Polja u programskom jeziku C

Cestar, Toni

Undergraduate thesis / Završni rad

2018

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:200:928882

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-18**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

ELEKTROTEHNIČKI FAKULTET

Stručni studij

POLJA U PROGRAMSKOM JEZIKU C

Završni rad

Toni Cestar

Osijek, 2018.

SADRŽAJ:

1. UVOD 1
1.1. Zadatak završnog rada 1
2. POLJA U PROGRAMSKOM JEZIKU C 2
2.1. Deklaracija i inicijalizacija polja 2
2.2. Memorija u programskom jeziku C 3
2.2.1. Statička alokacija memorije 3
2.2.2. Dinamička alokacija memorije 4
3. JEDNODIMENZIONALNA POLJA 6
3.1. Vektori 6
3.2. Polje znakova (engl. *string*) 9
3.3. Polje pokazivača 12
4. VIŠEDIMENZIONALNA POLJA 14
4.1. Dvodimenzionalna polja 14
4.2. Višedimenzionalna polja 20
5. ZAKLJUČAK 21
6. LITERATURA 22

1. UVOD

U programerskoj praksi često se zahtijeva rad s grupom elemenata istoga tipa poput imena osoba, vrijednosti očitanja pojedinih instrumenata, matematičkih modela raznih nizova i slično. Navedeni podaci se mogu jednostavno prikazati kao elementi unutar polja. Polje u programskom jeziku C se može definirati kao skup memorijskih lokacija u koje se mogu spremati podaci istog tipa i kojima se može pristupiti preko istog imena varijable. Ti elementi mogu biti cjelobrojni (engl. *integer*) ili realni brojevi jednostruke preciznosti (engl. *float*) koji pohranjuju brojeve ili nizovi znakovnih elemenata (engl. *char*), polja pokazivača te polja struktura (engl. *struct*). Inače, polje koje se sastoji od znakovnih elemenata se naziva *string*. Svi elementi unutar polja moraju biti istoga tipa podatka. Još jedan razlog za korištenje polja nasuprot korištenja više odvojenih varijabli je učinkovitiji im pristup i njihovo procesiranje. Završni rad će obraditi temu polja u programskom jeziku C uključujući karakteristike polja, načine zapisa polja u računalnu memoriju, načine njihove deklaracije, inicijalizacije, ispisa, te će se obraditi jednodimenzionalna i višedimenzionalna polja. Također će biti objašnjene i ostale osnovne funkcije za rad s poljima u programskom jeziku C.

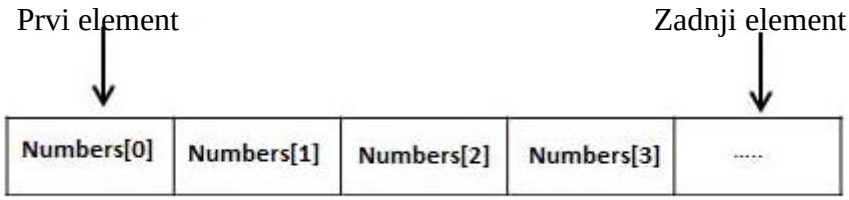
Nakon uvodnog poglavlja, poglavlje 2 opisuje zadatak završnog rada te daje općenit uvid u polja u programskom jeziku C uz tehnike upravljanja računalnom memorijom pri radu s poljima. Sljedeće poglavlje opisuje jednodimenzionalna polja koja se fokusiraju na vektore i polja znakova (engl. *string*), dok naredno poglavlje – višedimenzionalna polja – opisuje trodimenzionalna i polja s više od tri dimenzije. Zaključak daje osvrt na ciljeve postavljene u zadatku završnog rada i postignute rezultate.

1.1. Zadatak završnog rada

Zadatak završnog rada je objasniti polja u programskom jeziku C, njihove vrste, načine na koje se zapisuju u memoriju računala, kreiranje polja te osnovne funkcija za rad s poljima podataka.

2. POLJA U PROGRAMSKOM JEZIKU C

Programski jezik C daje strukturu podataka, polja, koja pohranjuju sekvencijalnu zbirku elemenata istog tipa fiksne veličine. Polja se koriste za pohranjivanje skupine podataka, ali je često korisnije razmišljati o polju kao zbirci varijabli istog tipa. Polja mogu sadržavati cijele brojeve, decimalne brojeve ili znakove, ovisno o tipu polja. Umjesto deklaracija pojedinih varijabli, kao što broj0, broj1, ..., i broj99, deklarira se jedno polje varijabli kao što su brojevi[100] i koriste se brojevi[0], brojevi[1]... brojevi[99] kao pojedine varijable. Specifičnom elementu u polju pristupa se preko njegovog indeksa poput brojevi[5] koji pristupa šestom elementu unutar polja budući da je indeks prvog elementa polja brojevi[0], a zadnjeg elementa polja brojevi[n-1] kao što se vidi na slici 2.1. Indeks ujedno označava i broj dimenzija polja, ako polje ima više dimenzija isto tako ima i više indeksa preko kojih se pristupa određenom elementu polja.



Sl. 2.1. Redoslijed elemenata u polju, prema [7].

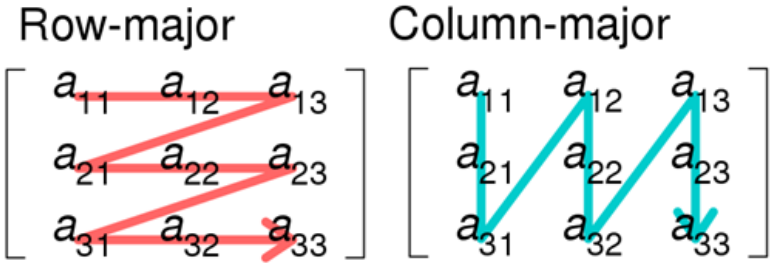
2.1. Deklaracija i inicijalizacija polja

