37. DINAMIČKA DODELA MEMORIJE

Prilikom delaracije promenljivih u programima, u zoni podataka operativne memorije, se odvaja potreban prostor na osnovu tipa podataka. U toku prevođenja programa se mora tačno znati koliko podataka će se koristiti prilikom izvršavanja programa. U slučaju nizova rezervisaće se memorijski prostor za smeštanje sa očekivanim maksimalnim brojem elemenata. Ovakav rad dovodi do nepotrebnog utroška memorije. Programski jezik C dozvoljava da se u toku izvršenja programa od operativnog sistema traži dodeljivanje memorijskog prostora.

Deo memorije koji se dodeljuje u toku izvršavanja programa naziva se dinamička zona memorije ili engl. "heap" memorija.

Osobine dinamičkih podataka - podaci smešteni u dinamičkoj zoni memorije nazivaju se dinamički podaci, postoje sve dok ih programer eksplicitno ne uništi ili do završetka programa kada se automatski uništavaju. **Dinamički podaci nemaju svoje indentifikatore već im se pristupa pomoću pokazivača iz zone podataka.**

Upravljanje dinamičkim podacima vrši se pomoću funkcija:

- `malloc()`,
- `calloc()`,
- `realloc()`,
- `free()`,

koje se nalaze u biblioteci "stdlib.h".

Funkcija `malloc()` alocira `size` bajtova u dinamičkoj zoni memorije. Ako uspešno obavi zadatak funkcija vraća pokazivač na alocirani prostor u memoriji. Ako dođe do greške biće vraćena vrednost NULL vrednost (ništa). Opšti oblik:

```
void *malloc(size_t size);
```

Primer: `malloc(10)` - rezerviše 10 bajtova

`malloc(sizeof(int))` - rezerviše 2 (4) bajta jer je sizeof(int) 2 ili 4, zavisno od arhitekture procesora. Pokazivač koji je rezultat funkcije je tipa void, znači da nije definisano za koji tip podatka u njemu može biti memorisana adresa. Međutim, rezultat funkcije se može pridružiti nekom pokazivaču preko cast operatora:

```
int *p;
p=(int *)malloc(sizeof(int));
```

Funkcija `free()` oslobađa prethodno alociranu memoriju na čiju početnu adresu sadrži pokazivač `buffer`. Pokazivač `buffer` dobio je adresu pozivom funkcije `malloc()`, `calloc()` ili `realloc()`. Opšti oblik:

```
void free(void *buffer);
```

Funkcija `calloc()` alocira blokove memorije. Opšti oblik:

```
calloc(n, element-size)
```

gde jh `n` broj elemenata, a `element-size` memorijska veličina jednog elementa.

Funkcija `realllоc()` vrši povećanje/smanjenje alocirane memorije . Opšti oblik:

```
realloc (p, new-size)
```

gde jh `p` pokazivač, a `new-size` nova količina alocirane memorije.

Primer - deklaracija pokazivača na tip podataka Student, a potom formira objekat na koga taj pokazivač pokazuje. Vrši se konverzija generičkog pokazivača u pokazivac tipa Student. U slučaju da se objekat nije mogao formirati prikazati poruku.

```
Student *p = NULL;
if(p = (Student *) malloc(sizeof(Student)) == NULL)
{
      printf("Memorija nije alocirana!"),
      return;
}
free(p);
```

Primer - program koji omogućava formiranje strukture radnik sa poljima:

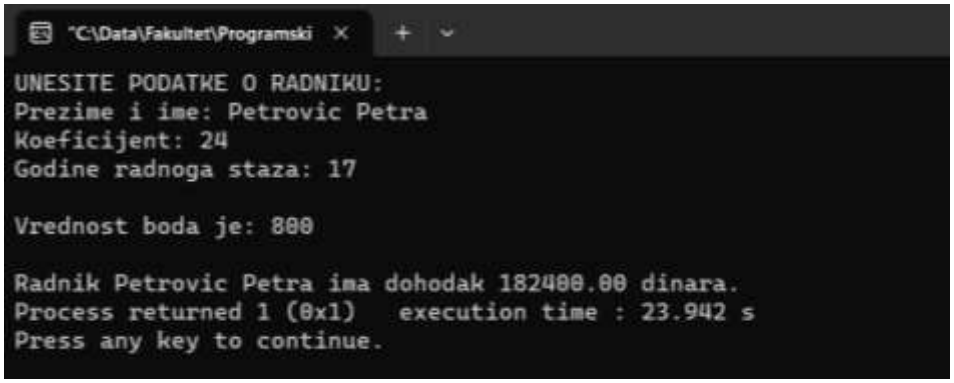
1) Prezime i ime - maksimalno 30 karaktera,
2) Koeficijent za radno mesto realan broj,
3) Godine radnog staža ceo broj,
4) Dohodak realan broj.

