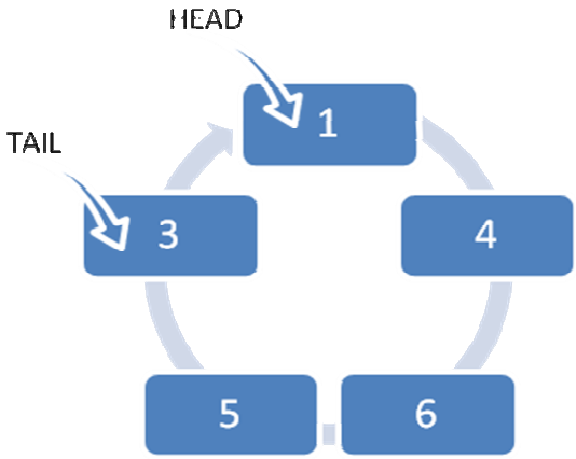
ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

Аудиториски вежби 7

Двојно поврзани и кружни листи (во C++)

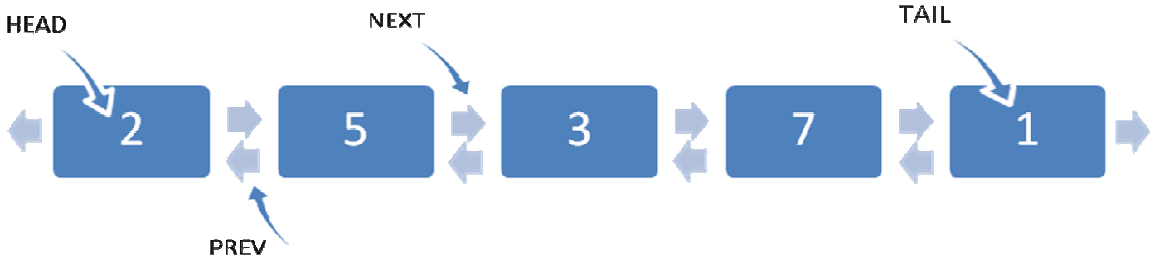
Единечно поврзана кружна листа е онаа кај која последниот јазол покажува кон првиот јазол.



Двојно поврзана листа е онаа кај која секој јазол има покажувач и кон претходникот и кон следбеникот. Ако листата е и кружна, тогаш последниот јазол покажува кон првиот јазол во листата и првиот јазол покажува кон последниот јазол во листата.

ЗАДАЧИ:

1. Да се имплементира двојно поврзана листа за која првиот јазол во лево ќе покажува на `NULL` и последниот јазол во десно ќе покажува на `NULL`. Да се креираат и функциите за поддршка на листата и во главната функција истите да се тестираат.

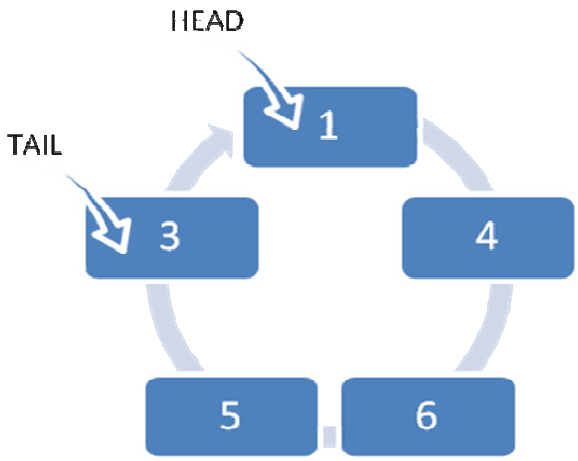



```
#include <iostream>
#include <stdlib.h>

using namespace std;

struct Jazol
{
	int info;
	Jazol *next;
	Jazol *prev;
};

struct DPLista
{
	Jazol *head;
	Jazol *tail;
	void init();
	void dodadiPrv(int el);
	void dodadiPosleden(int el);
	void brishiPrv();
	void brishiPosleden();
	void brishiLista();
```

```
};



void DPLista::init()
{
	head = tail = NULL;
}

void DPLista::dodadiPrv(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->next = head;
	pom->prev = NULL;
	head = pom;
	/* ako ovoj jazol e prviot jazol vo listata, kje bide i posleden */
	if (head->next == NULL)
		tail = head;
	else
	{
	/* ako noviot jazol ne e posleden jazol vo listata, prev poleto na prethodno
	prviot jazol mora da pokazuva kon noviot jazol*/

		(pom->next)->prev = pom;
	}
}

void DPLista::dodadiPosleden(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->next = NULL;

	/* ako listata bila prazna, ovoj jazol kje bide prv i posleden */
	if (head == NULL)
	{
		pom->prev = NULL;
		tail = head = pom;

	}
	else
	{
		pom->prev = tail;
		tail->next = pom;
		tail = pom;
	}
}

void DPLista::brishiPrv()
{
	if (head != NULL)
	{
		if (head->next == NULL)
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			Jazol *pom = head;
			head = head->next;
			head->prev = NULL;
			delete pom;
		}
```

```
	}
}

void DPLista::brishiLista()
{
	while (head != NULL)
		brishiPrv();
}



void DPLista::brishiPosleden()
{
	if (head != NULL)
	{
		if (head->next == NULL)
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			Jazol *pom = tail;
			tail = tail->prev;
			tail->next = NULL;
			delete pom;
		}
	}
}

int main()
{
	DPLista l1;
	l1.init();
	int brEl, info;
	cout << "Kolku elementi da sodrzhi listata? ";
	cin >> brEl;

	for (int i = 0; i < brEl; i++)
	{
		cin >> info;
		l1.dodadiPosleden(info);
	}

	l1.brishiPosleden();

	Jazol *dvizhi = l1.head;
	while (dvizhi != NULL)
	{
		cout << dvizhi->info << '\t';
		dvizhi = dvizhi->next;
	}
	cout << endl;

	dvizhi = l1.tail;
	while (dvizhi != NULL)
	{
		cout << dvizhi->info << '\t';
		dvizhi = dvizhi->prev;
	}
	cout << endl;

	l1.brishiLista();
	return 0;
}
```