Za deklaraciju polja u programskom jeziku C programer mora specificirati tip elementa, ime polja i broj elemenata koji će biti zapisani u polje. Računalna memorija se zauzima tijekom deklaracije polja, a veličina memorije ovisi o tipu elemenata polja i o broju elemenata polja. Oblik zapisa deklaracije polja je: *tip nazivPolja [veličinaNiza];*. primjer zapisa jednodimenzionalnog polja. *veličinaNiza* mora biti cjelobrojna konstanta veća od 0 i ona predstavlja broj elemenata koji se mogu zapisati u polje. *tip* mora biti tip podatka kojeg programski jezik C podržava poput *int*. *int* predstavlja cjelobrojni tip podatka i omogućuje spremanje jedino cijelih brojeva u polje. *nazivPolja* predstavlja ime polja i može sadržavati slova, brojeve i podvlaku (_). Prvi znak imena varijable mora biti slovo ili podvlaka, no preporučuje se koristiti slovo kao prvi znak da ne dolazi do sukoba sa standardnim sustavnim varijablama. Primjer deklaracije jednodimenzionalnog polja tipa *int*, imena *polje* i broj elemenata polja 10: *int polje [10];*. Kod inicijalizacije polja može se unijeti vrijednost svakog elementa polja odjednom: *int polje [3] = {5, 10, 15};*. Broj vrijednosti unutar vitičastih zagrada {} ne može biti veći od broja elemenata deklariranih u uglatim

zagradama []. U slučaju da je broj elemenata polja prazan (*veličinaNiza*), dobije se polje dovoljno veliko da sadrži sve inicijalizirane elemente: *int polje [] = {5, 10, 15};*. Za pridjeljivanje vrijednosti zadanom elementu unutar polja koristi se: *polje [1] = 10;*. Ovaj dio koda postavlja drugi element u polju na vrijednost 10. Ime polja je konstantan pokazivač na adresu prvog elementa u tome polju. Na primjer, u deklaraciji: *int a [10]; a* je pokazivač na *&a[0]*, što je ujedno i adresa prvog elementa polja *a*. Ime polja se može koristiti kao konstantan pokazivač. Na primjer, ** (a + 4)* je način pristupa podatku na lokaciji *a [4]*.

2.2. Memorija u programskom jeziku C

U računalnoj memoriji, elementi polja su spremljeni u sekvenci susjednih memorijskih blokova što omogućuje programu brži pristup vrijednostima unutar polja. Koriste se dva načina sekvencijalnog spremanja, *row-major* redoslijed i *column-major* redoslijed. Razlika između ovih redoslijeda je u redoslijedu spremanja elemenata polja u memoriju. Kod *row-major* redoslijeda uzastopni elementi reda se nalaze jedan pored drugoga, dok kod *column-major* redoslijeda uzastopni elementi stupca se nalaze jedan pored drugoga u memoriji računala kao što je prikazano na slici 2.2. Programski jezici C i C++ koriste *row-major* redoslijed za uzastopno spremanje elemenata reda u memoriju. Budući da su svi elementi unutar polja istog tipa podatka, memorijski blokovi su svi iste veličine. Svaki element polja okupira jedan memorijski blok. Veličina memorijskog bloka alocirana za spremanje elementa polja ovisi o tipu podatka polja budući da svaki tip podatka zauzima različitu količinu memorije što znači da ukupna veličina polja ovisi o tipu podatka polja i dimenzijama polja.



Sl. 2.2. *Row-major* redoslijed i *Colum-major* redoslijed metode spremanja višedimenzionalnih polja, prema [8].

2.2.1. Statička alokacija memorije

Programski jezik C ima tri različita tipa memorije, statičku, stog i hrpu (engl. *heap*). Statička memorija ustraje kroz cijeli program i uglavnom se koristi za spremanje globalnih

varijabli ili varijabli koje su navedene kao statičke. Na primjer: *int varijabla* može biti deklarirana izvan funkcije, što ju čini globalnom varijablom, što znači da joj se može pristupiti unutar cijelog programa. Globalne varijable su statične i postoji samo jedna istoimena varijabla unutar cijelog programa. Unutar funkcija varijable su dodijeljene na stog. Moguće je postaviti varijablu kao statičku ako ju navedemo kao takvu tako na primjer varijabla *static int varijabla;* koja se nalazi unutar funkcije se sprema u statičku memoriju iako se nalazi unutar funkcije čije se varijable spremaju na stog. Dakle, memorija stoga se koristi za spremanje varijabli koje se nalaze unutar funkcija, uključujući glavnu (engl. *main()*) funkciju. Struktura stog memorije je posljednji unutra, prvi izvan (engl. *Last-In, First-Out*). Svaki puta kada funkcija deklarira novu varijablu ona se sprema na stog i završetkom funkcije sve varijable deklarirane unutar te funkcije se brišu sa stoga i njihova memorija se oslobađa. Stog je posebno mjesto u memoriji kojom središnja procesorska jedinica *(*engl. *Central Processing Unit, CPU*) automatski upravlja te nema potrebe za ručnim alociranjem i oslobađanjem memorije. Svaki stog ima određenu količinu memorije čija veličina ovisi u operacijskom sustavu. Ako program pokuša staviti previše podataka na stog dolazi do prelijevanja stoga. Prelijevanje stoga se dogodi kada je sva dopuštena memorija stoga alocirana i novo alocirani podatci se izlijevaju u druge odjele memorije. Varijable zapisane u statičku memoriju se brišu po izlasku iz programa i memorija se oslobađa.

2.2.2. Dinamička alokacija memorije

Hrpa je dio memorije kojeg je potrebno unaprijed rezervirati u programskom kodu i u koji se sprema dinamički alocirana memorija. Memorija hrpe se alocira tijekom pokretanja programa te je pristup memoriji hrpe malo sporiji, no veličina memorije hrpe ograničena je samo veličinom virtualne memorije računala. Za razliku od ostalih memorija, memorijom hrpe računalo ne upravlja automatski već taj zadatak pada na korisnika, od alokacije memorije sve do oslobođenja. Za alociranje memorije na hrpu, koriste se funkcije *malloc()* i *calloc().* Alocirana memorija na hrpu ostaje alocirana sve dok se ne oslobodi preko funkcije *free().* U slučaju da se memorija ne oslobodi, program će imati gubitak memorije (engl. *memory leak*) što znači da ostali procesi neće moći koristiti dio memorije koji nije oslobođen. Većina današnjih operacijskih sustava ima sistem koji služi za oslobađanje memorije po završetku programa te time zaštićuju sustav od negativnih posljedica. Programski jezik C sam po sebi nema mogućnosti za dinamičku alokaciju memoriju, no unutar zaglavlja <*stdlib.h*> postoje četiri funkcije za dinamičku alokaciju memorije. Funkcija *malloc()* koja rezervira blok memorije određene veličine i vraća pokazivač podatkovnog tipa praznina (engl. *void*). Funkcija *calloc()* rezervira blok memorije specifične veličine i postavlja

alociranu memoriju na nulu. U slučaju da dinamički alocirana memorija preko funkcija *malloc()* ili *calloc()* nije dovoljna ili je ima previše ona se može promijeniti pomoću funkcije *realloc()*. Budući da se dinamički alocirana memorija preko funkcija *malloc()* i *calloc()* ne oslobađa sama od sebe koristimo funkciju *free()* da oslobodimo taj dio memorijskog prostora. Funkcija *free()* se koristi jednodimenzionalnih i višedimenzionalnih polja. Kod jednodimenzionalnih polja dinamički alocirana varijabla se oslobađa unošenjem imena varijable unutar zagrada *free* funkcije. Kod višedimenzionalnih polja dinamički alocirane varijable se oslobađaju korištenjem *free* funkcije unutar petlje tako da se svaki element polja pojedinačno oslobodi. Oslobađanje memorije dvodimenzionalnog i trodimenzionalnog polja prikazano je na slici 2.3.

