Факултет организационих наука, Универзитет у Београду

Семинарски рад из предмета Принципи програмирања

Имплементација двоструко спрегнуте циркуларне листе

Ментор:

Студент:

проф. др Саша Д. ЛазаревићМарко Мијаиловић 2022/0034

Београд, 2023.

Садржај:

[1.Увод 3](#_Toc135417708)

[2.Листе 4](#_Toc135417709)

[2.1.Једноструко спрегнуте листе 4](#_Toc135417710)

[2.2.Једноструко спрегнуте циркуларне листе 4](#_Toc135417711)

[2.3.Двоструко спрегнуте листе 4](#_Toc135417712)

[2.4.Двоструко спрегнуте циркуларне листе 5](#_Toc135417713)

[3.Двоструко спрегнуте циркуларне листе 6](#_Toc135417714)

[4.Имплементација двоструко спрегнуте циркуларне листе 7](#_Toc135417715)

[5.Закључак 8](#_Toc135417716)

[6.Литература 8](#_Toc135417717)

# 1.Увод

У овом семинарском раду ћу се бавити двоструко спрегнутим сиркуларним листама, њиховом применом и имплементацијом у програмском језику *C*. Размотрићу основне карактеристике двоструко спрегнутих циркуларних листи и објаснити зашто ове листе могу бити корисне у одређеним ситуацијама. Покрићу кључне концепте и операције које су потребне за имплементацију двоструко спрегнуте циркуларне листе. То укључује креирање листе, додавање и брисање елемената с почетка и краја листе, претрагу елемената у листи, замену елемената, сортирање листе, приказивање елемената листе у одговарајућем редоследу и још неке основне операције са листом. Такође обрадићу и неке операције које захтевају две листе као што су унија, разлика, пресек, поређење две листе и копирање из једне листе у другу.

# 2.Листе

Листе су једна од основних структура података у програмирању које омогућавају организацију и манипулацију колекцијом елемената. Листа представља потпуно уређену линеарну структуру података. Састоји се из чворова, при чему сваки чвор садржи податке и показивач на следећи чвор у листи. На тај начин, елементи листе се повезују и формирају секвенцијалну структуру.

Једна од предности коришћења листи је њихова флексибилност. За разлику од низова, листе могу динамички мењати величину током извршавања програма. То значи да се елементи могу додавати или уклањати из листе без потребе за предефинисањем фиксног броја елемената или реалокације меморије.

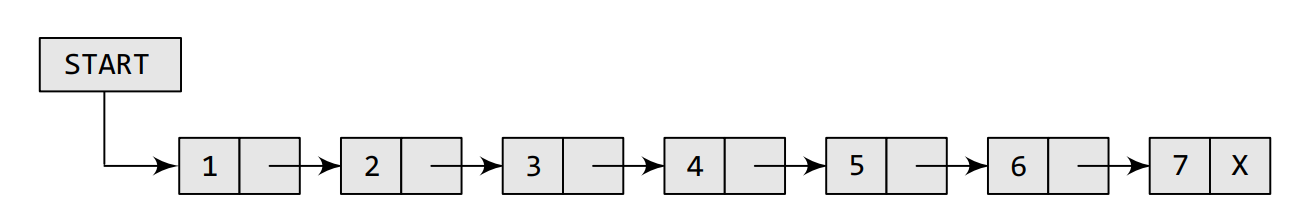
Постоји четири различите врсте листи: једноструко спрегнуте листе, једноструко спрегнуте циркуларне листе, двоструко спрегнуте листе и двоструко спрегнуте циркуларне листе.

## 2.1.Једноструко спрегнуте листе

Једноструко спрегнуте(повезане) листе су основни облик листа који се састоји од чворова који садрже податке и показивач на следећи чвор у листи.

Елементи листе су повезани једносмерно, од почетка према крају листе.