2. Да се реализира единечно–поврзана кружна листа. Да се реализираат функциите за поддршка на листата и да се тестира листата во главната функција.
Единечно–поврзана кружна листа претставува единечно поврзана листа кај која последниот јазол покажува кон првиот јазол.



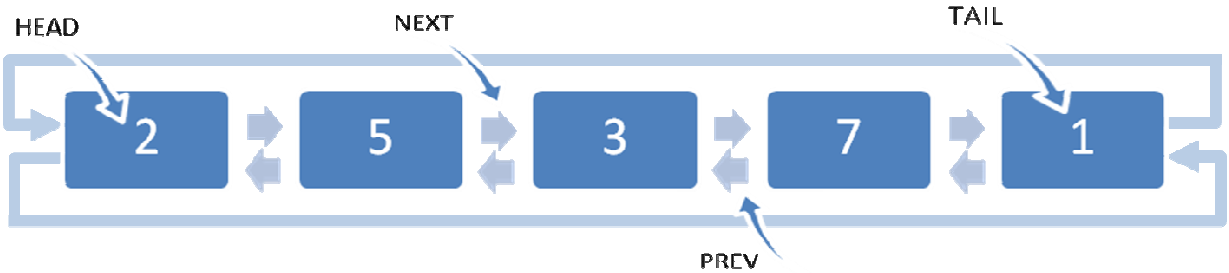

```
#include<iostream>
using namespace std;

struct Jazol
{
	int info;
	Jazol *link;
};
struct Lista
{
	Jazol *head;
	Jazol *tail; //za da go pamtime krajot
	int brel;
	void init();
	void kreirajLista(int el);
	void dodadiPrv(int el);
	void dodadiPosleden(int el);
	void brishiPrv();
	void brishiPosleden();
	void brishiLista();
	void pechati();
};

void Lista::init()
{
	head = tail = NULL; //kreirame prazna lista, bez jazli
	brel = 0;
}

void Lista::kreirajLista(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
	nov->link = nov; //jazolot pokazhuva kon sebe
	tail = head = nov;
	brel = 1;
}
```

```
void Lista::dodadiPrv(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
	if (head == NULL) //ako listata ne sodrzhi elementi
	{
		nov->link = nov; //jazolot pokazhuva kon sebe
		tail = head = nov;
	}
	else
	{
		nov->link = head;
		head = nov;
		tail->link = head;
	}
	brel++;
}

void Lista::dodadiPosleden(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
	if (head == NULL) //ako listata ne sodrzhi elementi
	{
		nov->link = nov; //jazolot pokazhuva kon sebe
		tail = head = nov;
	}
	else
	{
		tail->link = nov;
		tail = nov;
		tail->link = head;
	}
	brel++;
}

void Lista::brishiPrv()
{
	if (head->link == head)
	{
		delete head;
		head = tail = NULL;
	}
	if (head != NULL)
	{
		tail->link = head->link; //krajot na listata pokazhuva kon vtoriot jazol
		delete head; //osloboduvame memorija za prviot jazol
		head = tail->link; //pochetok stanuva vtoriot jazol
		brel--;
	}
}

void Lista::brishiPosleden()
{
	if (head->link == head)
	{
		delete head;
		head = tail = NULL;
	}
	if (head != NULL)
	{
		Jazol *pom = head;
		while (pom->link != tail)//zastanuvame na pretposledniot jazol
			pom = pom->link;
```

```
            pom->link = head; //pretposledniot jazol go povrzuvame so pochetokot na
listata
            delete tail; //osloboduvame memorija za posledniot jazol
            tail = pom; //kraj stanuva pretposledniot jazol
            brel--;
      }
}

void Lista::brishiLista()
{
      while (head != NULL)
            brishiPosleden();
}

void Lista::pechati()
{
      Jazol *pom = head;
      if (head != NULL)
            do
            {
                  cout << pom->info << '\t';
                  pom = pom->link;
            } while (pom != head);
      else { cout << "Listata e prazna"; }
      cout << endl;
}


int main()
{
      Lista l1;
      l1.init();
      l1.dodadiPosleden(4);
      l1.brishiPrv();
      l1.pechati();
      l1.dodadiPrv(2);
      l1.dodadiPosleden(5);
      l1.pechati();
      l1.brishiPrv();
      l1.pechati();
      l1.brishiPosleden();
      l1.pechati();
      l1.brishiLista();
}
```

3. Да се реализира двојно–поврзана кружна листа.