Oslobađanje memorije dvodimenzionalnog i trodimenzionalnog polja

```
// Oslobađanje memorije dvodimenzionalnog polja

for(int i = 0; i < N; i++)
    free(ptr[i]);
free(ptr);

// Oslobađanje memorije trodimenzionalnog polja

    for(int i=0;i<l;i++)
    {
        for(int j=0;j<m;j++)
        {
            free(arr3D[i][j]);
        }
        free(arr3D[i]);
    }
    free(arr3D);
```

Sl. 2.3. Prikaz oslobađanja memorije dvodimenzionalnog i trodimenzionalnog polja

3. JEDNODIMENZIONALNA POLJA

Jednodimenzionalna polja su polja podataka istoga tipa koji imaju samo jednu dimenziju, red ili stupac. Svaki element polja se označava cjelobrojnim indeksom koji odgovara njegovoj udaljenosti od prvog člana, koji ima indeks 0, dok zadnji član ima indeks za jedan manji od duljine polja. Elementima unutar polja se pristupa preko jednog indeksa. Jednodimenzionalna polja mogu biti polja brojeva, vektori, ili polja znakova poznatija kao *string.*

3.1. Vektori

Jednodimenzionalna polja podataka u koja se spremaju brojevi istoga tipa se nazivaju vektori. Spremljeni brojevi unutar vektora mogu biti cijeli brojevi, ako je tip polja na primjer *int*, ili decimalni brojevi, u slučaju da je tip podataka polja *float*. Najosnovnije operacije nad poljem podataka brojeva su deklaracija zadanog polja, učitavanje elemenata polja preko korisničkog unosa i ispis polja na ekran, kao što je prikazano na slici 3.1.

Primjer 1: Deklaracija, učitavanje i ispis vektora

```c
#include <stdio.h>

int main() {
    int a[10], i;


// Ucitavanje vektora
    for(i=0; i<10; i++) {
        printf("Ucitaj %d. broj polja:", i+1);
        scanf("%d", &a[i]);
    }

// Ispis vektora
    printf ("Polje: ");
    for (i=0; i<10; i++){
        printf ("%d ", a[i]);
    }

    return 0;
}
```

```
Ucitaj 1. broj polja:1
Ucitaj 2. broj polja:2
Ucitaj 3. broj polja:3
Ucitaj 4. broj polja:4
Ucitaj 5. broj polja:5
Ucitaj 6. broj polja:6
Ucitaj 7. broj polja:7
Ucitaj 8. broj polja:8
Ucitaj 9. broj polja:9
Ucitaj 10. broj polja:0
Polje: 1 2 3 4 5 6 7 8 9 0
```

Sl. 3.1. Programski kod za deklaraciju, učitavanje i ispis vektora i njegov ispis u komandnom sučelju

U 4. liniji koda sa slike 3.1. deklarira polje tipa *int*, ime polja je *a* i broj elemenata unutar polja je 10. 8. do 11. linija koda omogućuje korisniku unos cijelih brojeva u polje *a* sve dok se izvodi *for* petlja, što je u ovom slučaju deset puta. 14. do 17. linija koda ispisuje sve elemente polja na ekran u slijedu u kojem su zapisani u polju.

Kod ovakve deklaracije polje se zapisuje u memoriju stoga i memorija za polje se zauzima prilikom kompajliranja programa. U slučaju da nije poznata potrebna veličina polja podataka memorija se može dinamički alocirati preko funkcije *malloc()*. Funkcija *malloc()* omogućuje korisniku dinamičko zauzimanje potrebnog prostora memorije tijekom izvođenja programa, no ta se memorija mora i osloboditi preko funkcije *free()* kao što je prikazano na slici 3.2.

Primjer 2: Dinamička alokacija polja

```c
#include <stdio.h>
#include <malloc.h>

int main()
{

    int *a;
    int n;
    printf ("Unesite velicinu vektora: ");
    scanf ("%d", &n);
    int b = n * sizeof(int);
    a = (int *) malloc (b);

    for (int i=0; i<n; i++) {
        printf ("Unesite velicinu %d. elementa: ", i+1);
        scanf ("%d", &a[i]);
    }
    printf ("Vektor: ");
    for (int i=0; i<n; i++) {
        printf ("%d ", a[i]);
    }

    free (a);
    return 0;
}
```

```
Unesite velicinu vektora: 4
Unesite velicinu 1. elementa: 4
Unesite velicinu 2. elementa: 3
Unesite velicinu 3. elementa: 2
Unesite velicinu 4. elementa: 1
Vektor: 4 3 2 1
--------------------------------
```

Sl. 3.2. Programski kod za dinamičku alokaciju polja i njegov ispis u komandnom sučelju

U 11. liniji koda sa slike 3.2. program sprema potrebnu memoriju za zapis korisnički unesenog polja u varijablu *b*, nakon kojeg kod 12. linije zatraži dio memorije računala koji je jednak veličini varijable *b* za spremanje polja *a*.

Jednodimenzionalna polja brojeva se koriste za jednostavniji način spremanja podataka, no isto tako ona nam omogućuju lakšu obradu podataka zapisanih u polje. Često korištene operacije nad jednodimenzionalnim poljem podataka su pronalaženje najvećeg ili najmanjeg broja unutar polja koje je prikazano na slici 3.3 ili ispis svih parnih ili neparnih elemenata polja kao što se vidi na slici 3.4.

Primjer 3: Traženje najvećeg i najmanjeg broja u vektoru

```
14  //najveci
15
16     int temp = 0;
17
18     for(int i=0;i<10;i++)
19     {
20         if(a[i]>temp)
21         temp=a[i];
22     }
23
24   printf ("\n\nNajveci broj je: %d", temp);
25
26  //najmanji
27
28     int temp2 = a[0];
29
30     for(int i=0;i<10;i++)
31     {
32         if(a[i]<temp2)
33         temp2=a[i];
34     }
35
36   printf ("\n\nNajmanji broj je: %d", temp2);
37
38
```

```
Najveci broj je: 9

Najmanji broj je: 0
```

Sl. 3.3. Programski kod za najveći i najmanji broj u vektoru i njegov ispis u komandnom sučelju

18. do 22. linija koda sa slike 3.3 prolazi kroz cijelo polje i uspoređuje da li je svaki element polja veći od prethodnog i ako je to slučaj zapisuje ga u varijablu *temp* u kojoj je nakon izvođenja *for* petlja zapisan najveći broj polja.