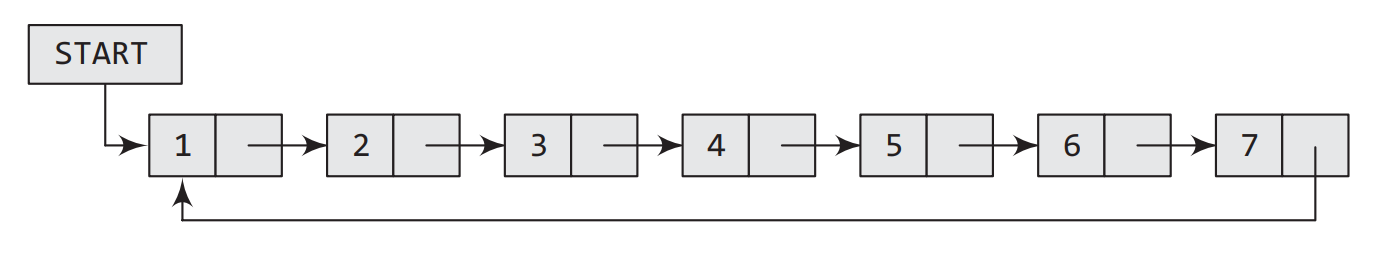
Предност ових листи је једноставна имплементација и ефикасно додавање и брисање елемената с почетка листе. Међутим, приступ елементима у средини или крају листе је спорији јер се листа мора линеарно претраживати.



## 2.2.Једноструко спрегнуте циркуларне листе

Једноструко спрегнуте циркуларне(кружне) листе су специфичан облик листа у којем последњи чвор показује на први чвор, чиме се формира цикличка структура.

Ово омогућава да се листа може кружно пролазити без прекида.

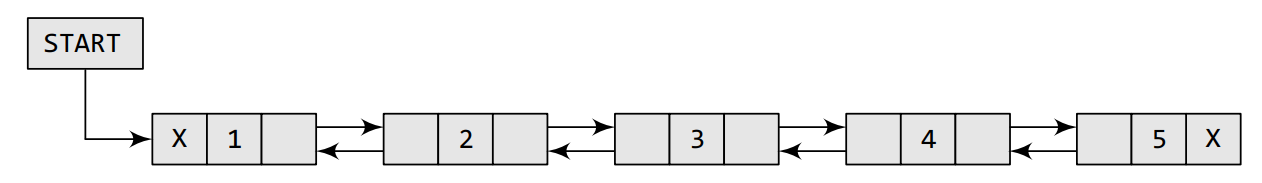


## 2.3.Двоструко спрегнуте листе

Двоструко спрегнуте(повезане) листе садрже чворове који имају показиваче на претходни и следећи чвор, чиме се омогућава двосмерна повезаност.

Ова структура омогућава ефикасно додавање и брисање елемената како с почетка, тако и с краја листе.

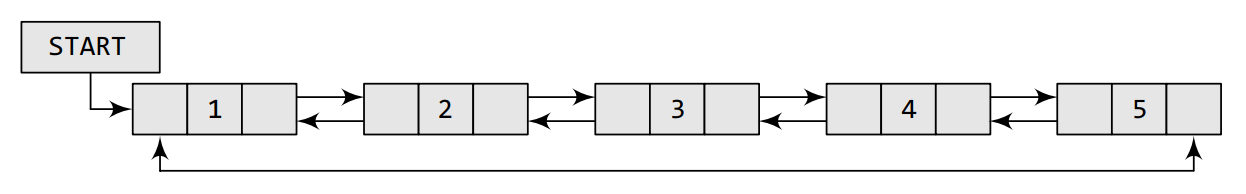
Притуп елементима у средини листе је бржи у односу на једноструко повезане листе јер се може кретати у оба смера. Међутим, двоструко повезане листе заузимају више меморије због додатних показивача.



## 2.4.Двоструко спрегнуте циркуларне листе

Двоструко спрегнуте циркуларне(кружне) листе су специфична врста листа која комбинује карактеристике двоструко спрегнутих и циркуларних листи.

Разлика између двоструко спрегнуте листе и циркуларне двоструко спрегнуте листе је у томе што циркуларне двоструко спрегнуте листа не садржи *NULL* у пољу претходног чвора првог чвора и пољу следећег чвора последњег чвора.Уместо тога, поље следећег чвора последњег чвора чува адресу првог чвора листе. Слично томе, поље претходног чвора првог чвора чува адресу последњег чвора.



# 3.Двоструко спрегнуте циркуларне листе

Двоструко спрегнуте циркуларне листе су веома корисна структура података која налази примену у различитим доменима програмирања и решавању проблема. Њихова специфична комбинација двоструко спрегнуте и циркуларне структуре пружа низ предности и омогућава ефикасно управљање подацима. Разумевање овог концепта омогућава програмерима да ефикасно управљају подацима и оптимизују перформансе својих програма.