Двојно–поврзана листа е онаа кај која секој јазол има покажувач и кон претходникот и кон следбеникот. Ако листата е и кружна, тогаш последниот јазол покажува кон првиот јазол во листата и првиот јазол покажува кон последниот јазол во листата.




```
#include<iostream>
```

```
using namespace std;

struct Jazol
{
	int info;
	Jazol *next; //pokazhuvach kon sledbenikot
	Jazol *prev; //pokazhuvach kon prethodnikot
};

struct Lista
{
	Jazol *head;
	Jazol *tail; //za da go pamtime krajot
	int brel;
	void init();
	void kreirajLista(int el);
	void dodadiPrv(int el);
	void dodadiPosleden(int el);
	void brishiPrv();
	void brishiPosleden();
	void brishiNaPozicija(int poz);
	void brishiLista();
	void pechati();
};

void Lista::init()
{
	head = tail = NULL; //kreirame prazna lista, bez jazli
	brel = 0;
}

void Lista::kreirajLista(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
	nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
	tail = head = nov;
	brel = 1;
}

void Lista::dodadiPrv(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
	if (head == NULL) //ako listata ne sodrzhi elementi
	{
		nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
		tail = head = nov;
	}
	else
	{
		//formirame 4 vrski
		nov->next = head;
		head->prev = nov;
		head = nov;
		tail->next = head;
		head->prev = tail;
	}
	brel++;
}

void Lista::dodadiPosleden(int el)
{
	Jazol *nov = new Jazol;
	nov->info = el;
```

```
	if (head == NULL) //ako listata ne sodrzhi elementi
	{
		nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
		tail = head = nov;
	}
	else
	{
		//formirame 4 vrski
		tail->next = nov;
		nov->prev = tail;
		tail = nov;
		tail->next = head;
		head->prev = tail;
	}
	brel++;
}

void Lista::brishiPrv()
{
	if (head->next == head)
	{
		delete head;
		head = tail = NULL;
		brel--;
	}
	if (head != NULL)
	{
		tail->next = head->next; //krajot na listata pokazhuva kon vtoriot jazol
(head->next)
		head->next->prev = tail; //vtoriot jazol pokazhuva kon posledniot
		delete head; //osloboduvame memorija za posledniot jazol
		head = tail->next;//pochetok stanuva vtoriot jazol
		brel--;
	}
}

void Lista::brishiPosleden()
{
	if (head->next == head)
	{
		delete head;
		head = tail = NULL;
		brel--;
	}
	if (head != NULL)
	{
		tail->prev->next = head; //pretposledniot jazol (tail->prev) pokazhuva kon
prviot
		head->prev = tail->prev;//prviot pokazhuva kon pretposledniot jazol
		delete tail; //osloboduvame memorija za prviot jazol
		tail = head->prev;  //kraj stanuva pretposledniot jazol
		brel--;
	}
}

void Lista::brishiNaPozicija(int poz) //prviot element e na pozicija 1
{
	if (poz > 0 && poz <= brel)
	{
		if (poz == 1) brishiPrv();
		else if (poz == brel) brishiPosleden();
		else
		{
			Jazol *pom = head;
			for (int i = 1; i < poz; i++)
```

```
                        pom = pom->next; //go dvizhime pokazhuvachot duri ne se
pozicionira na jazolot
                                                        //koj treba da se izbrishe
                    pom->next->prev = pom->prev;
                    pom->prev->next = pom->next;
                    delete pom;
                    brel--;
                }
        }
}

void Lista::brishiLista()
{
        while (head != NULL)
                brishiPosleden();
}

void Lista::pechati()
{
        Jazol *pom = head;
        if (head != NULL)
                do
                {
                        cout << pom->info << '\t';
                        pom = pom->next;
                } while (pom != head);
        else { cout << "Listata e prazna"; }
        cout << endl;
}

int main()
{
        Lista l1;
        l1.kreirajLista(3);
        l1.dodadiPosleden(4);
        l1.dodadiPrv(2);
        l1.dodadiPosleden(5);
        l1.dodadiPosleden(6);
        l1.pechati();
        l1.brishiPrv();
        l1.pechati();
        l1.brishiPosleden();
        l1.pechati();
        l1.brishiNaPozicija(2);
        l1.pechati();
        l1.brishiNaPozicija(1);
        l1.pechati();
        l1.brishiNaPozicija(1);
        l1.pechati();
        l1.brishiLista();
}
```

4. (вежба за колоквиум) Да се напише програма за симулација на опслужувањето на клиентите во маркет. Секоја каса во маркетот оперира со посебен **ред на чекање** со максимум од 20 клиенти, каде што за клиентите се познати името и презимето. Активните каси се чуваат во **поврзана листа** во растечки редослед во однос на бројот на касата. Структурата која го опишува јазолот каса од поврзаната листа треба да содржи име на касиер, реден број на каса, редот на чекање за соодветната каса и bool индикатор дали касата е отворена.
Да се имплементираат следните функции:

- Функција за додавање на клиент. Клиентот се додава во редот на чекање во касата со најмал број на клиенти. Ако има повеќе каси со најмал број на клиенти, тогаш клиентот се додава на касата со најмал реден број.
- Фунција за активирање на нова каса. Ако активираната каса има ист реден број со веќе постоечка каса во поврзаната листа, не се брише јазолот, туку само се менуваат податоците за касиер и bool индикаторот дали касата е отворена. Инаку, таа се додава како нов јазол на соодветна позиција во поврзаната листа сортирана според редните броеви на касите.
- Функција за затворање на каса, која како аргумент го прима редниот број на касата што треба да се затвори. Кога се затвора касата, не се примаат нови клиенти во нејзиниот ред на чекање, туку само може да се опслужуваат преостанатите клиенти во редот. Откако ќе се опслужат сите клиенти во редот, касата треба да се избрише.
- Функција за бришење на каса. Оваа функција треба да го прими како аргумент редниот број на касата и го брише соодветниот јазол од поврзаната листа.
- Функција за опслужување на корисник. Оваа функција како аргумент го прима редниот број на касата во која ќе се опслужи клиент. Го опслужува првиот клиент од редот на чекање и го отстранува.
- Функции за печатење на соодветните информации за касите и редовите на клиенти.