Primjer 4: Ispis neparnih i parnih brojeva u vektoru:

```
39  //neparni ispis
40
41    for(i=0; i<10; i++)
42        if(a[i]%2==1) {
43            printf ("\nNeparni broj: %d", a[i]);
44        }
45
46
47
48  //parni  ispis
49
50    for(i=0; i<10; i++)
51        if(a[i]%2==0) {
52            printf ("\nParni broj: %d", a[i]);
53
54        }
```

```
Neparni broj: 1
Neparni broj: 3
Neparni broj: 5
Neparni broj: 7
Neparni broj: 9
Parni broj: 2
Parni broj: 4
Parni broj: 6
Parni broj: 8
Parni broj: 0
```

Sl. 3.4. Programski kod za ispis neparnih u parnih brojeva u vektoru i njegov ispis u komandnom sučelju

U 42. liniji koda sa slike 3.4. program provjerava ostatak dijeljenja elementa polja s brojem dva. U slučaju da je ostatak jedan to znači da je element polja neparni broj te ga ispisuje. Budući da je unutar *for* petlje tu operaciju obavlja nad svakim elementom polja.

Unutar jednodimenzionalnog polja poredak elemenata nije zaključan te se može mijenjati po potrebi korisnika. Primjer algoritma sortiranja je *bubble* sortiranje. *Bubble* sortiranje je jednostavan algoritam svrstavanja koji više puta prolazi kroz polje podataka i uspoređuje svaki par elemenata te ako su u krivom redoslijedu zamjenjuje im poredak. *Bubble* sortiranje može sortirati elemente polje od najvećeg prema najmanjem ili od najmanjeg prema najvećem, kao što je prikazano na slici 3.5.

Primjer 5: Algoritam *bubble* sortiranja:

```
60      //bubblesort
61
62  int c, d, swap;
63
64  for (c = 0 ; c < ( 10 ); c++)
65  {
66    for (d = 0 ; d < 10 - c - 1; d++)
67    {
68      if (a[d] > a[d+1]) /* Za poredak od najveceg koristiti < */
69      {
70        swap   = a[d];
71        a[d]   = a[d+1];
72        a[d+1]= swap;
73      }
74    }
75  }
76
77  printf("\n\nBubblesort od najmanjeg:\n");
78
79  for ( c = 0 ; c < 10 ; c++ )
80    printf("%d\n", a[c]);
```

```
Bubblesort od najmanjeg:
0
1
2
3
4
5
6
7
8
9
```

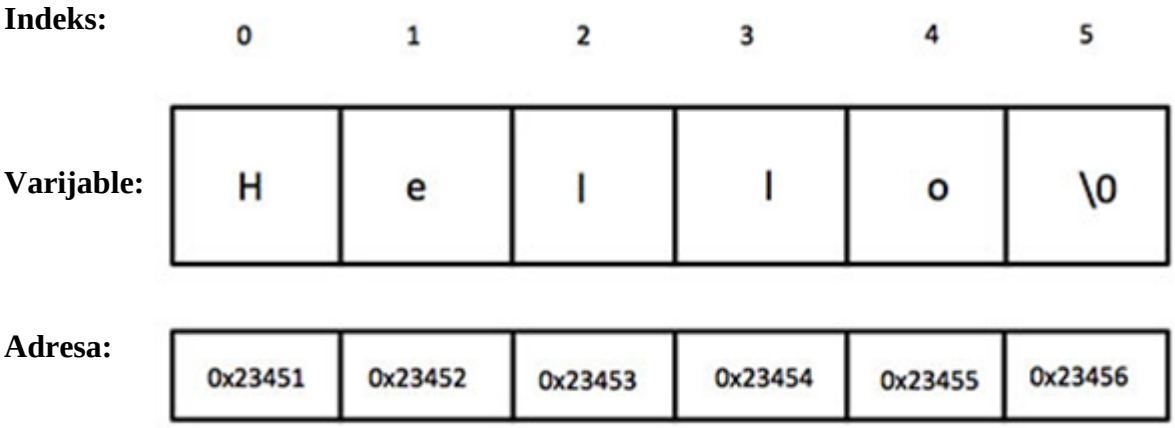
Sl. 3.5. Programski kod *bubble* sortiranja i njegov ispis u komandnom sučelju

U 68. liniji koda sa slike 3.5. program uspoređuje da li je prethodni element polja veći od sljedećeg. U slučaju da je, budući da sortira od najmanjeg elementa polja prema najvećemu, preko 69. do 75. linije koda program im zamjenjuje poredak. Algoritam prolazi kroz polje podataka sve dok ne poreda sve elemente polja.

3.2. Polje znakova (engl. string)

Polje znakova ili *string* je jednodimenzionalno polje znakova koje je završava „*null*“ znakom „\0“ (*null* predstavlja prazan unos). Programer ne stavlja *null* znak na kraju konstante polja znakova. C kompajler automatski stavlja „\0“ na kraj varijable polja znakova

tijekom inicijalizacije polja. Budući da na kraju polja kompajler stavlja *null* znak, tijekom deklaracije polja moramo postaviti veličinu polja za jedan više od broja znakova koje želimo spremiti u polje. Na primjer deklaracija polja znakova *hello* izgleda: *char hello[6] = {„H“, „e“, „l“, „l“, „o“, „\0“};* ili *char hello [ ] = „Hello“;*.



Sl. 3.6. Redoslijed elemenata u polju znakova

Osnovne operacije nad jednodimenzionalnim poljem znakova su deklaracija zadanog polja, inicijalizacija elemenata polja i ispis polja na ekran. Kod za te operacije je prikazan na slici 3.7.

Primjer 6: Deklaracija, inicijalizacija i ispis polja znakova:

```
1   #include <stdio.h>
2   #include <malloc.h>
3
4   int main()
5   {
6       char str[10] = "Hello";
7
8       printf ("%s", str);
9
10      return 0;
11  }
```

```
Hello
--------------------
```

Sl. 3.7. Programski kod za deklarirati, inicijalizirati i ispisati polje znakova i njegov ispis u komandnom sučelju

U 6. liniji koda sa slike 3.7. program deklarira jednodimenzionalno polje znakova imena *str* i duljine deset znakova, s tim da je zadnji znak *null*. Inicijalizacija je obavljena tijekom deklaracije u ovom slučaju i u polje su zapisani *Hello*. Kod 8. linije koda program ispisuje polje *str* na ekran.

U nekim slučajima potrebna duljina jednodimenzionalnog polja znakova nije poznata sve do unosa korisnika. U tom slučaju može se dinamički alocirati memorija potrebna za polje znakova. Funkcije koje obavljaju dinamičku alokaciju polja znakova su *malloc()* i *realloc()* i njihovo korištenje je prikazano na slici 3.8.