Једна од честих примена двоструко спрегнуте циркуларне листе је у имплементацији *cache* меморије. *Cache* меморија се користи за складиштење најчешће коришћених података ради бржег приступа. Коришћењем двоструко спрегнуте циркуларне листе, подаци се могу ротирати у петљи, омогућавајући константан приступ и оптимално искоришћавање *cache* меморије.

Двоструко спрегнута листа се може користити за имплементацију ундо/редо механизма у апликацијама. Свака акција коју корисник предузме се памти као чвор у листи, омогућавајући поништавање (*undo*) и враћање (*redo*) претходних акција.

Двоструко спрегнуте циркуларне листе се често користе за репродукцију песама у оба смера, у музичким плејерима. Сваки чвор у листи представља једну песму, а кружна структура омогућава непрекидну репродукцију песама у петљи, без потребе за прекидом репродукције.

Важно је напоменути да двоструко спрегнуте циркуларне листе нису универзално решење и не одговарају свим врстама проблема. Имају своје предности и мане које треба узети у обзир приликом разматрања њихове употребе.

Предности:

1.Ефикасно додавање и брисање елемената - Двоструко спрегнуте циркуларне листе омогућавају брзо додавање и брисање елемената како са почетка, тако и са краја листе. Ове операције се могу извршити у константном времену без потребе за линеарним претраживањем листе.

2.Двосмерна повезаност - Сваки чвор у двоструко спрегнутој циркуларној листи има показиваче на претходни и следећи чвор. То омогућава ефикасно кретање унапред и уназад кроз листу, што је корисно за операције као што су претрага, измена или манипулација елемената у средини листе.

3.Циркуларна структура - Двоструко спрегнуте циркуларне листе имају цикличку структуру, где се последњи чвор повезује са првим чвором. То омогућава кружни пролазак кроз листу без прекида, што је корисно у ситуацијама где је потребно обављати итерације преко листе у бесконачном циклу.

4.Флексибилност - Двоструко спрегнуте циркуларне листе омогућавају динамичко управљање подацима. Могу се лако проширивати додавањем нових чворова или смањивати уклањањем постојећих чворова без потребе за реалокацијом меморије.

Мане:

1.Повећана комплексност имплементације - Имплементација двоструко спрегнуте циркуларне листе захтева пажљиво управљање показивачима како би се одржала повезаност између чворова. Ова додатна комплексност може повећати могућност грешака у кодирању и отежати одржавање и дебаговање.

2.Повећана меморијска потрошња - Двоструко спрегнуте циркуларне листе заузимају више меморије у односу на једноструко повезане листе због додатних показивача на претходни чвор. Ово може бити значајан фактор уколико је меморијска ефикасност важна у одређеној апликацији.

3.Специфична примена - Двоструко спрегнуте циркуларне листе имају своје најбоље примене у одређеним сценаријима. У неким ситуацијама, као што су једноставне линеарно повезане структуре, могу бити прекомпликоване и непотребне. Пре него што се одлучимо за коришћење двоструко спрегнуте циркуларне листе, важно је пажљиво проценити захтеве апликације и узети у обзир предности и мане ове структуре података у односу на друге алтернативе.

4.Одржавање повезаности - Додатна повезаност и показивачи у двоструко спрегнутој циркуларној листи захтевају пажљиво одржавање како би се избегле грешке попут цикличких петљи или губитка података. Неправилно управљање показивачима може довести до непредвидљивог понашања листе.

# 4.Имплементација двоструко спрегнуте циркуларне листе

Функција за испис од почетка листе. Пролазимо кроз листу од главе и исписујемо сваки податак померајући помоћну (*pom*) на следећи елемент, док не дођемо поново до главе.

void ispisListeOdPocetka(P\_ELEMENT glava) {

if (glava == NULL) {

printc("Lista je prazna!",S\_CRVENA);

}

else {

P\_ELEMENT pom;

pom = glava;

printc("Lista je: ",S\_AQUA);

do {

printf("%d ", pom->info);

pom = pom->sled;

} while (pom != glava);

}

printf("\n\n");

}

Функција за испис од краја листе. Пролазимо кроз листу од репа(glava->pred) и исписујемо сваки податак померајући помоћну на претходни елемент, док не дођемо поново до репа.

void ispisListeOdKraja(P\_ELEMENT glava) {

if (glava == NULL) {

printf("Lista je prazna!");

