1. Pokazivači i reference

a. Adresni (dereferentni) operator (&) i operator reference (*)

Sve varijable, osim onih koje se smještaju u registre procesora, smještene su u memoriji računala na određenoj adresi. Dakle svaka memorijska lokacija ima adresu, i svakoj memorijskoj lokaciji možemo pristupiti putem te adrese, pa samim tim i varijabli pohranjenoj na toj adresi.

U trenutku deklariranja varijabla mora biti uskladištena na točno određenom mjestu u memoriji računala. U principu, prilikom programiranja ne određujemo na kojem će mjestu u memoriji biti uskladištena varijabla. Brigu o smještaju varijabli u memoriji računala vodi prevodioc tijekom prevođenja i operacijski sustav tijekom izvršenja programa. No, jednom pokrenut program ima dodjeljene adrese svim postavljenim varijablama i ponekad postoji potreba da se pouzdano zna memorijska lokacija uskladištene varijable.

Da bi se odredila memorijska lokacija neophodno je prije identifikacije varijable upisati **(&)** znak koji u doslovnom prijevodu znači «adresa».

Primjer:

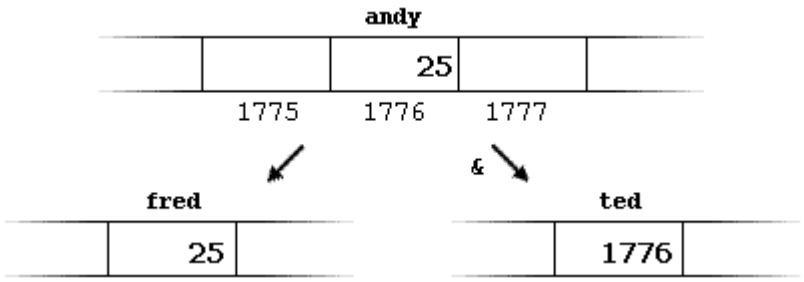
```
ted = &andy;
```

U ovom slučaju varijabli **ted** vrijednost će biti postavljena na memorijsku adresu varijable **andy** jer u slučaju dodavanja znaka **(&)** ispred imena varijable više ne dodijeljujemo vrijednosti varijablama već njenu memorijsku adresu u memoriji računala.

Pretpostaviti ćemo da je varijabla **andy** smještena na memorijskoj adresi **1776** i u sljedećem prijmeru valja uočiti:

```
 andy = 25;
fred = andy;
ted = &andy;
```

rezultat je prikazan na slici memorije:



Varijabla koja uskladištava vrijednost adrese druge varijable (kao **ted** u primjeru na slici) se naziva **pokazivač** (eng. pointer).

U Cu, pokazivač nije osnovni tip (int, float, ...) nego govorimo o pokazivaču na neki tip, npr **pokazivač na char, int, float ili double**.

Pri deklaraciji pokazivača, ispred imena pokazivača stavljamo unarni operator zvjezdica (*).

```
tip  *ime_var;
```

Primjer:

```
float  *niz;
int  *matrica;
```

Pristupanje vrijednosti na koju pokazuje pokazivač također se ostvaruje unarnim operatorom *.

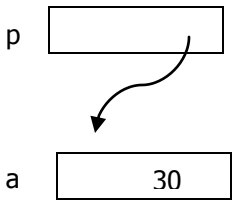
Primjer:

```
int  *m;
*m = 1001;
```

(naredbom **int *m;** definirali smo da je varijabla m pokazivač na vrijednost tipa int. Naredbom ***m = 1001;** vrijednost 1001 nismo upisali u varijablu m, nego smo je upisali **na memorijsku lokaciju** čija adresa se nalazi u varijabli m.)