Potrebno je uneti: prezime i ime, koeficijent, staž i da se za vrednost boda od k dinara, a da se na svaku godinu radnoga staža dohodak uvećava za 0.5%. Progam treba da ispiše koliki je dohodak radnika. Prilikom realizacije programa koristiti dinamičku zonu memorije.

```
#include <stdio.h>
#include <string.h>
#include<ctype.h>
#include <stdlib.h>
#define MAX 30

// Definisanje strukture
typedef struct radnik
{
    char prezimeime[MAX];
    double koeficijent;
    int staz;
    double dohodak;
} Tradnik;

void main()
{
    Tradnik *p; /* Deklarise se promenljiva p koja predstavlja pokazivac na
slozeni tip podatka Tradnik koji se nalazi u zoni podataka */
    printf("UNESITE PODATKE O RADNIKU:\n");

    double a;
    int b;

    /* Da bi se osiguralo korektno izvrsavanje programa, potrebno je
izvrsiti konverziju generickog pokazivaca u pokazivac tipa Tradnik, pa tek
tada izvrsiti dodelu adrese promenljivoj p. Ako promenljiva p bude imala
vrednost NULL, znaci da se objekat nije formirao usled nedostatka
operativne memorije. */
    if((p=(Tradnik *)malloc(sizeof(Tradnik)))==NULL)
    {
        printf("Nema dovoljno memorije!");
        return NULL;
    }

    /* Inicijalizacija elemenata structure */
    strcpy(p->prezimeime,"");
    p->koeficijent=0;
```

```
    p->staz=0;
    p->dohodak=0;

    /* Unos vrednosti */
    printf("Prezime i ime: ");
    gets(p->prezimeime);
    printf("Koeficijent: ");
    scanf("%lf",&a);
    p->koeficijent=a;
    printf("Godine radnoga staza: ");
    scanf("%d",&b);
    p->staz=b;
    double k=0;
    printf("\nVrednost boda je: ");
    scanf("%lf",&k);

    //izracunavanje dohotka
    p->dohodak=p->koeficijent*k;
    p->dohodak+=0.5*p->staz*p->dohodak;

    //ispis dohotka
    printf("\nRadnik %s ima dohodak %0.2f dinara.", p->prezimeime, p-
>dohodak);

    //oslobadjanje memorije
    free(p);
}
```

Rezultat izvršavanja:



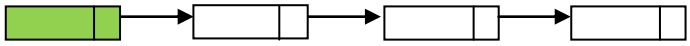
38. JEDNOSTRUKO SPREGNUTA LISTA, POJAM I DEFINICIJA, DODAVANJE ELEMENTA U LISTU

Dinamička struktura podataka u obliku jednostruko spregnute liste sadrži elemente liste koji su povezani pokazivačima na sledeći element. Elementi liste su objekti tipa struktura podataka. Da bi se moglo pristupati elementima liste potreban je pokazivač na prvi element liste, koji se naziva glava liste. Jednostruko spregnuta lista se intenzivno primenjuje u objektnom programiranju, gde su pojedini atributi klase liste objekata neke tipa neke druge klase.

Sa jednostruko spregnutim listama se mogu uraditi sledeće operacije:

- **inicijalizacija (kreiranje) liste,**
- **unos novog elementa u listu,**
- **prikaz elemenata iz liste,**
- **brisanje elementa iz liste,**
- **brisanje (uništavanje) liste (uz oslobađanje memorije).**

Grafički prikaz strukture tipa jednostruko spregnuta lista:



(Primer programa za rad sa listama je modifikovani primer iz M. Jurak: Programski jezik C, 2003/2004.)