}

else {

P\_ELEMENT pom;

pom = glava->pred;

printc("Lista je: ", S\_AQUA);

do {

printf("%d ", pom->info);

pom = pom->pred;

} while (pom != glava->pred);

}

printf("\n\n");

}

Фукнција додаје елемент на крај листе. Помоћу функције *malloc* обезбеђујемо меморију на *heap*-у за нови чвор. Постоје два случаја, као и код једноструко спрегнутих листи, када је листа празна и када већ садржи елементе. У зависности од тога мењамо одговарајуће показиваче.

int dodajElementNaKrajListe(P\_ELEMENT\* glava, int vrednost) {

P\_ELEMENT noviElement;

P\_ELEMENT rep=NULL;

noviElement = (P\_ELEMENT)malloc(sizeof(ELEMENT));

noviElement->info = vrednost;

if ((\*glava) == NULL) {

\*glava = noviElement;

noviElement->pred = \*glava;

noviElement->sled = \*glava;

rep = noviElement;

}

else {

//rep = \*glava;

//while (rep->sled != \*glava)

//{

// rep = rep->sled;

//}

rep = (\*glava)->pred;

noviElement->pred = rep;

rep->sled = noviElement;

(\*glava)->pred = noviElement;

noviElement->sled = \*glava;

rep = noviElement;

}

return noviElement->info;

}

Фукнција додаје елемент на почетак листе. Помоћу функције *malloc* обезбеђујемо меморију на *heap*-у за нови чвор. Постоје два случаја, као и код једноструко спрегнутих листи, када је листа празна и када већ садржи елементе. У зависности од тога мењамо одговарајуће показиваче.

int dodajElementNaPocetakListe(P\_ELEMENT\* glava, int vrednost) {

P\_ELEMENT noviElement;

P\_ELEMENT rep = NULL;

noviElement = (P\_ELEMENT)malloc(sizeof(ELEMENT));

noviElement->info = vrednost;

if ((\*glava) == NULL) {

\*glava = noviElement;

(\*glava)->sled = \*glava;

(\*glava)->pred = \*glava;

rep = \*glava;

}

else {

rep = \*glava;

while (rep->sled != \*glava)

{

rep = rep->sled;

}

rep->sled = noviElement;

noviElement->pred = rep;

(\*glava)->pred = noviElement;

noviElement->sled = \*glava;

\*glava = noviElement;

}

return noviElement->info;

}

Функција додаје елемент на прослеђену позицију. Прво проверавамо да ли је могуће додати елемент на задату позицију, односно да ли је унета позиција исправна.Ако није исписујемо одговарајућу поруку. Ако јесте померамо помоћну (*pom*), поставимо је на један елемент пре позиције и променимо показиваче.

void dodajElementNaZadatuPoziciju(P\_ELEMENT\* glava, int vrednost, int pozicija) {

P\_ELEMENT noviElement;

P\_ELEMENT pom;

int i = 1;

int n = 0;

if (\*glava != NULL) {

n = duzinaListe(\*glava);

}

if (pozicija == 1) {

dodajElementNaPocetakListe(glava, vrednost);

}

else if (pozicija < 1 || pozicija > n + 1) {

printc("Uneta pozicija je pogresna\n\n", S\_CRVENA);

return;

}

else {

noviElement = (P\_ELEMENT)malloc(sizeof(ELEMENT));

noviElement->info = vrednost;

pom = \*glava;

while (i < pozicija - 1) {

pom = pom->sled;

i++;

}

noviElement->pred = pom;

noviElement->sled = pom->sled;

pom->sled->pred = noviElement;

pom->sled = noviElement;

}

printc("Element je dodat\n\n", S\_AQUA);

}

Фунцкија омогућава унос листе док корисник не унесе 0, елементи се додају на крај позивом раније направљене функције.

void kreirajListu(P\_ELEMENT\* glava) {

int vrednost;

int i = 1;

while (1)

{

printf("Unesite %d. element.(unesite 0 za kraj unosa)\n",i);

scanf("%d", &vrednost);

i++;

if (vrednost == 0)

break;

dodajElementNaKrajListe(glava, vrednost);

}

printc("Lista je kreirana!\n\n",S\_AQUA);