Primjer:

```
int *p;
int a=30, b;
p=&a;  // varijabli p dodjeljuje se adresa na kojoj je zapisan broj 30 (sadržaj varijable a)
b=*p;  // varijabli b dodjeljuje se sadržaj memorijske lokacije čiju adresu sadrži pokazivač p (b
       poprima vrijednost 30)
```

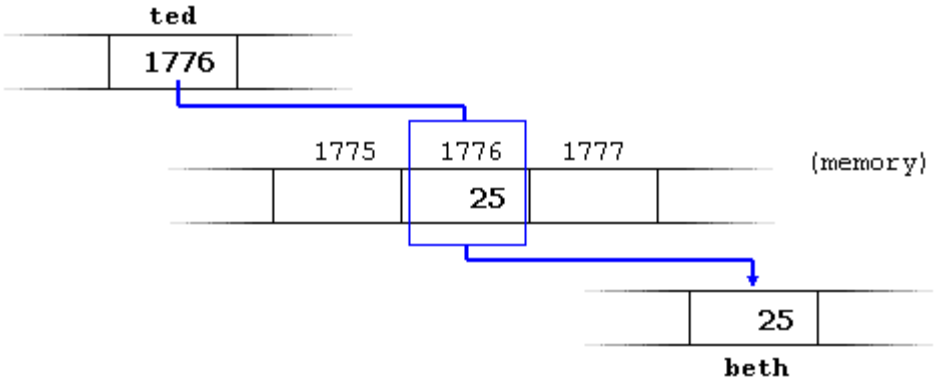


Korištenjem pokazivača možemo direktno pristupiti vrijednosti uskladištenoj u varijabli na koju pokazivač "gleda" ukoliko prije imena pokazivača koristimo oznaku **(*)**, koja može biti doslovno prevedena kao "vrijednost na koju pokazivač "gleda".

Primjer:

```
beth = *ted;
```

možemo zaključiti da varijabla **beth** ima vrijednost na koju pokauje pokazivač **ted**, te slijedeći primjere dolazimo do vrijednosti varijable **beth** koja ima iznos **25** jer pokazivač **ted** ima vrijednost **1776**, a vrijednost na memorijskoj lokaciji 1776 iznosi 25.



Neophodno je uočiti ulogu operatora reference **(*)** i izraz bez njegovog korištenja.

Primjer:

```
beth = ted;    // beth jednako ted ( 1776 )
beth = *ted;   // beth jednako vrijednosti na koju pokazuje ted ( 25 )
```

Adresni (dereferentni) operator (&)
koristi se kao prefix varijable i može biti protumačen kao "adresa", tj. &varijabla1 znači "adresa varijable1"
Operator reference (*)
označava da je neophodno izlučiti vrijednost koja je uskladištena na memorijskoj lokaciji sadržanoj u pokazivaču i može biti protumačen kao "vrijednost na koju gleda pokazivač"

Primjer:

```
andy = 25;
ted = &andy;
```

da slijedeći izrazi po relacijskim operatorima daju vrijednost **true**

```
andy == 25          - primjena asignacijskog operatora (vježba 8)
&andy == 1776       - adresni (dereferentni) operator (mem. lokacija var. andy)
ted == 1776         - budući da je ted = &andy
*ted == 25          - referentni operator vrijednost uskladištena na mem. lokaciji
                      1776 jednaka 25
```

iz svih ovih izraza da se zaključiti da je vrijednost izraza `*ted == andy` budući da je adresa na koju pokazuje pokazivač ted ostala ne promijenjena.

b. Deklariranje varijabli tipa pokazivač

Budući da pokazivač ima karakteristiku direktnog pokazivanja (referenciranja) na vrijednost koju sadrži njegova memorijska lokacija neophodno je jednoznačno odrediti tip varijable na koju pokazivač gleda prilikom njegove deklaracije. Tako deklaracija nije jednaka za pokazivač na varijablu tipa **int**, **float**, **double** i sl. kao u primjeru.

```
tip *ime_pokazivača;
```

gdje je ***tip,*** tip podataka na koju pokazivač pokazuje, a ne tip podataka samoga pokazivača.

Primjer:

```
int *broj;
char *znak;
float *velikibroj;
```

su tri deklaracije pokazivača gdje svaki od njih pokazuje na različiti tip podataka ali svaki od njih zauzima jednaku veličinu u memoriji (koja ovisi o operacijskom sustavu) dok različiti tipovi na koje pokazuju (int, char, float) zauzimaju različit prostor u memoriji.

Napomena: Prilikom deklaracije pokazivača **tip * ime_pokazivača**, treba imati na umu da je **(*)** samo način označavanja deklaracije i ne treba se miješati s oznakom **(*)** koja se primjenjuje prilikom korištenja referentnog operatora. Oznaka je ista, ali služe u dvije različite svrhe.

Primjeru izvornog koda s pokazivačima:

```
// pointer 1
#include <stdio.h>

int main ()
{
  int value1 = 5, value2 = 15;
  int * mypointer;

  mypointer = &value1;
  *mypointer = 10;
  mypointer = &value2;
  *mypointer = 20;
  printf("value1=%d – value2=%d",
value1, value2);
 return 0;
}
```

value1==10 / value2==20

promjena vrijednosti deklariranih varijabli korištenjem pokazivača

da bi se uočilo da pokazivač može imati više različitih vrijednosti u istom programu slijedi primjer:

```
// pointer 2
#include <stdio.h>


int main ()
{
  int value1 = 5, value2 = 15;
  int *p1, *p2;

  p1 = &value1; // p1 = adresa value1
  p2 = &value2; // p2 = adresa value2
  *p1 = 10;   // vrijednost na koju pokazuje p1 = 10
  *p2 = *p1;  // vrijednost na koju pokazuje p2 = vrijednost
na koju pokazuje p1
   p1 = p2;    // p1 = p2 (kopiranje vrijednosti pokazivača)
  *p1 = 20;   // vrijednost na koju pokazuje p1 = 20
  printf("value1=%d – value2=%d",   value1, value2);
 return 0;
}
```

value1==10
value2==20

c. Pokazivači i polja

Samo korištenje polja je vrlo usko povezano s jednim od pokazivača.
Da bismo mogli pristupati pojedinom elementu polja preko pokazivača, potrebno je deklarirati varijablu pokazivač na isti tip kao što je i polje i pridružiti mu početnu adresu polja.

Oznaka polja je identična memorijskoj adresi prvog elemena polja. Na prijmjeru možemo uočiti dvije deklaracije:

```
int numbers [20];
int *p;
```

tako da će asignacija biti valjana:

```
p = numbers;
```

U ovome primjeru pokazivač **p** i varijabla **numbers** su jednake i imaju ista svojstva, dok je jedina razlika u tome što pokazivaču možemo dodijeliti i druge vrijednosti, druge varijable i funkcije dok će varijabla **numbers** uvijek pokazivati na prvi od njenih 20 elemenata (u ovom primjeru) tipa **int** kako je i definirana.

Tako da, za razliku od pokazivača **p**, koji je običan varijabilni pokazivač, varijabla **n** je konstantni pokazivač što se može i pokazati na primjeru:

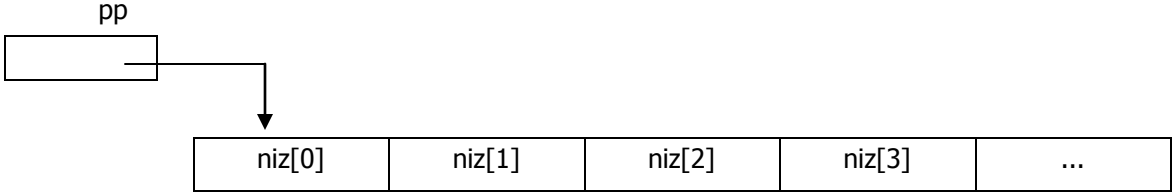
```
numbers = p;
```

gdje je izraz **netočan** jer je varijabla **numbers** polje int brojeva (konstantni pokazivač), a vrijednosti ne mogu biti dodijeljene na ovaj način.

Primjer:

```
int niz[5], *pp, a;
pp= &niz[0];
```

pokazivač pp sadrži adresu prvog člana niza, pa je naredbom a= *pp varijabli a pridružena vrijednost prvog elementa niza.