Definisanje liste objekata, kraja liste, pokazivača na prvi element:

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

/* oznaka kraja liste */
#define KRAJ (struct ime *) 0
/* deklaracija strukture */
struct ime
{
      char *p_ime;
      struct ime *next;
};

/* pokazivac na prvi element liste */
struct ime *start=KRAJ;
```

Struktura ime sadrži pokazivač na niz znakova p_ime i pokazivač na sledeći element u listi: next. Pokazivač na prvi element liste – start, definisan je kao globalna varijabla i inicijalizovan null pokazivačem koji označava kraj liste. **Lista je na ovaj način definisana i ne sadrži nijedan element**.
Element se u vezanu listu može dodati na bilo kom mestu. **Funkcija za unos,** koja dodaje novi element na kraj liste - Kod prvo alocira memoriju za novi element. Budući da će on biti zadnji u listi, njegov `next` član se postavlja na `KRAJ`. Zatim se alocira memorija za novo `ime` koje se učitava u memoriju `line` pomoću `scanf()` funkcije i kopira na rezervisanu memoriju sa `strcpy()`. Novi element je sada inicijaliziran i lokalna varijabla `novi` pokazuje na njega. Ukoliko je lista prazna, početak liste inicijaliziramo sa `novi`. Ako nije, nalazimo zadnji element i njegov `next` član postavljamo na `novi` koji time postaje zadnji element liste.

```
/* dodavanje novog elementa na kraj liste */
void unos()
{
      struct ime *zadnji, *novi;
      char line[128];
      printf("Dodavanje novog elementa na kraj liste.\n");
      novi = (struct ime *) malloc(sizeof(struct ime));
      if(novi == NULL)
      {
            printf("Nema dovoljno memorije ... ");
            return -1;
      }
      novi->next=KRAJ;
      printf("Unesite ime > ");
      scanf(" %[^\n]",line);
      novi->p_ime=(char *) malloc(strlen(line)+1);
      if(novi->p_ime == NULL)
      {
            printf("Nema dovoljno memorije ...");
            return -1; // exit(-1);
      }
      strcpy(novi->p_ime,line);
      if(start==KRAJ) /* prazna lista */
            start=novi;
      else
      {
            /* pronadji kraj liste */
            for(zadnji=start; zadnji->next != KRAJ; zadnji=zadnji-
            >next); // ne radi nista
            /* neka zadnji pokazuje na novi element */
            zadnji->next=novi;
      }
}
```

Ispis/štampanje liste - Kod jednostruko spregnute liste sve ispisivanje elemenata počinjati od prvog elementa. Ciklusom se prolazi kroz strukturu od prvog do poslednjeg elementa i nekom od funkcija C jezika se se prikazuju vrednosti polja (u primeru samo jednog, ali ih može biti i više) referenciranjem: naziv_objekta.ime_polja ili naziv_objekta->ime_polja.

```
void ispis()
{     /* Ispis liste */
      struct ime *element;
      int i=1;
      if(start == KRAJ )
      {
            printf("Lista je prazna.\n");
            return;
      }
      printf("Ispis liste \n");
      for(element=start; element!=KRAJ; element=element->next)
      {
            printf("%d. %s\n",i,element->p_ime); i++;
      }
}
```

39. TRAŽENJE ELEMENTA U JEDNOSTRUKO SPREGNUTOJ LISTI, BRISANJE ELEMENTA I OSLOBAĐANJE ZAUZETE MEMORIJE

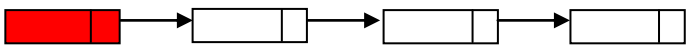
Dinamička struktura podataka u obliku jednostruko spregnute liste sadrži elemente liste koji su povezani pokazivačima na sledeći element. Za sortiranje jednostruko spregnute liste ne postoje posebne funkcije, već se koriste tehnike sortiranje niza, tj. vektora.

Pretraživanje jednostruko spregnute liste - Ukoliko lista nije prazna, u jednom for ciklusu, se prolazi kroz njene članove. Za pretraživanje koristimo for ciklus kojim se obilazi čitavu lista:

```
void trazi()
{ /* Pretrazivanje liste */
      struct ime *test;
      char line[128];
      int i;
      printf("Nalazenje imena u listi.\n");
      printf("Unesite ime ");
      scanf(" %[^\n]",line);
      i=1;
      for(test=start; test!=KRAJ; test=test->next)
      {
            if(!strcmp(test->p_ime,line))
            {
                  printf("Podatak je %d. u listi\n",i);
                  return;
            }
            i++;
      }
      printf("Podatak nije u listi.\n");
}
```

Funkcija za pretragu vrši upoređivanje upisanog imena sa svakim podatkom u listi linearno, počevši od prvog do zadnjeg elementa liste i ukoliko ga pronađe, ispisaće poruku Podatak je n. u listi. Ukoliko je pretraživanje neuspešno, ime nije u listi biće ispisana poruka: Podatak nije u listi.

Brisanje elementa iz liste:

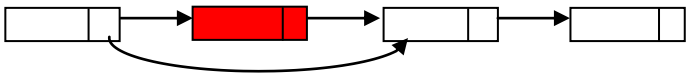


```
void brisi() /* Brisanje elementa iz liste */
{
      struct ime *test, *tmp;
      char line[128];
      int i;
      printf("Brisanje imena iz liste.\n");
      if(start == KRAJ)
      {
            printf("Lista je prazna.\n");
            return;
      }
      printf("Unesite ime ");
      scanf(" %[^\n]",line);
      /* ako je prvi element onaj koji trazimo */
```

```
        if( !strcmp(start->p_ime,line) )
        {
                printf("Podatak je 1. u listi\n");
                tmp=start;
                start=start->next;
                free(tmp->p_ime);
                free(tmp);
                return;
        }
        i=1;
        for(test=start; test!= KRAJ; test=test->next)
        {
                i++;
                if(!(tmp=test->next)) break; // element nije nadjen
                if( !strcmp(tmp->p_ime,line) )
                {
                        printf("Brisemo podatak br. %d u listi\n",i);
                        test->next=test->next->next;
                        free(tmp->p_ime);
                        free(tmp);
                        return;
                }
        }
        printf("Podatak nije u listi.\n");
        return;
}
```

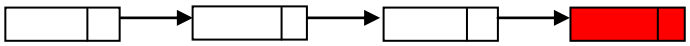
Element se iz vezane liste može izbrisati na bilo kom mestu. **Kod brisanja elementa iz liste prvo proveravamo da li je lista prazna**. **Ako nije, unosimo ime koje želimo izbrisati i provjeravamo da li je ono na prvom mestu u listi**, koje je nešto jednostavnije od brisanja proizvoljnog elementa. Dovoljno je samo start postaviti da pokazuje na drugi element: start=start->next.

Budući da je memorija za svaki element alocirana dinamički, prilikom brisanja elementa, treba je osloboditi. Stoga se stara vrednost pokazivača start pamti u lokalnoj varijabli tmp i memorija se oslobada pozivom funkcije: free(tmp).



Kod brisanja elementa na proizvoljnom mestu porebno je imati pokazivač na element koji prethodi elementu za brisanje jer njegov next član treba preusmeriti na element iza onog koji brišemo. Zato u petlji for(test=start; test != KRAJ; test=test->next) proveravamo da li tmp=test->next pokazuje na element za brisanje. Ako da, onda se test->next postavlja na test->next->next, prvi element iza onog koji brišemo i time je izvršeno izbacivanje iz liste. Ostaje još samo **delocirati memoriju izbrisanog elementa.**

U ovom kodu postoji problem zadnjeg elemeta. **Ako je to zadnji element, onda je tmp=test->next=KRAJ pa bi tmp->p ime u strcmp funkciji dalo grešku. Zato testiramo taj pokazivač i izlazimo iz petlje pomoću break naredbe ako smo na zadnjem elementu.**



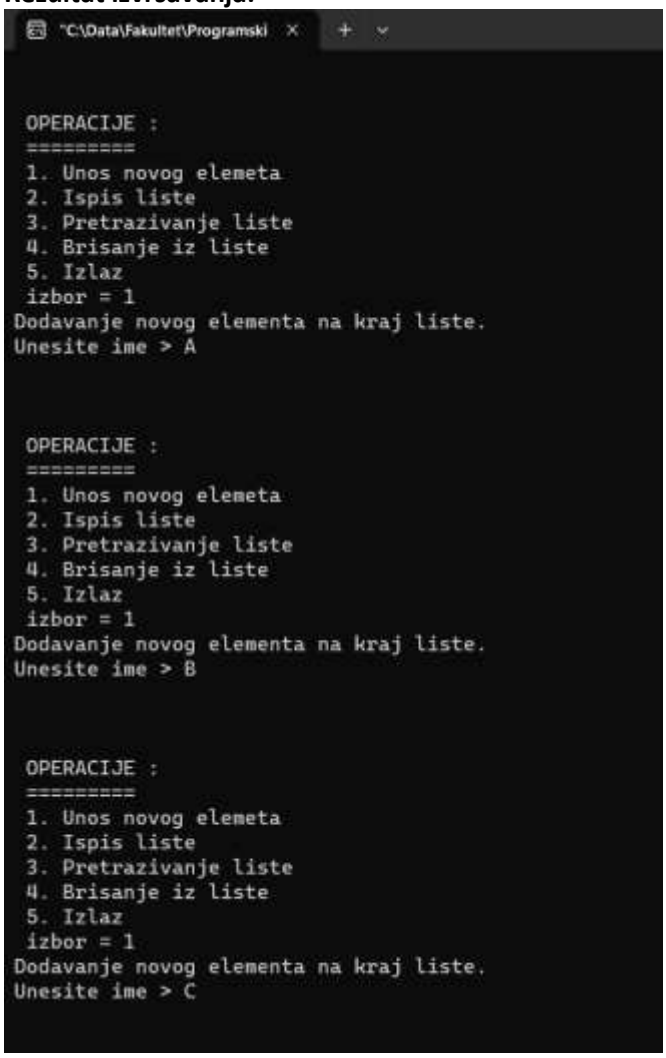
Meni programa sa mogućim operacijama:

```
int menu()
{
	int izbor;
	printf("\n\n\n");
	printf("\n OPERACIJE : ");
	printf("\n =========");
	printf("\n 1. Unos novog elemeta ");
	printf("\n 2. Ispis liste ");
	printf("\n 3. Pretrazivanje liste ");
	printf("\n 4. Brisanje iz liste ");
	printf("\n 5. Izlaz ");
	printf("\n izbor = ");
	scanf("%d",&izbor);
	return izbor;
}
```

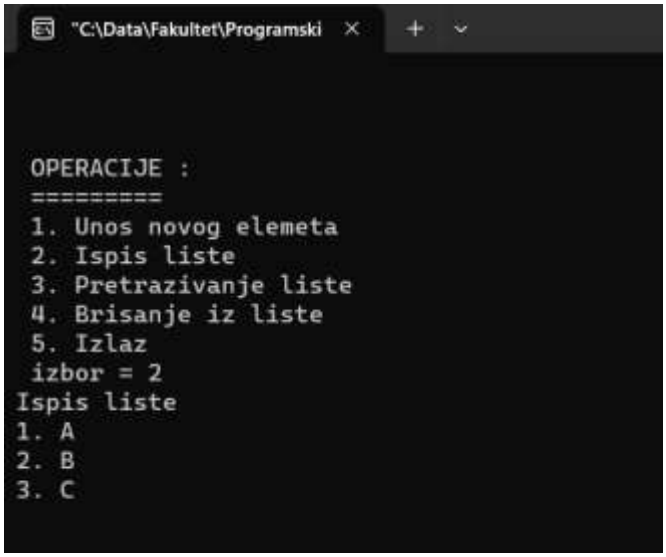
Prikaz glavnog programa:

```
int main()
{
	int izbor;
	do
	{
		switch(izbor=menu())
		{
			case 1:
				unos();
				break;
			case 2:
				ispis();
				break;
			case 3:
				trazi();
				break;
			case 4:
				brisi();
				break;
			case 5:
				break;
			default:
				printf("\n Pogresan izbor.");
		}
	}
	while(izbor!=5);
	return 0;
}
```

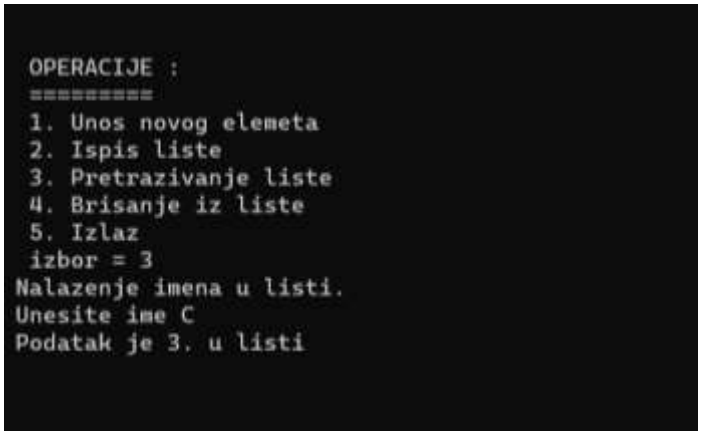
Rezultat izvršavanja:



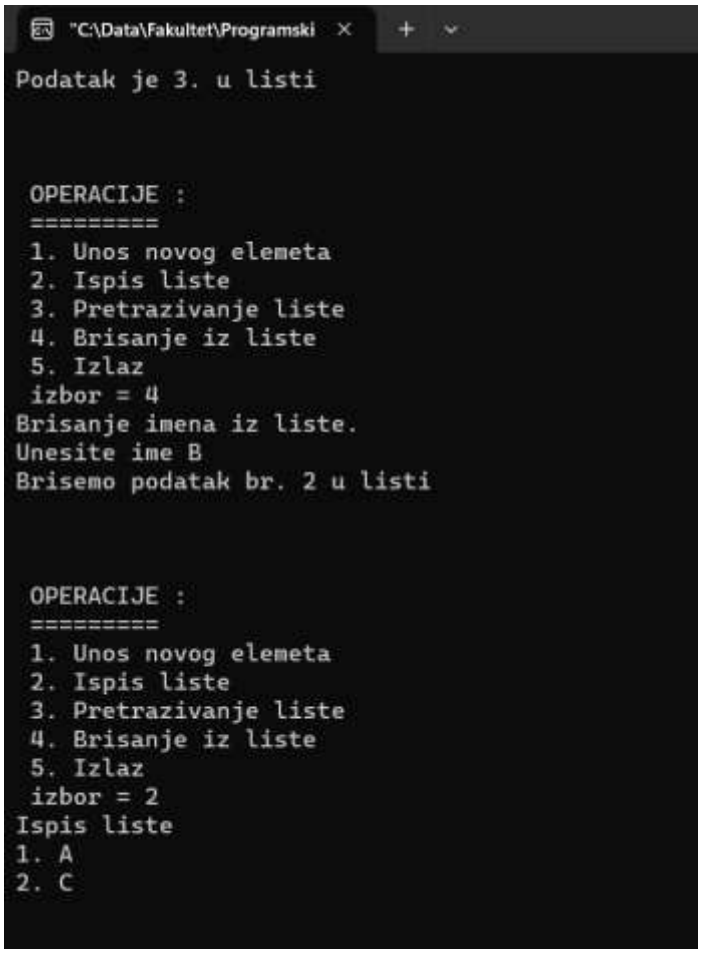
<- Unos elemenata



<- Ispis elemenata liste



<- Pretraga elementa

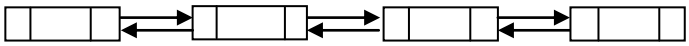


<- Brisanje elementa

40. DVOSTRUKO SPREGNUTA LISTA

Dinamička struktura podataka u obliku dvostruko spregnute liste sadrži elemente liste koji su povezani pokazivačima na sledeći element i na prethodni element. Elementi liste su objekti tipa struktura podataka koja sadrži određene elemente i dva pokazivača – na prethodni i sledeći element u listi. Vrednost pokazivača prvog i poslednjeg elementa na sledeći, tj. prethodni element su NULL.

Grafički prikaz dvostruko spregnute liste:



Dvostruko spregnuta lista je struktura koja je slična jednostruko spregnutoj listi. Sve što važi za jednostruko spregnutu listu, važi i za dvostruko spregnutu listu.

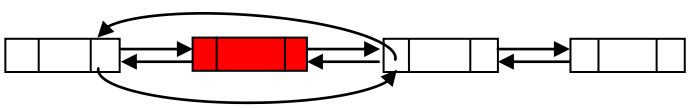
Osnovne karakteristike dvostruko spregnute liste:

- **omogućuje dodavanje elemenata,**
- **omogućuje brisanje elemenata,**
- **pristup elementima odvija se po redosledu,**
- **lista se može sortirati, ali ne mora.**

Definicija dvostruko spregnute liste:

```
/* oznaka kraja liste */
#define KRAJ (struct ime *) 0

/* deklaracija strukture */
struct ime
{
      char *p_ime;
      struct ime *previous;
      struct ime *next;
};

/* pokazivac na prvi element liste */
struct ime *start=KRAJ;
```

Osnovna prednost dvostruko spregnute liste u odnosu na jednostruku jeste mogućnost kretanja (tj. pristupa elementima) u oba smera. Kod jednostruko spregnute liste sve akcije moraju počinjati od prvog elementa. Ako treba ukloniti tekući element, u jednostruko spregnutoj listi moralo bi se ponovo krenuti od početka da bi se pronašao njegov prethodnik čiji pokazivač treba da se ažurira. Dvostruko spregnuta lista je pogodnija za navigaciju jer se iz tekućeg elementa može direktno pristupiti i njegovom pethodniku i njegovom sledbeniku, bez potrebe traženja. Dakle, osnovna prednost dvostruko spergnute liste u odnosu na jednostruko spregnutu je veća pogodnost za navigaciju, a cena toga su nešto sporije operacije zbog potrebe održavanja dva, a ne jednog pokazivača.



Dvostruko spregnuta lista omogućuje bolju primenu određenih metoda sortiranja, za razliku od jedostruko spregnute liste.

KORIŠTENA LITERATURA:

1. A. Hansen: Programiranje na jeziku C – Potpuni vodič za programski jezik C, Mikro knjiga, Beograd, 2000.
2. C.H. Pappas, W.H. Murray: C/C++ vodiš za programere, Mikro knjiga, Beograd, 1996.
3. M. Jurak: Programski jezik C, 2003/2004.
4. D. Ritchie, W. Kernighan: Programski jezik C, CET, Beograd, 2003.
5. C.L. Tondo, S.E. Gimpel: Programski jezik C rešenja zadataka, drugo izdanje, CET, Beograd, 1989.
6. V. Ćirić: Uvod u programiranje I programski jezik C, Univerzitet u Nišu, Elektronski fakultet, 2014.
7. B. Markoski: Praktikum rešenih zadataka iz programskog jezika C, Univerzitet u novom Sadu, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2013.
8. B. Markoski: Interni materijal za spremanje ispita iz Programskih jezika, Univerzitet u novom Sadu, Tehnički fakultet "Mihajlo Pupin", Zrenjanin, 2021/2022.
9. N. Jovanović: Uvod u programiranje, Viša poslovna škola, Blace, 2005.
10. D. Malbaški: Algoritmi i strukture podataka, Univerzitet Educons, Sremska Kamenica, 2015.
11. D. Malbaški: Programski jezici i strukture podataka, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Novi Sad.
12. B. Stojanović: Strukture podataka i algoritmi 2, Prirodno matematički fakultet Kragujevac, 2020.
13. A. Kelley, I. Pohl: A Book on C, Addison-Wesley, 1998.
14. N.M. jovanović: Uvod u programiranje, Viša poslovna škola, Blace, 2005.
15. T. Bailey: An Introduction to the C Programming Language and Software Design, Draft 0.6 (July 12, 2005).
16. https://slidetodoc.com/lekcija-01-uvodna-razmatranja-uvod-u-c-miljan/ M. Milošević, Uvod u C.
17. http://stackoverflow.com, public platform for software development.
18. http://poincare.matf.bg.ac.rs/~gordana//programiranjeI/Programski_Jezik_C_pokazivaci.pdf Programski jezik C – pokazivači, Matematički fakultet, Beograd.
19. http://tutorialspoint.com C Programing tutorial.
20. D. Drakulić: Osnove programskog jezika C sa zbirkom zadataka – skripta.
21. R. Goić: Programski jezik C - bilješke s predavanja, privremeni i nepotpuni materijali, Split, 2002.