Primjer 7: Dinamička alokacija memorije i ispis duljine polja znakova:

```
1   #include <stdio.h>
2   #include <malloc.h>
3   #include <string.h>
4
5   int main()
6   {
7     char *str, c;
8     int i = 0, j = 1;
9
10    str = (char*)malloc(sizeof(char));
11
12    printf("Unesite string : ");
13
14    while (c != '\n') {
15      c = getc(stdin);
16
17      str = (char*)realloc(str, j * sizeof(char));
18
19      str[i] = c;
20
21      i++;
22      j++;
23    }
24    str[i] = '\0';
25
26    printf("\nUneseni string je: %s", str);
27
28  int len;
29  len = strlen(str);
30
31  printf("\nDuzina stringa je: %d", len);
32
33    free(str);
34  }
```

```
Unesite string : Dinamicki alocirani string i duljina stringa.

Uneseni string je: Dinamicki alocirani string i duljina stringa.

Duzina stringa je: 46
--------------------------------
Process exited after 18.72 seconds with return value 0
Press any key to continue . . .
```

Sl. 3.8. Programski kod za dinamičku alokaciju polja podataka i njegovu duljinu i njegov ispis u komandnom sučelju

U 17. liniji koda sa slike 3.8. program dinamički zauzima memoriju za varijablu str veličine *char*. 14. linija koda je petlja koja se ponavlja sve dok ne dobije unos s tipkovnice koji predstavlja novi red što ujedno znači i kraj *string*-a. 15. linija koda uzima po jedan znak iz unosa i upisuje ga 19. linije koda u polje znakova na lokaciju *i*. 17. linija koda uzima polje znakova *str* i poveća njegovu

memoriju svakim prolaskom kroz petlju. 29. linija koda uzima duljinu polja znakova *str* i zapisuje ga u varijablu *len* preko funkcije *strlen.* Uz funkciju *strlen* koja mjeri duljinu polja znakova, često se koriste još dvije funkcije za obradu polja znakova: funkcija *strcpy* i funkcija *strcat*. Funkcija *strcpy* obavlja operaciju kopiranja polja znakova dok funkcija *strcat* spaja dva polja znakova u jedno te je rad tih funkcija prikazan na slici 3.9.

Primjer 8: Funkcije polja znakova *strcpy, strcat, strlen*:

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

int main () {

   char str1 [12] = "Hello";
   char str2 [12] = "World";
   char str3 [12];
   int  length ;

   strcpy(str3, str1);
   printf("String copy ( str3, str1) :  %s\n", str3 );

   strcat( str1, str2);
   printf("String concatenates ( str1, str2):   %s\n", str1 );

   length = strlen(str1);
   printf("String lenth (str1) :  %d\n", length );

   return 0;
}
```

```
String copy ( str3, str1) :  Hello
String concatenates ( str1, str2):   HelloWorld
String lenth (str1) :  10

--------------------------------
```

Sl. 3.9. Programski kod koji koristi funkcije *strcpy, strcat, strlen* i njegov ispis u komandnom sučelju

Funkcija *strcpy(s1, s2)* kopira polje znakova *s1* u polje znakova *s2*, funkcija *strcat(s1, s2)* dodaje polje znakova *s2* na kraj polja znakova *s1*, funkcija *strlen(s1)* vraća duljinu polja znakova *s1*. U 11. liniji koda sa slike 3.9. program kopira polje znakova *str1* u polje znakova *str3*. 14. linija koda ulanča polje znakova *str2* na kraj polje znakova *str1*. 17. linija koda postavlja vrijednost varijable *length* na duljinu polja znakova *str1*.

3.3. Polje pokazivača

U situaciji gdje se želi zadržati niz koji može pohraniti pokazivače na *int* ili bilo koji drugi tip podatka koriste se polja pokazivača. Opći način zapisa deklaracije polja pokazivača izgleda:

*tip *naziv[veličinaNiza];*. Unutar polja pokazivača svaki element je pokazivač koji pokazuje na vrijednost deklariranog tipa.

Primjer 9: Polje pokazivača:

```c
#include <stdio.h>

const int MAX = 3;

int main () {

   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++) {
      ptr[i] = &var[i];
   }

   for ( i = 0; i < MAX; i++) {
      printf("Vrijednost polja [%d] = %d\n", i, *ptr[i] );
   }

   return 0;
}
```

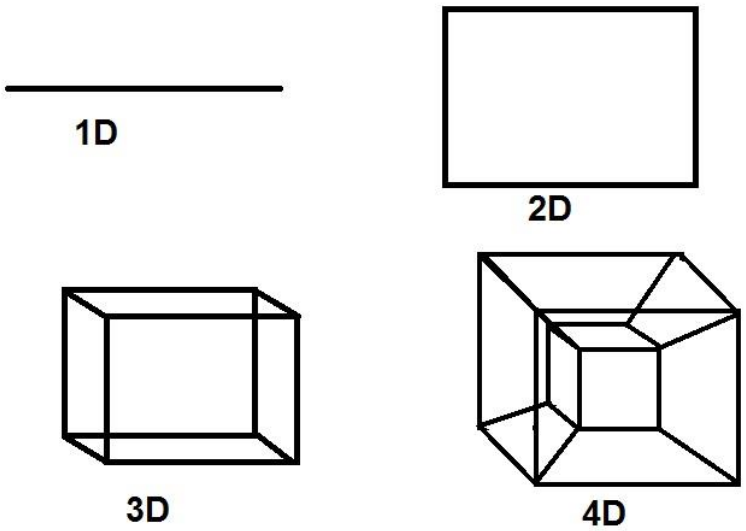
```
Vrijednost polja [0] = 10
Vrijednost polja [1] = 100
Vrijednost polja [2] = 200
```

Sl. 3.10. Programski kod za korištenje polje pokazivača i njegov ispis u komandnom sučelju

U 8. liniji koda deklariramo polje pokazivača *int *ptr[MAX]* gdje je *MAX* jednak tri. Unutar polja pokazivača **ptr* nalaze se tri pokazivača koji pokazuju na vrijednost tipa *int*. 10. do 11. linija koda pridjeljuje svakom pokazivaču polja **ptr* adresu elementa polja *var* te 14. do 15. linija koda ispisuju vrijednosti polja *var* preko polja pokazivača **ptr*.

4. VIŠEDIMENZIONALNA POLJA

Programski jezik C nam omogućuje izradu polja sa dvije ili više dimenzije. Više dimenzija u polju omogućuje zapis više podataka u jednu varijablu. Višedimenzionalna polja se deklariraju preko sljedeće sintakse: *tip ime_polja[d1][d2][d3]…[dn];*. *tip* predstavlja tip podataka zapisanih u polju. *ime_polja* je ime dodijeljeno polju koje deklariramo. *[d1][d2][d3]* su dimenzije našega polja sve do dimenzije *[dn]* koja predstavlja ukupan broj dimenzija i broj indeksa preko kojih se pristupa elementu polja.



Sl. 4.1. Prikaz dimenzija

4.1. Dvodimenzionalna polja

Dvodimenzionalna polja su najjednostavnija višedimenzionalna polja podataka koja su sastavljena od dvije dimenzije, reda i stupca, i njenim elementima se pristupa preko dva indeksa. Dvodimenzionalna polja brojeva se nazivaju matrice. Prvi indeks određuje redak matrice, a drugi indeks određuje stupac matrice. Prvi član dvodimenzionalnog polja (prvi redak, prvi stupac) označen je indeksom *[0][0]*, dok je posljednji član (posljednji redak, posljednji stupac) označen indeksom *[n-1][m-1]* gdje je *n* broj redaka i *m* broj stupaca matrice. Osnovne operaciju nad matricama su deklaracija matrice, unos elemenata u matricu i ispis matrice na ekran te kod za njih je prikazan na slici 4.2.

Primjer 9: Deklaracija, unos i ispis matrice:

```c
#include <stdio.h>

int main() {

    int iMatrica[3][3];
    int i;
    int j;

    printf("Unesite elemente matrice: \n");
    for(i=0;i < 3; i++) {
        for(j=0; j < 3; j++) {
            scanf("%d", &iMatrica[i][j]);
        }
    }
    printf("\nIspis matrice:\n");
    for(i=0; i < 3; i++) {
        for(j=0; j < 3; j++) {
            printf("%d ", iMatrica[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
Unesite elemente matrice:
1
2
3
4
5
6
7
8
9

Ispis matrice:
1 2 3
4 5 6
7 8 9
```

Sl. 4.2. Programski kod za deklaraciju, unos i ispis matrice i njegov ispis u komandnom sučelju

U 5. liniji koda sa slike 4.2. program deklarira matricu tipa podatka *int*, imena *iMatrica* i veličine 3*3 što znači da matrica ima tri retka i tri stupca što je ukupno devet elemenata. 12. linija koda zatraži od korisnika unos elemenata matrice, budući da je unutar *for* petlje unos se ponavlja za svaki element matrice. 18. linija koda ispisuje svaki element matrice na ekran budući da je unutar *for* petlje te linija koda *printf („\n“);* koji se nalazi između dvije *for* petlje omogućuje ispis matrice u matričnom obliku.

Isto kao i kod jednodimenzionalnih polja podataka, dvodimenzionalna polja podataka imaju statičku memoriju, stog memoriju i memoriju hrpe. U slučaju da nam je potrebna dinamički alocirana memorija kod višedimenzionalnih polja podataka ona se može zatražiti od računala preko funkcije *malloc()*. Sintaksa i primjer te funkcije je prikazan na slici 4.3.

Primjer 10: Dinamička alokacija memorije za matricu:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main() {
6
7      int a, b;
8
9      scanf ("%d", &a);
10     scanf ("%d", &b);
11     int *matrica = (int *) malloc (a*b*sizeof(int));
12
13      int i, j, count = 0;
14     for (i = 0; i <  a; i++)
15       for (j = 0; j < b; j++)
16          *(matrica + i*b + j) = ++count;
17
18     for (i = 0; i <  a; i++){
19
20       for (j = 0; j < b; j++){
21
22          printf("%d ", *(matrica + i*b + j));}
23          printf ("\n");}
24
25     free (matrica);
26     return 0;
27 }
```

```
5
7
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31 32 33 34 35

--------------------------------
```

Sl. 4.3. Programski kod za dinamičku alokaciju memorije kod matrice i njegov ispis u komandnom sučelju

U 11. liniji koda sa slike 4.3. program zatraži od računala memoriju veličine *a*b*sizeof(int)* te deklarira matricu pod nazivom *matrica* te veličine. Varijabla *a* predstavlja broj redaka matrice, varijabla *b* predstavlja broj stupaca matrice i *sizeof(int)* je veličina tipa podatka *int* koji je konstanta.

Često korištene operacije nad matricama uključuju pronalaženje najvećeg ili najmanjeg elementa matrice te ispis parnih i neparnih brojeva unutar matrice. Primjeri takvih operacija su prikazani na slikama 4.4. i 4.5.

Primjer 11: Najveći broj u matrici:

```
16     int maximum = Matrica[0][0];
17
18     for(int c = 0 ; c < 3 ; c++ )
19     {
20        for(int d = 0 ; d < 3 ; d++ )
21        {
22           if ( Matrica[c][d] > maximum )
23              maximum = Matrica[c][d];
24        }
25     }
26
27      printf ("\nNajveci broj je: %d", maximum);
```

```
Unesite elemente matrice:
1
2
3
4
5
6
7
8
9

Najveci broj je: 9
```

Sl. 4.4. Programski kod za najveći broj u matrici i njegov ispis u komandnom sučelju

23. i 24. linija koda prolazi kroz svaki element matrice te provjerava da li je veći od varijable *maximum* i ako je postavlja varijablu na vrijednost tog elementa. Jednim prolaskom kroz matricu

varijabla *maximum* ima vrijednost najvećeg elementa u matrici. U slučaju da nam je potreban najmanji broj unutar matrice znak veći od (>) se zamijeni sa znakom manji od (<).

Primjer 12: Parni i neparni brojevi matrice:

```
16    for(int c = 0 ; c < 3 ; c++ )
17    {
18       for(int d = 0 ; d < 3 ; d++ )
19       {
20          if ((Matrica[c][d] % 2) == 0)
21          {printf ("Parni broj: %d \n", Matrica[c][d]);}
22
23       }
24    }
25
26       for(int c = 0 ; c < 3 ; c++ )
27    {
28       for(int d = 0 ; d < 3 ; d++ )
29       {
30
31       if ((Matrica[c][d] % 2) == 1)
32          {printf ("Neparni broj: %d \n", Matrica[c][d]);}
33       }
34    }
```

```
Unesite elemente matrice:
1
2
3
4
5
6
7
8
9
Parni broj: 2
Parni broj: 4
Parni broj: 6
Parni broj: 8
Neparni broj: 1
Neparni broj: 3
Neparni broj: 5
Neparni broj: 7
Neparni broj: 9
```

Sl. 4.5. Programski kod za parne i neparne brojeve matrice i njegov ispis u komandnom sučelju

U 20. linija koda sa slike 4.5. program provjerava da li je ostatak dijeljenja elementa matrice sa brojem dva jednak nuli, što ga čini parnim brojem te se kasnije ispisuje, dok 31. linija provjerava da li je ostatak dijeljenja elementa matrice sa brojem dva jednak jedan, što ga čini neparnim brojem te ga kasnije ispisuje.

Elemente matrice, kao elemente dvodimenzionalnog polja, je moguće poredati po veličini. Primjer takve funkcije je funkcija *qsort* koja sortira matricu od najmanjeg elementa prema najvećem. Primjer rada funkcije *qsort* je prikazan na slici 4.6.