Да се напише главна програма што ќе ги тестира сите горенаведени функционалности.

```
#include <iostream>
#include <string>
#include <stdlib.h> //zaradi exit -1

using namespace std;
#define MAX 20

struct Klient
{
	string ime, prezime;
};

struct Queue
{
	Klient array[MAX];
	int front, rear;
	void init();
	int size();
	bool isEmpty();
	bool isFull();
	void enqueue(Klient e);
	Klient dequeue();
	void print();
};

void Queue::init()
{
	front = 0;
	rear = -1;
}

int Queue::size()
```

```
{
	return rear - front + 1;
}

bool Queue::isFull()
{
	return (rear == MAX - 1);
}

bool Queue::isEmpty()
{
	return (rear == -1);
}

void Queue::enqueue(Klient e)
{
	if (isFull())
	{
		cout << "Nema mesto" << endl;
		return;
	}
	cout << "Klientot " << e.ime << " " << e.prezime << " vleguva vo redot. " << endl;
	array[++rear] = e;
}

Klient Queue::dequeue()
{
	if (isEmpty())
	{
		cout << "Redot e prazen" << endl;
		return {};
	}
	Klient pom = array[front];
	cout << "Klientot " << pom.ime << " " << pom.prezime << " e opsluzen. " << endl;
	for (int i = front; i < rear; i++)
		array[i] = array[i + 1];
	rear--;
	return pom;
}

void Queue::print()
{
	if (isEmpty())
	{
		cout << "Redot e prazen" << endl;
		return;
	}
	for (int i = front; i <= rear; i++)
		cout << "Klient " << i << ": " << array[i].ime << " " << array[i].prezime <<
endl;
}

struct InfoKasa
{
	string imeKasier;
	Queue redKlienti;
	bool otvorena;
	int brKasa;
};

struct Kasa
{
	InfoKasa info;
	Kasa *next;
};
```

```
struct KLista
{
	Kasa *head;
	void init();
	void dodadiKasa(InfoKasa el);
	void brishiKasa(int brKasa);
	Kasa *vratiKasa(int brKasa);
	void brishiLista();
	void print();
	void opsluzhiKlient(int brKasa);
	void zatvoriKasa(int brKasa);
	void dodadiKlient(Klient kl);
};

void KLista::init()
{
	head = NULL;
}

Kasa* KLista::vratiKasa(int brKasa)
{
	if (head == NULL)
		return NULL;
	else
	{
		Kasa *dvizhi = head;
		while (dvizhi != NULL)
		{
			if (dvizhi->info.brKasa == brKasa)
				return dvizhi;
			dvizhi = dvizhi->next;
		}
	}
	return NULL;
}

void KLista::dodadiKasa(InfoKasa el)
{
	if (head == NULL)
	{
		Kasa *pom = new Kasa;
		pom->info = el;
		head = pom;
		pom->next = NULL;
	}
	else
	{
		Kasa *barana = vratiKasa(el.brKasa);
		if (barana == NULL)
		{
			Kasa *dvizhi, *pred;
			dvizhi = pred = head;
			Kasa *pom = new Kasa;
			pom->info = el;

			if (el.brKasa < head->info.brKasa)
			{
				head = pom;
				pom->next = dvizhi;
			}
			else
			{
				while (dvizhi != NULL && dvizhi->info.brKasa < el.brKasa)
				{
```

```
                                pred = dvizhi;
                                dvizhi = dvizhi->next;
                        }
                        pred->next = pom;
                        pom->next = dvizhi;
                }
        }
        else
        {
                cout << "Kasata vekje postoi, smena na kasier! " << endl;
                barana->info.imeKasier = el.imeKasier;
                barana->info.otvorena = el.otvorena;
        }
    }
}

void KLista::brishiKasa(int brKasa)
{
    Kasa *brishi = vratiKasa(brKasa);
    if (brishi != NULL)
    {
        if (head == brishi)
        {
            head = brishi->next;
            delete brishi;
        }
        else
        {
            Kasa *dvizhi = head;
            while (dvizhi->next != brishi)
                dvizhi = dvizhi->next;

            dvizhi->next = brishi->next;
            delete brishi;
        }
    }
}

void KLista::brishiLista()
{
    if (head != NULL)
    {
        while (head->next != NULL)
        {
            Kasa *pom = head;
            head = head->next;
            delete pom;
        }
    }
}

void KLista::print()
{
    cout << endl << "Informacii za kasite: " << endl;
    for (Kasa *pom = head; pom != NULL; pom = pom->next)
    {
        cout << "Kasa: " << pom->info.brKasa << ", kasier: " << pom->info.imeKasier
<< endl;
        pom->info.redKlienti.print();
        cout << endl;
    }
}

void KLista::opsluzhiKlient(int brKasa)
{
```

```
	Kasa *pom = vratiKasa(brKasa);
	if (pom != NULL)
	{
		pom->info.redKlienti.dequeue();
		if (!(pom->info.redKlienti.size()) && !(pom->info.otvorena))
			brishiKasa(brKasa);
	}
}

void KLista::dodadiKlient(Klient kl)
{
	if (head != NULL)
	{
		Kasa *dvizhi = head;
		Kasa *najdobar = head;
		int minSize = MAX;
		while (dvizhi != NULL)
		{
			int qSize = dvizhi->info.redKlienti.size();
			if ((qSize < minSize) && (dvizhi->info.otvorena))
			{
				minSize = qSize;
				najdobar = dvizhi;
			}
			dvizhi = dvizhi->next;
		}
		najdobar->info.redKlienti.enqueue(kl);
	}
}

void KLista::zatvoriKasa(int brKasa)
{
	Kasa *pom = vratiKasa(brKasa);
	if (pom != NULL)
		pom->info.otvorena = false;
}

int main()
{
	Klient klienti[10] = {
{"A.","A."},{"B.","B."},{"C.","C."},{"D.","D.",},{"E.","E."},{"F.","F."},{"G.","G."},{"H.
","H."},{"I.", "I."},{"J.","J."} };
	KLista market;
	market.init();

	InfoKasa kasa1;
	kasa1.brKasa = 1;
	kasa1.imeKasier = "Olivia";
	kasa1.otvorena = true;
	kasa1.redKlienti.init();
	market.dodadiKasa(kasa1);

	InfoKasa kasa2;
	kasa2.brKasa = 5;
	kasa2.imeKasier = "Tony";
	kasa2.otvorena = true;
	kasa2.redKlienti.init();
	market.dodadiKasa(kasa2);

	for (int i = 0; i < 10; i++)
		market.dodadiKlient(klienti[i]);

	market.print();

	while (true)
```

```
	{
		cout << "Izberi edna od slednite mozhnosti: " << endl;
		cout << "d (nov klient), k (nov kasier), o (opsluzi klient), z (zatvori
kasa), p (print), q (izlez)" << endl;
		cout << "-> ";
		char izbor;
		cin >> izbor;
		if (izbor == 'd')
		{
			cout << "Izbravte da dodadete klient. " << endl;
			Klient kl;
			cout << "Ime: ";
			cin >> kl.ime;
			cout << "Prezime: ";
			cin >> kl.prezime;
			market.dodadiKlient(kl);
		}
		else if (izbor == 'k')
		{
			cout << "Izbravte da dodadete kasa. " << endl;
			InfoKasa nova;
			cout << "Ime na kasier: ";
			cin >> nova.imeKasier;
			cout << "Broj na kasa: ";
			cin >> nova.brKasa;
			nova.otvorena = true;
			nova.redKlienti.init();
			market.dodadiKasa(nova);
		}
		else if (izbor == 'o')
		{
			cout << "Izbravte da opsluzite klient. " << endl;
			int brkasa;
			cout << "Br. na kasa: ";
			cin >> brkasa;
			market.opsluzhiKlient(brkasa);
		}
		else if (izbor == 'z')
		{
			cout << "Izbravte da zatvorite kasa. " << endl;
			int brkasa;
			cout << "Br. na kasa: ";
			cin >> brkasa;
			market.zatvoriKasa(brkasa);
		}
		else if (izbor == 'p')
		{
			cout << "Izbravte da printate informacii. " << endl;
			market.print();
		}
		else if (izbor == 'q')
			break;
		else
			continue;
	}

	market.brishiLista();
	return 0;
}
```

5. Да се напише функција за работа со **единечно поврзана кружна листа**, која на влез добива една пополнета листа и цел број кој одговара на максимален број на премини преку крајот на листата. Функцијата треба да ја изминува листата според предефиниран распоред даден во глобална низа mozhniPremini. Пред изминувањето на листа се креира покажувач кој се

поставува на почетниот елемент во листата. Потоа, при изминувањето се чита вредност од mozhniPremini (при првата итерација тоа е почената вредност во mozhniPremini) и покажувачот се поместува во десно во листата за онолку елементи колку е прочитаната вредност. По застанувањето се копира јазолот на кој се застанало и новиот јазол се вметнува пред јазолот на кој се случило застанувањето. Потоа постапката се повторува со тоа што се отчитува следната вредност од mozhniPremini. Постапката прекинува кога бројот на преминувања преку крајот на листата ќе го надмине влезниот аргумент во функцијата. Изминувањето на листата да се направи во глобална функција premini.

