Pokazivači u C jeziku

Kusak, Matej

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:337263

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-25**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Preddiplomski stručni studij elektrotehnike, smjer Informatika

Pokazivači u C jeziku

Završni rad

Matej Kusak

Osijek, 2017.

Sadržaj

1. UVOD 2
1.1. Zadatak završnog rada 2
2. DEKLARACIJA POKAZIVAČA 3
2.1. Pokazivači tipa void 3
2.2. Inicijalizacija 4
2.3. Pokazivači i adrese 5
2.4. Aritmetika pokazivača 6
3. PRISTUP PODACIMA PUTEM POKAZIVAČA 7
3.1. Pokazivači na funkcije 7
3.2. Pokazivači na pokazivač 8
3.3. Pokazivači i polja 8
3.3.1. 2D polja 10
3.3.2. Argumenti 12
4.POKAZIVAČI KAO ARGUMENTI FUNKCIJA 13
4.1. Pokazivači i strukture 13
5. PROGRAM 14
6. ZAKLJUČAK 15
LITERATURA 16
SAŽETAK 17
ABSTRACT 17
ŽIVOTOPIS 18

1.UVOD

Cilj ovog završnog rada je napraviti program koji će raditi s poljima i pokazivačima na njih gdje će se vidjeti primjeri kako se kreira nova matrica, učitavanje vrijednosti nove matrice iz datoteke, ispis matrice te množenje matrica. Uz to ispisuju se i najveće vrijednosti svakog retka predane matrice te najveće vrijednosti svakog stupca predane matrice. Sami pojam pokazivač, prijevod je engleskog izraza *pointer*, kojim se označava nešto ili pokazuje na nešto. U programskim jezicima pokazivači su varijable koje sadrže adresu objekta na koji pokazuju, jer je svojom adresom i veličinom tj. tipom podataka svaki objekt određen. Oni su glavni dio C jezika, tj. snažan alat koji se koristi iz razloga jer samo preko njih možemo izvesti neki račun ili programsku tehniku i ujedno daju kompaktan i efikasan kod. Kada imamo pokazivač na neku varijablu ili objekt njihovom primjenom su omogućene napredne programske tehnike kao što su polimorfizam, dinamičko alociranje memorije te apstraktni tip podataka. Da bi se lakše mogle shvatiti ove tehnike, u sljedećim će poglavljima one biti objašnjene te pokazana njihova primjena.

1.1. Zadatak završnog rada

Obraditi temu pokazivača u programskom jeziku C. Napisati program u dogovoru s mentorom koji će pokazati praktičnu primjenu pokazivača u C-u.

2. DEKLARACIJA POKAZIVAČA

Deklaracija pokazivača je jako slična deklaraciji varijable, kod pokazivača se samo između oznake tipa i samog imena pokazivača obavezno upisuje operator dereferenciranja (operator indirekcije) '*'.

Primjerice,

```
int*p;      //deklariran pokazivač p tipa int
unsigned*q; //deklariran pokazivač q tipa unsigned
```

2.1. Pokazivači tipa void

Jezik C razlikuje više pokazivačkih tipova i tip pokazivača se određuje na osnovu tipa podataka na koji pokazuje. Ovo znači da pokazivači čuvaju informaciju o tipu onoga na što ukazuju, s izuzetkom pokazivača tipa void koji nema informaciju o tipu podataka na koji ukazuje.

U nastavku je naveden kratak primjer s ovim tipom pokazivača.

```
#include <stdio.h>
int mail()
      {
      int x=1;
      int pomocnaVarijabla =10;
      void*py;
      void*nesvrstan;   //pokazivač 'nesvrstan', tipa void pokazuje na
                        //određeni tip te se može preusmjeriti na objekt
                        //bilo kojeg tipa
      py=&pomocnaVarijabla;
      nesvrstan=&x;
      nesvrstan=py;
      nesvrstan=&x;     //prije samog ispisa, pokazivaču tog tipa(void)
                        // treba dodijeliti tip pokazivača int (int*) kako
                        // bi prevoditelj znao pravilnoisčitati sadržaj
                        // memorije na koju taj pokazivač pokazuje
      printf("%d\n",*(int*)nesvrstan);
      return0;
}
```

2.2. Inicijalizacija

Treba znati da je pokazivače prije upotrebe potrebno inicijalizirati, što zapravo znači da im se dodjeljuju vrijednosti adrese memorijskog objekta. Pri inicijalizaciji, tip pokazivača mora biti jednak tipu tog memorijskog objekta, a to je moguće adresnim operatorom '&'.

Na primjer, u programu:

```
int sum;     // deklaracija varijable: sum
int*s;       // ovdje je deklariran pokazivač na objekt tipa int
sum=55;      // inicijalizacija varijable sum
s=&sum;      // ovdje je s inicijaliziran na adresu varijable sum
```

prvo se izvodi deklaracija varijable *sum* te pokazivača s, zatim varijabli *sum* se dodjeljuje vrijednost 55 te inicijaliziramo pokazivač s da pokazuje na tu varijablu.

Svaka varijabla koja je deklarirana u C-u ima vlastitu adresu kojoj se može pristupiti putem adresnog operatora '&'.

