Računske vježbe 7

Programiranje I

1. Napisati program koji od elemenata dinamičkog niza formira druga dva dinamička niza, niz negativnih i niz pozitivnih brojeva (uključujući i nulu). Prikazati elemente tako dobijenih nizova.

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

int main()
{
    int *arr, *posArr, *negArr, i, j = 0, k = 0, length, posLength = 0;

    puts("Unesite duzinu niza:");
    scanf("%d", &length);

    arr = (int *)malloc(length * sizeof(int));
    if(arr == NULL) // NULL je konstanta obicno jednaka 0
    {
        puts("Nema dovoljno memorije na raspolaganju!");
        exit(1);
    }

    puts("Unesite elemente niza:");
    for(i = 0; i < length; i++)
    {
        scanf("%d", arr + i);
        if(arr[i] >= 0)
            posLength++;
    }

    posArr = (int *)malloc(posLength * sizeof(int));
    negArr = (int *)malloc((length - posLength)*sizeof(int));

    if((posArr == NULL) || (negArr == NULL))
    {
        puts("Nema dovoljno memorije na raspolaganju!");
        exit(1);
    }

    for(i = 0; i < length; i++)
        if(arr[i] >= 0)
            posArr[j++] = arr[i];
        else
            negArr[k++] = arr[i];

    free(arr);

    printf("Niz pozitivnih brojeva: ");
    for(i = 0; i < posLength; i++)
        printf("%d ", posArr[i]);
```

```
46
47      printf("\nNiz negativnih brojeva: ");
48      for(i = 0; i < length - posLength; i++)
49          printf("%d ", negArr[i]);
50
51      free(posArr);
52      free(negArr);
53  }
```

Za dinamičku alokaciju memorije uključujemo zaglavlje `stdlib.h`. Na predavanjima smo naučili da se memorija za niz dinamički zauzima na sledeći način:

```
malloc(duzina * sizeof(tip));
```

upotrebom funkcije `malloc()` koja kao argument uzima veličinu memorije u bajtovima, a vraća pokazivač na `void` koji pokazuje na početak bloka zauzete memorije. Kako nam ovaj tip ne odgovara, moramo ga eksplicitno konvertovati u odgovarajući tip upotrebom **cast** operatora:

```
tip *a;
a = (tip *)malloc(duzina * sizeof(tip));
```

Poziv funkcije `malloc` nam ne garantuje da će memorija stvarno biti zauzeta. Naime, ako ova funkcija ne uspije da zauzme memoriju ona vraća `NULL` pokazivač koji nam u kontekstu memorije govori da pokazivač ni na šta ne pokazuje. Ovo se neće dešavati često, ali se mora provjeriti kao nepovoljan ishod što smo i učinili u zadatku.

2. Svaki red fajla `brojevi.txt` sadrži po jedan cijeli broj. Napisati program koji štampa najmanji i najveći cijeli broj u tom fajlu.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <limits.h>
4
5  int main()
6  {
7      FILE *file;
8      int min = INT_MAX, max = INT_MIN, number; // protumacite
9      file = fopen("brojevi.txt", "r");
10     if(file == NULL)
11     {
12         printf("Greska pri otvaranju fajla!");
13         exit(1);
14     }
15     while(fscanf(file, "%d", &number) != EOF)
16     {
17         if(number < min) min = number;
18         if(number > max) max = number;
19     }
20     fclose(file);
21     printf("Najmanji je broj %d, a najveci %d.\n", min, max);
22 }
```

Programski jezik `C` poznaje tip podatka ,,pokazivač na fajl". Da bi se radilo sa ovim tipom u `C`-u mora biti uključena programska biblioteka `stdio.h`. Pokazivači na fajl se deklarišu kao:

```
FILE *file;
```

gdje je `FILE` novi tip podatka sa kojim se susrećemo (biće uskoro jasnije), a `file` je ime pokazivačke promjenljive. Ova promjenljiva se inicijalizuje pomoću funkcije `fopen`:

```
file = fopen("brojevi.txt", "r");
```