Како дел од програмата да се напишат структури за имплементација на јазол и листа и функциите за поддршка на листата: функција за иницијализација, функции додавање и бришење на елемент на почеток од листата, додавање и бришење на елемент на крај од листата, бришење на цела листа, печатење на елементите од листата и бришење на јазол после даден јазол (адресата на јазолот после кој треба да се направи бришењето се добива на влез од функцијата).

Влез:

Lista1: |10|<->|2|<->|5|<->|7|<->|13|

mozhniPremini[10] = {2, 5, 7, 4, 8, 2, 6, 9, 3, 1}

premini(Lista1, 3)

Излез:

Lista1: |10|<->|10|<->|2|<->|5|<->|5|<->|5|<->|5|<->|7|<->|13|

Објаснување: При изминувањето се отчитува 2 од низата mozhniPremini и се стига до јазолот со вредност |5| во листата (тој јазол е за две позиции десно од почетниот јазол), по што се додава јазол со истата вредност пред него. Потоа се отчитува 5 од mozhniPremini и се стига до новиот јазол кој беше додаден со претходниот чекор (притоа е направен еден премин преку крајот на листата) и се додава уште еден јазол со вредност |5|. Постапката се повторува се додека не се направат 3 премини преку крајот на листата.

Решение:

```
#include <iostream>
using namespace std;

int mozhniPremini[10] = {2, 5, 7, 4, 8, 2, 6, 9, 3, 1};

struct jazol {
	int info;
	jazol *link;
};

struct EPKLista {
	jazol *head;
	jazol *tail;

	void init(){
		head = tail = NULL;
	}

	void dodadiPrv(int x){
		jazol *nov = new jazol;

		nov->info = x;

		if(head == NULL){
			nov->link = nov;
			head = tail = nov;
			return;
```

```
        }

        nov->link = head;
        tail->link = nov;

        head = nov;
}

void dodadiPos(int x){
        jazol *nov = new jazol;

        nov->info = x;

        if(head == NULL){
                nov->link = nov;
                head = tail = nov;
                return;
        }

        nov->link = head;
        tail->link = nov;

        tail = nov;
}

void brishiPrv(){
        if(head == NULL){
                return;
        }

        if(head == tail){
                delete head;
                head = tail = NULL;
                return;
        }

        jazol *pom = head;

        tail->link = head->link;
        head = head->link;

        delete pom;
}

void brisiPos(){
        if(head == NULL)
                return;

        if(head == tail){
                delete head;
                head = tail = NULL;
                return;
        }

        jazol *pom = tail;

        jazol *dvizi = head;

        while(dvizi->link != tail)
                dvizi = dvizi->link;

        dvizi->link = head;
        tail = dvizi;

        delete pom;
}

void brisiLista(){
```

```
		while(head != NULL)
			brishiPrv();
	}

	void pechati(){
		if(head == NULL)
			return;

		jazol *dvizi = head;

		while(dvizi != tail){
			cout<<"|"<<dvizi->info<<"|->";
			dvizi = dvizi->link;
		}

		cout<<"|"<<tail->info<<"|"<<endl;

	}

	void brisiJazol(jazol *pom){
		if (pom == NULL)
			return;

		if(pom->link == NULL){
			return;
		}

		jazol *izol = pom->link;
		if (izol == head)
			brishiPrv();
		else if (izol == tail)
			brisiPos();
		else {
			pom->link = izol->link;

			delete izol;
		}
	}

	void duplirajJazol(jazol *pom){
		if(pom == head){
			dodadiPrv(head->info);
			return;
		}
		jazol *dvizi = head;
		while(dvizi->link!=pom){
			dvizi = dvizi->link;
		}

		jazol *nov = new jazol;
		nov->info = pom->info;
		nov->link = pom;
		dvizi->link = nov;
	}
};

void premini(EPKLista &L1, int n){
	int brPrem = 0;
	int skok, i = 0;
	jazol *dvizi = L1.head;

	while(brPrem<n && i < 10){
		skok = mozhniPremini[i++];
		while(skok>0){
			if(dvizi == L1.tail)
				brPrem++;
			dvizi = dvizi->link;
			skok--;
```

```
                }
                L1.duplirajJazol(dvizi);
        }
}

int main(){
        EPKLista L1;

        L1.init();
        L1.dodadiPrv(10);
        L1.dodadiPos(2);
        L1.dodadiPos(5);
        L1.dodadiPos(7);
        L1.dodadiPos(13);

        premini(L1,3);

        L1.pechati();
        L1.brisiLista();

        return 0;
}
```

6. Да се напише функција за собирање на два броја со помош на двојно поврзани листи. Секој број е претставен преку двојно поврзана листа. Секој јазол во листата содржи едноцифрена целобројна вредност, чија тежина во бројот е одредена според редоследот на јазолот во листата. Имено, првиот јазол има најмала тежина, последниот јазол најголема. Функцијата, која ги добива листите од броевите како аргументи, треба да направи собирање на двата броја и резултатот да го смести во нова двојно поврзана листа, која ќе биде проследена во главната функција за печатење. Внимавајте на преносот.