Kako pp pokazuje na prvi član niza, moramo moći i ostalima. Postoje dva načina kako pristupamo pojedinim elementima niza:

1. Način:

```
int niz[5], *pp, a;
pp= &niz[0];
pp++;  // pokazivač se pomakne na sljedeći el u nizu a zatim se vrijednost tog el
           niza pridruži var a
a= *pp;
```

2. Način:

```
int niz[5], *pp, a;
pp= &niz[0];
a= *(pp+1);  // nema pomicanja pokazivača, ali var a opet je pridružena vrijednost
                  drugog el niza
```

Primjer:

```
// pointer 3
#include <stdio.h>


int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    printf("&d, ", numbers[n]);
 return 0;
}
```

```
10, 20, 30,
40, 50,
```

d. Inicijalizacija pokazivača

Prilikom deklaracije pokazivača možemo i eksplicitno odrediti na koju varijablu želimo vezati pokazivač.

```
int number;
int *tommy = &number;
```

je ekvivalentno:

```
int number;
int *tommy;
tommy = &number;
```

Prilikom asignacije pokazivača uvijek se dodijeljuje adresa na koju pokazivač "gleda", nikada vrijednost varijable na koju pokazuje. Ne treba smetnuti s uma da operator reference koristi istu oznaku, ali nije istoznačnica, stoga:

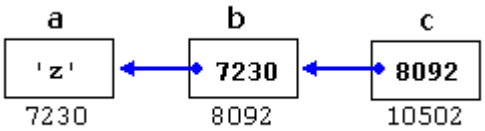
```
int number;
int *tommy;
*tommy = &number; (neispravno!)
```

e. Pokazivač na pokazivač

Programski jezik C++ dozvoljava korištenje pokazivača na pokazivač, tako da u tom slučaju pokazuje na vrijednost. Da bi se to postiglo dovoljno je koristiti operator reference (*) :

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```

tako da za proizvoljno odabrane memorijska mjesta **7230**, **8092** i **10502**, vrijedi:



Prema tome, varijabla **c** može imati tri različita stanja, svako od njih svojom vrijednošću jedinstvena u ovisnosti o korištenju operatora tako da je:

```
c je varijable tipa (char **) s vrijednošću 8092
*c je varijable tipa (char*) s vrijednošću 7230
**c je varijable tipa (char) s vrijednošću 'z'
```

f. Pokazivači na funkcije

Pokazivači donose veliku prednost u korištenju funkcija gdje omogućavaju prijenos funkcije (metode) kao parametra drugoj funkciji kako ona ne može biti prenesena operatorom reference. Prilikom deklaracije pokazivača ne funkciju moramo je deklarirati kao prototip funkcije s iznimkom da ime funkcije mora biti u zagradama i upisanim (*) prije njenog imena kao u primjeru.

```
// pointer 3
#include <stdio.h>

int addition (int a, int b)
{ return (a+b); }

int subtraction (int a, int b)
{ return (a-b); }

int (*minus)(int,int) = subtraction;

int operation (int x, int y, int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
  return (g);
}

int main ()
{
  int m,n;
  m = operation (7, 5, addition);
  n = operation (20, m, minus);
  printf("&d ", n);
  return 0;
}
```

8

U primjeru, **minus** je globalni pokazivač na funkciju koja ima dva parametra tipa **int**, te istovremeno pokazuje na funkciju **subtraction** u jednoj liniji koda:

```
int (* minus)(int,int) = subtraction;
```

2. Rad s datotekama

a. Implementacija podrške rada s datotekama

Programski jezik C++ posjeduje slijedeće zaglavne datoteke kako bi se omogućio rad s datotekama

* ofstream: Stream klasa kako bi se pisalo u datoteku
* ifstream: Stream klasa kako bi se čitalo iz datoteke
* fstream: Stream klasa kako bi se pisalo/čitalo iz datoteke

Ove klase su izvedene direktno ili indirektno iz klase istram i ostream. Više informacija o strukturi izvedenih klasa koje omogućavaju rad s datotekama možete pronaći na web odredištu
http://www.cplusplus.com/ref/iostream .

Ako u programu želimo koristiti datoteke, potrebno je deklarirati datotečnu varijablu:
FILE *datotečna_varijabla
Nakon toga otvaramo datoteku za pristup (da možemo pristupati njenom sadržaju ili takav upisivati).

Funkcije koje omogućuju korištenje datoteka direktno su vezane uz korištenje Stream klasa te se njihova osnovna implementacija nalazi u zaglavnoj datoteci <stdio.h>.

Funkcije koje valja koristiti za manipulaciju datotekama počinju s oznakom **f**, kao što su:
datotečna_varijabla=fopen (ime_datoteke, nacin_otvaranja) // otvara datoteku
fclose (znak, datotecna_varijabla) // zatvara datoteku
fprintf (file, format, podatak1, podatak2, …) // formatirani upis
feof,
ferror,
fflush,
fgetc,
fgetpos,
fgets,
fopen,
fputc,
fputchar,
fputs,
fread,
freopen,
fscanf, **fseek**, **fsetpos** i slično.

Stdio.h stream je predstavljem pokazivačem na Datotečnu strukturu koja sadrži interne informacije o obilježja i indikatore datoteke. Općenito, podatci koji se nalaze u strukturi nisu direktno referencirani. Prilikom korištenja funkcija u **stdio.h**, pokazivači na Datotečnu strukturu koriste se samo kao argumenti ulazno/izlaznih fukcija.

Osnovne fukcije koje se najčešće koriste prilikom pristupa podatcima na disku računala su: fopen, fclose, fread, fwrite, fputc, fgetc.

b. Opis osnovnih funkcija

fopen

definicija: FILE * fopen (const char * filename, const char * mode);

Otvara datoteku čije je ime uskladišteno u varijabli *filename* i vraća kao rezultat pokazivač na dotičnu datoteku (stream). Dostupne funkcije na datoteci definirane su parametrom *mode*.

Parametri:

- **filename**, ime datoteke koju se želi otvoriti. Ovaj parameter mora biti u skladu sa specifikacijama operativnog sistema i može sadržavati cjelokupnu putanju do datotetke.
- **mode**, tip pristupa podatcima i može biti: **"r"** – otvara datoteku za čitanje, **"w"** – kreira praznu datoteku za pisanje. Ukoliko postoji datoteka njezin sadržaj je nepovratno obrisan, **"a"** – dodavanje na kraj u postojeću datoteku ukoliko postoji, u suprotnom je datoteka kreirana, **"r+"** – otvara datoteku za čitanje i pisanje. Datoteka mora biti prethodno kreirana, **"w+"** – kreira praznu datoteku za čitanje i pisanje. Ukoliko datoteka s identičnim imenom postoji, njezin sadržaj je obrisan prije otvaranja datoteke, **"a+"** – otvara datoteku za čitanje i dodavanje. Sve operacije dodavanja se odvijaju na kraju datoteku tako čuvajući prethodno upisani sadržaj.

Parametar **mode** služi također ukoliko želimo otvoriti datoteku kao tekst ili binarnu datoteku. Više informacija o tekst i binarnom obliku manipulacije možete pronaći na web odredištu:
http://www.cplusplus.com/ref/cstdio/fopen.html

Povratna vrijednost: Ukoliko je datoteka uspješno otvorena funkcija će vratiti pokazivač na datoteku. U suprotnom je vraćen pokazivač vrijednosti NULL.

Primjer:

```
/* fopen primjere */
#include <stdio.h>
int main () {
  FILE * pFile; //deklaracija pokazivača
  pFile = fopen ("datoteka.txt","wt"); //postavljanje pokazivača
  if (pFile!=NULL) {
    fputs ("fopen primjer",pFile);
    fclose (pFile);
  } return 0;
}
```

fread

definicija: int fread (void * buffer, size_t size, size_t count, FILE * stream);

Čitanje bloka podataka iz stream-a te njihovo spremanje u postavljeni buffer (privremenu memoriju). Indikator pozicije u stream-u povećava se s brojem pročitanih bajtova.

Parametri:

- buffer – pokazivač na ciljanu strukturu s minimalnom vrijednošću od (size*count) bajtova
- size – veličina u bajtovima svakog sloga za čitanje
- count – broj slogova, svaki definiran svojom veličinom u bajtovima
- stream – pokazivač na otvorenu datoteku

Povratna vrijednost: Ukupna vrijednost pročitanih slogova. Ukoliko se povratni broj razlikuje od broja slogova povratna vrijednost 'kraj datoteke' se vraća kao rezultat. Za provjeru stanja koristiti funkcije **feof** i **ferror**.

fprintf

definicija: int fprintf (FILE * stream , const char * format [, argument , ...]);

Upisuje formatirane podatke u stream kao sekvencu formatiranih argumenata.

Parametri:

- stream – pokazivač na otvorenu datoteku
- format – string koji treba biti upisan. Opcionalno može sadržavati argumente formatiranja. Argumenti formatiranja mogu biti slijedeći:

tip	Ispis	primjer
c	Znak	**a**
d ili **i**	Decimalni integer s predznakom	**392**
e	Inžinjerska notacija (mantisa/eksponent) koristeći **e** znak	**3.9265e2**
E	Inžinjerska notacija (mantisa/eksponent) koristeći **E** znak	**3.9265E2**
f	Decimalni float	**392.65**
g	Skraćeno %e ili %f	**392.65**
G	Skraćeno %E ili %f	**392.65**
o	Oktalni broj s predznakom	**610**
s	String znakova	**sample**
u	Bez predznaka decimalni integer	**7235**
x	Bez predznaka heksadecimal integer	**7fa**
X	Bez predznaka heksadecimal integer (velika slova)	**7FA**
p	Adresa argumenta	**B800:0000**
n	Nema ispisa.	

Povratna vrijednost: Ukoliko je operacija uspješna vraćeni broj jednak je broju upisanih znakova. U suprotnom vraćena je negativna vrijednost broja.

Primjer:

```
/* fprintf primjer */
#include <stdio.h>
int main () {
   FILE * pFile;
   int n;
   char name [100];
   pFile = fopen ("datoteka.txt","w");
   for (n=0 ; n<3 ; n++) {
     puts ("please, enter a name: "); //ekvivaletno printf
     gets (name); //ekivaletno scanf
     fprintf (pFile, "Ime %d [%-10.10s]\n",n,name);
   }
   fclose (pFile);
   return 0;
}
```

fscanf

definicija: int fscanf (FILE * stream , const char * format [, argument , ...]);

Čitanje podataka iz stream-a s trenutne pozicije i spremanje vrijednosti u određenu varijablu. Pozicije određene pojedinim argumentima su popunjene vrijednostima pojedinih tipova podataka.

Parametri:

- stream – pokazivač na otvorenu datoteku Pointer to an open file.
- format – string koji može sadržavati neke od slijedećih tipova: - znakovi koji čine praznine (funkcija ignorira sve prazne znakove: novi red, razmak i sl.), znakovi koji ne čine praznine (u pravilu 'korisni' podatci).

Povratna vrijednost: broj uspješno pročitanih vrijednosti. Ova vrijednost ne uključuje zanemarena polja.

Primjer:

```
// fscanf primjer
#include <stdio.h>
int main (){
  char str [80];
  float f;
  FILE * pFile;

  pFile = fopen ("datoteka.txt","w+");
  fprintf (pFile, "%f %s", 3.1416, "PI");
  rewind (pFile);
  fscanf (pFile, "%f", &f);
  fscanf (pFile, "%s", str);
  fclose (pFile);
  printf ("Procitano: %f i %s \n",f,str);
  return 0;
}
```

fclose

definicija: int fclose (FILE * stream);

Zatvara (datoteku) tj. stream koji je povezan imenom datoteke te briše sve vezane objekte (buffer).

Parametri:

- stream – pokazivač na Datotečnu strukturu koju valja zatvoriti.

Povratna vrijednost: Ukoliko je stream uspješno zatvoren povratna vrijednost je 0, u suprotnom povratna vrijednost je EOF.

Primjer:

```
/* fclose primjer */
#include <stdio.h>
int main (){
  FILE * pFile;
  pFile = fopen ("myfile.txt","wt");
  fprintf (pFile, "fclose primjer");
  fclose (pFile);
  return 0;
}
```