Na primjeru sljedećeg koda kao rezultat će biti ispisana adresa varijable x, prilikom svakog novog pokretanja, adrese će se promijeniti:

```
#include <stdio.h>
int main(void){
      double x=333;
      double *px; // ovdje se dodatno definira varijabla px kao pokazivač
                  // na tip double, gdje zvijezdica govori da se radi o
                  // pokazivaču na tip double, a ne o varijabli tog tipa
      px=&x;      // px pokazuje na adresu varijable x
      printf("Vrijednost od x=%f, adresa od x=%p\n",x,px);
      return 0;
}
```

Vrijednost na koju pokazivač pokazuje možemo dohvatiti putem operatora dereferenciranja *, što vidimo u sljedećem kodu:

```
double x=5,y;      // vrijednost tipa double
double*px;         // pokazivač na tip double
px=&x;             // px sada ima adresu varijable x
y=*px;             // y prima vrijednost varijable x (=5)
```

prethodni program bi se mogao zapisati u obliku

```
#include <stdio.h>
int main(void){
      doubley=5;
      double*py;
      py=&y;
      printf("Vrijednost od y=%f, adresa od y=%p\n",*py,py);
      return0;
}
```

Vidi se da * ima različito značenje u deklaraciji i u izvršnoj naredbi. U deklaraciji varijable ona ukazuje da je varijabla tipa pokazivač na dani tip. U izvršnoj naredbi ona predstavlja operator dereferenciranja.

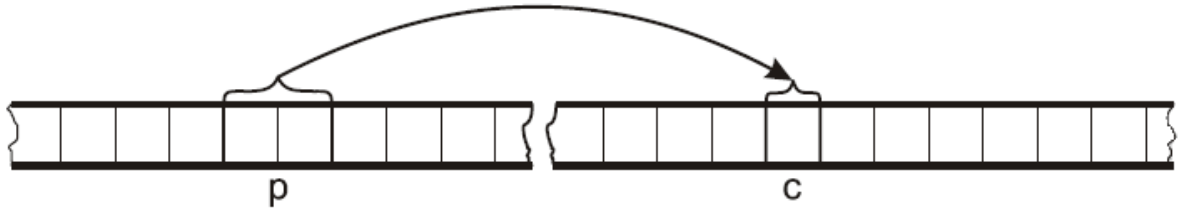
Operator indirekcije koji se označava znakom '*', zapisuje se ispred izraza čija vrijednost predstavlja neku adresu. Pomoću operatora indirekcije dobiva se vrijednost koja je zapisana na toj adresi.

Što znači ako je: y=555, tada je i *(&y)=555 jer operator indirekcije ispisuje vrijednosti koje se nalaze na adresama varijabli, u ovom slučaju varijable y.

2.3. Pokazivači i adrese

Memoriju računala može se gledati kao niz adresiranih memorijskih ćelija, kojima se može pristupati pojedinačno ili u povezanim grupama. Adresu varijable može se odrediti pomoću posebnog operatora&(*adresni operator)*, kojeg koristimo kao unarni operator koji se zapisuje ispred samog imena varijable. [Stojnović, 2016]

Najmanja memorijska jedinica (ćelija) kojoj se može pristupiti je bajt. Jedan bajt memorije je dovoljan za čuvanje jednog *char* podatka, dva bajta mogu čuvati *short* cjelobrojni tip podatka, a četiri uzastopna bajta mogu formirati *long*. Slično, za smještanje pokazivača se najčešće koriste grupe od dvije ili četiri uzastopne ćelije. Na sljedećoj slici je shematski prikazana situacija u memoriji kada pokazivač pokazuje na promjenjivi tip *char*.



Slika 2.3. Primjer memorije računala gdje pokazivač *p* pokazuje na adresu od *c* [Stojnović, 2016]

Da bi se dobila adresa nekog promjenjivog tipa u memoriji, koristimo operator &. Tako izraz:

```
p=&c;
```

pokazivaču *p* dodjeljuje adresu *c* i tada se kaže da *p* „pokazuje na“ *c*.

2.4. Aritmetika pokazivača

Pretpostavlja se da pokazivač *pa* pokazuje na broj u cjelobrojnom polju *a* veličine 5, može se pisati *pa = a*, tako će pokazivač *pa* pokazivati na prvi element u polju *a* što se vidi u sljedećem kodu:

```
#include <stdio.h>
int main()
{
int*pa;                         // pokazivač pa
int a[5]={10,20,30,40,50};      // cjelobrojno polje od 5 brojeva
int y;                          // cjelobrojna varijabla
    pa = a;                     // pokazivač pa pokazuje na prvi element u
                                // polju a

int i =3;                       // varijabla koja se koristi kao index kod
                                // polja
    y =*(pa + i);               // y se postavlja na vrijednost na trećem
                                // mjestu u polju a
    printf("%d\n,y);            // ispis vrijednost trećeg
return0;                        // mjesta u polju a40
}
```

Da se `y =*(pa+i);` napisao bez zvjezdice u obliku `y=(pa+i),` kao rezultat bi se dobiva memorijska adresa, a ne vrijednost trećeg mjesta u polju.