Пример:
Треба да се соберат броевите 1457 и 489.
L1: 7 <-> 5 <-> 4 <-> 1
L2: 9 <-> 8 <-> 4
Резултат: 6 <-> 4 <-> 9 <-> 1

Решение:

```
#include <iostream>

using namespace std;

struct Jazol
{
        int info;
        Jazol *next;
        Jazol *prev;
};

struct DPLista
{
        Jazol *head;
        Jazol *tail;
        void init();
        void dodadiPrv(int el);
        void dodadiPosleden(int el);
        void brishiPrv();
        void brishiPosleden();
        void brishiLista();
        void pechati();
};
```

```
void DPLista::init()
{
	head = tail = NULL;
}

void DPLista::dodadiPrv(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->next = head;
	pom->prev = NULL;
	head = pom;
	if (head->next == NULL)
		tail = head;
}

void DPLista::dodadiPosleden(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->next = NULL;

	if (head == NULL)
	{
		pom->prev = NULL;
		tail = head = pom;

	}
	else
	{
		pom->prev = tail;
		tail->next = pom;
		tail = pom;
	}
}

void DPLista::brishiPrv()
{
	if (head != NULL)
	{
		if (head->next == NULL)
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			Jazol *pom = head;
			head = head->next;
			head->prev = NULL;
			delete pom;
		}
	}
}

void DPLista::brishiLista()
{
	while (head != NULL)
		brishiPrv();
}



void DPLista::brishiPosleden()
{
	if (head != NULL)
	{
		if (head->next == NULL)
```

```
		{
			delete head;
			head = tail = NULL;
		}
		else
		{
			Jazol *pom = tail;
			tail = tail->prev;
			tail->next = NULL;
			delete pom;
		}
	}
}

void DPLista::pechati()
{
    for(Jazol *pom = head; pom != NULL; pom = pom->next)
        cout << pom->info;
    cout << endl;
}

void soberi(DPLista l1, DPLista l2, DPLista *zbir) {
	int sum = 0, carry = 0;

	Jazol *j1 = l1.head;
	Jazol *j2 = l2.head;

	for (; j1 != NULL && j2 != NULL; j1 = j1->next, j2 = j2->next) {
		sum = j1->info + j2->info + carry;
		carry = sum / 10;
		sum = sum % 10;
		zbir->dodadiPosleden(sum);
	}

	for (; j1 != NULL; j1 = j1->next) {
		sum = j1->info + carry;
		carry = sum / 10;
		sum = sum % 10;
		zbir->dodadiPosleden(sum);
	}

	for (; j2 != NULL; j2 = j2->next) {
		sum = j2->info + carry;
		carry = sum / 10;
		sum = sum % 10;
		zbir->dodadiPosleden(sum);
	}
	if (carry!=0)
		zbir->dodadiPosleden(carry);
}

int main()
{
	DPLista l1, l2, zbir;
	l1.init();
	l2.init();
	zbir.init();

	l1.dodadiPrv(1);
	l1.dodadiPrv(4);
	l1.dodadiPrv(5);
	l1.dodadiPrv(7);

	l2.dodadiPrv(4);
	l2.dodadiPrv(8);
	l2.dodadiPrv(9);

	soberi(l1, l2, &zbir);
```

```
        cout << "Zbirot e: " << endl;
        zbir.pechati();

        l1.brishiLista();
        l2.brishiLista();
        zbir.brishiLista();
        return 0;
}
```

7. За потребите на апликација за слушање на песни, потребно е да се напише програма која работи со двојно поврзани кружни листи. Податочниот тип кој се чува за секоја музика е името на песната. За двојно поврзаните кружни листи треба да се напишат и соодветните функции за иницијализација, креирање на листа, додавање на елемент на прва позиција, додавање на елемент на последна позиција, бришење на елемент на прва позиција, бришење на елемент на последна позиција, бришење на било која позиција, бришење на листа, печатење на листа и читање на било која позиција. Потоа, треба да се овозможи спојување на две музички листи со иста должина во една нова музичка листа и тоа наизменично. Последната песна од втората листа треба да биде прва, по неа следува првата песна од првата листа, следна е претпоследната песна од втората листа, па втората песна од првата листа, за на крај претпоследната песна да биде првата песна од втората листа и последната песна да биде последната песна од првата листа.

Пример:

Lista 1	Lista 2	Lista 3
Pesna L1 0	Pesna L2 0	Pesna L2 6
Pesna L1 1	Pesna L2 1	Pesna L1 0
Pesna L1 2	Pesna L2 2	Pesna L2 5
Pesna L1 3	Pesna L2 3	Pesna L1 1
Pesna L1 4	Pesna L2 4	Pesna L2 4
Pesna L1 5	Pesna L2 5	Pesna L1 2
Pesna L1 6	Pesna L2 6	Pesna L2 3
		Pesna L1 3
		Pesna L2 2
		Pesna L1 4
		Pesna L2 1
		Pesna L1 5
		Pesna L2 0
		Pesna L1 6

Решение:

```
#include <iostream>
#include <string>
using namespace std;
struct Jazol
{
        string info;
        Jazol* next; //pokazhuvach kon sledbenikot
        Jazol* prev; //pokazhuvach kon prethodnikot
};
struct Lista
{
        Jazol* head;
        Jazol* tail; //za da go pamtime krajot
        int brel;
        void init();
        void kreirajLista(int el);
        void dodadiPrv(string el);
        void dodadiPosleden(string el);
        void brishiPrv();
        void brishiPosleden();
        void brishiNaPozicija(int poz);
        void brishiLista();
```

```
        void pechati();
        string citajNaPozicija(int poz);
};
void Lista::init()
{
        head = tail = NULL; //kreirame prazna lista, bez jazli
        brel = 0;
}
void Lista::kreirajLista(int el)
{
        Jazol* nov = new Jazol;
        nov->info = el;
        nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
        tail = head = nov;
        brel = 1;
}
void Lista::dodadiPrv(string el)
{
        Jazol* nov = new Jazol;
        nov->info = el;
        if (head == NULL) //ako listata ne sodrzhi elementi
        {
                nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
                tail = head = nov;
        }
        else
        {
                //formirame 4 vrski
                nov->next = head;
                head->prev = nov;
                head = nov;
                tail->next = head;
                head->prev = tail;
        }
        brel++;
}
void Lista::dodadiPosleden(string el)
{
        Jazol* nov = new Jazol;
        nov->info = el;
        if (head == NULL) //ako listata ne sodrzhi elementi
        {
                nov->next = nov->prev = nov; //jazolot pokazhuva kon sebe
                tail = head = nov;
        }
        else
        {
                //formirame 4 vrski
                tail->next = nov;
                nov->prev = tail;
                tail = nov;
                tail->next = head;
                head->prev = tail;
        }
        brel++;
}
void Lista::brishiPrv()
{
        if (head->next == head)
        {
                delete head;
                head = tail = NULL;
        }
        if (head != NULL)
        {
                tail->next = head->next; //krajot na listata pokazhuva kon vtoriot jazol

                        head->next->prev = tail; //vtoriot jazol pokazhuva kon posledniot
                delete head; //osloboduvame memorija za posledniot jazol
```

```
                head = tail->next;//pochetok stanuva vtoriot jazol
                brel--;
        }
}
void Lista::brishiPosleden()
{
        if (head->next == head)
        {
                delete head;
                head = tail = NULL;
        }
        if (head != NULL)
        {
                tail->prev->next = head; //pretposledniot jazol (tail->prev) pokazhuva kon

                        head->prev = tail->prev;//prviot pokazhuva kon pretposledniot jazol
                delete tail; //osloboduvame memorija za prviot jazol
                tail = head->prev; //kraj stanuva pretposledniot jazol
                brel--;
        }
}
void Lista::brishiNaPozicija(int poz) //prviot element e na pozicija 1
{
        if (poz > 0 && poz <= brel)
        {
                if (poz == 1) brishiPrv();
                else if (poz == brel) brishiPosleden();
                else
                {
                        Jazol* pom = head;
                        for (int i = 1; i < poz; i++)
                                pom = pom->next; //go dvizhime pokazhuvachot duri ne se
                        //pozicionira na jazolot
                                pom->next->prev = pom->prev;
                        pom->prev->next = pom->next;
                        delete pom;
                        brel--;
                }
        }
}
string Lista::citajNaPozicija(int poz) //prviot element e na pozicija 1
{
        if (poz >= 0 && poz < brel)
        {
                Jazol* pom = head;
                for (int i = 0; i < poz; i++) {
                        pom = pom->next;
                }
                return pom->info;
        }
}


void Lista::brishiLista()
{
        while (head != NULL)
                brishiPosleden();
}
void Lista::pechati()
{
        Jazol* pom = head;
        if (head != NULL)
                do
                {
                        cout << pom->info << '\t';
                        pom = pom->next;
                } while (pom != head);
        else { cout << "Listata e prazna"; }
        cout << endl;
```

```
}

int main()
{
	Lista l1;
	Lista l2;
	Lista l3;
	l1.kreirajLista(7);
	l2.kreirajLista(7);
	for (int i = 0; i < 7; i++) {
		l1.dodadiPosleden("Pesna L1 "+to_string(i)+"\n");
	}
	for (int i = 0; i < 7; i++) {
		l2.dodadiPosleden("Pesna L2 " + to_string(i) + "\n");
	}
	l3.kreirajLista(l2.brel + l1.brel);

	for (int i = 0; i < l1.brel; i++) {
		l3.dodadiPosleden(l1.citajNaPozicija(i));
		l3.dodadiPosleden(l2.citajNaPozicija(l1.brel - i - 1));
	}

	cout << "Lista 1\n";
	l1.pechati();
	cout << "Lista 2\n";
	l2.pechati();
	cout << "Lista 3\n";
	l3.pechati();
}
```