}

Функција брише елемент са почетка листе. Постоје два случаја када постоји један елемент у листи и када их има више. У зависности од тога мењамо одговарајуће помазиваче, а затим помоћу функције *free* ослобађамо меморију елемента који бришемо.

void obrisiElementSaPocetkaListe(P\_ELEMENT\* glava) {

P\_ELEMENT pom;

pom = \*glava;

P\_ELEMENT rep;

rep = \*glava;

if (\*glava == NULL) {

printc("Lista je prazna!\n\n",S\_CRVENA);

return;

}

while (rep->sled != \*glava)

{

rep = rep->sled;

}

if (\*glava == (\*glava)->sled) {

\*glava = NULL;

rep = NULL;

free(pom);

}

else {

\*glava = (\*glava)->sled;

(\*glava)->pred = rep;

rep->sled = \*glava;

free(pom);

}

}

Функција брише елемент са краја листе. Постоје два случаја када постоји један елемент у листи и када их има више. У зависности од тога мењамо одговарајуће помазиваче, а затим помоћу функције *free* ослобађамо меморију елемента који бришемо.

void obrisiElementSaKrajaListe(P\_ELEMENT\* glava) {

P\_ELEMENT pom;

P\_ELEMENT rep;

rep = \*glava;

if (\*glava == NULL) {

printc("Lista je prazna!\n\n",S\_CRVENA);

return;

}

while (rep->sled != \*glava)

{

rep = rep->sled;

}

pom = rep;

if (\*glava == (\*glava)->sled) {

\*glava = NULL;

rep = NULL;

free(pom);

}

else {

rep = rep->pred;

rep->sled = \*glava;

(\*glava)->pred = rep;

free(pom);

}

}

Функција брише елемент листе који се налази на прослеђеној позицији. Прво проверавамо да ли је позиција исправна. Ако је унета позиција 1 или *n* користимо раније направљене функције. У супротном користимо помоћну (*pom*) и бришемо одговарајући елемент.

void obrisiElementNaPoziciji(P\_ELEMENT\* glava, int pozicija) {

P\_ELEMENT pom;

pom = \*glava;

int n = duzinaListe(\*glava);

int i = 1;

if (pozicija<1 || pozicija>n) {

printc("Pozicija ne postoji\n\n",S\_CRVENA);

}

else if(pozicija==1){

obrisiElementSaPocetkaListe(glava);

}

else if (pozicija == n) {

obrisiElementSaKrajaListe(glava);

}

else {

while (i<pozicija)

{

pom = pom->sled;

i++;

}

pom->pred->sled = pom->sled;

pom->sled->pred = pom->pred;

free(pom);

}

}

Функција брише елемент листе који има прослеђену вредност. Пролазимо кроз листу и проверавамо да ли се јавља унета вредност. Ако се јавља користимо претходно креирану функцију за брисање елемнта са позиције и бришемо његово прво појављивање у листи, у супротном исписујемо одговарајућу поруку.

void obrisiElementSaVrednoscu(P\_ELEMENT\* glava, int vrednost) {

//SAMO PRVI sa tom vrednostcu brise

P\_ELEMENT pom;

pom = \*glava;

int daLiSeJavlja = 0;

int i = 1;

if (\*glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

do {

if (pom->info == vrednost) {

obrisiElementNaPoziciji(glava, i);

pom = \*glava;

i--;

daLiSeJavlja = 1;

}

else {

pom = pom->sled;

i++;

}

} while (pom != \*glava);

if (!daLiSeJavlja) {

printc("Uneta vrednost ne postoji u listi\n\n", S\_CRVENA);

return;

}

}

Функција брише све елементе листе који имају прослеђену вредност.

void obrisiSveElementeSaVrednoscu(P\_ELEMENT\* glava, int vrednost) {

//SVE brise

P\_ELEMENT pom;

pom = \*glava;

int j = 1;

int daLiSeJavlja = 0;

int n = duzinaListe(\*glava);

if (\*glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

for (int i = 0; i <= n; i++)

{

if (pom->info == vrednost) {

obrisiElementNaPoziciji(glava, j);

pom = \*glava;

i = 0;

j = 1;

n--;

daLiSeJavlja = 1;

}

else {

pom = pom->sled;

j++;

}

}

if (!daLiSeJavlja) {

printc("Uneta vrednost ne postoji u listi\n\n", S\_CRVENA);

return;

}

}

Функција брише целу листу. Пролази кроз целу листу и брише један по један елемент док не дође до краја.

void obrisiCeluListu(P\_ELEMENT\* glava) {

P\_ELEMENT pom;

P\_ELEMENT tekuci;

if (\*glava == NULL) {

//printc("Lista je prazna!\n\n",S\_CRVENA);

return 0;

}

else{

tekuci = (\*glava)->sled;

while (tekuci != \*glava) {

pom = tekuci->sled;

free(tekuci);

tekuci = pom;

}

free(\*glava);

\*glava = NULL;

}

printc("Lista je obrisana\n", S\_AQUA);

}

Основне аритметичке функције за листу радимо слично као и за низ (дужина, максимум, минимум, сума, средња вредност)

int duzinaListe(P\_ELEMENT glava) {

int n = 0;

P\_ELEMENT pom;

if (glava == NULL) {

return n;

}

else {

pom = glava;

do {

n++;

pom = pom->sled;

} while (pom != glava);

}

return n;

}

int maxListe(P\_ELEMENT glava) {

int max;

if (glava == NULL) {

//printf("Lista je prazna, pa nema maksimum!\n");

return 0;

}

else {

P\_ELEMENT pom;

pom = glava;

max = pom->info;

do {

pom = pom->sled;

if (max < pom->info) {

max = pom->info;

}

} while (pom != glava);

}

return max;

}

int minListe(P\_ELEMENT glava) {

int min;

if (glava == NULL) {

//printf("Lista je prazna, pa nema minimum!\n");

return 0;

}

else {

P\_ELEMENT pom;

pom = glava;

min = pom->info;

do {

pom = pom->sled;

if (min > pom->info) {

min = pom->info;

}

} while (pom != glava);

}

return min;

}

int sumaListe(P\_ELEMENT glava) {

int suma = 0;

if (glava == NULL) {

return 0;

}

else {

P\_ELEMENT pom;

pom = glava;

do {

suma += pom->info;

pom = pom->sled;

} while (pom != glava);

}

return suma;

}

double srednjaVrednostListe(P\_ELEMENT glava) {

double srvr = 0;

int n = duzinaListe(glava);

int suma = sumaListe(glava);

if (!n) {

return 0;

}

else {

srvr = (double)suma / n;

return srvr;

}

}

Функција мења елементе на прослеђеним позицијама.

void ZamenaElemenataListe(P\_ELEMENT glava, int pozicija1, int pozicija2) {

int n = duzinaListe(glava);

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

if (pozicija1<1 || pozicija1>n || pozicija2<1 || pozicija2>n) {

return;

}

P\_ELEMENT prvi = glava;

P\_ELEMENT drugi = glava;

for (int i = 1; i < pozicija1; i++)

{

prvi = prvi->sled;

}

for (int i = 1; i < pozicija2; i++)

{

drugi = drugi->sled;

}

int pom;

pom = prvi->info;

prvi->info = drugi->info;

drugi->info = pom;

}

Функција поставља нове вредсноти на све елементе са прослеђеном старом вредношћу.

void promeniElementPoVrednosti(P\_ELEMENT glava, int staraVrednost, int novaVrednost) {

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

if (glava != NULL) {

P\_ELEMENT pom;

pom = glava;

do {

if (pom->info == staraVrednost) {

pom->info = novaVrednost;

}

pom = pom->sled;

} while (pom != glava);

}

}

Функција мења вредност елемента на прослеђеној позицији.

void promeniElementPoPoziciji(P\_ELEMENT glava, int vrednost, int pozicija) {

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

if (glava != NULL) {

P\_ELEMENT pom;

pom = glava;

for (int i = 1; i < pozicija; i++)

{

pom = pom->sled;

}

pom->info = vrednost;

}

}

Фунцкија проверава да ли су сви елементи у листи различити.

int daLiSuRazliciti(P\_ELEMENT glava) {

int daLiSePojavljuje = 0;

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

if (glava != NULL) {

P\_ELEMENT predhodni;

P\_ELEMENT tekuci;

predhodni = glava;

tekuci = glava->sled;

do {

do {

if (tekuci != predhodni && predhodni->info == tekuci->info) {

daLiSePojavljuje = 1;

return 0;//printf("Nisu svi elementi u listi razliciti");

}

tekuci = tekuci->sled;

} while (tekuci != glava);

predhodni = predhodni->sled;

} while (predhodni != glava);

}

if (!daLiSePojavljuje)

return 1; //printf("Svi elementi u listi su razliciti");

}

Функција проверава да ли прослеђена вредност постоји у листи и исписујемо одговарајућу поруку.

void daLiPostojiUListi(P\_ELEMENT glava, int vrednost) {

int i = 0;

int br = 0;

int daLiSePojavljuje = 0;

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

if (glava != NULL) {

P\_ELEMENT pom;

pom = glava;

do {

i++;

if (pom->info == vrednost) {

printf("%d se nalazi na %d. poziciji u listi\n", vrednost, i);

daLiSePojavljuje = 1;

br++;

}

pom = pom->sled;

} while (pom != glava);

}

if (!daLiSePojavljuje)

printf("%d se ne nalazi u listi\n", vrednost);

else

printf("%d je %d puta nadjen u listi\n\n",vrednost,br);

}

Функција сортира листу.

void SortirajListu(P\_ELEMENT glava) {

int pom;

if (glava != NULL) {

P\_ELEMENT predhodni;

P\_ELEMENT tekuci;

predhodni = glava;

tekuci = glava->sled;

do {

do {

if (predhodni->info < tekuci->info) {

pom = predhodni->info;

predhodni->info = tekuci->info;

tekuci->info = pom;

}

tekuci = tekuci->sled;

} while (tekuci != glava);

predhodni = predhodni->sled;

} while (predhodni != glava);

}

printc("Lista je sortirana\n\n",S\_AQUA);

}

Функција обрће листу (*reverse*).

void ObrniListu(P\_ELEMENT\* glava) {

if (\*glava != NULL) {

P\_ELEMENT predhodni = \*glava;

P\_ELEMENT pom = \*glava;

P\_ELEMENT tekuci = (\*glava)->sled;

predhodni->sled = predhodni;

predhodni->pred = predhodni;

while (tekuci != \*glava)

{

pom = tekuci->sled;

tekuci->sled = predhodni;

predhodni->pred = tekuci;

(\*glava)->sled = tekuci;

tekuci->pred = \*glava;

predhodni = tekuci;

tekuci = pom;

}

\*glava = predhodni;

}

printc("Lista je obrnuta\n\n", S\_AQUA);

}

Функција избацује дупликате из листе. Упоређујемо вредности и бришемо дупликате.

void izbaciDuplikate(P\_ELEMENT\* glava) {

P\_ELEMENT pom;

P\_ELEMENT pom2;

if (\*glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

pom = \*glava;

pom2 = (\*glava)->sled;

do {

do {

if (pom!=pom2 && pom->info == pom2->info) {

obrisiElementSaVrednoscu(glava, pom->info);

pom = \*glava;

pom2 = (\*glava)->sled;

}

pom2 = pom2->sled;

} while (pom2 != \*glava);

pom = pom->sled;

} while (pom != \*glava);

printc("Duplikati izbaceni\n\n", S\_AQUA);

}

Функција исписује фреквенцију свих елемената који се појављују у листи.

void frekvencija(P\_ELEMENT glava) {

P\_ELEMENT kopija = NULL;

kopirajListu(&glava, &kopija);

P\_ELEMENT pom;

P\_ELEMENT pom2;

pom = kopija;

pom2 = kopija;

int vrednost = 0;

int i = 0;

if (glava == NULL) {

printc("Lista je prazna!\n\n", S\_CRVENA);

return;

}

do {

pom = kopija;

do {

if (pom == NULL)

return;

vrednost = pom->info;

if (pom->info == pom2->info) {

i++;

}

pom2 = pom2->sled;

} while (pom2 != kopija);

printf("%d je nalazi na %d mesta u listi\n",vrednost, i);

obrisiSveElementeSaVrednoscu(&kopija, pom->info);

pom2 = kopija;

i = 0;

pom = pom->sled;

} while (pom != kopija);

printf("\n");

}

Функција копира листу1 у листу2. Прво бришемо листу2, затим пролазећи кроз листу1 додајемо јој елементе на крај.

void kopirajListu(P\_ELEMENT\* glava1, P\_ELEMENT\* glava2) {

obrisiCeluListu(glava2);

P\_ELEMENT tekuci = \*glava1;

if (\*glava1 != NULL) {

do {

dodajElementNaKrajListe(glava2, tekuci->info);

tekuci = tekuci->sled;

} while (tekuci != \*glava1);

}

}

Функција пореди две листе. Користимо функцију за разлику, ако нема разлике следи да су исте.

int uporediListe(P\_ELEMENT glava1, P\_ELEMENT glava2) {

P\_ELEMENT glava3 = NULL;

razlika(glava1, glava2, &glava3);

int n = duzinaListe(glava3);

if (!n) {

return 1;

}

else {

return 0;

}

}

Функција прави листу3 која представља унију листе1 и листе2. Пролазимо кроз листу1, па затим кроз листу2 и додајемо њихове елементе на крај листе3.

void unija(P\_ELEMENT glava1, P\_ELEMENT glava2,P\_ELEMENT\* glava3) {

P\_ELEMENT pom = glava1;

P\_ELEMENT pom2 = glava2;

if (glava1 == NULL && glava2 == NULL) {

\*glava3 = NULL;

return;

}

if (glava1 == NULL) {

do {

dodajElementNaKrajListe(glava3, pom2->info);

pom2 = pom2->sled;

} while (pom2 != glava2);

return;

}

else if (glava2 == NULL) {

do {

dodajElementNaKrajListe(glava3, pom->info);

pom = pom->sled;

} while (pom != glava1);

return;

}

else {

do {

dodajElementNaKrajListe(glava3, pom->info);

pom = pom->sled;

} while (pom != glava1);

do {

dodajElementNaKrajListe(glava3, pom2->info);

pom2 = pom2->sled;

} while (pom2 != glava2);

return;

}

printc("Izvrsena unija\n\n",S\_AQUA);

}

Функција прави листу3 која представља пресек листе1 и листе2. Поредимо елементе две листе и оне који су једнаки убацујемо у листу3.

void presek(P\_ELEMENT glava1, P\_ELEMENT glava2, P\_ELEMENT\* glava3) {

P\_ELEMENT pom = glava1;

P\_ELEMENT pom2 = glava2;

if (glava1 == NULL || glava2 == NULL) {

\*glava3 = NULL;

return;

}

else {

do {

do {

if (pom->info == pom2->info) {

dodajElementNaKrajListe(glava3, pom->info);

}

pom2 = pom2->sled;

} while (pom2 != glava2);

pom = pom->sled;

} while (pom != glava1);

}

printc("Izvrsen presek\n\n", S\_AQUA);

}

Функција прави листу3 која представља разлику листе1 и листе2. Поредимо елементе две листе и оне који се јављају у листи1, а не постоје у листи2 додајемо на крај листе3.

void razlika(P\_ELEMENT glava1, P\_ELEMENT glava2, P\_ELEMENT\* glava3) {

P\_ELEMENT pom = glava1;

P\_ELEMENT pom2 = glava2;

int daLisePojavljuje = 0;

if (glava1 == NULL || (glava1==NULL && glava2==NULL)) {

\*glava3 = NULL;

return;

}

else if (glava2 == NULL) {

do {

dodajElementNaKrajListe(glava3, pom->info);

pom = pom->sled;

} while (pom != glava1);

return;

}

else {

do {

daLisePojavljuje = 0;

do {

if (pom->info == pom2->info) {

daLisePojavljuje = 1;

break;

}

pom2 = pom2->sled;

} while (pom2 != glava2);

if (!daLisePojavljuje) {

dodajElementNaKrajListe(glava3, pom->info);

}

pom = pom->sled;

} while (pom != glava1);

return;

}

printc("Izvrsena razlika\n\n", S\_AQUA);

}

У функцији void Liste() је обједињена цела имплементација двоструко спрегнутих циркуларних листи помоћу менија.

# 5.Закључак

Двоструко спрегнуте циркуларне листе су корисна структура података са бројним применама. Њихове предности укључују ефикасно додавање и брисање елемената, двосмерну повезаност, флексибилност и циркуларну структуру. Међутим, њихова употреба захтева пажљиво разматрање и процену специфичних захтева проблема како би се донела добра одлука о одабиру одговарајуће структуре података.

# 6.Литература

https://www.codeblog.rs/clanci.php?p=tutorijal\_implementacija\_liste\_cpp

https://ciksiti.com/hr/chapters/10765-implementation-of-doubly-linked-list-c

https://www.geeksforgeeks.org/applications-advantages-and-disadvantages-of-circular-doubly-linked-list/

https://www.softwaretestinghelp.com/doubly-linked-list-in-java/