3. PRISTUP PODACIMA PUTEM POKAZIVAČA

Programski jezik C smatra se zapravo jezikom niske razine i to iz razloga što omogućuje korištenje numeričkih vrijednosti adresa varijabli i funkcija pomoću kojih se može indirektno manipulirati podacima i izvođenje programa, za koju svrhu postoje posebni operatori. Adresni operator '&' koji određuje adresu i operator indirekcije (unarni operator) '*' koji daje sadržaj operanda, te specijalni tip varijabli koje se nazivaju pokazivačke varijable ili pokazivači. [Mateljan, 2005]

3.1. Pokazivači na funkcije

Funkcije se također gleda kao memorijski objekti pa ih se može deklarirati i inicijalizirati pokazivače na funkcije.

Pokazivač na funkciju deklarira se kao:

```
tip_pod(*ime)(tip1 arg1,..., tip_n arg_n);
```

Ovdje je *ime* pokazivač na funkciju koja uzima *n* argumenata od*tip_1* do *tip_n* i vraća vrijednost tipa *tip_pod*.

Pravilo je da se pokazivač za funkciju, kojuse definira u obliku:

```
oznaka_tipa F(list_ parametara);
```

definira na sljedeći način:

```
oznaka_tipa (*pF)(list_ parametara);
```

Kada se ime funkcije napiše bez zagrada, to predstavlja adresu funkcije, te pridjelom vrijednosti:

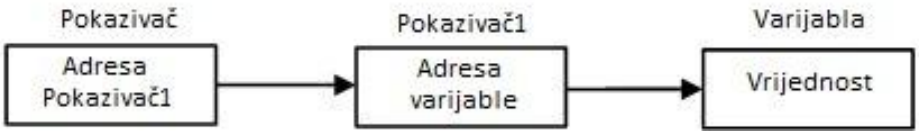
```
pF = F;
```

zapravo se inicijalizira pokazivač pF na adresu funkcije 'F'. Nakon inicijalizacije pokazivača, njegovom indirekcijom se može izvesti i poziv funkcije na sljedeći način:

```
(*pF)(lista_argumenata);
```
ili na još jednostavniji način:
```
pF(lista_argumenata);
```
jer, zagrade zapravo predstavljaju operator poziva funkcije, no kompajlerće sam izvesti indirekciju pokazivača, ako se iza njega napišu zagrade kako je u primjeru iznad.

3.2. Pokazivači na pokazivače

Pokazivač na pokazivač je zapravo oblik višestrukih neizravnosti, tj. lanac pokazivača. Kada definira se pokazivač na pokazivač, prvi pokazivač sadrži adresu drugog pokazivača, koji sadrži adresu varijable na koju zapravo pokazuje.



Slika 3.2.1. Pokazivač na pokazivač

Takav pokazivač se deklarira sljedećom linijom koda:
```
int **pokazivac;
```

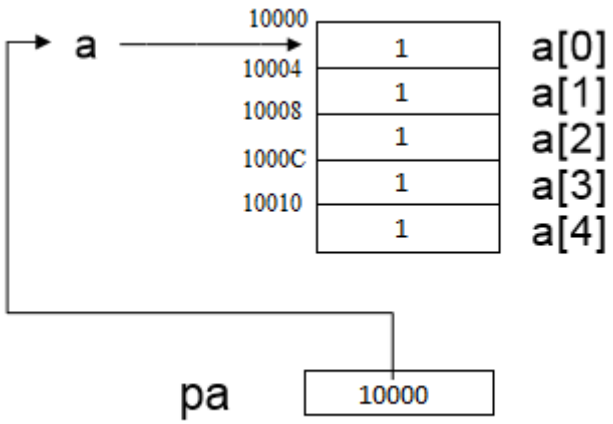
3.3. Pokazivači i polja

Kod jednodimenzionalnih polja članovi polja su poredani u niz jedan iza drugoga, a indeks svakog od članova odgovara njegovoj udaljenosti od prvog člana. Svaki od članova niza se označava cjelobrojnim indeksom i to tako da prvi član niza ima indeks 0, a posljednji član indeks za jedan manji od duljine polja.

Deklaracija polja: *int a[5];* definira polje *a* veličine 10 elemenata s elementima polja *a[0], a[1], ..., a[4]*.

Pokazivač na cjelobrojni tip:

```
int*pa;
pa =&a[0];  // pokazivač pa pokazuje na prvi element polja a Slika 3.3.2.,
            // što se još može zapisati i kao pa=a
```

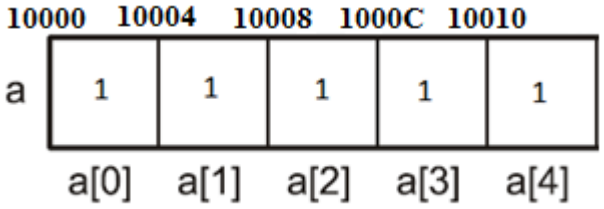


Slika 3.3.2. Pokazivač *pa* pokazuje na adresu 10000, tj. na prvi element polja [Ipšić, 2009]

Svaka operacija koja se može obaviti korištenjem nizova, može se obaviti i pomoću pokazivača. Varijanta koja koristi pokazivače je brža no teža za razumijevanje.

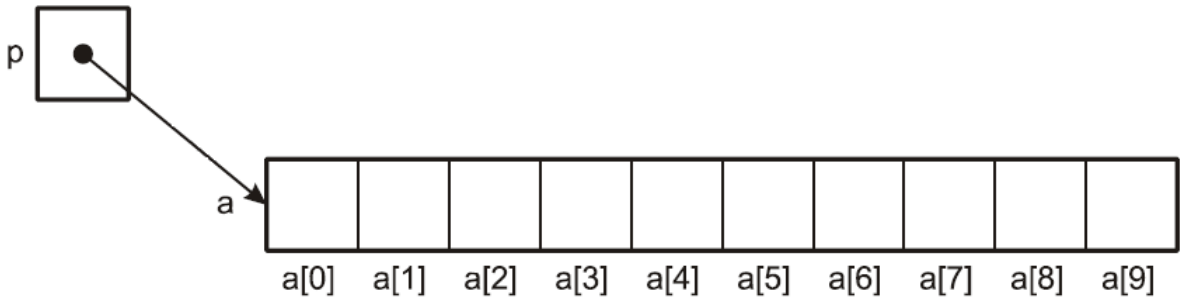
Deklaracija: int a[5];

definira niz *a* veličine 5, odnosno niz od 5 uzastopnih cijelih brojeva *a[0]*, *a[1]*, ..., *a[4]*.



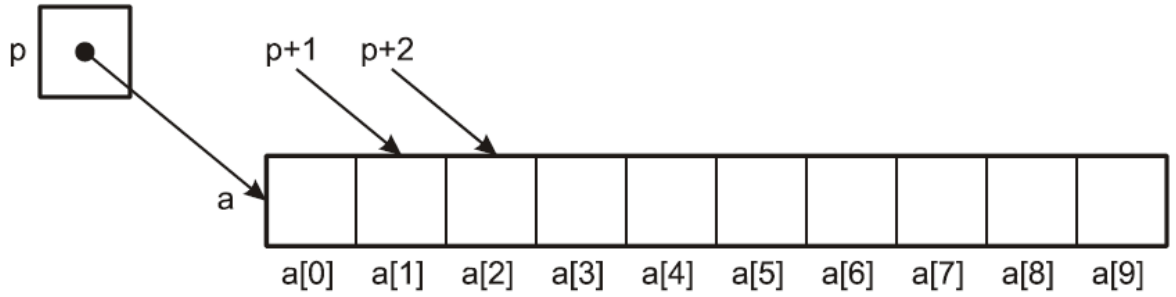
Slika 3.3.3. Shematski prikaz niza *a* u memoriji računala sa adresama od 10000 do 10010 i njihovim vrijednostima [Stojnović, 2016]

Ukoliko je *p* pokazivač na cijeli broj, deklariran kao *int *p;* tada linija *p = &a[0];* postavlja pokazivač *p* da pokazuje na nulti element niza *a*, ili drugačije rečeno pokazivač *p* sadrži adresu elementa *a[0]*.



Slika 3.3.4 Pokazivač *p* pokazuje na nulti element niza [Stojnović, 2016]

Ukoliko pokazivač *p* pokazuje na određeni element niza, po definiciji *p+1* pokazuje na sljedeći element, *p+i* pokazuje na *i*-ti element nakon *p*, a *p-i* pokazuje na *i*-ti element ispred *p*. Tako, ukoliko *p* pokazuje na *a[0]*, onda *(p+1)predstavlja sadržaj elementa *a[1]*, a*(p+i) sadržaj elementa *a[i]*.



Slika 3.3.5 Pristupanje elementima niza pomoću pokazivača [Stojnović, 2016]

Ovakav način pristupa elementima niza može se koristiti bez obzira na tip elemenata niza, tako da *p+1* uvijek predstavlja sljedeći element niza, a *p+i* predstavlja *i*-ti element nakon *p*, bilo da se radi o nizu cijelih brojeva ili nekom drugom.

3.3.1. 2D polja

Polje je ograničen blok memorije, tako je 2D polje veličine *m*n* definirano:

```
int A[m][n];
```

i njegovim elementima se može pristupiti sa:

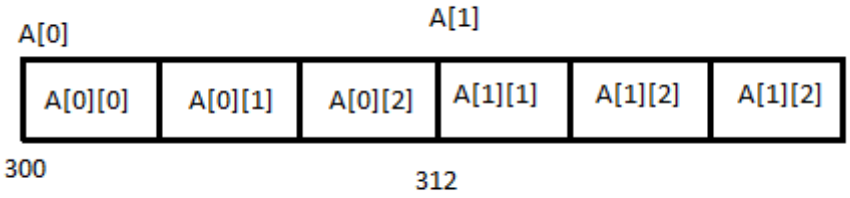
```
A[i][j];
```

gdje je *i* indeks retka, a *j* indeks stupca.

2D polja zapravo se može zamisliti kao dva jednodimenzionalna polja, što znači da je svaki red u 2D polju jedno jednodimenzionalno polje koje se može zapisati jedno za drugim.
Stoga u 2D polju:

```
int A[2][3];
```

može se zamisliti da je A[0] adresa nultog reda, a A[1] adresa prvog reda. Kao što je prikazano na *Slici 3.1.1.*



Slika 3.3.1.1 Prikaz 2D polja razdvojenih na jednodimenzionalna

Ime polja vraća pokazivač na prvi element polja, tu prvi element nije integer već jednodimenzionalno polje od tri integera.
Kada bi se napisalo:

```
int *p=A;
```

Kompajler će izbaciti grešku i *A* će vratiti pokazivač na jednodimenzionalno polje od tri integera, a ne samo pokazivač na integer.

Pravilan zapis je:

```
int (*p)[3]=A;
```

Tako naredbom:

```
print A;
```
ili `&A[0];`
dobije se ispis adrese nultog reda u memoriji što je *300.*

Za pristup prvom elementu nultog jednodimenzionalnog polja:

```
print *A;
```
ili

```
A[0];
```
ili
```
&A[0][0];
```

3.3.2. Argumenti

Pri pozivu funkcije, u pravilu programskog jezika C prenosi se vrijednost stvarnog argumenta, dok se u funkciji taj parametar tretira kao lokalna varijabla koja opet ima vrijednost tog stvarnog argumenta. U slučaju kad je parametar funkcije pokazivač, tada je stvarni argument funkcije zapravo adresa objekta koji ima isti tip kao i taj pokazivač. Da bi se pristupilo tom objektu i mijenjalo njegov sadržaj koristi se indirekciju pokazivača, što znači da taj objekt možemo tretirati kao varijablu koja se koristi i u funkciji.

Taj način prenošenja parametara funkcije prikazan je u sljedećem programu:

```
#include <stdio.h>

void Inc(int*pVar){           // ovdje se inkrementira vrijednost
                              // varijable, čiju adresu prenosimo
                              // u funkciju kao vrijednost tog pokazivača,
                              // a varijabli pristupamo pomoću indirekcije
                              // tog istog pokazivača(pVar)

(*pVar)++;
}
int main(){
      int var =5;
      Inc(&var);              printf ("var=%d\n", var);
      return0;
}
```

4. POKAZIVAČI KAO ARGUMENTI FUNKCIJA

Kao argumente funkcije osim samih pokazivača mogu se prikazati i polja. Kod jednodimenzionalnih polja članovi su poredani u niz jedan iza drugoga, a indeks svakoga od članova odgovara njegovoj udaljenosti od prvog člana. Svaki od članova niza se označava cjelobrojnim indeksom i to tako da prvi član niza ima indeks 0, a posljednji član indeks za jedan manji od duljine polja. Što se tiče dvodimenzionalnog polja, može ga se predočiti tablicom sa zadanim brojem redaka i stupaca. Položaj člana unutar dvodimenzionalnog polja označen je sa dva cjelobrojna indeksa, prvi indeks određuje redak, a drugi stupac. Osim polja, strukture također mogu biti argumenti funkcije. Sama struktura je skup varijabli koje imaju zajedničko ime. Za razliku od polja, elementi strukture ne moraju biti istoga tipa te se pomoću njih lakše organiziraju podaci koji čine cjelinu ili objekt.

4.1. Pokazivači i strukture

Pokazivač na strukturu definira se kao i pokazivač na osnovne tipove varijabli. Da bi pristupili članovima strukture mora se koristiti operator pristupa. Za pristup članovima struktura i klasa koristi se operator .(točka) i ->. Operator točka se koristi za pristup članovima strukture, dok se operator -> koristi za pristup članu strukture koji je pozvan od strane pokazivača što se vidi u sljedećem primjeru koda.

```
#include <stdio.h>
typedef struct osoba
{
	int dob;
	float tezina;
};
int main()
{
	struct osoba,*osobaPtr, osoba1;
	osobaPtr=osoba1;                    //pokazivač pokazuje na
                                        //memorijsku adresu od osoba1

	printf("Unesi dob: ");
	scanf("%d",&(osobaPtr).dob);
	printf("Unesi tezinu: ");
	scanf("%d",&(osobaPtr).tezina);
	printf("Ispis: ");
```

```
    scanf("%d",&(osobaPtr).dob,(*osobaPtr).tezina);
    return0}
```

5. PROGRAM

Zadatak je napisati program koji će raditi s 1D i 2D poljima i pokazivačima na njih. U programu je potrebno deklarirati strukturu koja će za svoje članove imati broj redaka matrice, broj stupaca matrice te podatke matrice. Potrebno je deklarirati nekoliko funkcija.

newMatrix - Kreira novu matricu dimenzija predanih u parametrima, te vraća adresu te matrice. Unutar funkcije potrebno je dinamički alocirati memoriju za podatke u matrici.

newRandom - Kreira novu matricu dimenzija predanih u prva dva parametra. Kreiranu matricu potrebno je popuniti slučajnim brojevima, gdje druga dva parametra predstavljaju minimalnu i maksimalnu vrijednost raspona unutar kojega će biti kreirani slučajni brojevi.

newFromFile - Kreira novu matricu čije će vrijednosti učitati iz datoteke te vraća adresu te matrice. Parametar koji se predaje je obično polje znakova koje predstavlja naziv datoteke u kojoj se nalazi zapisana matrica te broj redaka i stupaca matrice.

newFromSTDIN - Kreira novu matricu dimenzija predanih u parametrima, te traži od korisnika upis vrijednosti elemenata matrice s tipkovnice.

matrixPrint - ispis matrice u matričnom obliku.

matrixMultiply - Obavlja množenje dvije matrice. Vratiti rezultantu matricu, ili NULL ako dimenzije predanih matrica nisu ispravne.

matrixHighestValuesRow - vraća 1D polje s najvećim vrijednosti svakog retka predane matrice

matrixHighestValuesColumn - vraća 1D polje s najvećim vrijednosti svakog stupca predane matrice.

6. ZAKLJUČAK

Pokazivači zauzimaju glavno mjesto u oblikovanju C programa. Pokazivač je zapravo varijabla koja sadrži adresu i ako je to adresa varijable, tada se kaže da pokazivač "pokazuje" na tu varijablu. U radu s njima koriste se dva određena operatora: adresni operator '&' i operator indirekcije '*'. Kada bi se adresni operator napisalo ispred imena varijable, tada on u izrazu vraća adresu te varijable, dok operator indirekcije, ako ga se upiše ispred imena pokazivača, kao ispis vraća sadržaj varijable na koju taj pokazivač pokazuje.

Pokazivači i nizovi su u posebnom odnosu, ako je ime niza napisano bez uglatih zagrada tada ono predstavlja pokazivačku konstantu koja pokazuje na prvi element tog niza. Kao karakter indeksnog operatora su uglate zagrade, jer kada ih se navodi iza imena pokazivača, uzrokuju da se na dalje, s tim pokazivačem može postupati kao s nizom. Da bi se niz prenijelo u funkciju zapravo mora se prenijeti pokazivač na prvi element niza u funkciju, tj. adresu prvog elementa niza. Kako funkcija zna adresu niza, na dalje u njoj možemo koristiti naredbe za mijenjanje sadržaja u elementima niza.

LITERATURA

[1] Programski jezik C, https://web.math.pmf.unizg.hr/~singer/Prog_Add/c.pdf [svibanj, 2017]

[2] Pokazivači i nizovi, https://sr.scribd.com/doc/312230538/SPA2-Pokazivaci-i-nizovi-pdf [svibanj, 2017]

[3] Programiranje, http://www.riteh.uniri.hr/zav_katd_sluz/zr/nastava/racprog/download/predavanja/Programiranje_6_2010.pdf [svibanj, 2017.]

[4] C programiranje fesb, http://www.tenis-as.com/download-eknjige/c-programiranje-fesb.pdf [svibanj, 2017.]

[5] Mala škola pokazivača, http://poincare.matf.bg.ac.rs/~jelenagr/2d/MalaSkolaPokazivaca.pdf [svibanj, 2017]

SAŽETAK

Naslov: Pokazivači u C jeziku

U ovom završnom radu cilj je bio što više objasniti kako rade pokazivači te što je sve s njima moguće raditi. Objašnjeno primjerima i komentarima, te napravljeni program kako je dobiven zadatak da se odrade funkcije koje su također objašnjene i prikazane u kodu za lakše razumijevanje. Najveći problem s kojim sam se susrećemo je zapravo bila dinamička alokacija memorije. Za uspješno rješavanje problema se moralo naučiti mnogo toga o načinu na koji se strukture dinamički alociraju i kako se pristupa njihovim elementima.

Ključne riječi: pokazivači, programski jezik, memorija, dinamičko alociranje

ABSTRACT

Title: *Pointers in C*

In this thesis, the goal was to explain how the pointers are working and what's all possible with them. A program was created, explained by examples and comments, given as a task to process functions that are also described and displayed in the code for easier understanding. The biggest problem I encountered was actually the dynamic allocation of memory. For a successful problem solving, I had to learn a lot about how structures are dynamically allocated and how can I access their elements.

Keywords: *pointers, programming language, memory, dynamic allocation*

ŽIVOTOPIS

Matej Kusak rođen je 18.04.1993. u Koprivnici. Osnovnu školu pohađao u Pitomači (Osnovna škola Petra Preradovića). Sudjelovao na natjecanjima iz Matematike i Engleskog jezika. 2012. godine, nakon završene Strukovne škole u Đurđevcu, smjer Tehničar za računalstvo, upisao preddiplomski stručni studij Elektrotehnike, smjer Informatika na Elektrotehničkom fakultetu u Osijeku, kao redovan student. Praksu obavljao u Virtual d.o.o u Đurđevcu.

PRILOG A

Glavni dio programa

```
#include <stdio.h>      //sadrži osnovne funkcije za input/output
#include <stdlib.h>     //sadrži osnovne funkcije za input/output
#include <string.h>     //služi za rukovanje tekstualnim nizovima
#include <time.h>       //da bi dobili nasumičnu vrijednost koristi se
                        // funkcija rand() koja treba ovu bibilioteku
#include "functions.h"  //header datoteka koja mora biti ovdje uključena

#define len1 (sizeof matrica1/ sizeof (Matrix))
#define len2 (sizeof matrica2/ sizeof (Matrix))
#define len3 (sizeof matrica3/ sizeof (Matrix))

int main(int argc,char*argv[])
{
      int n, m;
      //Matrix je struktura i nalazi se u functions.h, a ovdje se dinamički
      //alociraju četiri
      //matrice, dinamička alokacija znači da nije unaprijed definirana
      //veličina u memoriji
      //nego se može mijenjati
      Matrix *matrica1,*matrica2,*matrica3,*rezMat;
      //naziv .txt datoteke iz koje se čitaju brojevi
      char* fileName ="matrix.txt";

      printf("\nUnesite broj redaka matrice:\n");
      scanf("%d",&n);
      printf("\nUnesite broj stupaca matrice:\n");
      scanf("%d",&m);

      // Kreiranje matrica, svaka matrica je ranije dinamički alocirana ali
      //je prazna, kada
      //stavimo: ime_matrice = funkcija(parametri),  prosljeđuju se
      //parametre u funkciju
      //te se zatim matrica popunjava
      rezMat = newMatrix(n, m);
      matrica1 = newRandom(n, m);
      matrica2 = newFromSTDIN(n, m);
      matrica3 = newFromFile(fileName, n, m);

      // Množenje matrica
      matrixMultiply(&matrica2,&matrica3,&rezMat, len1,
      len2, len3, n, m);
      //Najveća vrijednost svakog retka predane matrice
      matrixHighestValuesRow(matrica1, n, m);
      //Najveća vrijednost svakog stupca predane matrice
      matrixHighestValuesColumn(matrica1, n, m);

      // Oslobađanje memorije, obzirom da su sve matrice dinamički
      //alocirane, oslobađanje memorije je obavezno
      freeMemory(matrica1, len1);
      freeMemory(matrica2, len2);
      freeMemory(matrica3, len3);
```

```
	printf("\nProgram je zavrsen, memorija dealocirana.\n");
	return0;}
```

***Source* datoteka**

Ovdje se nalaze konkretne funkcije

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "functions.h"

//FUNKCIJE
//Alociranje memorije za matricu
Matrix* newMatrix(int n,int m)
{
int i;
	Matrix* matrica = malloc(sizeof(Matrix));
	matrica->M = m;
	matrica->N = n;

	// Alociranje redaka
	matrica->values = malloc(n *sizeof(int*));
	// Alociranje stupaca
	for(i =0; i < matrica->N; i++)
		matrica->values[i]= malloc(m *sizeof(int));

//Inicijalizacija svih elemenata na nulu. Radi se prvo malloc(), a zatim
//inicijalizacija svih elemenata (umjesto koristenja calloc() funkcije)
	for(n =0; n < matrica->N; n++)
	{
		for(m =0; m < matrica->M; m++)
		{
			matrica->values[n][m]=0;
		}
	}
	return matrica;
}

// Popunjavanje i printanje matrice nasumicnim vrijednostima
Matrix* newRandom(int n,int m)
{
	int i, j;
	Matrix* matrica = malloc(sizeof(Matrix));
	matrica->M = m;
	matrica->N = n;

	// Alociranje redaka
	matrica->values = malloc(n *sizeof(int*));

	// Alociranje stupaca
	for(i =0; i < matrica->N; i++)
		matrica->values[i]= malloc(m *sizeof(int));

	// Odabir raspona slucajnog broja
	srand(time(NULL));
	int min, max;
	printf("\nUnesite minimalnu vrijednost raspona slucajnog broja:\n");
	scanf("%d",&min);
```

```
	printf("\nUnesite maksimalnu vrijednost raspona slucajnog broja:\n");
	scanf("%d",&max);

	// Postavljanje vrijednosti svih elemenata matrice
	for(n =0; n < matrica->N; n++)
	{
		for(m =0; m < matrica->M; m++)
		{
			int value = rand()%(max +1- min)+ min;
			matrica->values[n][m]= value;
		}
	}

	printf("\n--Matrica sa nasumicnim vrijednostima u zadanom rasponu.--
");

// Poziv funkcije matrixPrint u koju se predaju adresa matrice i njene
//dimenzije
	matrixPrint(&(*matrica), n, m);

	return matrica;
}
// Rucni unos matrice
Matrix* newFromSTDIN(int n,int m)
{
	int i, value;
	Matrix* matrica = malloc(sizeof(Matrix));
	matrica->M = m;
	matrica->N = n;

	// Alociranje redaka
	matrica->values = malloc(n *sizeof(int*));

	// Alociranje stupaca
	for(i =0; i < matrica->N; i++)
		matrica->values[i]= malloc(m *sizeof(int));
	i =0;
	printf("Matrica s rucnim unosom elemenata.\n");

	// Petlja koja omogucuje rucni unos svih elemenata matrice
	for(n =0; n < matrica->N; n++)
	{
		for(m =0; m < matrica->M; m++)
		{
			printf("Unesite %d. element matrice.\n", i +1);
			scanf("%d",&value);
			matrica->values[n][m]= value;
			i++;
		}
	}

	printf("\n--Matrica s elementima koje korisnik unosi.--");
	matrixPrint(&(*matrica), n, m);

	return matrica;
}

// Ucitavanje matrice iz tekstualne datoteke
Matrix* newFromFile(char* fileName,int n,int m)
{
```

```
	int i, value;
	FILE *fp;
	Matrix* matrica = malloc(sizeof(Matrix));
	matrica->M = m;
	matrica->N = n;

	// Alociranje redaka
	matrica->values = malloc(n *sizeof(int*));

	// Alociranje stupaca
	for(i =0; i < matrica->N; i++)
		matrica->values[i]= malloc(m *sizeof(int));

	// Otvaranje tekstualne datoteke cija putanja je predana u parametri-
	//ma funkcije
	fp = fopen(fileName,"r");

	// Postavljanje vrijednosti svih elemenata matrice
	for(n =0; n < matrica->N; n++)
	{
		for(m =0; m < matrica->M; m++)
		{
			fscanf(fp,"%d",&value);
			matrica->values[n][m]= value;
		}
	}

	printf("\n--Matrica ucitana iz tekstualne datoteke.--");
	matrixPrint(&(*matrica), n, m);

	fclose(fp);
	return matrica;
}

// Mnozenje matrica
void matrixMultiply(Matrix *matrica2, Matrix *matrica3, Matrix *rezMat,int
n,int m)
{
	int i =0, j =0, k =0;

	matrica2->M = m;
	matrica2->N = n;
	matrica3->M = m;
	matrica3->N = n;
	rezMat->M = m;
	rezMat->N = n;

	printf("--Mnozenje matrice koju ste unijeli s matricom ucitanom iz
datoteke.--\n");

	// Provjera uvjeta mogu li se matrice množiti. Ako ne, vraca se NULL;
	//ako da, ide se u mnozenje
	if(matrica2->N != matrica3->M)
	{
		printf("Matrice se ne mogu pomnoziti, dimenzije ne odgovara-
ju!\n");
		returnNULL;
	}
	else
	{
```

```
            printf("Dimenzije matrica odgovaraju, mogu se pomnoziti.\n");

            // Trostruka for petlja koja vrsi mnozenje matrica po matema-
            //tickoj formuli
            for(i =0; i < matrica2->N; i++)
            {
                  for(j =0; j < matrica3->M; j++)
                  {
                        for(k =0; k < matrica2->M; k++)
                        {
                              rezMat->values[i][j]+= matrica2->values[i][k]*
matrica3->values[k][j];
                        }
                  }
            }

            matrixPrint(&(*rezMat), n, m);
      }
}

// Ispis matrice
void matrixPrint(Matrix *matrica,int n,int m)
{
      printf("\n");
      printf("Matrica:\n");

      // Petlja koja prolazi kroz matricu i ispisuje sve elemente
      for(n =0; n < matrica->N; n++)
      {
            for(m =0; m < matrica->M; m++)
            {
                  printf("%d \t", matrica->values[n][m]);
            }
            printf("\n");
      }
      printf("\n\n");
}

// Trazenje retka s najvecom vrijednosti
void matrixHighestValuesRow(Matrix *matrica,int n,int m)
{
      matrica->M = m;
      matrica->N = n;

      int max[20];
      int i, t;

      // Petlja koja trazi najvecu vrijednost u svakom retku
      for(n =0; n < matrica->N; n++)
      {
            t = matrica->values[n][0];
            for(m =0; m < matrica->M; m++)
            {
                  if(matrica->values[n][m]> t)
                  {
                        t = matrica->values[n][m];
                        max[n]= t;
                  }
            }
      }
```

```
	// Ispis 1D polja koje sadrzi najvece vrijednosti
	printf("\Najvece vrijednosti u svakom retku:");
	for(i =0; i < n; i++)
	{
		printf("\n%d", max[i]);
	}
	printf("\n\n");
}

// Trazenje stupca s najvecom vrijednosti
void matrixHighestValuesColumn(Matrix *matrica,int n,int m)
{
	matrica->M = m;
	matrica->N = n;

	int max[20];
	int i, t;

	// Petlja koja trazi najvecu vrijednost u svakom stupcu
	for(n =0; n < matrica->N; n++)
	{
		t = matrica->values[0][m];
		for(m =0; m < matrica->M; m++)
		{
			if(matrica->values[n][m]> t)
			{
				t = matrica->values[n][m];
				max[m]= t;
			}
		}
	}

	// Ispis 1D polja koje sadrzi najvece vrijednosti
	printf("\Najvece vrijednosti u svakom stupcu:");
	for(i =0; i < n; i++)
	{
		printf("\n%d", max[i]);
	}
	printf("\n\n");
}

// Oslobadjanje dinamicki alocirane memorije
void freeMemory(Matrix *matrica)
{
	int i, value;

	// Prolazak kroz svaki redak i brisanje svakog elementa
	for(i =0; i < matrica->N; i++)
		free(matrica->values[i]);

	free(matrica->values);

	// Oslobadjanje strukture "matrica"
	free(matrica);
}
```

Header datoteka

Ovdje su definirane funkcije i struktura

```
// Deklaracija strukture Matrix
typedefstruct
{
	int N;
	int M;
	int**values;
} Matrix;

// Deklaracija funkcija
Matrix* newMatrix(int,int);
Matrix* newRandom(int,int);
Matrix* newFromSTDIN(int,int);
Matrix* newFromFile(char*,int,int);
void matrixMultiply(Matrix *, Matrix *, Matrix *,int,int);
void matrixHighestValuesRow(Matrix*,int,int);
void matrixHighestValuesColumn(Matrix*,int,int);
void matrixPrint(Matrix*,int,int);
void freeMemory(Matrix*);
```

PRILOG B

Dokumenti:

Matej Kusak – pokazivači u C jeziku.doc
Matej Kusak – pokazivači u C jeziku.docx
Matej Kusak – pokazivači u C jeziku.pdf

Datoteke:

main.c
munctions.c
munctions.h
test2.exe