Primjer 13: Sortiranje od najmanjeg elementa matrice:

```
41   qsort(Matrica, 9, sizeof(int), compar);
42
43
44   int compar(const void *a, const void *b) {
45      int ia = *((int*)a);
46      int ib = *((int*)b);
47      return ia - ib;
48   }
```

```
Unesite elemente matrice:
9
5
1
8
6
2
3
7
4

Ispis matrice:
9 5 1
8 6 2
3 7 4

Ispis matrice:
1 2 3
4 5 6
7 8 9
```

Sl. 4.6. Programski kod za sortiranje elemenata matrice od najmanjeg i njegov ispis u komandnom sučelju

Funkcija *qsort()* sortira elemente matrice od najmanjeg do najvećeg. Ova se funkcija nalazi unutar <*stdlib.h*> biblioteke.

Matrica može biti kvadratna, u slučaju da su broj redaka matrice i broj stupaca matrice jednaki. Kod kvadratne matrice mogu se koristiti dijagonale, ravne linije koje povezuju suprotne kutove kvadra. Kod kvadratne matrice postoje dvije vrste dijagonala, glavna i sporedna. Glavna dijagonala prolazi kroz elemente kojima su indeksi reda i stupca jednaki, dok sporedna dijagonala prolazi elementima kojima je zbroj indeksa za jedan manji od broja redaka ili broja stupaca matrice. Kod za korištenje glavne i sporedne dijagonale i njihov zbroj se nalazi na slici 4.7.

Primjer 14: Zbroj glavne i sporedne dijagonale kvadrate matrice:

```
for (i=0;i<3;i++){
for (j=0;j<3;j++){

        if(i==j)   d+=Matrica[i][j];     //glavna dijagonala
        if(i+j==2) s+=Matrica[i][j];     //sporedna dijagonala

    }
}

printf ("\nZbroj glavne i sporedne dijagonale je: %d", d+s);
```

```
Unesite elemente matrice:
1
2
3
4
5
6
7
8
9

Ispis matrice:
1 2 3
4 5 6
7 8 9

Zbroj glavne i sporedne dijagonale je: 30
```

Sl. 4.7. Programski kod za zbroj glavne i sporedne dijagonale kvadratne matrice i njegov ispis u komandnom sučelju

Linija koda *if (i==j) d+=Matrica[i][j]* sa slike 4.7. gleda da li su varijable *i* i *j* jednake što znači se taj element matrice nalazi na glavnoj dijagonali te ako jesu dodaje ih u varijablu *d*. Element se nalazi na sporednoj dijagonali u slučaju *if (i+j==i-1) s+=Matrica[i][j];* te ako je izjava točna dodaje vrijednost elementa u varijablu *s*. Na kraju programa varijable *d* i *s* se zbrajaju i njihov rezultat se ispisuje na ekran.

Matrica se može ispuniti preko unosa s tipkovnice ili se može ispuniti nasumičnim brojevima. Funkcija za generiranje nasumičnih brojeva je *rand()*. Funkcija *rand()* se koristi uz zaglavlje *<time.h>* koje sadrži funkcije vremena i datuma te daje pristup manipulaciji istih. Popunjavanje matrice nasumičnim brojevima prikazano je na slici 4.8.

Primjer 15: Matrica nasumičnih brojeva:

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

int main () {
    int a, b;
    scanf ("%d%d", &a, &b);

        int matrica[a][b];

    for (int i=0; i<a; i++){

        for (int j=0; j<b; j++){
        int r = rand() % (100);
        matrica[i][j] = r;
    }}

        printf ("Random matrica: \n");
    for (int i=0; i<a; i++){

        for (int j=0; j<b; j++){

        printf ("  %d  ", matrica[i][j]);
    }printf ("\n");}

    return 0;
}
```

```
4
6
Random matrica:
  41    67    34    0    69    24
  78    58    62    64    5    45
  81    27    61    91    95    42
  27    36    91    4    2    53

--------------------------------
```

Sl. 4.8. Programski kod za izradu matrice nasumičnih brojeva i njegov ispis u komandnom sučelju

Linija koda *#include <time.h>* uključuje zaglavlje *<time.h>* potrebno za funkciju *rand()*. 14. linija koda kreira varijablu *r* koja ima vrijednost između 0 i 99. Preko 15. linije koda element matrice dobiva vrijednost nasumične varijable *r* te budući da *for* petlja prolazi cijelom matricom svaki element dobiva novu nasumičnu vrijednost varijable *r*.

4.2. Višedimenzionalna polja

Višedimenzionalna polja poput trodimenzionalnih polja ili polja s više dimenzija se koriste kada zapisujemo objekte sa više karakteristika. Na primjer može se koristiti trodimenzionalno polje za spremanje visine, duljine i širine svake sobe na svakom katu ili polje sa četiri dimenzije za zapis vremenske prognoze te spremamo dan, temperaturu, vlagu i pritisak zraka. Broj *for* funkcija za unos i ispis polja ovisi o broju indeksa polja, za svaki indeks ili dimenziju polja potrebna je jedna *for* funkcija, tako da se kod unosa i ispisa trodimenzionalnog polja koriste tri *for* funkcije za unos i ispis elemenata iz polja kao što se vidi na slici 4.9.

Primjer 16: Unos i ispis 3D polja:

```c
#include <stdio.h>
int main()
{
    int i, j, k, polje[2][3][2];

    printf("Unesite vrijednosti elemenata: \n");

    for(i = 0; i < 2; ++i) {
        for (j = 0; j < 3; ++j) {
            for(k = 0; k < 2; ++k ) {
                scanf("%d", &polje[i][j][k]);
            }
        }
    }

    printf("\nIspis vrijednosti:\n\n");

    for(i = 0; i < 2; ++i) {
        for (j = 0; j < 3; ++j) {
            for(k = 0; k < 2; ++k ) {
                printf("polje[%d][%d][%d] = %d\n", i, j, k, polje[i][j][k]);
            }
        }
    }

    return 0;}
```

```
Ispis vrijednosti:

polje[0][0][0] = 1
polje[0][0][1] = 2
polje[0][1][0] = 3
polje[0][1][1] = 4
polje[0][2][0] = 5
polje[0][2][1] = 6
polje[1][0][0] = 7
polje[1][0][1] = 8
polje[1][1][0] = 9
polje[1][1][1] = 0
polje[1][2][0] = 1
polje[1][2][1] = 2

