ulaznih uređaja kao što je na primer tastatura. Standardna C biblioteka obezbeđuje ogroman broj funkcija koje se mogu koristiti u ovu svrhu. U ovoj lekciji će biti prikazan dobar deo standardne biblioteke C jezika koji se posvećen ulazno-izlaznim operacijama i koji se često naziva *I/O* biblioteka. Sve funkcije koje se koriste za rad sa fajlovima su deklarisanе u fajlu *stdio.h*.

Standardna biblioteka implementira jednostavan model tekstualnog ulaza i izlaza. Ulaz i izlaz se modeluju tzv. tokovima (engl. *stream*) podataka (obično pojedinačnih bajtova ili karaktera). Standardni ulaz obično čine podaci koji se unose sa tastature. Podaci koji se upućuju na standardni izlaz se obično prikazuju na ekranu. Pored standardnog izlaza, postoji i standardni izlaz za greške na koji se obično upućuju poruke o greškama nastalim tokom rada programa i koji se, takode, obično prikazuje na ekranu. Iako se svaka datoteka može razmatrati kao niz bajtova, obično razlikujemo datoteke koje sadrže tekstualni sadržaj od datoteka koje sadrže binarni sadržaj. U zavisnosti od toga da li se u datoteci nalazi tekstualni ili binarni sadržaj, razlikuju se dva različita načina pristupa.

U programima koje pišemo obično mogu da se jave sintaksne (*compile-time*) greške i izvršne (*run-time*) greške. Izvršne greške je teže pronaći i otkloniti od sintakasnih grešaka. Za mnoge programere, korigovanje grešaka je najteži deo programiranja. Međutim, korigovanje se može načiniti mnogo lakšim ako se koriste određene tehnike koje će biti opisani u ovom poglavlju.

Jezik C++ se razlikuje od C-a pre svega podrškom objektno orijentisanom programiranju. Međutim, u njemu ima i niz novih opcija i mogućnosti, koje nisu objektnog karaktera, zbog kojih pisanje programa koji nisu objektnе prirode, lakše sprovesti u C++-u nego u C-u.

# Datoteke i tokovi u C-u

*datoteke, tokovi, stdin, stdout, stderr*

- 
- *Uvod u datoteke i tokove*
  - *Standardne datoteke i osnovne funkcije za rad sa fajlovima*

01

# UVOD U DATOTEKE I TOKOVE

*Pod datotekom podrazumevamo željeni prostor na memorijskom medijumu koji sadrži određene informacije. C jezik tretira datoteku kao strukturu koja se definiše u fajlu `stdio.h`*

Do sada opisane funkcije za ulaz i izlaz su uglavnom čitale sa standardnog ulaza i pisale na standardni izlaz, koji su automatski definisani i od strane operativnog sistema i kojima program automatski ima pristup. Iako je, mehanizmima preusmeravanja, bilo moguće izvesti programski pristup lokalnim datotekama, mehanizam preusmeravanja je veoma ograničen jer ne daje mogućnost istovremenog čitanja (ili pisanja) datoteke i standardnog ulaza kao ni mogućnost istovremenog čitanja (ili pisanja) više datoteka. Zbog toga, jezik C nudi direktnu podršku za rad sa lokalnim datotekama, bez potrebe za korišćenjem usluga preusmeravanja operativnog sistema. Sve potrebne deklaracije i za rad sa datotekama nalaze u zaglavlju `<stdio.h>`.

Pojam fajl je u C-u uveden zbog potrebe povezivanja programa sa ulazno-izlaznim uređajima, koji omogućavaju registrovanje podataka na magnetnom disku, magnetnoj traci, ekranu monitora, itd.

Pod datotekom podrazumevamo željeni prostor na memorijskom medijumu koji sadrži određene informacije. C jezik tretira fajl kao strukturu, koja se definiše u fajlu `stdio.h`.

```
struct iobuf
{
    char *ptr;
    int cnt;
    char *base;
    char flag;
    char file;
};

#define FILE struct iobuf;
```

Svi programi sa kojima ćemo raditi u nastavku će koristiti predhodno definisani tip `FILE`.

# STANDARDNE DATOTEKE I OSNOVNE FUNKCIJE ZA RAD SA FAJLOVIMA

*Kada se C program pokrene, operativni sistem otvara tri toka podataka i to: standardni ulaz, standardni izlaz i standardni izlaz za greške*

Kada se C program pokrene, operativni sistem otvara tri toka podataka (standardni ulaz, standardni izlaz i standardni izlaz za greške), kao da su datoteke i obezbeđuje pokazivače kojima im se može pristupati. Ti pokazivači se nazivaju:

```
FILE* stdin;  
FILE* stdout;  
FILE* stderr;
```

Kao što smo već napomenuli, da bi smo mogli da koristimo funkcije za rad sa fajlovima, neophodno je da uključimo standardnu biblioteku **stdio.h**. Funkcije iz standardne biblioteke **stdio.h** koje možemo da koristimo za rad sa tekstualnim fajlovima su:

- **fopen** – otvara tekstualni fajl
- **fclose** – zatvara tekstualni fajl
- **feof** – detektuje marker kraja fajla.
- **fscanf** – učitava formatirani ulaz iz fajla
- **fprintf** – štampa formatirani izlaz u fajl.
- **fgets** – učitava string iz fajla.
- **fputs** – štampa string u fajl.
- **fgetc** – učitava karakter iz fajla.
- **fputc** – štampa karakter u fajl.



# Otvaranje i zatvaranje datoteka

<i>otvaranje fajla, zatvaranje fajla, fopen, fclose</i>

- 
- *Otvaranje datoteke*
  - *Zatvaranje datoteka*

02

# OTVARANJE DATOTEKE

*Za otvaranje datoteke se koristi funkcija `fopen()` koja kao rezultat vraća pokazivač preko koga zatim pristupamo datoteci*

Pre nego što se započne rad sa fajlovima potrebno je otvoriti fajl radi čitanja ili upisivanja funkcijom `fopen()`. Zaglavlje ove funkcije je:

```
FILE *fopen(char *fname, char *mode);
```

Prvi parametar ove funkcije je ime fajla koji treba otvoriti. Drugi parametar je mode koji definiše način na koji se koristi fajl (režim pristupa). Funkcija `fopen()` vraća pokazivač na fajl pomoću koga zatim pristupamo datoteci. Ako fajl ne može da se otvori funkcija `fopen()` vraća vrednost `NULL`. Kontrola da li je fajl uspešno otvoren može se realizovati sa:

```
if(inf=fopen("test","r")!=NULL)
```

Najčešće se koriste sledeći specifikatori pristupa nekoj datoteci (fajlu):

r	Fajl se otvara samo za čitanje
w	Fajl se otvara za upis (sadržaj se briše)
a	Fajl se otvara za dodavanje sadržaja (pri čemu se stari sadržaj čuva)
r+	Otvora fajl istovremeno i za čitanje i za upis.
w+	Otvora fajl istovremeno i za čitanje i za upis. Prvo skraćuje fajl na dužinu jednaku nuli a ukoliko fajl ne postoji onda ga kreira.
a+	Otvora fajl istovremeno i za čitanje i za upis. Kreira fajl ukoliko ne postoji. Čitanje se vrši od početka fajla dok se upis vrši na kraj, tj. vrši se dodavanje sadržaja i čuvanje starog sadržaja.

Slika-1 Specifikatori koji se koriste pri otvaranju fajlova

Ukoliko imate u planu da radite sa binarnim datotekama onda je poželjno koristiti sledeće specifikatore pristupa:

```
"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"
```

# ZATVARANJE DATOTEKA

*Zatvaranje datoteke se obavlja korišćenjem funkcije `fclose()` čiji je argument pokazivač na već otvorenu datoteku*

Kada se prekida rad sa datotekama ona se mora zatvoriti, bez obzira da li je ulazna ili izlazna, funkcijom:

```
fclose(pf);
```

Prototip funkcije je:

```
int fclose( FILE *fp );
```

gde je `pf` pokazivač na predhodno otvoreni fajl. Ovim se oslobađa bafer koji se koristio pri radu sa fajlovima. Pod baferizacijom se podrazumeva privremeno čuvanje ulaznih i izlaznih podataka u delu memorije koji se naziva bafer. Funkcija `fclose()` vraća `0` ako je zatvaranje izvršeno uspešno i `EOF` (`-1`) u suprotnom. Takođe, kreirana struktura `FILE` nije više potrebna i uklanja se iz memorije. S obzirom na to da većina operativnih sistema ograničava broj datoteka koje mogu biti istovremeno otvorene, dobra je praksa zatvarati datoteke čim prestanu da budu potrebne. U slučaju da se program normalno završi, funkcija `fclose()` se poziva automatski za svaku otvorenu datoteku.

# Funkcije za rad sa datotekama

<i>datoteke, ulaz, izlaz, tekst, karakteri, formatirani I/O</i>

- 
- Funkcije za rad sa katekterima i datotekama*
  - Funkcije za rad sa tekstom i datotekama*
  - Formatirani ulaz/izlaz i datoteke*
  - Sistemske funkcije za rad sa datotekama*
  - Funkcije za rad sa binarnim datotekama*

03

# UVODNA RAZMATRANJA

## *Programski jezik C podržava veliki broj funkcija koje olakšavaju rad sa tekstualnim i binarnim datotekama*

U nastavku će biti opisane sledeće grupe funkcija:

- Funkcije za rad sa karakterima
- Funkcije za rad sa tekstom
- Formatirani ulaz i izlaz
- Sistemske funkcije za rad sa datotekama
- Funkcije za rad sa binarnim datotekama

# Funkcije za rad sa katekterima i datotekama

<i>datoteke, karakteri, getc, fgetc, putc, fputc</i>

---

➤ *Funkcije za rad sa karakterima: getc() i putc()*

➤ *Funkcija ungetc()*

03

# FUNKCIJE ZA RAD SA KARAKTERIMA: GETC() I PUTC()

*Nakon što se otvori datoteka, iz nje se čita ili se u nju piše sadržaj. To mogu biti i pojedinačni karakteri*

## ❑ Funkcije **getc()** i **fgetc()**

Funkcija **getc()** učitava jedan karakter iz datoteke (učitavanje se obavlja iz datoteke). Funkcija ima sledeće zaglavlje:

```
int getc(FILE *pf)
```

Promenljiva **pf** dobija vrednost pomoću funkcije **fopen()**. Argument funkcije je pokazivač na strukturu **FILE**, koji identifikuje željenu datoteku, a vraća vrednost tipa **int**. Kada funkcija **getc()** dostigne kraj fajla tada vraća **EOF**. Osim prethodne, postoji i sledeći oblik funkcije za čitanje karaktera iz fajla:

```
int fgetc( FILE * fp );
```

Funkcija **fgetc()** radi po istom principu kao i **getc()**, s tim što je **getc()** definisana kao makro-funkcija dok je **fgetc()** definisana kao obična funkcija. Makro **getc()** se uglavnom koristi jer je brži od poziva funkcije. Međutim, u nekim slučajevima, kada je **fp** složeni izraz koji može da ima prateće efekte, poželjnije je koristiti funkciju jer se može desiti da makro svoje argumente obradi više od jednom.

## ❑ Funkcije **putc()** i **fputc()**

Funkcija **putc()** upisuje karakter u datoteku. Zaglavlje date funkcije je:

```
int putc(char c, FILE *pf)
```

Argument **c** je karakter koji se upisuje u fajl na koji pokazuje argument **pf**. Osim makro-funkcije **putc()** moguće je koristiti i funkciju **fputc()**:

```
int fputc( int c, FILE *fp );
```

U nastavku je dat primer kojim se sadržaj fajla "**test**" (formiran od velikih slova) šifriran šalje u fajl "**šifra**". Znak se šifrira tako što se zamenjuje sledećim **ASCII** znakom, a znak '**Z**' zamenjuje sa '**A**'.

```
#include<stdio.h>
void main()
{
    FILE *inf, *outf; /*pokazivači na fajlove*/
    int c;
    inf=fopen("test","r");
    outf=fopen("šifra","w");

    while((c=getc(inf))!=EOF) /*uzima se znak iz
inf*/
    {
        if('A'<c && c<'Z')
            c++;
        else
            c='A';
        putc(c,outf);
    }
}
```

# FUNKCIJA UNGETC()

*Funkcija ungetc() ubacuje karakter u tok (stream) tako da on postaje dostupan prilikom naredne operacije čitanja podataka iz toka*

Prototip funkcije je:

```
int ungetc (int c, FILE *fp);
```

i ona „**vraća**“ karakter u datoteku a kao rezultat daje identifikator pozicije datoteke tako da naredni poziv operacije čitanja nad ovom datotekom očita upravo vraćeni karakter. Ovaj karakter može, ali ne mora, biti jednak poslednjem pročitanoj karakteru datoteke. Iako ova funkcija utiče na naredne operacije čitanja datoteke, ona ne menja fizički sadržaj same datoteke (naime vraćeni karakteri se najčešće smeštaju u memorijski bafer odakle se dalje čitaju, a ne u datoteku na disku). Funkcija **ungetc** kao rezultat daje **EOF** ukoliko je došlo do greške, a vraćeni karakter inače.

Naredni primer čita karaktere iz datoteke dok su u pitanju cifre i pri tom gradi dekadni ceo broj. Pošto poslednji karakter koji je pročitao nije cifra, on se (uslovno rečeno) vraća u tok kako ne bi bio izostavljen prilikom narednih čitanja.

```
#include <stdio.h>

int readint(FILE* fp)
{
    int val = 0, c;
    while (isdigit(c = getc(fp)))
        val = 10*val + (c - '0');
    ungetc(c, fp);
}
```



# Funkcije za rad sa tekstom i datotekama

<i>datoteke, tekst, fgets, fputs</i>

- 
- *Funkcije za rad sa tekstom: fgets() i fputs()*
  - *Implementacija funkcija fgets(), fputs i getline()*

03

# FUNKCIJE ZA RAD SA TEKSTOM: FGETS() I FPUTS()

*Standardna biblioteka definiše i funkcije za rad sa tekstualnim datotekama liniju po liniju*

## ❑ Funkcija **fgets()**

Zaglavlje ove funkcije je sledeće:

```
char *fgets(char *string, int maxl, FILE *pf)
```

Ova funkcija čita iz fajla string (na koga pokazuje pf) do sledećeg znaka nove linije ili dok ne očita **maxl-1** znakova.

Drugi argument predstavlja maksimalnu dužinu učitanoog stringa.

Na kraju stringa se dopisuje završni znak. Ostatak linije preko maksimalne dužine se ne ignoriše već se čita sledećim pozivom funkcije fgets(). Kada naiđe na znak **EOF** ova funkcija vraća vrednost **NULL**.

Razlika između **gets()** i **fgets()** je u tome što **gets()** zamenjuje znak nove linije sa završnim znakom, a **fgets()** čuva znak nove linije ako ga ima i posle njega dopisuje završni znak.

## ❑ Funkcija **fputs()**

Zaglavlje funkcije fputs() je:

```
int fputs(char *string, FILE *pf)
```

Ova funkcija upisuje u fajl (na koji pokazuje pf) string. Funkcija vraća **EOF** u slučaju greške. Slično **puts()** ova funkcija ne šalje završni znak stringu koji predaje. Za razliku od **puts()** funkcija **fputs()** ne dopisuje znak za novu liniju stringu koji predaje.

**Primer:** Napisati program koji korišćenjem funkcije **fgets()** čita fajl "test" i ispisuje ga na ekranu. Maksimalna dužina linije koja se čita je 20 znakova.

```
#include<stdio.h>
#define MAXL 20
void main()
{
    FILE *pf;
    char *string[MAXL];
    pf=fopen("test","r");
    while(fgets(string, MAXL, pf)!=NULL)
        puts(string);
    fclose(pf);
}
```

# IMPLEMENTACIJA FUNKCIJA FGETS(), FPUTS I GETLINE()

*Implementacija bibliotečkih funkcija nije značajno različita od implementacije ostalih funkcija, što je ilustrovano implementacijama funkcija fgets() i fputs()*

U nastavku je data implementacija funkcije `fgets()` u obliku doslovno preuzetom iz jedne realne implementacija C biblioteke:

```
/* fgets: get at most n chars from iop */
char *fgets(char *s, int n, FILE *iop)
{
    register int c;
    register char *cs;
    cs = s;
    while (--n > 0 && (c = getc(iop)) !=
EOF)
        if ((*cs++ = c) == '\n')
            break;
    *cs = '\0';
    return (c == EOF && cs == s) ? NULL : s;
}
```

Implementacija funkcija `fputs()` ima sledeći oblik:

```
/* fputs: put string s on file iop */
int fputs(char *s, FILE *iop)
{
    register int c;
    while (c = *s++)
        putc(c, iop);
    return ferror(iop) ? EOF : 0;
}
```

Korišćenjem funkcije `fgets()` jednostavno je napraviti funkciju koja čita liniju po liniju sa standardnog ulaza, vraćajući dužinu pročitane linije, odnosno nulu kada liniju nije moguće pročitati:

```
/* getline: read a line, return length */
int getline(char *line, int max)
{
    if (fgets(line, max, stdin) == NULL)
        return 0;
    else
        return strlen(line);
}
```

# Formatirani ulaz/izlaz i datoteke

<i>datoteke, formatirani I/O, fprintf, fscanf</i>

- 
- *Formatirani ulaz i izlaz: funkcije fscanf() i fprintf()*
  - *Primeri upotrebe ulazno/izlaznih funkcija za čitanje i upis*

03

# FORMATIRANI ULAZ I IZLAZ: FUNKCIJE FSCANF() I FPRINTF()

*Za formatirani upis odnosno očitavanje podataka iz datoteka mogu se koristiti funkcije `fscanf()` i `fprintf()` koje su po prirodi identične funkcijama `scanf()` i `printf()`*

Za formatirani ulaz i izlaz mogu se koristiti funkcije `fscanf` i `fprintf`. One su identične funkcijama `scanf` i `printf`, osim što je prvi argument **FILE** pokazivač koji ukazuje na datoteku iz koje se čita, odnosno u koju se piše. Formatirajuća niska (niz karaktera) je u ovom slučaju drugi argument.

```
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)
```

U nastavku je dat primer upotrebe funkcije `fprintf()`:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    FILE * fp;

    fp = fopen ("file.txt", "w+");
    fprintf(fp, "%s %s %s %d", "We", "are", "in",
2015);

    fclose(fp);
    return(0);
}
```

Nakon izvršenja programa kreiraće se datoteka `file.txt` u kojoj će biti upisan sledeći tekst:

```
We are in 2015
```

U cilju boljeg uvida u funkciju `fscanf()` dat je i sledeći primer:

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
    char str1[10], str2[10], str3[10];
    int year;
    FILE * fp;

    fp = fopen ("file.txt", "w+");
    fputs("We are in 2012", fp);

    rewind(fp);
    fscanf(fp, "%s %s %s %d", str1, str2, str3,
&year);

    printf("Read String1 |%s|\n", str1 );
    printf("Read String2 |%s|\n", str2 );
    printf("Read String3 |%s|\n", str3 );
    printf("Read Integer |%d|\n", year );

    fclose(fp);
}
```

Rezultat programa biće:

```
Read String1 |We|
Read String2 |are|
Read String3 |in|
Read Integer |2015|
```

# PRIMERI UPOTREBE ULAZNO/IZLAZNIH FUNKCIJA ZA ČITANJE I UPIS

*Pri čitanju datoteka koristite identifikatore `r` i `r+` dok za upis podataka u datoteku koristite identifikatore `w` i `w+`*

## ❑ Upisivanje podataka u datoteku

Probajmo sledeći primer. Povedite računa da imate dostupan `/tmp` direktorijum, u suprotnom ga morate kreirati pre izvršenja programa na računaru.

```
#include <stdio.h>
void main()
{
    FILE *fp;

    fp = fopen("/tmp/test.txt", "w+");
    fprintf(fp, "This is testing for fprintf...\n");
    fputs("This is testing for fputs...\n", fp);
    fclose(fp);
}
```

Nakon izvršenja prethodnog programa kreiraće se tekstualni fajl `test.txt` u direktorijumu `/tmp` i u njemu će biti upisane dve linije korišćenjem dve različite funkcije.

## ❑ Čitanje iz datoteke

U nastavku je dat primer koji korišćenjem standardnih funkcija vrši učitavanje podataka iz već kreiranog fajla (datoteke):

```
#include <stdio.h>
void main()
{
    FILE *fp;
    char buff[255];

    fp = fopen("/tmp/test.txt", "r");
    fscanf(fp, "%s", buff);
    printf("1 : %s\n", buff );

    fgets(buff, 255, (FILE*)fp);
    printf("2: %s\n", buff );

    fgets(buff, 255, (FILE*)fp);
    printf("3: %s\n", buff );
    fclose(fp);
}
```

Nakon izvršavanja prethodnog koda biće učitani fajl kreiran u prethodnom primeru i proizvešće se sledeći rezultat:

```
1 : This
2 : is testing for fprintf...
3 : This is testing for fputs...
```

Analizirajmo malo detaljnije šta se desilo u ovom primeru. Prvi `fscanf()` metod će očitati samo reč **This** jer posle ove reči postoji praznina, Sledeći poziv je za funkciju `fgets()` koja će očitati celu liniju (dok ne naiđe na **EOL** – **end of line** karakter). Konačno poslednji poziv `fgets()` će očitati celu drugu liniju fajla.

# Sistemske funkcije za rad sa datotekama

<i>datoteke, exit, feof, ferrror, remove, rename, fseek, ftell</i>

- 
- *Osnovne sistemske funkcije*
  - *Nasumičan pristup sadržaju fajla*

03

# OSNOVNE SISTEMSKE FUNKCIJE

## *Osnovne sistemske funkcije koje olakšavaju rad sa datotekama su: `exit()`, `feof()`, `ferror()`, `remove()` i `rename()`*

U nastavku će biti ukratko opisane sistemske funkcije koje olakšavaju rad sa fajlovima kao što su: `exit()`, `feof()`, `ferror()`, `remove()` i `rename()`.

Funkcija `exit()` se koristi za prekid izvršavanja programa u slučaju greške. Ako se sa `exit()` obratimo iz funkcije koju poziva glavni program, prekida se izvršavanje celog programa, a ne samo funkcije. Ova funkcija zatvara sve fajlove otvorene sa `fopen()`. Argument funkcije ukazuje na grešku.

Funkcija `feof` vraća vrednost tačno (ne-nula) ukoliko se došlo do kraja date datoteke.

```
int feof(FILE *fp);
```

Jedan od mogućih oblik upotrebe funkcije `feof` je:

```
if( feof(fp) ) break ;
```

Funkcija `ferror` vraća vrednost tačno (ne-nula) ukoliko je došlo do greške u radu sa datotekom. Prototip funkcije je:

```
int ferror(FILE *fp)
```

Brisanje fajla je moguće uraditi korišćenjem funkcije `remove()`:

```
char buffer[101];
/* ...*/
if(remove(buffer) == 0)
    printf("File %s deleted.\n", buffer);
else
    fprintf(stderr, "Error deleting the file
%s.\n", buffer);
```

Korišćenjem funkcije `rename()` moguće je izmeniti naziv fajla:

```
#include<stdio.h>
int main(void)
{
    char buffer_old[101], buffer_new[101];

    printf("Current filename: ");
    gets_s(buffer_old, 100);

    printf("New filename: ");
    gets_s(buffer_new, 100);

    if(rename(buffer_old, buffer_new) == 0)
    {
        printf("%s has been rename
%s.\n", buffer_old, buffer_new);
    }
    else
    {
        fprintf(stderr, "Error
renaming %s.\n", buffer_old);
    }
}
```



# NASUMIČAN PRISTUP SADRŽAJU FAJLA

*Za određivanje trenutne pozicije u fajlu se koristi funkcija `ftell()` dok se za postavljanje pozicije na kojoj će se obaviti naredno čitanje/upis se koristi funkcija `fseek()`*

Slučajan ili nasumičan pristup fajlu se odnosi na mogućnost čitanja ili menjanja informacije na bilo kom mestu u otvorenom fajlu. Ovo je moguće uraditi korišćenjem indikatora koji se odnosi na poziciju u fajlu.

## ❑ Određivanje trenutne pozicije

Sledeća funkcija kao rezultat vraća trenutnu poziciju fajla kome je izvršen pristup:

```
long ftell( FILE * fp );
```

Funkcija `ftell` vraća trenutnu poziciju u datoteci (u obliku pomeraja od početka fajla izraženog u broju bajtova) na koju pokazuje `fp`. Za binarni tok (fajl), pozicija se odnosi na broj karaktera u fajlu pre trenutne pozicije, tj. pomeraj trenutnog karaktera u odnosu na početak fajla. Funkcija `ftell()` vraća `-1` ako je nastala neka greška.

## ❑ Postavljanje pozicije pristupa fajlu

```
int fseek( FILE * fp, long offset, int origin );
```

Funkcija `fseek` služi za pozicioniranje na mesto u datoteci sa koga će biti pročitano ili na koje će biti upisan sledeći podatak. Iako primena ovih funkcija nije striktno ograničena na binarne datoteke, one se najčešće koriste sa binarnim datotekama. Funkcija `fseek` kao drugi argument dobija pomeraj izražen u broju bajtova, dok su za treći argument dozvoljene vrednosti `SEEK_SET` koja označava da se pomeraj računa u odnosu na početak datoteke, `SEEK_CUR` koji označava da se pomeraj računa u odnosu na tekuću poziciju i `SEEK_END` koji označava da se pomeraj računa u odnosu na kraj datoteke. Vrednost drugog parametra, `offset`, može biti negativna. Međutim, rezultujuća pozicija u fajlu mora biti pozitivna vrednost.

Ukoliko se uspešno pozove, funkcija `fseek()` briše indikator fajla koji se odnosi na kraj fajla (`EOF`) i vraća nulu kao rezultat (vrednost različita od nule znači da se desila greška).

```
void rewind( FILE *fp );
```

Funkcija `rewind()` postavlja indikator pozicioniranja na početak fajla i briše `EOF` indikator, kao i indikator greške.

# Funkcije za rad sa binarnim datotekama

<i>Binarne datoteke, fread, fwrite</i>

---

➤ *Ulazno/izlazne funkcije za rad sa binarnim fajlovima*

03

# ULAZNO/IZLAZNE FUNKCIJE ZA RAD SA BINARNIM FAJLOVIMA

*Za čitanje podataka iz binarnog fajla se koristi funkcija fread() dok se za upis podataka u binarni fajl koristi funkcija fwrite()*

Standardna biblioteka nudi funkcije za direktno čitanje i pisanje bajtova u binarne datoteke. Programeru su na raspolaganju i funkcije za pozicioniranje u okviru datoteka, pa se binarne datoteke mogu koristiti i kao neka vrste memorije sa slobodnim pristupom. Funkcija **fread** se koristi za čitanje niza slogova iz binarne datoteke, a funkcija **fwrite** za pisanje niza slogova u binarnu datoteku:

```
size_t fread(void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);

size_t fwrite(const void *ptr, size_t size_of_elements,
             size_t number_of_elements, FILE *a_file);
```

Obe funkcije treba biti korišćenje za čitanje ili upis blokova memorije - obično nizova i struktura.

Za ove funkcije, prvi argument je adresa na koju se smeštaju pročitani slogovi iz datoteke, odnosno u kojoj su smešteni slogovi koji će biti upisani u datoteku. Drugi argument predstavlja veličina jednog sloga, treći broj slogova, a četvrti pokazivač povezane datoteke. Funkcije vraćaju broj uspešno pročitanih, odnosno upisanih slogova.

Sledeći primer ilustruje primenu ovih funkcija. U narednom primeru se deklariše struktura **rec** koja ima članice **x,y i z** of koje su celobrojnog tipa. U glavnom programu se vrši otvaranje

(**fopen**) fajla za čitanje (**w**).

```
#include<stdio.h>
struct rec
{
    int x,y,z;
};
void main()
{
    int counter;
    FILE *ptr_myfile;
    struct rec my_record;

    ptr_myfile=fopen("test.bin","wb");
    if (!ptr_myfile)
    {
        printf("Unable to open
file!");
        return 1;
    }
    for ( counter=1; counter <= 10;
counter++)
    {
        my_record.x= counter;
        fwrite(&my_record,
sizeof(struct rec), 1, ptr_myfile);
    }
    fclose(ptr_myfile);
}
```

Zatim se ispituje da li je fajl uspešno otvoren, a ako nije onda se poruka štampa korisniku i prekida se izvršenje programa. U **for** petlji se setuje parametar **x** svake od definisanih strukturnih promenljivih, a zatim se podatak o strukturi štampa u datoteku .

Na kraju se vrši zatvaranje fajla (koje se ne sme zaboraviti).

# Dijagnostika i korigovanje grešaka

<i>Analiza koda, dijagnostika koda, korigovanje grešaka</i>

- 
- *Osnovne metode za dijagnostiku*
  - *Karakteristike i prednosti procesa dijagnostike*
  - *Krajnje mere i sugestije za dobro obavljenu dijagnozu grešaka*

04

# OSNOVNE METODE ZA DIJAGNOSTIKU

## Osnovne tehnike za detektovanje grešaka su: metoda štampanja (*write*) i korišćenje alata za dijagnostiku

U programima koje pišemo obično mogu da se jave sintaksne (*compile-time*) greške i izvršne (*run-time*) greške. Izvršne greške je teže pronaći i otkloniti od sintakasnih grešaka. Mogu se koristiti dve tehnike za pronalaženje izvršnih grešaka:

- ❑ ubaciti izlazne instrukcije na ključnim tačkama programa (štampanje instrukcije), tj. korišćenje “**WRITE tehnika**”
- ❑ koristiti program koji se zove “*debugger*“, koji omogućuje kontrolisanje izvršavanja vašeg programa dok se izvršava, i na taj način se vrši pronalaženje grešaka (*buggs*)

Dijagnostika i korigovanje greške, *debugging*, je proces identifikacije uzroka tj. korena neke greške, i samo korigovanje te greške. Kod nekih projekata, *debugging* (korigovanje grešaka, debugiranje) nosi 50% od ukupnog vremena razvoja softvera. Za mnoge programere *korigovanje grešaka* je najteži deo programiranja. Međutim, debugiranje se može načiniti mnogo lakšim ako se koriste određene tehnike koje će biti opisane u ovom poglavlju. Softverski defekti (*bugs*) su greške u softveru odnosno greške programera (*errors, defects, faults*).

Debugiranje omogućuje dijagnozu defekata. Međutim, nisu svi programeri vešti u programiranju. Najbolji dijagnostičari mogu biti tri puta brži od onih najlošijih. Takođe kod korigovanja pronađenih defekata, može često doći do pravljenja novih defekata. Na slici ispod je dato jedno poređenje između jednog najboljeg i jednog najlošijeg programera, za jedan slučaj gde je bilo 12 defekata u softveru. Ovo nisu statistički podaci, već samo jedan primer, ali ilustrativan primer:

	Najbolji programer	Najlošiji programer
Vreme dedefektovanja u minutima	5	15
Broj nepronadjenih defkata	0	4 od 12
Dodati novi defekti	0	11

Slika-1 Poređenje dobrog i lošeg programera u fazi dijagnostike i korekcije grešaka

Ovo govori da je dijagnoza greške disciplina kojoj treba posvetiti veliku pažnju, pa se stoga programeri dodatno obučavati u toj oblasti

# KARAKTERISTIKE I PREDNOSTI PROCESA DIJAGNOSTIKE

*Iako je dijagnostika greške ponekada težak proces za programere, to je i prilika da se nauči puno toga: o samom programu, o algoritmu, kao i o tipovima i uzrocima grešaka koje se prave*

Nije dovoljno reći da se dijagnoza greški može obaviti pomoću ubacivanja serije instrukcija štampanja u program, u cilju nalaženja greške. Međutim, iako je otklanjanje grešaka teško za programere, ono je veoma korisno i predstavlja i pogodnost tj. priliku za nešto korisno, naime:

- ako je napravljena greška u programu, to je prilika da se nauči nešto o tom programu.
- takođe to je prilika da se nauči o tipovima i uzrocima grešaka koje se prave.
- to je prilika da se nauče tehnike dijagnoze greški, i
- prilika da se nauče tehnike korigovanja greški, gde se traži tačna dijagnoza i lečenje uzroka a ne simptoma problema

Neefikasni prilazi dijagnozi greške su:

- pronalaženje greške pogađanjem, pomoću slučajno tj. haotično izabranih instrukcija štampanja u programu, i pomoću slučajnih tj. haotičnih izmena u programu
- izbegavanje da se udubi duboko u problem, već se površno pokušava ispraviti problem

Dijagnoza i korigovanje greške sastoji se od nalaženja greške i popravke greške. Međutim, nalaženje greške i razumevanje uzroka greške je obično 90% posla.

# KRAJNJE MERE I SUGESTIJE ZA DOBRO OBAVLJENU DIJAGNOZU GREŠAKA

*Ponekad je potrebno primeniti krajnje mere, koje će svakako rešiti problem dijagnoze greške ako se nije uspelo pomoću testova i hipoteza o grešci*

## Krajnje mere za dijagnozu greške

Ponekad je potrebno primeniti krajnje mere, koje će svakako rešiti problem dijagnoze greške ako se nije uspelo pomoću testova i hipoteza o grešci. Ove krajnje mere su:

- uraditi kompletnu proveru dizajna i kodiranja dela programa koji ne radi dobro
- odbaciti deo programa koji ne radi dobro, i ponovo ga iz početka dizajnirati i kodirati
- odbaciti ceo program i ponovo ga dizajnirati i kodirati iz početka
- uraditi kompilaciju sa kompletnom informacijom o debugiranju
- pooštriti testiranje softverskih jedinica, i testirati softverske jedinice u izolaciji
- koristiti **debugger** da se provere široki opsezi test parametara
- promeniti kompajler
- opremiti program sa štampanjima i prikazima.
- kompilaciju i izvršavanje programa obaviti negde drugde, u drugom okruženju
- integrisati program u malim komadima, i kompletno testirati svaki taj komad.

## Korigovanje grešaka:

Teže je pronaći grešku nego korigovati grešku. Medjutim, kod korigovanja greške često se prave nove greške. Sledeće su sugestije za izbegavanje pravljenja novih greški kod korigovanja postojećih greški:

- pre korigovanja potrebno je razumeti problem, toliko razumeti npr. da se može predvideti svako pojavljivanje te greške
- takodje, razumeti ceo program, tj. kontekst problema, jer onda se može rešiti problem kompletno a ne delimično
- potvrditi da je dijagnoza korektna, pre nego se pristupi korigovanju greške, tj. proveriti ispravnost hipoteze greške, odnosno obaroriti ostale hipoteze koje nisu tačne
- memorisati originalni izvorni kod, u cilju poređenja sa novim izvornim kodom
- korigovati problem, a ne samo simptom.
- menjati program samo ako ste sigurni da je ispravna promena, jer ako se program menja po sistemu probe i greške, onda to nije uopšte efikasan metod
- vršiti jednu po jednu promenu programa, a ne više njih istovremeno, jer je to vrlo komplikovano i izaziva nove greške
- proveriti izvršene korekcije

# Alati za dijagnostiku grešaka

<i>dijagnostika grešaka, IDE za dijagnostiku, debugger</i>

- 
- *Osnovne opcije alata za dijagnostiku*
  - *Prednosti i mane alata za dijagnostiku*

05



# OSNOVNE OPCIJE ALATA ZA DIJAGNOSTIKU

*Kod velikih programa sa mnogo linija koda metoda štampanja ne daje dobre rezultate ali je tu za vas neki od alata za dijagnostiku*

Za male programme, **WRITE** tehnika je uspešna. Ali, kod velikih programa ova tehnika nije efikasna. Kod velikih programa programer ne zna gde da postavi izlazne (štampanje) instrukcije. Proces dodavanja štampanja, pa kompilacija, pa egzekucija je vrlo dosadan tj. spor i neefikasan proces. Takođe, kod problema sa pokazivačima tj. "pointerima" (*pointers*), pronalaženje greške je posebno teško jer pokazivač prikazan u heksadecimalnom formatu ne znači ništa. Zato, umesto **WRITE** tehnike može se primeniti specijalni program koji se zove *debugger*, ali je pre toga potrebno naučiti kako se koristi ovaj program.

U sledećoj tabeli date su komande koje se mogu koristiti u *debugger* programu. Ove komande su raspoložive u okviru **Visual C++**, **Netbeans IDE** i **Code::Blocks** okruženja:

Start executing in debugger	startovanje programa pomoću <i>debugger-a</i>
Step in:	izvršavanje jedne instrukcije tj. <i>single-stepping</i> (koračanje)
Step over (Next step):	preskakanje metode koračanja
Continue:	nastaviti izvršavanje programa
View variable:	prikazivanje promenljive
Set breakpoint:	zaustavljanje programa u određenoj instrukciji
Add watch:	prikazati promenljivu kad se program zaustavi
Program reset:	startovanje programa iz početka pomoću <i>debugger-a</i>

Slika-1 Osnovne opcije alata za korigovanje grešaka u uobičajenom integrisanom razvojnom okruženju (IDE)

# PREDNOSTI I MANE ALATA ZA DIJAGNOSTIKU

*Alati za dijagnostiku su korisno sredstvo ali pod uslovom da programer ima dobru kontrolu nad kodom koji testira i da je svestan tipa problema koji je nastao*

Programi za dijagnostiku greške, *debuggers*, omogućuju:

- da se postave zaustavne tačke u programu, da se program jednostavno zaustavi kod određene linije u programu ili da se zaustavi kada se nekoj promenljivoj zadaje vrednost
- omogućuju izvršavanje programa liniju po liniju, tzv. koračanje kroz program ili podprogram, ili preskakanje podprograma
- omogućuju izvršavanje programa unazad do tačke gde je defekt započeo.
- takodje omogućuju štampanje raznih iskaza u toku izvršavanja programa, npr 'pogledaj ovde' ili slično
- pomoću njih se mogu ispitivati podaci u programu , npr. dinamički alocirana matrica podataka.

Postoje kontroverzna mišljenja o dijagnostičkim programima. Neki programeri su protiv njihove upotrebe, argumentujući da se defekti brže i tačnije otkrivaju pomoću razmišljanja i tehnika opisanih u prethodnim glavama teksta, i da se mentalnom egzekucijom programa, a ne dijagnostičkim programom, trebaju otkrivati defekti. Međutim, ovo mišljenje da treba odbaciti dijagnostičke programe nije u redu, preterano je. Ali isto tako, treba reći da se svako oruđe može koristiti ispod ili iznad svojih mogućnosti, i to važi i za dijagnostičke programe.

Dakle, ne možemo se potpuno osloniti na dijagnostičke programe misleći da će oni umesto vas otkriti greške. Naime, dijagnostički program je samo jedno pomoćno sredstvo, i on nije zamena za razmišljanje kao glavni alat dijagnostike greške. Ali ni razmišljanje nije totalna zamena za dijagnostički program. Kod interaktivne dijagnostike se koristi *debugger* da se menjaju razni parametri u programu u toku izvršavanja programa. Međutim, interaktivni *debugger* ima nedostatak a to je da ohrabruje prilaz pokušaj-i-pogreši, *trial-and-error*, tj haotično otkrivanje greški umesto sistematskog prilaza.

# Uvod u C++

<i>C++, sintaksa, tipovi podataka, deklaracija promenljivih</i>

- 
- *Rezervisane reči; Logički tip podatka*
  - *Prvi C++ program*
  - *Pregled tipova podataka; Lokalne i globalne promenljive*
  - *Modifikator const, razmeštanje opisa promenljivih, konverzija podataka*
  - *Komentari u C++-u*

06

# REZERVISANE REČI; LOGIČKI TIP PODATKA

*C++ ima dodatne rezervisane reči u odnosu na C, a takođe ima i definisan logički tip podatka `bool` koji u C-u ne postoji*

U jeziku C++ kao dodatak u odnosu na jezik C imamo sledeće najčešće korišćene službene reči:

<code>asm</code>	<code>bool</code>	<code>catch</code>	<code>char</code>	<code>class</code>	<code>delete</code>
<code>explicit</code>	<code>export</code>	<code>false</code>	<code>friend</code>	<code>inline</code>	<code>mutable</code>
<code>namespace</code>	<code>new</code>	<code>operator</code>	<code>private</code>	<code>protected</code>	<code>public</code>
<code>template</code>	<code>this</code>	<code>throw</code>	<code>true</code>	<code>try</code>	<code>typename</code>
<code>using</code>	<code>virtual</code>				

C++ dozvoljava deklaraciju promenljivih bilo gde u programu, ne samo na početku bloka. Najčešće je to pre prvog pojavljivanja promenljive.

## ❑ Logički tip `bool`

U C++-u za veličine logičkog tipa mogu se umesto 0 i 1 koristiti vrednosti `true` (tačno) i `false` (netačno). Logička konstanta `false` je jednaka nuli, a svaka druga vrednost se tretira kao tačna. Kada se `true` konvertuje u ceo broj rezultat je 1. Za opis promenljivih logičkog tipa koristi se rezervisana reč `bool`.

# PRVI C++ PROGRAM

*U C++-u imamo, kao dodatak u odnosu na C, tokove za rad sa ulazom i izlazom*

Analizirajmo sledeći primer u programskom jeziku C++. Linija 1 je specijalan tip iskaza koji se naziva predprocesorska naredba ili direktiva. U ovom slučaju, mi govorimo kompajleru da želimo da koristimo `iostream` biblioteku. Biblioteka `iostream` sadrži deo koda koji opisuje kompajleru šta operacije `cout` i `endl` znače. Drugim rečima, u C++ je neophodno uključiti biblioteku `iostream` u kojoj se nalaze funkcije za štampanje podataka na ekranu. U liniji 3 se deklarise `main()` funkcija, kao u prethodno pokazanim C primerima.

```
1 #include <iostream>
2
3 int main()
4 {
5     using namespace std;
6     cout << "Hello world!" << endl;
7     return 0;
8 }
```

Linije 4 i 8 ukazuju kompajleru koje linije pripadaju `main` funkciji. Sve linije koda koje se nalaze između otvorene vitičaste zagrade u liniji 4, i zatvorene vitičaste zagrade u liniji 8 predstavljaju deo `main()` funkcije. U liniji 5 je naš prvi iskaz. Opis operacija `cout` i `endl` se nalazi u okviru biblioteke `iostream`. Međutim, u okviru biblioteke `iostream`, postoji tačno određeni imenski prostor `std` u okviru koga je navedena definicija ovih funkcija za štampanje. Stoga je neophodno korišćenjem iskaza `using` ukazati kompajleru da treba da pogleda unutar odeljka sa nazivom `std` (skraćeni za `standard`) i da pokuša da pronađe definiciju za `cout` i `endl`, ukoliko prilikom kompajliranja nije uspeo nigde drugde da pronađe ove definicije. Ovaj iskaz je neophodan kompajleru da bi uspeo lakše da pronađe definiciju operacija `cout` i `endl`, koje koristimo u liniji 6. U liniji 6 se nalazi iskaz štampanja. `cout` je specijalni objekat koji predstavlja konzolu/ekran. Simbol `<<` je operator izlaza, koji ukazuje kompajleru da ono što sledi nakon simbola treba biti oštampano na ekranu/konzoli. `endl` je specijalan symbol koji kursor prebacuje u sledeći red konzole. Linija 7 je naredba povratka (`return statement`). Nakon završetka rada programa, šalje se povratna vrednost sistemu koja opisuje da li program izvršen uspešno ili ne. U ovom konkretnom primeru, naredbom `return` se vraća vrednost nula operativnom sistemu što znači da je "sve izvršeno propisno!". Praksa je da se koriste nenulte vrednosti kada je nešto pošlo po zlu, odnosno kada program zbog nekih problema treba da prekine sa radom.

# PREGLED TIPOVA PODATAKA; LOKALNE I GLOBALNE PROMENLJIVE

*C++ ima i logički tip podatka bool kao dodatak u odnosu C. Takođe ima mogućnost pristupa globalnoj promenljivoj istog naziva u unutrašnjem bloku korišćenjem operatora ::*

- ❑ Tipovi podataka. Za razliku od C-a, C++ ima kao dodatak sledeće tipove podataka:

Tip	Ključna reč
Logički tip	bool
Prošireni karakter	wchar_t

Slika-1 Dodatni tipovi podataka u C++ -u o odnosu na C

U sledećoj tabeli je dat kompletan prikaz tipa promenljive u jeziku C++, koliko memorije zauzima, i koja je minimalna odnosno maksimalna vrednost koja staje u odgovarajući tip.

Tip	Memorija	Opseg vrednosti
char	1byte	-127 to 127 or 0 to 255
unsigned char	1byte	0 to 255
signed char	1byte	-127 to 127
int	4bytes	-2147483648 to 2147483647
unsigned int	4bytes	0 to 4294967295
signed int	4bytes	-2147483648 to 2147483647
short int	2bytes	-32768 to 32767
unsigned short	2 bytes	0 to 65,535
signed short	2 bytes	-32768 to 32767
long int	4bytes	-2,147,483,647 to 2,147,483,647
signed long int	4bytes	Isto kao long int
unsigned long int	4bytes	0 to 4,294,967,295
float	4bytes	+/- 3.4e +/- 38 (~7 cifara)
double	8bytes	+/- 1.7e +/- 308 (~15 cifara)
long double	8bytes	+/- 1.7e +/- 308 (~15 cifara)
wchar_t	2 or 4 bytes	1 prošireni karakter

Slika-2 Tipovi, memorija i opseg vrednosti podataka u C++-u

- ❑ Lokalne i globalne promenljive

U lekcijama o C-u smo pričali o oblasti vidljivosti promenljive i rekli smo da one mogu biti globalne i lokalne. Takođe smo rekli da u slučaju da u programu imamo lokalnu i spoljašnju promenljivu istog naziva onda je u bloku u kome je deklarirana lokalna promenljiva ona i ta koja je vidljiva. Tada je spoljašnja promenljiva nedostupna iako joj se doseg proteže i kroz taj deo programa. Globalnoj promenljivoj se tada može pristupiti korišćenjem operatora za razrešenje oblasti vidljivosti, odnosno takozvanog „scope resolution“ operatora tj. dvostruke dvotačke (::). Ovaj operator omogućava korišćenje globalne promenljive i u slučaju da je zaklanja lokalna promenljiva istog naziva. Pogledajmo primer:

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

char c = 'a'; // global variable

int main() {
    char c = 'b'; //local variable

    cout << "Local c: " << c << "\n";
    cout << "Global c: " << ::c << "\n"; //using scope
    resolution operator

    return 0;
}
```

# MODIFIKATOR CONST, RAZMEŠTANJE OPISA PROMENLJIVIH, KONVERZIJA PODATAKA

*C++ omogućava deklaraciju promenljivih bilo gde u okviru bloka a ne samo na početku. Takođe omogućava eksplicitnu konverziju u takozvanom funkcijskom obliku*

## ❑ Modifikator const

Slično statičkim promenljivama, promenljive opisane kao `const` nisu dostupne u drugim fajlovima projekta. C++ kompajler najčešće promenljivu opisanu kao `const` tretira kao konstantu deklarisanu direktivom `#define`, tj. prosto vrši zamenu promenljive vrednošću kojom je inicijalizovana. Prednost upotrebe `const` je kontrola slaganja tipova što može preduprediti mnoge probleme.

## ❑ Razmeštanja opisa promenljivih unutar bloka

U jeziku C morate sve opise lokalnih promenljivih unutar bloka smestiti pre prvog operatora. U C++-u nemate takvo ograničenje. Tako, promenljive možete opisati u bilo kojoj liniji bloka, ali pre prvog korišćenja. Promenljive možete opisati i u delu za inicijalizaciju operatora `for`:

```
for(int i =0; i < MAX; i++)
```

U ovom slučaju treba napomenuti da je promenljiva `i` vidljiva i nakon završetka `for` petlje.

## ❑ Eksplicitna konverzija podataka.

Eksplicitna konverzija u C-u se piše u obliku:

*(tip) <izraz>*, odnosno:

```
x = (int )3.5 + (int) 5.6;
```

U C++-u osim ovog moguće je i takozvani funkcijski oblik:

*tip(<izraz>)*, odnosno:

```
x = int (3.5) + int (5.6);
```

# KOMENTARI U C++-U

*C++ kao dodatak u odnosu na C ima jednolinijski komentar, koji je standardan za sve C++ kompajlere*

Komentari predstavljaju pomoćni tekst u C/C++ programu koji se ignoriše od strane kompajlera. Komentarom se u C-u smatra svaki niz simbola koji počinje parom znakova `/*` a završava se znakovima `*/` kao što je prikazano u nastavku:

```
/* my first program in C */
```

U C++ u postoji još jedan oblik komentara koji se naziva jednolinijski komentar. On počinje simbolom `//` i proteže se do kraja linije, tj ukazuje da kompajler treba da ignoriše sve što se nalazi od simbola do kraja linije. Na primer:

```
cout << "Hello world!" << endl; // Everything from here to the right is ignored.
```

Običaj je da se jednolinijski komentari koriste sa ciljem da se na brz način opiše jedna od linija koda.

```
cout << "Hello world!" << endl; // cout and endl live in the iostream library  
cout << "It is very nice to meet you!" << endl; // these comments make the code hard to read  
cout << "Yeah!" << endl; // especially when lines are different lengths
```



# Standardni ulaz i izlaz u C++-u

*standardni tokovi, klase za I/O, funkcija get, funkcija getline()*

- 
- *Ulaz/ izlaz u C++-u (tokovi – streams)*
  - *Ulaz/ izlaz u C++-u – osnovni primer*
  - *Korišćenje cin i cout objekta kod nizova*
  - *Ulazne funkcije get() i getline()*
  - *Funkcija ignore()*
  - *Pravila kod učitavanja karaktera i problema sa*

07

# ULAZ/ IZLAZ U C++-U (TOKOVI – STREAMS)

*Osnovne klase u C++-u koje obezbeđuju rad sa ulazno/izlaznim operacijama su: `istream`, `ostream` i `iostream`*

Kao i kod programskog jezika C, ulazno izlazna funkcionalnost nije ni deo C++ jezika, ali je omogućena korišćenjem standardne biblioteke (u okviru imenskog prostora `std`). U nekom od prethodnih primera je pokazano uključivanje biblioteke `iostream` sa ciljem da se koriste standardni objekti `cin` i `cout` kao prost ulaz/izlaz. U okviru ove lekcije biće detaljnije obrađena biblioteka `iostream`. Pri uključivanju zaglavlja `iostream`, programer dobija pristup celoj hijerarhiji klasa koje obezbedju I/O funkcionalnost (uključivanje klase koja se naziva `istream`).

Klasa `istream` je osnovna klasa za rad sa ulaznim tokovima. Kod ulaznih tokova, operator izvlačenja (`>>`) se koristi za izvlačenje neke vrednosti iz toka (`stream`-a). Na primer, kada korisnik pritisne dugme na tastaturi, odgovarajući znak ulazi u tok (`stream`), a zatim program korišćenjem odgovarajućeg objekata (`cin`) i operatora (u ovom slučaju `>>`) može da pokupi ono što se nalazi u toku. Za razliku od klase `istream`, klasa `ostream` je osnovna klasa za rad sa izlaznim tokovima. Kod ulaznih tokova, operator ubacivanja (`<<`) se koristi za ubacivanje nekog podatka u tok. Na primer, nakon što programer/korisnik ubaci novu vrednost u tok podataka on će se prikazati u konzoli (na monitoru, koji u ovom slučaju vrši ulogu korisnika). Konačno, imamo klasu `iostream` koja omogućava rad sa ulazno/izlaznim funkcijama i predstavlja neku vrstu integracije funkcija koje su obuhvaćene klasama `istream` i `ostream`.

C++ dolazi u paketu sa četiri objekta za rad sa standardnim tokovima, od kojih smo prva dva već imali prilike da vidimo u prethodnim primerima:

- ❑ `cin` — standardni ulaz (obično je to tastatura)
- ❑ `cout` — standardni izlaz (obično je to monitor)
- ❑ `cerr` — standardna greška (obično je to tastatura), obezbeđuje izlaz bez bafera (`buffer`)
- ❑ `clog` — standardna greška (obično je to monitor), obezbeđuje izlaz sa baferom

# ULAZ/ IZLAZ U C++-U – OSNOVNI PRIMER

*Da bi smo mogli da koristimo ulazno/izlazne funkcije u C++-u neophodno je pretprocesorkom naredbom uključiti klasu koja obezbeđuje ovu funkcionalnost ( `#include <iostream >` )*

Pošto smo uključili biblioteku `'iostream'`, onda možemo da koristimo operator `<<`, da uputimo izlazne podatke na ekran tj. konzolu. Ovaj operator se još zove i „insertor”, *inserter*, – on insertuje na ekran šta god da je iza instrukcije *cout*, npr.

```
cout << initTime;
```

Ova instrukcija šalje sadržaj promenljive `'initTime'` na odgovarajući ‘prozor’ na ekranu. Naravno, možemo niz promenljivih da uputimo na ekran sa samo jednom instrukcijom, npr.

```
cout << initTime << endl << finalTime << endl;
```

Takođe, možemo da pomoću *cout* da šampamo tekst na ekran, koristeći operator navođenja: `“`, tj. stavljajući tekst između ova dva operatora: `“ ...”`, npr.

```
cout << “The initial time: “ << initTime << endl;
```

U nastavku je dat osnovni primer korišćenja ulazno/izlaznih funkcija u C++-u:

```
#include <iostream>

int main()
{
    using namespace std;
    // First we'll use the insertion operator on cout to print text to the monitor
    cout << "Enter your age: " << endl;

    // Then we'll use the extraction operator on cin to get input from the user
    int nAge;
    cin >> nAge;

    if (nAge <= 0)
    {
        // In this case we'll use the insertion operator on cerr to print an error message
        cerr << "Oops, you entered an invalid age!" << endl;
        exit(1);
    }
}
```

# KORIŠĆENJE CIN I COUT OBJEKTA KOD NIZOVA

*Operator cin ima mogućnost da očita ceo niz karaktera odjednom dok, suprotno, operator cout ima mogućnost da oštampa ceo niz karakter na ekran odjednom*

Korišćenjem **cin** objekta moguće je dodeliti vrednost elementima niza na dva načina. Prvi način je da se svakom članu niza pojedinačno dodeli neka vrednost, što je ilustrovano sledećim primerom:

```
#include <iostream>
using namespace std;
const int MAX = 3;
int main ()
{
    char grades[MAX];
    for (int i = 0; i < MAX; i++)
    {
        cout << "Enter grade for test #" << i + 1 <<
        ":";
        cin >> grades[i];
    }
    return 0;
}
```

Ova tehnika je logičan izbor kada se svaki element niza razlikuje od drugog, kao što je to slučaj u prethodnom primeru gde imamo različite ocene (od A do E) za svaki od tekstova. Ponekad su, međutim, članovi niza povezani, kao što je slučaj sa nizom karaktera koji predstavlja ime neke osobe.

Ono što ovde treba napomenuti je da postoje bitne razlike između niza karaktera i niza numeričkih tipova podataka kad je u pitanju upotreba **cin** i **cout** objekta.

Naime, **cin** objekat u sklopu sa operatorom izvlačenja **>>** ima mogućnost da dodeli vrednost svim članovima nekog niza karaktera. Takođe, **cout** objekat u sklopu sa operatorom ubacivanja **<<** može istovremeno da oštampa sve elemente niza, što ilustruje sledeći program:

```
#include <iostream>
using namespace std;
int main ()
{
    char name[ 80] = {'J', 'e', 'f', 'f', '/0' };
    cout << "Enter your name: ";
    cin >> name;
    cout << "Your name is " << name;
    return 0;
}
```

Mogući ulaz i izlaz programa bi imao sledeći oblik:

```
Enter your name: Jeff
Your name is Jeff
```

Ovaj pristup ima prednosti jer ne zahteva korišćenje petlji za unos i štampanje podataka. Operacija dodele vrednosti se vrši u jednom koraku korišćenjem **cin** objekta u sklopu sa operatorom izvlačenja iz toka **>>**. Na sličan način, štampanje se vrši u jednom koraku korišćenjem **cout** objekta u sklopu sa operatorom ubacivanja u tok **<<**.

# ULAZNE FUNKCIJE GET() I GETLINE()

*Funkcija `get()` se koristi za unos karaktera sa ulaznog toka, dok se funkcija `getline()` koristi za unos teksta*

U okviru `iostream`-a postoji veliki broj funkcija za olakšano učitavanje podataka. Jedna od najčešće korišćenih je funkcija `get()` koja jednostavno hvata karakter sa ulaznog toka (`stream`-a). U nastavku je jedan prost primer korišćenja funkcije `get()`:

```
#include <iostream>
int main()
{
    char ch;
    while (cin.get(ch))
        cout << ch;

    return 0;
}
```

Funkcija `get()` ima i verziju za rad sa stringovima koja osim imena stringa (teksta) kao argument sadrži i maksimalni broj karaktera koje čitamo u jednom pozivu funkcije:

```
#include <iostream>
int main()
{
    char strBuf[11];
    cin.get(strBuf, 11);
    cout << strBuf << endl;

    return 0;
}
```

Osim funkcije `get()`, postoji i funkcija `getline()` koja čita celu liniju do prelaska u sledeći red.

```
#include <iostream>
int main()
{
    char strBuf[11];
    // Read up to 10 characters
    cin.getline(strBuf, 11);
    cout << strBuf << endl;

    // Read up to 10 more characters
    cin.getline(strBuf, 11);
    cout << strBuf << endl;
    return 0;
}
```

Funkcija `getline()` je dobra zamena za učitavanje stringova korišćenjem `cin` objekta. Nju ima smisla koristiti kada se tekst koji unosite sastoji iz praznina, kao u sledećem slučaju

```
Enter your name: Jeff Kent
```

S obzirom da postoji praznina između reči, vrednost promenljive koja služi za prihvatanje teksta korišćenjem `cin` objekta u sklopu sa operatorom izvlačenja `>>` će biti samo „Jeff“. Osim funkcije `getline()` moguće je koristiti predefinisani oblik funkcije `get()`:

```
cin.get(name, 80, '\n');
```

# FUNKCIJA IGNORE()

*Funkcija ignore() se koristi sa ciljem da se iz ulaznog bafera učita karakter koji će biti ignorisan. Najčešće se koristi kod unosa karaktera za kojim sledi pritisak tastera ENTER*

Pretpostavimo da imao sledeći program koji se odnosi na unos karaktera i ispitivanje uslova da li ili ne želite da nastavite sa izvršavanjem vašeg program, što će naravno biti čest slučaj tokov vašeg programerskog poziva. U nastavku je dat primer gde se učitavanje karaktera vrši korišćenjem `cin.get(ch)` funkcije:

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

int main(void)
{
    char ch;
    do {
        cout << "Press Q or q to
quit, any other key to continue: ";
        cin.get(ch);
        if (ch != 'Q' && ch != 'q')
            cout << "You
want to continue?\n";
        else
            cout << "You
quit";
    } while (ch != 'Q' && ch != 'q');
    return 0;
}
```

Problema u ovom primeru nastaje kada nakon odgovarajućeg karaktera pritisnete taster **ENTER**. Naime, pritiskom na **ENTER** u bafer se upisuje i karakter koji se odnosi na novu liniju `'\n'`.

Pretpostavimo da smo kao prvi karakter uneli **x** a zatim pritisnuli **ENTER**. U prvom prolazu kroz petlju `do while`, funkcija `cin.get()` će očitati karakter **x**, dok će u drugom prolasku kroz petlju biti očitani karakter nove linije `'\n'` ne dozvoljavajući nam da unesemo karakter po želji.

Rešenje je da se obriše karakter nove linije iz ulaznog bafera pre poziva `get` funkcije. To se može uraditi korišćenjem funkcije `ignore` koja je funkcija članica `cin` objekta.

Poziv funkcije `ignore` koja nema nijedan argument će omogućiti da se sledeći karakter iz ulaznog bafera očita ali i ignoriše, što znači da neće biti dodeljen niti jednoj promenljivoj. To je upravo ono što smo i želeli. Moguće je koristiti funkciju `get()` bez argumenata i imaće isti efekat kao i korišćenje funkcije `ignore()` bez argumenata. Stoga će sledeća dva iskaza proizvesti isti rezultat:

```
cin.ignore();
cin.get();
```

# PRAVILA KOD UČITAVANJA KARAKTERA I PROBLEMA SA KARAKTEROM NOVE LINIJE

*Unos karaktera u program može uneti dosta zabune i problema, naročito kada za unosom sledi pritisnut taster ENTER pa je stoga korisno poznavati sledeća pravila*

U nastavku su data tri prosta pravila kojih se treba pridržavati da bi se izbegli problemi izazvani karakterom nove linije koji ostaje u ulaznom baferu (pretpostavka je da ne postoji dobar programerski razlog da se karakter nove linije ostavi u ulaznom baferu.).

## □ Pravilo #1:

Nakon učitavanja karaktera korišćenjem `cin` objekta i operatora izvlačenja iz toka (`>>`) treba da sledi poziv funkcije `ignore`. Naime, objekat `cin` sa operatorom ekstrakcije (izvlačenja iz toka, `>>`) uvek ostavlja karakter nove linije u ulaznom baferu pa ga je stoga neophodno obrisati korišćenjem funkcije `ignore`. Na primer:

```
char ch;
cin >> ch;
cin.ignore();
```

## □ Pravilo #2:

Nakon upotrebe funkcije `getline` nikako ne treba da sledi poziv funkcije `ignore` bez argumenata. Naime, funkcija `getline` po prirodi čita celu liniju zajedno sa karakterom nove linije, tako da njega briše iz ulaznog bafera, pa stoga ne treba koristiti funkciju `ignore` nakon `getline`.

## □ Pravilo #3:

Nakon korišćenja funkcije `get` koja ima jedan argumenta, kao npr `cin.get(ch)`, treba proveriti da li se karakter nove linije nalazi u ulaznom baferu. Ukoliko je tako onda treba koristiti funkciju `ignore`, u suprotnom ne. Naime, funkcija `get` koja ima jedan argument, i to karakter, će ostaviti karakter `'\n'` u ulaznom baferu ukoliko unesete karakter i pritisnete `ENTER` ali neće ako pritisnete samo `ENTER`, Stoga je poželjno ispitati da li je potrebno obrisati bafer. Na primer:

```
char ch;
cin.get(ch);
if (ch != '\n')
    cin.ignore();
```

# Formatiranje izlaznih podataka u C++-u

*funkcije i manipulatori, širina polja, poravnanje, popunjavanje*

- 
- *Formatiranje pri štampanju korišćenjem ostream i ios*
  - *Primer formatiranja korišćenjem ostream-a i ios-a*
  - *Podešavanje širine polja, praznih polja i poravnavanje*
  - *Primer podešavanja širine polja, ispunje praznih polja i poravnavanja*

08



# FORMATIRANJE PRI ŠTAMPANJU KORIŠĆENJEM OSTREAM I IOS

*Funkcije i manipulatori jezika C++ za formatiranje izlaza su definisani u okviru klasa ostream i ios. Neophodno je uključiti ove klase u program radi korišćenja pomenutih opcija*

Korišćenjem standardnih manipulatora, moguće je promeniti preciznost i format pri štampanju decimalnih brojeva. Postoji nekoliko opcija formatiranja koje se mogu koristiti samostalno ili u kombinaciji sa drugim, pa ćemo pokazati neki od primera.

Neki od manipulatora koji će često biti korišćeni su:

- ❑ **fixed** - koristi decimalni zapis za štampanje vrednosti
- ❑ **scientific** - koristi eksponencijalni zapis za štampanje vrednosti
- ❑ **setprecision(int)** – podešava preciznost za ispis realnih brojeva (definisan u **iosmanip.h**)

U slučaju upotrebe fiksnog ili eksponencijalnog zapisa, preciznost se odnosi na broj decimalnih mesta koja se koriste za ispis decimalnog dela. U slučaju da je preciznost manja od broja cifara, izvršiće se zaokruživanje broja.

# PRIMER FORMATIRANJA KORIŠĆENJEM OSTREAM-A I IOS-A

*Preciznost pri štampi realnih brojeva u C++-u se reguliše korišćenjem funkcije `setprecision()`*

U nastavku je dat primer formatiranja štampe relanih brojeva korišćenjam manipulatora `fixed` i `setprecision()`:

```
cout << fixed << endl;
cout << setprecision(3) << 123.456 << endl;
cout << setprecision(4) << 123.456 << endl;
cout << setprecision(5) << 123.456 << endl;
cout << setprecision(6) << 123.456 << endl;
cout << setprecision(7) << 123.456 << endl;

cout << scientific << endl;
cout << setprecision(3) << 123.456 << endl;
cout << setprecision(4) << 123.456 << endl;
cout << setprecision(5) << 123.456 << endl;
cout << setprecision(6) << 123.456 << endl;
cout << setprecision(7) << 123.456 << endl;
```

Rezultat prethodnog koda je:

```
123.456
123.4560
123.45600
123.456000
123.4560000

1.235e+002
1.2346e+002
1.23456e+002
1.234560e+002
1.2345600e+002.
```

# PODEŠAVANJE ŠIRINE POLJA, PRAZNIH POLJA I PORAVNAVANJE

*Korišćenjem manipulatora i funkcija članica moguće je u programskom jeziku C++ formatirati tekst da bude u pogodnoj formi oštampan na ekranu ili u datoteci*

Obično, kada se štampaju brojevi, ne vodi se računa o prostoru koji će da okružuje oštampana polja. Međutim, moguće je vršiti različita poravnavanja pri štampanju brojeva. Da bi smo opisali manipulatore za poravnavanje teksta, neophodno je da prvo uvedemo pojam širine polja. Širina polja (**width**) predstavlja broj ostavljenih mesta za štampanje promenljive. Ukoliko broj koji štampamo ima manje cifara od trenutne širine polja, on će prilikom štampanja biti poravnat sa desne ili leve strane (u zavisnosti kako je podešeno). Ukoliko je broj cifara vrednosti koju štampamo veći od širine polja, u tom slučaju ne dolazi do odsecanja broja već do automatskog povećanja širine polja štampe.

## Manipulatori:

- ❑ **internal** - Levo poravnanje znaka, i desno poravnanje vrednosti
- ❑ **left** - Levo poravnanje znaka (+/-) i vrednosti
- ❑ **right** - Desno poravnanje znaka (+/-) i vrednosti
- ❑ **setfill(char)** - Postaviti karakter koji ispunjava prazna polja u okviru formata za štampanje (definisan u **io manip.h**)
- ❑ **setw(int)** - Postaviti šitinu polja za unos i ispis parametara (definisan u **io manip.h**)

## Funkcije članice:

- ❑ **fill()** - Vraća trenutni karakter za ispunjavanje praznina
- ❑ **fill(char)** - Postavlja novi a vraća kao rezultat stari karakter za ispunjavanje
- ❑ **width()** - Vraća trenutnu širinu polja
- ❑ **width(int)** - Postavlja novu a vraća kao rezultat staru širinu polja

# PRIMER PODEŠAVANJA ŠIRINE POLJA, ISPUNE PRAZNIH POLJA I PORAVNAVANJA

*U cilju korišćenja prethodnih manipulatora za ispis teksta, neophodno je pre toga podesiti širinu polja*

Ovo je moguće uraditi ili korišćenjem funkcije `width(int)` ili korišćenjem manipulatora `setw()`.

```
cout << -12345 << endl; // print default value with no field width
cout << setw(10) << -12345 << endl; // print default with field width
cout << setw(10) << left << -12345 << endl; // print left justified
cout << setw(10) << right << -12345 << endl; // print right justified
cout << setw(10) << internal << -12345 << endl; // print internally justified
```

Kao rezultat se dobija:

```
-12345
  -12345
-12345
  -12345
- 12345
```

Ovde treba napomenuti da `setw()` i `width()` utiču samo na naredni izraz koji se koristi za štampu. Stoga ih je neophodno koristiti pre svakog manipulatora za format štampe. U nastavku je dat primer korišćenja funkcije `fill` za ispunjavanje praznina, pa da pogledamo kako ona funkcioniše:

```
cout.fill('*');
cout << -12345 << endl; // print default value with no field width
cout << setw(10) << -12345 << endl; // print default with field width
cout << setw(10) << left << -12345 << endl; // print left justified
cout << setw(10) << right << -12345 << endl; // print right justified
cout << setw(10) << internal << -12345 << endl; // print internally justified
```

Kao rezultat se dobija:

```
-12345
***-12345
-12345***
****-12345
-****12345
```

# Vežbe – Rad sa fajlovima u C-u

<i>Datoteke, funkcije za ulaz/izlaz,</i>

- 
- *Primer: Otvaranje tekstualnog fajla i upisivanje*
  - *Primer. Upisivanje podataka o učenicima u fajl*
  - *Primer. Upisivanje podataka o učenicima u fajl*
  - *Primer. Upisivanje u fajl i učitavanja iz fajla*
  - *Primeri. Rad sa komandnom linijom; Dodavanje podataka u postojeći fajl*

09

# PRIMER: OTVARANJE TEKSTUALNOG FAJLA I UPISIVANJE

*U nastavku je dat osnovni primer otvaranja, upisivanja i zatvaranja datoteke sa detaljnim objašnjenjem*

Pogledajmo i detaljno analizirajmo sledeći primer za rad sa fajlovima

```
#include<stdio.h>
int main()
{
    FILE *ptr_file;
    int x;

    ptr_file =fopen("output.txt", "w");

    if (!ptr_file)
        return 1;

    for (x=1; x<=10; x++)
        fprintf(ptr_file,"%d\n", x);

    fclose(ptr_file);
    return 0;
}
```

Analizirajmo primer liniju po liniju radi boljeg razumevanja:

```
ptr_file =fopen("output", "w");
```

Funkcija `fopen` otvara fajl "output.txt" u modu za upis (`w`). Ukoliko fajl ne postoji on će biti kreiran. Ali morate obratiti posebnu pažnju. Ukoliko fajl postoji njegov sadržaj će biti uništen. Funkcija `fopen` vraća pokazivač na fajl, koji je smešten u promenljivoj `ptr_file`. Ukoliko nije moguće otvoriti fajl iz nekog razloga promenljiva `ptr_file` će imati vrednosti `NULL`.

```
if (!ptr_file)
```

Ovaj uslovni iskaz nakon funkcije `fopen` će proveriti da li je fajl uspešno otvoren ili ne. Ukoliko fajl nije uspešno otvoren program prekida sa radom i vraća 1 (što označava da je nešto pošlo po zlu u programu).

```
fprintf(ptr_file,"%d\n", x);
```

U okviru `for` petlje koja se izvršava 10 puta, iskaz `fprintf` na sličan način kako to funkcija `printf()` radi sa obilčnim izlazom (konzolom) štampa podatke u fajl na koji pokazuje pokazivač `ptr_file`.

```
fclose(ptr_file);
```

Funkcija `fclose` zatvara fajl. Ova funkcija mora biti navedena pra kraja programa, naročito ako vršite upis u fajl tako da se ona ne sme zaboraviti. Takođe treba poveriti računa da se umesto `fclose` ne navede funkcija `close()` koja takođe postoji u standardnoj C biblioteci. Funkcija `close()` neće zatvoriti fajl kako treba pa se u jednom momentu može desiti da program jednostavno prekine sa radom.

# PRIMER. UPISIVANJE PODATAKA O UČENICIMA U FAJL

*Napisati program koji učitava n strukturnih promenljivih ucenik (ime, adresa, razred, odeljenje) i zapisuje podatke u datoteku UCENIK.TXT Podaci jednog ucenika su u jednom redu.*

```
#include<stdio.h>
#include<conio.h>

struct ucenik
{
    char ime[35];
    char adresa[50];
    unsigned raz;
    unsigned ode;
};

void citaj(struct ucenik *o)
{
    printf("Unesite ime osobe:");
    gets(o->ime);
    printf("Unesite adresu osobe:");
    gets(o->adresa);
    printf("Unesite razred i odeljenje ucenika:");
    scanf("%u%u",&o->raz,&o->ode);
    while(getchar()!='\n');
}

void main()
{
    struct ucenik x;
    FILE *f;
    int i,n;

    f=fopen("ucenik.txt","w");
    printf("Unesi broj ucenika: ");
```

# PRIMER. UPISIVANJE PODATAKA O UČENICIMA U FAJL

*Napisati program koji učitava n strukturnih promenljivih ucenik (ime, adresa, razred, odeljenje) i zapisuje podatke u datoteku UCENIK.TXT Podaci jednog ucenika su u jednom redu.*

```
scanf("%d",&n);
while(getchar()!='\n');

for(i=0;i<n;i++)
{
printf("Unesi podatke za %d. ucenika:\n",i+1);
citaj(&x);
fprintf(f,"%15s%20s%6d%6d\n",
x.ime,x.adresa,x.raz,x.ode);
}

fclose(f);

printf("\n\nZavršen unos podataka...\nPritisni neki taster...");
getch();
```



# PRIMER. UPISIVANJE U FAJL I UČITAVANJA IZ FAJLA

*Napisati C program koji u datoteku upisuje prvih 10 prirodnih brojeva, a zatim se iz iste datoteke čitaju brojevi dok se ne stigne do kraja i ispisuju se na standardni izlaz*

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
    int i;
    FILE* f = fopen("podaci.txt", "w");
    if (f == NULL)
    {
        printf("Greska prilikom otvaranja datoteke podaci.txt za pisanje\n");
        exit(1);
    }
    for (i = 0; i<10; i++)
        fprintf(f, "%d\n", i);

    fclose(f);

    f = fopen("podaci.txt", "r");
    if (f == NULL)
    {
        printf("Greska prilikom otvaranja datoteke podaci.txt za citanje\n");
        exit(1);
    }
    while(1)
    {
        int br;
        fscanf(f, "%d", &br);
        if (feof(f)) break;
        printf("Procitano : %d\n", br);
    }
}
```

# PRIMERI. RAD SA KOMANDNOM LINIJOM; DODAVANJE PODATAKA U POSTOJEĆI FAJL

*Nazivi datoteka sa kojima se operiše mogu biti prosleđeni kao argumenti komandne linije*

Napisati program koji kopira fajl čije se ime zadaje kao prvi argument komandne linije u fajl čije se ime zadaje kao drugi argument komandne linije. Uz svaku liniju se zapisuje i njen broj.

```
#include <stdio.h>
#define MAX_LINE 256

int getline(char s[], int lim)
{
    char* c = fgets(s, lim, stdin);
    return c==NULL ? 0 : strlen(s);
}

void main(int argc, char* argv[])
{
    char line[MAX_LINE];
    FILE *in, *out;
    int line_num;
    if (argc != 3)
    {
        fprintf(stderr, "Upotreba : %s ulazna_datoteka
        izlazna_datoteka\n", argv[0]);
        return 1;
    }

    if ((in = fopen(argv[1], "r")) == NULL)
    {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[1]);
        return 1;
    }
    if ((out = fopen(argv[2], "w")) == NULL) {
        fprintf(stderr, "Neuspesno otvaranje datoteke %s\n", argv[2]);
        return 1;
    }
    line_num = 1;
    while (fgets(line, MAX_LINE, in) != NULL)
    {
        fprintf(out, "%-3d : \t", line_num++);
        fputs(line, out);
    }
    fclose(in);
    fclose(out);
}
```

Primer. Napisati C program koji u datoteku dadaje tekst.

```
#include <stdio.h>
void main()
{
    FILE* datoteka;
    if (
    (datoteka=fopen("dat.txt","a"))==NULL)
    {
        fprintf(stderr, "Greska :
        nisam uspeo da otvorim dat.txt\n");
        return 1;
    }
    fprintf(datoteka, "Zdravo svima\n");
    fclose(datoteka);
}
```

# PRIMER. PROGRAM KOJI PRAVI KOPIJU ZADATOG FAJLA

*Napisati program koji podatke iz jednog fajla prebacuje u drugi fajl sa nazivom KOPIJA. Takođe oštampati podatke na ekran korišćenjem standardnog toka stdout*

```
#include <stdio.h>

int main(void)
{
    FILE *ulaz, *izlaz;
    char c, fajl[50];

    /* izlaz na stdout i ulaz sa stdin */
    fprintf(stdout, "unesi ime fajla\t");
    fscanf(stdin, "%s", fajl);

    /* isto kao printf("unesi ime fajla\t");
    scanf("%s", fajl); */

    /* otvaranje fajlova */
    if ((ulaz = fopen(fajl, "r")) == NULL)
    {
        fprintf(stderr, "greska pri otvaranju ulaznog fajla\n");
        return 1;
    }

    if ((izlaz = fopen("KOPIJA", "w")) == NULL)
    {
        fprintf(stderr, "greska pri otvaranju izlaznog fajla\n");
        return 1;
    }

    /* prepisivanje ulaznog fajla u izlazni i na stdout */
    while (!feof(ulaz))
    {
        c=fgetc(ulaz);
        fputc(c, izlaz);

        fputc(c, stdout); /* putchar(c); */
    }

    /* zatvaranje fajlova */
    fclose(ulaz);
    fclose(izlaz);
    return 0;
}
```

# PRIMER. KOPIRANJE SADRŽAJA JEDNE DATOTEKE U DRUGU

*Napisati program koji kopira sadržaj jedne datoteke u drugu i pri tome omogućava da svaka reč započinje velikim slovom dok su ostala slova mala.*

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_DUZINA_IMENA_DATOTEKE 100
#define MAX_VELICINA_DATOTEKE 100000

void main( void )
{
    /* prostor za unosenje naziva ulazne datoteke koju otvaramo */
    char u_datoteka[ MAX_DUZINA_IMENA_DATOTEKE ];
    /* prostor za unosenje naziva izlazne datoteke koju otvaramo */
    char i_datoteka[ MAX_DUZINA_IMENA_DATOTEKE ];
    /* mesto za privremeno smestanje datoteke za korigovanje*/
    char prostor[ MAX_VELICINA_DATOTEKE ];
    /* pomocna karakтерна promenljiva - za citanje iz datoteke*/
    char ch;
    /* promenljive za brojac */
    int i, br;
    /* pokazivaci na ulaznu i izlaznu datoteku */
    FILE *p_ulazna, *p_izlazna;

    /* unosi se ime ulazne datoteke */
    printf( "Upisi naziv ulazne datoteke:" );
    scanf( "%s", u_datoteka );
    fflush( stdin );

    /* otvaramo ulaznu datoteku za citanje */
    p_ulazna = fopen( u_datoteka, "r" );

    if ( p_ulazna == NULL ) /* ako je doslo do neke greske */
    {
        printf( "Uneto ime datoteke ne postoji ili ne moze da se otvori!\n" );
        return; /* izadji iz programa */
    }
}
```

# PRIMER. KOPIRANJE SADRŽAJA JEDNE DATOTEKE U DRUGU

*Napisati program koji kopira sadržaj jedne datoteke u drugu i pri tome omogućava da svaka reč započinje velikim slovom dok su ostala slova mala.*

```
/* inicijalizuj brojac na NULU */
br = 0;

/* citaj redom karaktere sve dok ne dodjes do kraja datoteke (EOF)*/
while ( ( ch = fgetc( p_ulazna ) ) != EOF )
    prostor[ ++br ] = ch; // karaktere smestaj u ovaj niz

// ako je ulazna datoteka prazna, nema smisla nista dalje raditi
if ( br == 0 )
{
    printf( "Ulazna datoteka je prazna!\n" );
    return; /* izadji iz programa */
}

/* postavi 'i' na jedinicu */
i = 1;

/* Prvi znak - ako je slovo mora biti veliko */
/* Ovo je klasican izuzetak u programiranju ... */
/* ... jer u glavnoj petlji ispitujemo i prethodni karakter*/
prostor[i] = toupper( prostor[i] );
fprintf( p_izlazna, "%c", prostor[i] );

/* upisi u datoteku zeljene izmene... */
for ( i=2; i<=br; i++ )
{
    if ( islower( prostor[i] ) != 0 )
        if ( isalpha( prostor[i-1] ) == 0 )
            prostor[i] = toupper( prostor[i] );

    if ( isupper( prostor[i] ) != 0 )
        if ( isalpha( prostor[i-1] ) != 0 )
            prostor[i] = tolower( prostor[i] );

    // Upisi u izlaznu datoteku
    fprintf( p_izlazna, "%c", prostor[i] );
}

// Zatvaranje datoteka
fclose( p_ulazna );
fclose( p_izlazna );
}
```

# PRIMER. PRETVARANJE VELIKIH SLOVA U MALA

*Napisati program koji učitava datoteku i pretvara sva slova u velika ili mala u zavisnosti od izbora korisnika.*

```
/* '#include' i '#define' su predprocesorske direktive */
#include <string.h> /* ... za rad sa stringovima */
#include <stdio.h> /* ... za standardni ulaz i izlaz */
#include <ctype.h> /* ... za neke druge funkcije npr.'toupper'*/

/* definiira maksimalne velicine datoteke - 100000 karaktera */
#define MAX_VELICINA_DATOTEKE 100000
/* 'main' je oznaka za pocetak glavne funkcije, odnosno programa */
void main( void )
{
    /* predvidjeni memorijski prostor za naziv datoteke koju otvaramo */
    /* u nasem slucaju 100 karaktera*/
    char datoteka[100];

    /* karakterna promenljiva za izbor, u koju cemo uneti...
    ... oznaku za mala ili velikaslova */
    char izbor;

    /* Pomocne promenljive */
    char ch;
    int i, br;

    /* memorijski prostor za privremeno smestanje korigovane datoteke... */
    char prostor[ MAX_VELICINA_DATOTEKE ];

    /* pointer na datoteku */
    FILE *p_datoteka;

    /* 'printf' ispisuje tekst na ekranu ... */
    printf( "Upisi naziv datoteke:" );
    /* Unosi se ime datoteke*/
    scanf( "%s", datoteka );
```

# PRIMER. PRETVARANJE VELIKIH SLOVA U MALA

*Napisati program koji učitava datoteku i pretvara sva slova u velika ili mala u zavisnosti od izbora korisnika.*

```
fflush( stdin );

/* 'fopen' otvara datoteku za citanje (READ)- "r" */
p_datoteka = fopen( datoteka, "r" );

/* ako je doslo do neke greske (== NULL) ... */

if ( p_datoteka == NULL )
{
    printf( "Uneto ime datoteke ne postoji ili ne moze da se otvori!\n" );
    return; /* izadji iz programa*/
}

/* Meni za unos izbora*/
printf( "Meni:\n"
        "V - pretvori sve u velika slova\n"
        "M - pretvori sve u mala slova\n\n"
        "Izbor? : " );

scanf( "%c", &izbor );

fflush( stdin );

// uneti karakter pretvara u veliko slovo
izbor = toupper( izbor );

/* Ako nismo upisali zeljeno slovo */
if ( izbor != 'V' && izbor != 'M' )
{
    printf( "Nije uneto odgovarajuce slovo!\n" );
    return; /* izadji iz programa*/
}
```

# PRIMER. PRETVARANJE VELIKIH SLOVA U MALA

*Napisati program koji učitava datoteku i pretvara sva slova u velika ili mala u zavisnosti od izbora korisnika.*

```
/* inicijalizujemo brojac na NULU */
br = 0;

/* while petlja*/
while ( ( ( ch = fgetc( p_datoteka ) ) != EOF ) && ( br < MAX_VELICINA_DATOTEKE ) )
{
    /* Menjamo mala u VELIKA ili VELIKA u mala */
    if ( izbor == 'V' ) ch = toupper( ch );
    else                ch = tolower( ch );
    /* izmenu belezimo u nizu */
    prostor[ ++br ] = ch;
}

/* 'fclose' zatvara datoteku na koju pokazuje POINTER p_datoteka*/
fclose( p_datoteka );
/* inicijalizujemo pointer na NULU za svaki slucaj */
p_datoteka = NULL;
/* otvaramo datoteku ali ovoga puta za pisanje ("w") */
p_datoteka = fopen( datoteka, "w" );
/* ako je doslo do neke greske (== NULL) ... */
if ( p_datoteka == NULL )
{
    printf( "Datoteke ne moze da se otvori za pisanje!\n" );
    return; /* izadji iz programa*/
}

/* upisujemo iz karakternog niza 'prostor[]' izmenjenu datoteku*/
for ( i=1; i<=br; i++ )
    fprintf( p_datoteka, "%c", prostor[i] );
/* zatvoramo (fclose) datoteku */
fclose( p_datoteka );
}
```



# PRIMER. ZAMENA TABULATORA SA PRAZNYM POLJIMA

*Napisati program koji učitava datoteku i zamenjuje tabulatore sa blenkovima, pri čemu broj blenkova zadaje korisnik.*

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

#define MAX_VELICINA_DATOTEKE 100000

void main( void )
{
    char datoteka[100]; /* naziv datoteke koju otvaramo */
    char broj; /* broj blanko karaktera koji zelimo da umetnemo */
    /* Pomocne promenljive */
    char ch;
    int i, br;
    /* mesto za privremeno smestanje korigovane datoteke */
    char prostor[ MAX_VELICINA_DATOTEKE ];
    FILE *p_datoteka; /* pokazivac na datoteku koju otvaramo */

    /* unosi se ime datoteke, i to max do 100 karaktera */
    printf( "Upisi naziv datoteke:" );
    scanf( "%s", datoteka );
    fflush( stdin );

    /* otvaramo datoteku za citanje */
    p_datoteka = fopen( datoteka, "r" );

    if ( p_datoteka == NULL ) /* ako je doslo do neke greske */
    {
        printf( "Uneto ime datoteke ne postoji ili ne moze da se otvori!\n" );
        return; /* izadji iz programa*/
    }

    /* unosi slovo koje ce mo da ispitujemo */
    do
    {
        printf( "Upisi broj blanko karaktera za zamenu tabulatora:" );
        scanf( "%c", &broj );
        fflush( stdin );
    } /* ako nismo uneli slovo, vraca nas ponovo na unos */
    while ( isdigit( broj ) == 0 );
```

# PRIMER. ZAMENA TABULATORA SA PRAZNYM POLJIMA

*Napisati program koji učitava datoteku i zamenjuje tabulatore sa blenkovima, pri čemu broj blenkova zadaje korisnik.*

```
/* inicijalizujemo brojac na NULU */
br = 0;

/* Radi dok god ne dosegemo kraj datoteke */
while ( ( ch = fgetc( p_datoteka ) ) != EOF )
{
    /* Ako smo naisli na znak za tabulator... */
    if ( ch == '\t' )
        for ( i = 1; i <= broj - '0'; i++ ) prostor[ ++br ] = ' ';
    else /* a ako nismo... */
        prostor[ ++br ] = ch;
}

/* zatvori datoteku */
fclose( p_datoteka );

/* inicijalizujemo pointer na NULU za svaki slucaj */
/* jer ce mo ponovo otvarati datoteku */
p_datoteka = NULL;

/* otvaramo datoteku ali ovoga puta za pisanje */
p_datoteka = fopen( datoteka, "w" );

if ( p_datoteka == NULL ) /* ako je doslo do neke greske */
{
    printf( "Datoteke ne moze da se otvori za pisanje!\n" );
    return; /* izadji iz programa*/
}

/* upisi u datoteku zeljene izmene... */
for ( i=1; i<=br; i++ )
    fprintf( p_datoteka, "%c", prostor[i] );

/* zatvori datoteku */
fclose( p_datoteka );
}
```

# PRIMER. ČITANJE IZ DATOTEKE I SORTIRANJE REČI U ABECEDNOM PORETKU

*Napisati program koji učitava datoteku i na ekran štampa sve reči poređavši ih prema abecednom redosledu.*

```
#include <string.h>
#include <stdio.h>
#include <ctype.h>

void main(void)
{
    char datoteka[100]; /* naziv datoteke koju otvaramo */
    char rec[1000][100]; /* string u koji se izdvajaju reci */
    char pom[100];
    FILE *p_datoteka; /* pokazivac na datoteku */
    int i, j, broj;

    /* unosi se ime datoteke, i to max do 100 karaktera */
    printf( "Upisi naziv datoteke:" );
    scanf( "%s", datoteka );

    p_datoteka = fopen( datoteka, "r" ); /* otvaramo datoteku za citanje*/
    if ( p_datoteka == NULL ) /* ako je doslo do neke greske */
    {
        printf( "Uneto ime datoteke ne postoji ili ne moze da se otvori!\n" );
        return; /* izadji iz programa*/
    }

    /* citaj redom stringove (reci) sve dok ne dodjes do kraja datoteke (EOF)*/
    /* istovremeno stavlja ih redom u niz...*/
    for ( broj=0; fscanf( p_datoteka, "%s", rec[broj] ) != EOF; broj++ )
    ;

    /* Umanji broj reci za jedan (ovo je trik)*/
    --broj;

    /* klasican bubble-sort */
```

## PRIMER. ČITANJE IZ DATOTEKE I SORTIRANJE REČI U ABECEDNOM PORETKU

*Napisati program koji učitava datoteku i na ekran štampa sve reči poređavši ih prema abecednom redosledu.*

```
/* ispisi rezultat na ekran */  
for ( i=0; i<=broj; i++ )  
    printf( "%s\n", rec[i] );  
  
/* zatvori datoteku */  
fclose( p_datoteka );  
}
```

# DATOTEKE - ZADACI ZA SAMOSTALAN RAD

*U nastavku su dati zadaci za samostalno vežbanje kada je u pitanju rad sa datotekama*

Napisati program koji učitava datoteku **ulaz.txt**, iz nje uklanja suvišne razmake između reči i ispravljeni tekst čuva u datoteku **izlaz.txt**.  
Upisati u datoteku tri broja. Pročitati datoteku i sabrati sve brojeve.

Napisati program koji iz datoteke **ulaz.txt** učitava deset brojeva koji će biti zapamćeni u niz. Nakon unosa program treba da sortira unete brojeve po veličini, od najvećeg ka najmanjem, i da ih oštampa u datoteci **izlaz.txt**.

Napisati program kojim se čita datoteka: znak po znak; i određuje broj pojavljivanja malih slova u datoteci bez **w**, **x**, **y**.

# Vežba – Alati za dijagnostiku grešaka

<i>Debugger, VisualStudio, Netbeans</i>

---

*Dijagnostika grešaka - Visual Studio 2010*

*Dijagnostika grešaka - Netbeans IDE*

10

# UVODNA RAZMATRANJA

*Najveći broj integrisanih razvojnih okruženja sadrži u sebi veoma moćne alate za dijagnostiku grešaka*

U nastavku će ukratko biti opisane opcije koje vam nude sledeća integrisana razvojna okruženja kada je u pitanju detekcija i korigovanje grešaka u programu:

- Visual Studio 2010 Debugger
- Netbeans C/C++ IDE Debugger

# Dijagnostika grešaka - Visual Studio 2010

*dijagnostika, korigovanje grešaka, debugger, VisualStudio*

- *Pokretanje alata i postavljanje tačke zaustavljanja*
- *Opcije za izvršavanje koda „korak po korak“*
- *Uslovna tačka prekida*
- *Prozori za praćenje*
- *Prozori za posmatranje proizvoljnih i lokalnih promenljivih*

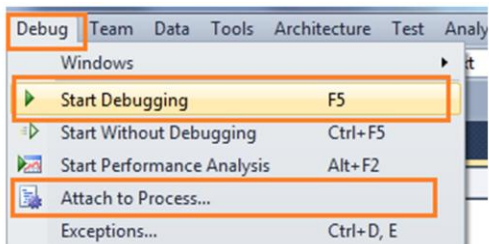
# 10



# POKRETANJE ALATA I POSTAVLJANJE TAČKE ZAUSTAVLJANJA

*Debugiranje programa može započeti tako što se iz menija **Debug** izabere "Start Debugging" ili jednostavno pritiskom na F5.*

Debugiranje programa može započeti tako što se iz menija **Debug** izabere "Start Debugging" ili jednostavno pritiskom na F5, što će startovati vaš program. Ukoliko ste u nekoj od linija naveli tačku zaustavljanja (breakpoint) izvršenje programa će početi i program će se pauzirati u toj liniji. U suprotnom će se program izvršiti do kraja ili do linije gde će se desiti neočekivani prekid.



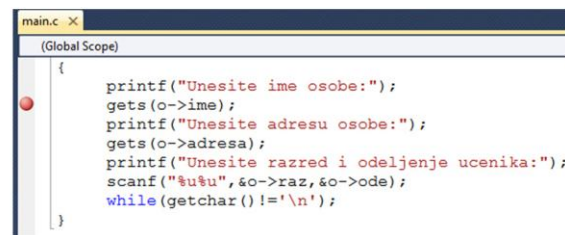
Slika-1 Postupak pokretanja alata

Debugiranje bilo kog programa počinje postavljanjem tačke zaustavljanja (breakpoint) u liniji koda gde mislimo da se javila greška.

## □ Tačka zaustavljanja – Breakpoint

Breakpoint se koristi sa ciljem da se Alat za korigovanje obavesti kada i gde treba da pauzira program u toku izvršavanja. Tačka zaustavljanja se postavlja ili postavljanjem kursora u liniji u kojoj želimo da se program zaustavi i pritiskom na F9; ili klikom na "side bar" koda u pravcu linije gde želimo da zaustavimo program.

Naravno, pre postavljanja tačke zaustavljanja treba otprilike znati šta nije u redu sa kodom. U trenutku kad **debugger** stigne u tačku zaustavljanja programer dobija mogućnost da ispita šta nije u redu sa kodom korišćenjem alata za ispravljanje grešaka.



Slika-2 Postavljanje tačke zaustavljanja

## □ Debugiranje korišćenjem tačke zaustavljanja

Već smo postavili **breakpoint** u liniji koda u kojoj želimo da se naš program pauzira. Kada program stigne do te tačke i kada se pauzira, imamo nekoliko opcija pomoću kojih možemo da ispitamo tačnost koda. Ove opcije su navedene na sledećoj slici i deo su takozvanog **Breakpoint Toolbar**-a.



Slika-3 Paleta opcija za rad sa **debugger**-om

# OPCIJE ZA IZVRŠAVANJE KODA „KORAK PO KORAK“

*Opcije Step over i Step Into omogućavaju praćenje programa korak po korak. Step Over odmah odradjuje liniju poziva funkcije dok Step Into ulazi u prvu liniju pozvane funkcije*

## ❑ Opcija **Step Over**

"**Step Over**" [ F10 ] – ova komanda omogućava izvršavanje koda liniju po liniju. Ukoliko se u ovoj liniji nalazi poziv funkcije onda će se izvršiti ceo kod funkcije a program će se zaustaviti u prvoj liniji nakon poziva funkcije. **Step Over** će izvršiti ceo metod odjednom. U nastavku je dat primer u kome se program zaustavio u liniji gde se nalazi poziv funkcije `citaj()`. Izborom opcije **StepOver** alat **debugger** neće ući u blok funkcije već će preći na sledeću liniju nakon poziva funkcije `citaj()` a to je linija gde je poziv `fprintf()`.

```
for(i=0;i<n;i++)
{
    printf("Unesi podatke za %d. ucenika:\n",i+1);
    citaj (&x);
    fprintf(f, "%15s%20s%6d%6d\n",
        x.ime, x.adresa, x.raz, x.ode);
}
```

Slika-4 Opcija Step Over, Prečica je F10

## ❑ Opcija **Step Into**

Ova opcija izvršava jednu liniju koda. Ukoliko je u toj liniji poziv funkcije, onda će ući u funkciju i pauzirao program u prvoj liniji funkcije. Prečica je **F11**, a na sledećoj slici je prikazan rezultat korišćenja ove opcije iz linije tačke prekida koja je prikazana na prethodnoj slici.

```
void citaj(struct ucenik *o)
{
    printf("Unesite ime osobe:");
    gets(o->ime);
    printf("Unesite adresu osobe:");
    gets(o->adresa);
    printf("Unesite razred i odeljenje ucenika:");
    scanf("%u%u", &o->raz, &o->ode);
    while(getchar() != '\n');
```

Slika-5 Opcija Step Into – Prečica F11

## ❑ Opcija **Step Out**

Ova opcija izvršava liniju po liniju koda. Ukoliko se tačka prekida nalazi u okviru bloka neke funkcije (kao na primer u okviru funkcije `citaj` sa slike-5) onda će se izborom ove opcije izvršiti sve linije u okviru bloka i program će se zaustaviti u prvoj liniji bloka pozivaoca funkcije nakon poziva odgovarajuće funkcije u kojoj smo pozvali **Step Out**. Ukoliko u bloku postoji još jedna linija prekida, kao što i jeste u našem primeru, program će se zaustaviti na toj tački, a tek sledeći poziv ove iste opcije će vratiti tok izvršavanja programa van funkcije. Prečica za ovu opciju je **Shift - F11**.

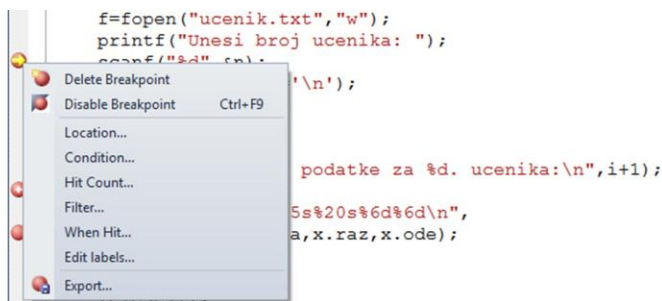
## ❑ Opcija **Continue**

Ova opcija nastavlja proces izvršavanja programa nakon dostizanja tačke zaustavljanja. Prečica za ovu opciju je **"F5"**.

# USLOVNA TAČKA PREKIDA

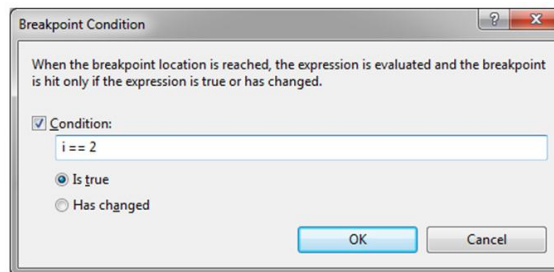
*Visual Studio nam omogućava uslovnu tačku prekida, odnosno da se program zaustavi ako je zadovoljen neki uslov*

Pretpostavimo da smo u petlji i da želimo da se zaustavimo u nekom proizvoljnom koraku petlje jer nam se čini da za taj korak naš kod ne radi kako treba. **Visual Studio** nam omogućava uslovnu tačku prekida, odnosno da se program zaustavi ako je zadovoljen neki uslov. Da bi smo ovo uradili neophodno je da prvo postavimo tačku prekida u liniji kojoj želimo da pauziramo program. Zatim, jednostavno, treba pritisnuti desni taster miša na crvenu tačku koja označava tačku prekida (**breakpoint**). Otvoriće se dopunski meni u kome treba samo kliknuti na polje **"Condition"**.



Slika-6 Postavljanje uslovne tačke prekida

Klikom na **"Condition"** u okviru dodatnog menija, otvoriće se dijalog za uslovnu tačku prekida (Slika-7).



Slika-7 Dijalog Breakpoint Condition

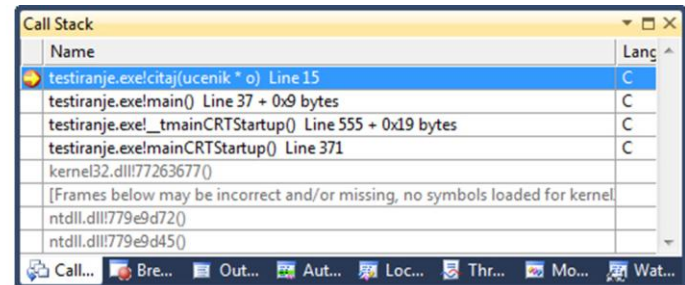
# PROZORI ZA PRAĆENJE

*Prozori za praćenje, odnosno prozori za istragu, omogućavaju praćenje promenljivih u trenutku kada debugger dostigne tačku zaustavljanja tj. breakpoint*

Ovi prozori su takozvani prozori za istragu. Prvi sledeći korak nakon dostizanja tačke prekida je da se ispita vrednost objekata ili promenljivih u programu. **Visual Studio** vam omogućava da postavljanjem kursora miša preko imena promenljive dobijate dodatne informacije o toj promenljivoj kao što su tip podatka, njena vrednost itd. Osim te opcije postoje i različite vrste prozora za praćenje a neki od njih su **Autos**, **Local**, itd, i oni će biti opisani u nastavku.

## Prozor **Call Stack**

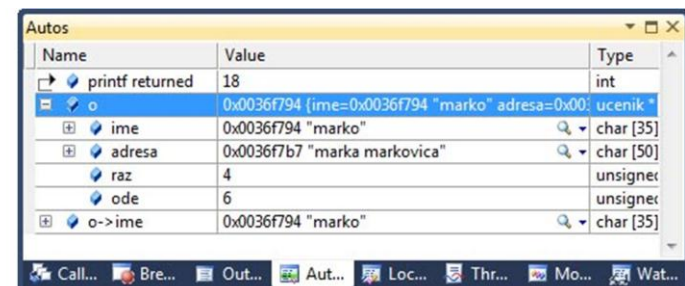
Ova opcija povećava produktivnost tokom dijagnostike i korekcije grešaka. Ukoliko u programu imate višestruki poziv funkcija iz drugih funkcija odnosno veliki broj ugnježdenih funkcija u programu ili u fazi debugiranja, imaćete često potrebu da proverite iz kog bloka je koji poziv izvršen. Stek poziva, odnosno **"Call Stack"** vam omogućava ove opcije. Prozor **Call Stack Window** prikazuje hijerarhiju poziva funkcija u vašem programu.



Slika-8 Prozor Call Stack

## Prozor **Autos**

Automatske promenljive su one promenljive koje po automatizmu detektuje **VS debugger** u toku procesa dijagnostike. **Visual Studio** je taj koji određuje koji objekti i promenljive su bitne za tekući segment koda i na osnovu toga pravi listu **"Auto"** promenljivih. Prečica za prikaz **Auto** promenljivih je **"Ctrl+D+A"**.



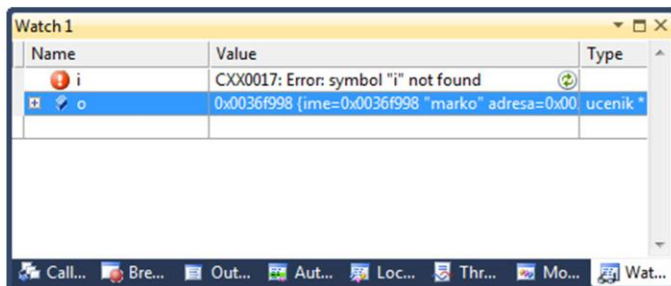
Slika-9 Prozor Autos

# PROZORI ZA POSMATRANJE PROIZVOLJNIH I LOKALNIH PROMENLJIVIH

*Prozor Watch se koristi za praćenje pojedinačnih objekata ili promenljivih, dok se u prozoru Locals prate lokalne promenljive bloka u kome je zaustavljen program*

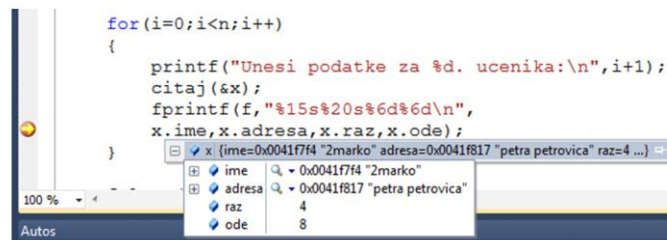
## ❑ Prozor Watch

Prozor Watch se koristi za praćenje pojedinačnih objekata ili promenljivih. Proizvoljne promenljive se dodaju ručno u listu ovog prozora. Može se u ovaj prozor dodati onoliko promenljivih koliko je to potrebno. Dodavanje promenljive se vrši desnim klikom miša na ime promenljive i izborom opcije **"Add To Watch"** ili jednostavno prevlačenjem selektovane promenljive iz tekst editora u ovaj prozor.



Slika-10 Prozor Watch

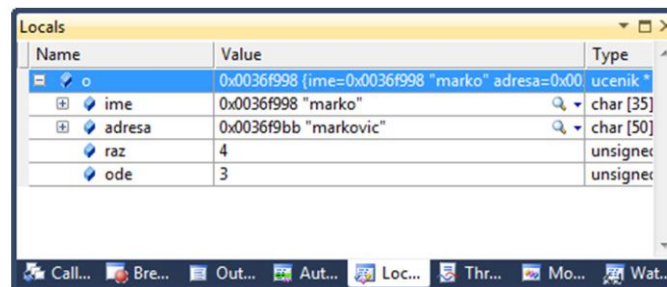
Isti efekat se može postići postavljanjem kursora miša iznad naziva promenljive **"name"** koja se nalazi u liniji u kojoj je tačka prekida, što se može videti na slici 11.



Slika-11 Informacija o promenljivoj postavljanjem kursora na ime

## ❑ Prozor Locals

U ovom prozoru se automatski prikazuje lista promenljivih koje su lokalne u okviru opsega bloka u kome je pauziran program.



Slika-12 Prozor Locals

# Dijagnostika grešaka - Netbeans IDE

<i>dijagnostika, korigovanje grešaka, debugger, Netbeans</i>
<i>20 min</i>

- 
- *Postupak korigovanja grešaka i opcije za izvršavanje programa „korak po korak“*
  - *Pokretanje alata i dodavanje promenljivih u prozor za praćenje*
  - *Ostale opcije za praćenje promenljivih i izvršavanje koda*

10



# POSTUPAK KORIGOVANJA GREŠAKA I OPCIJE ZA IZVRŠAVANJE PROGRAMA „KORAK PO KORAK“

*Netbeans IDE ima skoro identičnu podršku za dijagnostiku kao i Visual Studio. Pokretanje alata se vrši izborom opcije **Run->Debug Main Project***

U ovom poglavlju će biti opisane tehnike razvojnog okruženja **Netbeans IDE** koje se koriste za korigovanje grešaka u vašem programu kada nešto krene po zlu.

## Postavljanje tačke zaustavljanja

Tačka zaustavljanja (**breakpoint**) se može postaviti klikom miša na levi tab u liniji u kojoj želimo da se zaustavimo. Nakon klika primetićete roze kvadrat, a i cela linija će biti obojena u roze. Ponovnim klikom na već postavljen **breakpoint** on će biti deaktiviran.








Slika-1 Postavljanje tačke zaustavljanja

Postoji nekoliko načina da startuje vaš program u **debug** modu:

- Izaberite **Debug Main Project** u meniju **Run** da bi se program izvršio do prve tačke zaustavljanja. Ukoliko niste postavili ni jednu tačku zaustavljanja u program onda će program izvršiti do kraja.

- Izaberite **Step Into** u meniju **Run** da bi se program izvršavao linija po linija. U ovom slučaju će se program paузirati u prvoj liniji. U ovoj tački možete da analizirate vrednost promenljivih a i da nastavite sa izvršavanjem programa.
- Izaberite **Run to Cursor** u meniju **Run** da bi se program izvršio do linije u kojoj se trenutno nalazi kursor, gde će program biti paузiran.

Jednom kad je paузiran rad programa, može se pratiti tok njegovog izvršavanja korišćenjem opcija koje su navedene u sledećoj tabeli:

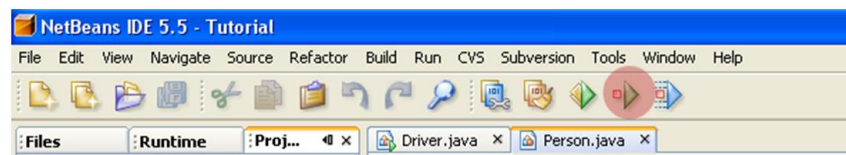
<b>Step Over</b> (  ) -	Izvršava jednu liniju koda. Ukoliko je u toj liniji poziv funkcije, izvršiće se funkcija bez ulaska u njen blok.
<b>Step Into</b> (  ) -	Izvršava jednu liniju koda. Ukoliko je u toj liniji poziv funkcije, onda će ući u funkciju i paузiraće program u prvoj liniji funkcije.
<b>Step Out</b> (  ) -	Izvršava jednu liniju koda. Međutim, ukoliko je linija koda unutar nekog metoda, onda će se izvršiti sve linije metoda i vratiti se u prvu liniju bloka odakle je pozvan metod.
<b>Run to Cursor</b> (  ) -	Izvršava program dok ne dođe do linije u kojoj se nalazi kursor.
<b>Continue</b> (  ) -	Nastavlja sa izvršavanjem programa do sledeće tačke zaustavljanja ili do kraja programa.

Slika-2 Opcije za praćenje toka izvršavanja programa

# POKRETANJE ALATA I DODAVANJE PROMENLJIVIH U PROZOR ZA PRAĆENJE

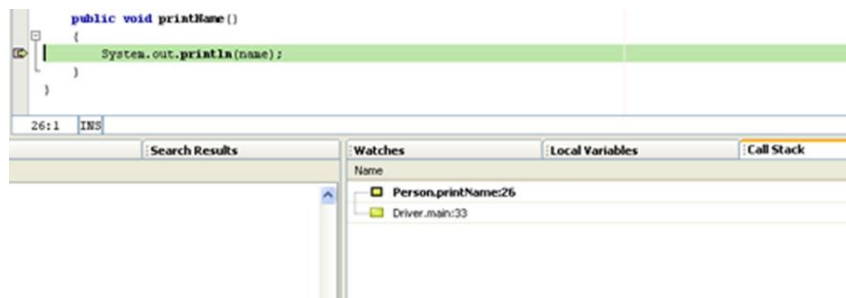
*Nakon pokretanja alata za dijagnostiku program se zaustavlja u tački prekida. Nakon toga je moguće vršiti praćenje promenljivih korišćenjem sličnih prozora kao u Visual Studio-u*

Pretpostavimo da smo debugiranje započeli klikom na dugme "Debug Main Project". To je dugme koje je na sledećoj slici osenčeno crvenom bojom.



Slika-3 Paleta sa dugmetom za startovanje dijagnostike

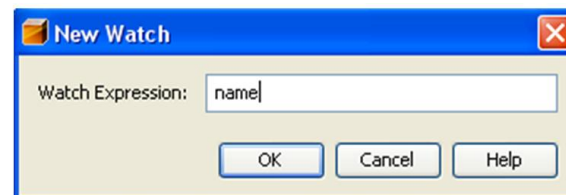
Nakon pokretanja programa u modu za korekcije (debug modu) primetićete da se program pauzira u tački u kojoj ste postavili breakpoint. Znaćete da je program pauziran jer je linija obojena u zeleno. Osim toga, ako pogledate prozor "call stack" na dnu Netbeans aplikacije primetićete listu metoda koje su pozvane redom do mesta tačke zaustavljanja.



Slika-4 Izgled programa kad se dostigne tačka zaustavljanja

Za zadati primer sa Slike-4, koji se odnosi na jednu Java aplikaciju, primetićete da se u steku poziva (call stack-u) nalaze dve funkcije: main method klase Driver, iz koga je pozvan metod printName metod klase Person.

Sada kada smo dostigli tačku zaustavljanja, možemo da pratimo bilo koju promenljivu koja je dostupna u tom trenutku. Tu se misli na sve lokalne promenljive i eventualno globalne promenljive koje se vide u bloku gde smo postavili tačku zaustavljanja. U ovom primeru imamo samo jednu promenljivu "name" tako da ćemo samo nju dodati u prozor za praćenje. To radimo klikom na tab "Watches" a zatim Run > New Watch u glavnom meniju. Otvoriće se sledeći dijalog New Watch (Slika-5), pa ćemo u polju za unos Watch Expression da upišemo "name" a zatim ćemo kliknuti na OK.



Slika-5 Dijagram za dodavanje promenljive u prozor za praćenje

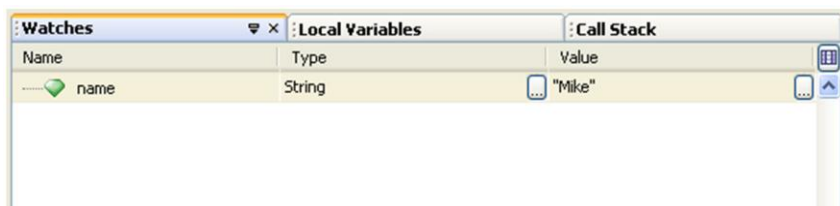
Nakon pritiska na dugme OK primetićete da se u prozoru Watches pojavila nova promenljiva



# OSTALE OPCIJE ZA PRAĆENJE PROMENLJIVIH I IZVRŠAVANJE KODA

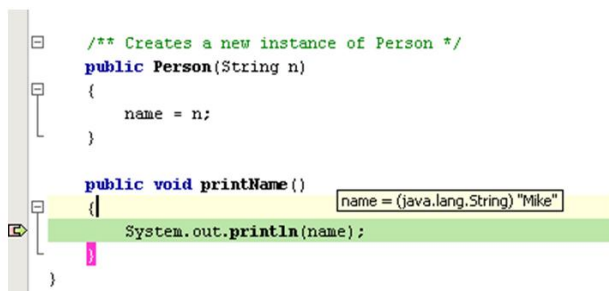
*Takođe je moguće, postavljanjem kursora miša iznad naziva promenljive, pratiti promene promenljive u toku izvršavanja programa*

Takođe se može primetiti da promenljiva **name** u tom trenutku ima vrednost "Mike".



Slika-6 Prozor Watch za praćenje promena promenljive

Isti efekat se može postići postavljanjem kursora miša iznad naziva promenljive **"name"** koja se nalazi u liniji u kojoj je tačka prekida, što se može videti na narednoj slici (Slika-7).



Slika-7 Postavljanje kursora miša preko naziva promenljive

## Prozor za lokalne promenljive

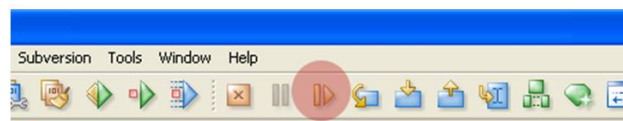
Prozor **Local variables** prikazuje ime, tip podatka i vrednost svih promenljivih koje su u opsegu odnosno koje su vidljive u bloku u kome smo pauzirali program kao i statičke promenljive. Da bi se aktivirao ovaj program treba u **Windows** meniju izabrati **Debugging -> Local Variables**.



Slika-8 Prozor za lokalne promenljive

## ❑ Opcija **Continue**

Nakon što smo završili sa analizom ove linije koda možemo dati programu instrukciju da nastavi sa radom, a to radimo klikom na dugme **continue** u paleti na vrhu prozora programa **Netbeans** (Slika-9). Program će se završiti uobičajeno.



Slika-9 Dugme za nastavak izvršavanja programa

# Vežba – Uvod u C++

<i>C++, sintaksa, ulazno/izlazne funkcije</i>

- 
- *Izgled prvog C++ programa*
  - *Primer. Štampanje podataka na ekran*
  - *Primer. Korišćenje petlji u C++-u*
  - *Primer. Formatiranje pri štampanju podataka - precision*
  - *Primer. Učitavanja stringova u C++ program*

11

# IZGLED PRVOG C++ PROGRAMA

*Ovako izgleda kod prvog programa koji ispisuje poruku Pozdrav:*

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main(int argc, char** argv) {
    cout << "Pozdrav!!!";
    return 0;
}
```

U prvom delu koda (**#include** deo) uključujemo biblioteke koje su potrebne u radu programa. **cstdlib** je biblioteka koja se generiše kada se napravi novi projekat u **NetBeans**-u i nju ne treba dirati. Biblioteka **iostream** je biblioteka koju smo mi uključili i koja nam je potrebna za rad ovog programa. Pomoću ove biblioteke možemo u kodu koristiti komande:

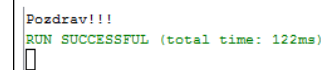
- **cout** - Komanda za ispis poruke na konzoli
- **cin** - Komanda za unos podataka preko konzole u program

Metoda **main** počinje sa:

```
int main(int argc, char** argv)
```

od nje počinje program i sve što se nalazi između vitičastih zagrada (**{}**) predstavlja kod koji se izvršava u njoj. U ovom programu ona sadrži dve linije koda, prva je za ispis poruke, a druga za gašenje programa.

Program u **NetBeansu** pokrećemo na dugme **Run Project**, i rezultat rada programa izgleda ovako(Slika-1):



```
Pozdrav!!!
RUN SUCCESSFUL (total time: 122ms)
|
```

Slika-1 Prikaz ispisa na konzoli

# PRIMER. ŠTAMPANJE PODATAKA NA EKRAN

*Korišćenje funkcije cout u C++- za štampanje rezultata na ekran*

```
#include <iostream>
using namespace std;
int main()
{
    //deklaracija tri celobrojne promenljive
    int initTime;
    int finalTime;
    int timeChange;
    //zadavanje vrednosti celobrojnim promenljivama
    initTime = 12;
    finalTime = 15;
    //operacija oduzimanja, gde je znak minus u stvari operator oduzimanja
    timeChange = finalTime - initTime;
    //prikazivanje teksta i promenljivih veličina na ekranu
    cout << "The initial time is " << initTime << endl;
    cout << "Final time is " << finalTime << endl;
    cout << "Time change is " << timeChange << endl;
    return 0;
}
```

# PRIMER. KORIŠĆENJE PETLJI U C++-U

## Upotreba petlji i funkcija za štampanje podataka u programskom jeziku C++

Na primer, u sledećem programu se od korisnika zahteva da unese ukupan broj prodavaca, kao i broj prodaja koji je svaki od njih izvršio, i broj proizvoda u toku svake od izvršenih prodaja, a zatim se vrši štampanje podatka o prosečnom broju prodaja za svakog prodavca.

```
#include <iostream>
using namespace std;
int main(void)
{
    int persons, int numSales;
    cout << "Enter number of salespersons: ";
    cin >> persons;
    cout << "Enter number of sales per salesperson: ";
    cin >> numSales;
    for (int x = 1; x <= persons; x++)
    {
        int sale, total = 0;
        float average;
        for (int y = 1; y <= numSales; y++)
        {
            cout << "Enter sale " << y << " for salesperson " << x << ": ";
            cin >> sale;
            total += sale;
        }
        average = (float) total / numSales;
        cout << "Average sales for salesperson #" << x << " is " << average << endl;
    }
    return 0;
}
```

The input and output could be

The input and output could be

```
Enter number of salespersons: 2
Enter number of sales per salesperson: 3
Enter sale 1 for salesperson 1: 4
Enter sale 2 for salesperson 1: 5
Enter sale 3 for salesperson 1: 7
Average sales for salesperson #1 is 5.33333
Enter sale 1 for salesperson 2: 8
Enter sale 2 for salesperson 2: 3
Enter sale 3 for salesperson 2: 4
Average sales for salesperson #2 is 5
```

# PRIMER. FORMATIRANJE PRI ŠTAMPANJU PODATAKA - PRECISION

*U sledećem primeru je opisan postupak korišćenja specifikatora precision za podešavanje preciznosti pri štampanju decimalnog realnog broja*

U nastavku je dat izvorni kod primera:

```
// Fig. 15.9: Fig15_09.cpp
// Controlling precision of floating-point values.
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;

#include <iomanip>
using std::setprecision;

#include <cmath>
using std::sqrt; // sqrt prototype

int main()
{
    double root2 = sqrt( 2.0 ); // calculate square root of 2
    int places; // precision, vary from 0-9

    cout << "Square root of 2 with precisions 0-9." << endl
         << "Precision set by ios_base member function "
         << "precision:" << endl;

    cout << fixed; // use fixed-point notation

    // display square root using ios_base function precision
    for ( places = 0; places <= 9; places++ )
    {
        cout.precision( places );
        cout << root2 << endl;
    } // end for

    cout << "\nPrecision set by stream manipulator "
         << "setprecision:" << endl;

    // set precision for each digit, then display square root
    for ( places = 0; places <= 9; places++ )
        cout << set precision( places ) << root2 << endl;

    return 0;
}
```

Izlaz programa je:

```
Square root of 2 with precisions 0-9.
Precision set by ios_base member function
precision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562

Precision set by stream manipulator setprecision:
1
1.4
1.41
1.414
1.4142
1.41421
1.414214
1.4142136
1.41421356
1.414213562
```

# PRIMER. UČITAVANJA STRINGOVA U C++ PROGRAM

*U sledećem primeru je dato poređenje korišćenja standardnog toka objekta cin i funkcije za učitavanje cin.get()*

U nastavku je dat izvorni kod primera:

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;

int main()
{
    // create two char arrays, each with 80 elements
    const int SIZE = 80;
    char buffer1[ SIZE ];
    char buffer2[ SIZE ];

    // use cin to input characters into buffer1
    cout << "Enter a sentence:" << endl;
    cin >> buffer1;

    // display buffer1 contents
    cout << "\nThe string read with cin was:" << endl
         << buffer1 << endl << endl;

    // use cin.get to input characters into buffer2
    cin.get( buffer2, SIZE );

    // display buffer2 contents
    cout << "The string read with cin.get was:" <<
endl
         << buffer2 << endl;
    return 0;
} // end main
```

Izlaz programa će biti:

```
Enter a sentence:
Contrasting string input with cin and cin.get

The string read with cin was:
Contrasting

The string read with cin.get was:
string input with cin and cin.get
```

# PRIMER. FORMATIRANI IZLAZ - KORIŠĆENJE SPECIFIKATORA ZA PORAVNAVANJE KAO I ZA ŠTAMPANJE REALNIH BROJEVA

*U nastavku su dati primeri korišćenja specifikatora left i right za poravnavanje linije koja se štampa, kao i korišćenje fixed i scientific specifikatora za format štampe realnih brojeva*

Primer 1. Korišćenje specifikatora za poravnavanje

```
#include <iostream>
using std::cout;
using std::endl;
using std::left;
using std::right;
#include <iomanip>
using std::setw;
int main()
{
    int x = 12345;

    // display x right justified (default)
    cout << "Default is right justified:" << endl
         << setw( 10 ) << x;

    // use left manipulator to display x left
    justified
    cout << "\n\nUse std::left to left justify x:\n"
         << left << setw( 10 ) << x;

    // use right manipulator to display x right
    justified
    cout << "\n\nUse std::right to right justify
x:\n"
         << right << setw( 10 ) << x << endl;
    return 0;
}
```

Primer 2. Korišćenje specifikatora za štampanje realnih brojeva u običnom, fiksno i eksponencijanom zapisu.

```
#include <iostream>
using std::cout;
using std::endl;
using std::fixed;
using std::scientific;

int main()
{
    double x = 0.001234567;
    double y = 1.946e9;

    cout << "Displayed in default format:" <<
endl
         << x << '\t' << y << endl;

    cout << "\nDisplayed in scientific format:"
<< endl
         << scientific << x << '\t' << y << endl;

    cout << "\nDisplayed in fixed format:" <<
endl
         << fixed << x << '\t' << y << endl;
    return 0;
}
```



# SIMULIRANJE IZDAVANJA RAČUNA OD STRANE JEDNOG RESTORANA

*Da li biste na osnovu dosadašnjeg znanja mogli da se uhvatite u koštac sa jednim profesionalnim problemom?*

Jednom restoranu je neophodno da se napiše program koji će da simulira generisanje i štampanje računa nakon obavljene usluge. Program treba od korisnika da zahteva da unese cenu obroka. Program treba da izračuna visinu poreza (8.25% od cene obroka) i da doda taj porez na ukupnu cenu koju treba da oštampa na ekranu. Program treba zatim da ispita koliki je novac dala mušterija, i nakon toga da sračuna kusur i da ispiše poruku na ekranu (novac koji je mušterija dala, cena obroka i kusur).

Pseudo kod za prethodno definisani problem se sastoji iz sledećih koraka:

- Uneti cenu obroka.
- Izračunati porez i ukupnu cenu obroka.
- Oštampati cenu, porez i ukupnu cenu.
- Uneti podatak o novcu koji je dala mušterija
- Izračunati kusur.
- Oštampati količinu novca koju je dala mušterija, ukupnu cenu i kusur.

# SIMULIRANJE IZDAVANJA RAČUNA OD STRANE JEDNOG RESTORANA – IMPLEMENTACIJA

*Korišćenjem odgovarajućih tipova podataka i manipulatora za štampanje podatka programskog jezika C++ moguće je izvršiti postavljeni problem*

U nastavku je dat kompletan **source** kod za pomenuti problem štampanja računa

```
// prb02-1.cpp
// This program simulates a simple cash register.
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    const double SALES_TAX_RATE = 0.0825;

    double meal_price,    // Price of meal entered by user
           sales_tax,    // Amount of sales tax
           total,        // Total bill: meal_price + sales_tax
           amt_tendered, // Amount received from customer
           change;       // Change: amt_tendered - total

    // Set up output stream for currency output
    cout << setprecision(2)
         << setiosflags(ios::fixed)
         << setiosflags(ios::showpoint);

    // Display banner and obtain price
    cout << "*** C++-Side Restaurant ***" << endl << endl;
    cout << "Enter the price of the meal: $";
    cin >> meal_price;
    cout << endl;

    // Calculate tax and total price
    sales_tax = meal_price * SALES_TAX_RATE;
    total = meal_price + sales_tax;
```

Rezultat izvršavanja programa je prikazan na sledećoj slici:

```
Program Output

*** C++-Side Restaurant ***

Enter the price of the meal: $52.95

Price of Meal:  52.95
Sales Tax:      4.37
-----
Total Amount:   57.32

Enter amount tendered: $70.00

Amount Tendered: $ 70.00
Total Amount:    $ 57.32
-----
Change:          $ 12.68

*** Thank You ***
```

Slika-2 Rezultat prethodnog koda

# SIMULIRANJE IZDAVANJA RAČUNA OD STRANE JEDNOG RESTORANA – IMPLEMENTACIJA

*Korišćenjem odgovarajućih tipova podataka i manipulatora za štampanje podatka programskog jezika C++ moguće je izvršiti postavljeni problem*

U nastavku je dat kompletan `source` kod za pomenuti problem štampanja računa

```
// Display price tax and total
cout << endl;
cout << "Price of Meal: " << setw(6) << meal_price << endl;
cout << "Sales Tax: " << setw(10) << sales_tax << endl;
cout << "-----" << endl;
cout << "Total Amount: " << setw(7) << total << endl;

// Obtain amount tendered
cout << endl;
cout << "Enter amount tendered: $";
cin >> amt_tendered;
cout << endl;

// Calculate change
change = amt_tendered - total;

// Display amounts and change
cout << endl;
cout << "Amount Tendered: $" << setw(7) << amt_tendered
    << endl;
cout << "Total Amount:      $" << setw(7) << total << endl;
cout << "-----" << endl;
cout << "Change:                $" << setw(7) << change << endl;

// Print closing banner
cout << endl << endl;
cout << "**** Thank You ****" << endl;

return 0;
}
```

Rezultat izvršavanja programa je prikazan na sledećoj slici:

```
Program Output

*** C++-Side Restaurant ***

Enter the price of the meal: $52.95

Price of Meal:  52.95
Sales Tax:      4.37
-----
Total Amount:   57.32

Enter amount tendered: $70.00

Amount Tendered: $ 70.00
Total Amount:    $ 57.32
-----
Change:          $ 12.68

*** Thank You ***
```

Slika-2 Rezultat prethodnog koda

# ZADACI ZA SAMOSTALAN RAD

*Koristeći predavanja i vežbe za ovu nedelju uraditi sledeće zadatke:*

1. Napisati program koji za uneti karakter ispisuje njegovu numeričku vrednost. Uraditi ovo korišćenjem `cout` i `cin`.
2. Definisati `PI` kao konstantu pretprocesora; Napisati program koji za uneti poluprečnik izračunava i prikazuje obim i površinu kruga; Koristiti `PI`.
3. Napisati program koji računa godišnju kamatu na iznos depozita u banci. Unosimo količinu novca i kamatnu stopu na mesečnom nivou, a aplikacija nam kaže koliko smo novca zaradili od kamate za godinu dana. Razmisliti koje tipove podataka je optimalno koristiti za ovaj problem.
4. Napisati C++ program koji ispisuje dva broja i pita korisnika koji od ta dva broja je veći. Ukoliko je prvi broj veći korisnik treba da unese DA, a ukoliko nije NE. Nakon unosa program govori korisniku da li je unos bio tačan ili nije.
5. Napisati C++ aplikaciju koja od korisnika zahteva da unese dva cela broja, i nakon toga unese jedan od aritmetičkih operatora. Program proverava koji operater je unet i prikazuje rezultat nakon primene operatora nad unetim varijablama preko terminala. Primer:
  - Unesite A: 2
  - Unesite B: 3
  - Unesite aritmetički operator: +
  - Rezultat sabiranja vrednosti 2 i 3 je 5
6. Napraviti program u koji se unosi broj od 1 do 7, a on ispisuje na konzoli dan u nedelji. Ako je unet neki drugi broj, program ispisuje grešku.

# Zaključak

# O FAJLOVIMA

## *Na osnovu svega navedenog možemo izvesti sledeći zaključak*

Podaci se nazivaju trajnim ako opstaju i nakon završetka programa, odnosno i nakon gašenja računara. Podaci koji se čuvaju u promenljivama programa nisu trajni, pošto se **RAM** memorija, gde se promenljive čuvaju, briše kada program (odnosno računar) prestane sa radom. Stoga je neophodno čuvati podatke na hard disk-u računara ili na nekom spoljašnjem medijumu za skladištenje podataka, kako bi se ti podaci po potrebi mogli ponovo dobiti.

U ovoj lekciji je opisano kako je moguće podatke načiniti trajnim tako što ih čuvamo u datoteke odnosno fajlove. Osim toga je opisano kako je kasnije moguće pristupiti tip podacima i očitati ih iz datoteke.

Datoteka je skup ili kolekcija podataka. Datoteka se čuvaju u trajnom skladištu, kao što je na primer hard-disk, CD-ROM, ili neki drugi medijum za skladištenje podataka.

U datotakama se podaci mogu čuvati u dva različita formata: tekstualnom i binarnom. U tekstualnim datotekama se čuvaju podaci koji su prethodno konvertovani u stringove sadržane od **ASCII** karaktera. Suprotno tome, binarni fajlovi čuvaju podatke u istom formatu u kojem su oni smešteni u **RAM** memoriji, tako da se podaci sastoje iz nula ili jedinica. Programi kao što je **Notepad** mogu da se koriste za prikaz i promenu tekstualnih fajlova. U binarnim fajlovima mogu da se čuvaju složeni podaci, pa se binarni fajlovi koriste za složenije programe kao što je procesiranje teksta rad sa bazama podataka itd.

Postupak pristupa datoteci, bilo da se koristi za čitanje ili upis, se sastoji iz nekoliko koraka. Prvo, naravno treba otvoriti datoteku da bi se uspostavila veza izmedju otvorenog fajla i pokazivača u vašem programu. Drugi korak je naravno čitanje odnosno upis. Treći i konačni korak je zatvaranje datoteke, da bi se sistemski reusrsi koji uspostavljaju vezu izmedju datoteke i vašeg programa oslobodili i da bi se sprečili problemi deljenog pristupa datoteci prouzrokovani slučajem kada se iz jednog dela programa želi pristupiti datoteci koja je u nekom ranijem delu koda otvorena ali nije još uvek zatvorena.

# O OSNOVNIM KARAKTERISTIKAMA C++ JEZIKA

## *Možemo izvesti sledeći zaključak*

Veoma puno profesionalnog softvera je napisano u C++-u. C++ omogućuje objektnu orijentaciju kod razvoja softvera i omogućuje shvatanje objektno orijentacije, koja je univerzalna i za druge o. o. programske jezike. Međutim, u njemu ima i niz novih opcija i mogućnosti, koje nisu objektnog karaktera, zbog kojih pisanje programa koji nisu objektno prirode, lakše sprovesti u C++-u nego u C-u.

C++ ima dodatnu mogućnost učitavanja podataka sa standardnog ulaza odnosno upisa podataka na standardni ulaz. U tu svrhu je moguće koristiti objekat `cin` da bi se dodelila vrednost uneta od strane korisnika, promenljivoj u većem programu. Takođe je moguće koristiti standardne funkcije `cin` objekta: `get` i `getline` za učitavanje karaktera odnosno niza karaktera.

Opisani su problemi koji nastaju pri učitavanju karaktera kada se desi da karakter nove linije ostane u baferu i izvedena su tri pravila kojih se treba pridržavati. Prvo pravilo je da nakon poziva `cin` objekta sa operatorom izvlačenja treba da sledi poziv funkcije `ignore` bez argumenata. Drugo pravilo je da nikako ne treba koristiti funkciju `ignore` nakon poziva funkcije `getline`. Treće pravilo je da nakon korišćenja funkcije `get` koja ima jedan karakter treba proveriti da li je u ulaznom baferu ostao karakter nove linije ili ne, što je moglo da se desi u slučaju korišćenja tastera `ENTER` nakon unosa karaktera.

Opisani su razni modifikatori i specifikari koji mogu biti korišćenji u cilju formatiranja izlaznih podataka. U tu svrhu je moguće koristiti običan, decimalni i eksponencijalni zapis realnih brojeva. Zatim, moguće je vršiti podešavanje širine polja i popunjavanje praznih polja, a moguće je vršiti odgovarajuća poravnavanja: desno, levo, centrirano, itd.