koja prima dva argumenta. Prvi argument je putanja do fajla, a drugi je režim pristupa fajlu. Putanja do fajla može biti apsolutna (specificira lokaciju u odnosu na root, recimo "C:\\fajl.txt") ili relativna (u odnosu na tekući direktorijum, kao u našem primjeru). Režim pristupa fajlu naglašava da li želimo da samo pročitamo sadržaj fajla ili da pak nešto u njega upišemo itd. U našem slučaju je "r", što je skraćeno od *read*, odnosno namjeravamo samo da čitamo iz fajla. Režimi pristupa su jako važni jer njima osiguravamo da se eventualna greška programera ne reflektuje na fajl sa kojim radimo. Funkcija vraća `FILE` pokazivač čija vrijednost može biti `NULL` ukoliko je došlo do greške prilikom otvaranja fajla te je, prije nego krenemo u rad sa fajlom, potrebno provjeriti da li je otvaranje bilo uspješno. Sve funkcije za upis i čitanje iz fajla vrše pomjeranje pozicije za čitanje i upis tako da je naredna pozicija tamo gdje je stalo prethodno čitanje. Drugim riječima, funkcija `fscanf` koju koristimo za čitanje iz fajla kao prvi argument prima pokazivač na fajl, koji se u ovom slučaju ponaša kao tok (engl. *stream*) i koji se u svakom pozivu dodatno pomjera. Dakle, funkcija radi slično kao i `scanf`, samo što unos ne vrši korisnik preko komandnog prozora, već se on automatski dobija iz fajla. Kada tok dođe do kraja fajla (naiđe na `EOF` karakter odnosno *end of file*) čitanje se obustavlja. Da bi promjene koje su se vršile nad fajlovima bile pravilno ažurirane, fajlovi se moraju, prije završetka programa, obavezno zatvoriti. To se vrši pozivom funkcije `fclose`:

```
fclose(file);
```

3. Svaki red fajla `rijeci.txt` sadrži po jednu riječ. Napisati program koji formira fajl `malaSlova.txt` prepisujući one riječi iz fajla `rijeci.txt` koje sadrže samo mala slova. Nakon datih obrada zatvoriti predmetne fajlove.

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

int main()
{
    int i, ind;
    char line[20];
    FILE *fileRead, *fileWrite;
    if((fileRead = fopen("rijeci.txt", "r")) == NULL)
    {
        puts("Greska pri otvaranju prvog fajla!");
        exit(1);
    }
    if((fileWrite = fopen("malaSlova.txt", "w")) == NULL)
    {
        puts("Greska pri otvaranju drugog fajla!");
        exit(1);
    }
    while(fgets(line, 20, fileRead) != NULL)
    {
        ind = 1;
        int len = strlen(line);
        // ako postoji novi red (enter) na kraju linije fajla ukloniti ga
        if(line[len - 1] == '\n')
        {
            line[len - 1] = '\0';
            len -= 1;
        }
        for(i = 0; i <= len - 1; i++)
        {
```

```
            if(line[i] < 'a' || line[i] > 'z' )
            {
                ind = 0;
                break;
            }
        }
        if(ind == 1)
            fprintf(fileWrite, "%s\n", line);
            // drugi nacin:
            // fputs(str, fb);
            // paziti jer fputs ne stavlja novi red na kraju
    }
    _fcloseall();
}
```

Sjetimo se da izraz:

```
a = b;
```

u programskom jeziku `C` ima vrijednost (koju?) što smo i iskoristili prilikom otvaranja fajla. Za učitavanje jedne linije fajla u promjenljivu koja predstavlja string koristili smo funkciju `fgets`:

```
fgets(line, 20, fileRead)
```

čiji je prvi argument pokazivač na niz karaktera koji predstavlja lokaciju na koju će se pročitani string upisati. Drugi argument je maksimalni broj karaktera koji se može pročitati prije zaustavljanja (uključujući i poslednji terminacioni), dok je treći pokazivač na fajl. Zaustavlja se kada se pročita `n-1` karaktera, pročita karakter za novi red ili se pročita `EOF`, šta god se prvo dogodi. Ova funkcija, ukoliko je čitanje izvršeno uspješno, vraća isti pokazivač koji smo joj proslijedili, odnosno `NULL` pokazivač (sadržaj proslijeđenog pokazivača ne mijenja) ukoliko je došlo do greške. Nekoliko je razloga zašto smo se odlučili za `fgets`, a ne za `fscanf`. Ključni razlog leži u tome što je `fgets` sigurniji jer kontroliše maksimalan broj karaktera koji se mogu učitati u jednoj iteraciji. Kao i u slučaju `scanf`, `fscanf` se zaustavlja nakon prve pojave bjeline što nam često neće odgovarati. Analogno `fscanf`, upis u fajl vršimo pomoću funkcije `fprintf`. Kako u ovom zadatku radimo sa dva pokazivača na fajl, istovremeno ih zatvaramo na sledeći način:

```
_fcloseall();
```

što rezultuje kao da smo zatvorili oba pojedinačno:

```
fclose(fileRead);
fclose(fileWrite);
```