--------------------------------
```

Sl. 4.9. Programski kod za unos i ispis 3D polja i njegov ispis u komandnom sučelju

U 4. liniji koda sa slike 4.9. program deklarira trodimenzionalno polje s veličinom dimenzija 2, 3 i 2. Polje ukupno sadrži 12 elemenata. 11. linija koda zatraži unos elemenata od korisnika za svaki element polja te 21. linija ispisuje svaki element polja na ekran.

5. ZAKLJUČAK

Polja u programskom jeziku C dio su programskog jezika C koja služe kao alat za jednostavniju pohranu podataka istoga tipa u susjedne memorijske lokacije te za jednostavnije izvođenje operacija nad njima. Obično se koriste za zapis digitalne fotografije, zapise tablica, zapis crvena-zelena-plava (engl. *Red Greed Blue, RGB*) aditivnog modela boje. Postoje tri različita tipa memorije u programskom jeziku C za pohranu polja, to su statička memorija, memorija stoga i memorija hrpe. Način na koji programski jezik C sprema podatke u navedena tri tipa memorije ovisi o tome kako i gdje je polje deklarirano i omogućuje korisniku učinkovito korištenje memorijskih blokova za spremanje polja. U statičku memoriju se zapisuju globalne varijable, na stog se spremaju sve varijable deklarirane unutar funkcija, uključujući glavnu funkciju, dok je potrebnu memoriju hrpe, u koju se sprema dinamički alocirana memorija, potrebno unaprijed rezervirati te tom memorijom računalo ne upravlja automatski već taj zadatak pada na korisnika, od alokacije memorije sve do oslobođenja. Jednodimenzionalna polja brojeva, vektori, koriste se za spremanje brojeva u niz nad kojim se vrše potrebne matematičke funkcije. U slučajevima gdje model podataka zahtjeva organizaciju istih u dvodimenzionalno polje koriste se dvodimenzionalni zapisi, često zvani matrice, koji pohranjuju podatke u retke i stupce. Manipulacija poljima podataka često je zahtjevna i u tu svrhu su u ovom radu prikazane osnovne funkcije za rad nad njima. U ovom radu prikazane su funkcije za deklaraciju, učitavanje i ispis jednodimenzionalnih i višedimenzionalnih polja te za pronalaženje parnih i neparnih brojeva, najvećih i najmanjih brojeva, unošenje nasumičnih brojeva, zbroj dijagonala dvodimenzionalnog polja i funkcija za sortiranje čitavih polja. Prethodno navedene funkcije su prikazane iz razloga što se one koriste za rješavanje najčešćih problema kod polja podataka. Za polje znakova dodatno su navedene funkcije *strcpy*, *strcat* i *strlen* koje obavljaju operacije kopiranja, dodavanja i mjerenja duljine polja znakova. Navedeni programski kod omogućuje unos dvodimenzionalnih polja te obavlja funkcije za ispis, najveći broj u retku, najveći broj u stupcu te množenje odabranih dvodimenzionalnih polja u slučaju da im dimenzija odgovaraju pravilu.

6. LITERATURA

[1] Skupina autora, C Programming Tutorial: C – Arrays [online], TutorialsPoint, Hyderabad, India, 2018., dostupno na https://www.tutorialspoint.com/cprogramming/c_arrays.htm [4.4.2018.]

[2] Michael Wirth, Memory in C – The stack, the heap, and static [online], The Craft of Coding, Guelph, Ontario, Canada, 2013., dostupno na https://craftofcoding.wordpress.com/2015/12/07/memory-in-c-the-stack-the-heap-and-static/ [4.4.2018.]

[3] Prof. dr. sc. Ivo Ipšić, Programiranje [online], Sveučilište u Rijeci, TEHNIČKI FAKULTET, Rijeka, Hrvatska, 2018., dostupno na http://www.riteh.uniri.hr/zav_katd_sluz/zr/nastava/racprog/download/audit/polja.pdf [4.4.2018.]

[4] Skupina autora, Bubble sort [online], Wikipedia, San Francisco, California, 2003., dostupno na https://en.wikipedia.org/wiki/Bubble_sort [4.4.2018.]

[5] Skupina autora, C Programming Tutorial: C – Arrays [online], TutorialsPoint, Hyderabad, India, 2018., dostupno na https://www.tutorialspoint.com/cprogramming/c_strings.htm [4.4.2018.]

[6] Skupina autora, C Programming Tutorial: C – Arrays [online], TutorialsPoint, Hyderabad, India, 2018., dostupno na https://www.tutorialspoint.com/cprogramming/c_strings.htm [4.4.2018.]

[7] Skupina autora, C Programming Tutorial: C – Arrays [online], TutorialsPoint, Hyderabad, India, 2018., dostupno na https://www.tutorialspoint.com/cprogramming/images/arrays.jpg [23.4.2018.]

[8] Skupina autora, Row- and column-major order [online], Wikipedia, San Francisco, California, 2003., dostupno na https://en.wikipedia.org/wiki/File:Row_and_column_major_order.svg [23.4.2018.]

SAŽETAK

Polja u programskom jeziku C se koriste kao jednostavniji zapis grupe podataka istog tipa i nad njima je lakše izvoditi matematičke operacije i funkcije. Iz tog razloga česti programerski izazov je učinkovit rad s poljima u programskom jeziku C. U ovom radu implementirane su funkcije za neke od najčešćih operacija s poljima, a to su inicijalizacija, učitavanje, ispis, pronalaženje parnih i neparnih brojeva te najvećih i najmanjih brojeva među njima, umnožak matrica itd. Programski jezik C ima tri različita tipa memorije, koji se implementiraju u statičku i dinamičku memoriju te memoriju hrpe. U statičku memoriju se zapisuju globalne varijable, na stog se spremaju varijable koje se nalaze unutar funkcija, dok je hrpa dio memorije kojeg je potrebno unaprijed rezervirati u programskom kodu i kojom računalo ne upravlja automatski već taj zadatak pada na programera. Polja u programskom jeziku C mogu biti jednodimenzionalna, što su vektori i polja podataka ili višedimenzionalna, poput matrica.

Ključne riječi: polje, memorija, vektor, polje podataka, matrica.

ARRAYS IN PROGRAMMING LANGUAGE C

ABSTRACT

Arrays in programming language C are used as a simplified mechanism to store a group of data of the same type making easier to perform mathematical operations and functions over them. For this reason, a frequent programming challenge is the effective work with arrays in the programming language C. In this paper, functions for some of the most common array operations are implemented, such as initialization, loading, printing, finding odd and even numbers, the largest and smallest numbers among them, multiplication of matrices, etc. The programming language C has three different types of memory, which are implemented as the static, dynamic memory and the heap memory. Global variables are stored in the static memory, the variables contained in functions are stored on stack, while the heap memory is a part of the memory that needs to be reserved in advance in the program code and it is not managed by the computer automatically, instead this task falls on the programmer. Arrays in the programming language C can be one-dimensional, which are vectors and strings or are multi-dimensional, such as matrices.

Keywords: array, memory, vector, string, matrix.

ŽIVOTOPIS

Toni Cestar rođen je u Koprivnici 3. 8. 1993. godine. Trenutno živi u Đurđevcu. Osnovno obrazovanje je stekao u Osnovnoj školi Grgura Karlovčana u Đurđevcu, nakon kojega je pohađao Gimnaziju Dr. Ivana Kranjčeva u Đurđevcu. Nakon gimnazije upisuje Fakultet elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Trenutno studira na Preddiplomskom stručnom studiju Elektrotehnike, smjer informatika.

Potpis: