ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

ПОДАТОЧНИ СТРУКТУРИ И ПРОГРАМИРАЊЕ 2023/2024

Аудиториски вежби 6

Единечно поврзани листи (во C++)

Единечно поврзани листи

Низите се едноставен податочен тип во кој елементите се чуваат на последователни меморискилокации.

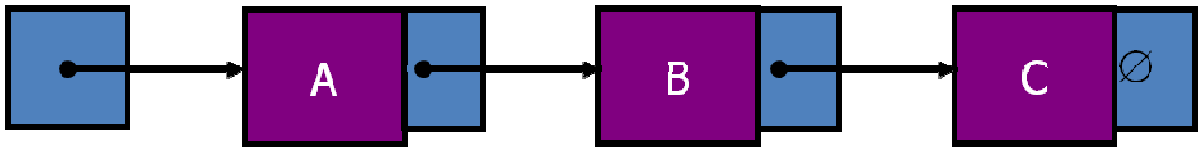
Статички дефинираните **низи** не можат да ја менуваат својата големина во текот на извршување на програмата.
Пример: `int nizaSt[12]; /*низата зафаќа простор за 12 елементи без разлика дали истите ќе се користат или не и не може да се проширува или намалува*/`

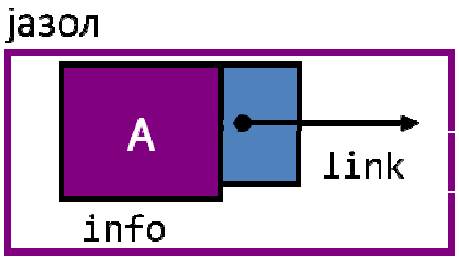
Динамички дефинираните **низи** може да ја менуваат својата големина во текот на извршување на програмата, но за да ја зголемат/намалат големината, меморискиот простор кој претходно се чувал за истите мора да се ослободи и да се алоцира нов (поголем/помал) меморискипростор во кој се копираат дел од елементите од старата низа и се додаваат/ одземаат нови елементи.
Пример: `int *nizaDin = new int[brEl];`

Од претходно споменатото може да заклучиме дека низите се едноставен начин за чување на податоци во определен редослед, но не се многу прилагодливи.
Додавањето и бришењето на елементи во низата, значи поместување на останатите елементи.
Алтернативна и меморискипооптимална варијанта на низите се поврзаните листи.

Поврзаната листа, во најопшта форма, претставува множество од јазли кои се поврзани во некаков редослед. Секој јазол освен информација (**info**), содржи и покажувач (адреса) кон јазолот со кој е поврзан (**link**). Секој јазол се алоцира динамички, независно од останатите и **не** е сместен на последователна мемориска локација во однос на неговиот претходник/ следбеник. Редоследот во поврзувањето на јазлите е определен исклучиво од покажувачите.
Додавањето, односно отстранувањето на јазол од поврзаната листа се сведува само на преповрзување на линкови, нема потреба од поместување на елементи.

Поврзаната листа во која секој јазол содржи адреса само на неговиот следбеник, се нарекува **единечно поврзана листа**. Последниот јазол во единечно поврзаната листа покажува кон `NULL`. Ако последниот јазол во единечно поврзана листа покажува кон првиот јазол, тогаш станува збор за **кружно поврзана** единечна листа.
Поврзаната листа не е јазол, туку покажувач кон јазол (head)!





ЗАДАЧИ:

1. Да се напишат потребните структури и функции за имплементирање на единечно поврзана листа и поддршка на функции за креирање на листа со еден елемент, бришење и додавање на елементи на почеток и крај на листата. Да се напише програма што ќе ги тестира сите фунции.

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

using namespace std;

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

struct EPLista
{
	Jazol *head;
	void init();
	void kreirajLista(int el);
	void dodadiPrv(int el);
	void dodadiPosleden(int el);
	void brishiPrv();
	void brishiPosleden();
	void pechatiLista();
	void brishiLista();
};

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

void EPLista::kreirajLista(int el)
{
	head = new Jazol;
	head->info = el;
	head->link = NULL;
}

void EPLista::dodadiPrv(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->link = head;
	head = pom;
}

void EPLista::dodadiPosleden(int el)
{
	Jazol *dvizhi = head, *pom = new Jazol;
	pom->info = el;
	if (head == NULL)
		head = pom;
	else
	{
		while (dvizhi->link != NULL)
			dvizhi = dvizhi->link;
		dvizhi->link = pom;
	}
	pom->link = NULL;
```

```
}



void EPLista::brishiPrv()
{
	if (head != NULL)
	{
		if ((head)->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol *pom = head;
			head = head->link;
			delete pom;
		}
	}
}

void EPLista::brishiPosleden()
{
	if (head != NULL)
	{
		if (head->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol *pom = head, *brishi;
			while (pom->link->link != NULL)
				pom = pom->link;

			brishi = pom->link;
			pom->link = NULL;
			delete brishi;
		}
	}
}

void EPLista::pechatiLista()
{
	cout << "Elementite vo listata se: ";
	for (Jazol *pom = head; pom != NULL; pom = pom->link)
		cout << pom->info << '\t';
	cout << endl;
}

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

int main()
{
	EPLista l1;
	l1.init();

	l1.dodadiPrv(1);
	l1.dodadiPosleden(7);
```

```
	l1.dodadiPosleden(8);

	l1.brishiPrv();
	cout << "Prviot jazol vo listata ima info pole " << l1.head->info << endl;
	l1.pechatiLista();

	EPLista l2;
	l2.kreirajLista(2);
	l2.dodadiPosleden(4);
	l2.dodadiPosleden(5);
	l2.brishiPosleden();
	l2.pechatiLista();

	l1.brishiLista();
	l2.brishiLista();
	l1.pechatiLista();
	l2.pechatiLista();

	return 0;
}
```

2. Да се напише функција која како аргумент добива веќе пополнета единечно поврзана листа чии јазли содржат цели броеви, а треба да формира две нови единечно поврзани листи. Во првата листа ќе се сместат јазлите кои носат информација за непарни броеви, додека во втората листа ќе се сместат јазлите кои носат информација за парните броеви.

Забелешка: Во двете резултантни листи да се врши одново алоцирање на меморија за новите јазли. Искористете ги структурите и функциите за работа со единечно поврзани листи од претходната задача.

```
void podeliSporedParnost(EPLista &L) {
	EPLista LP, LNP;
	LP.kreirajLista();
	LNP.kreirajLista();

	Jazol * dvizi = L.head;

	while (dvizi != NULL) {
		if (dvizi->info % 2 == 0) {
			LP.dodadiPosleden(dvizi->info);
		}
		else {
			LNP.dodadiPosleden(dvizi->info);
		}
		dvizi = dvizi->link;
	}

	cout << "Parnata lista sodrzi:" << endl;
	LP.pechatiLista();
	cout << "Neparnata lista sodrzi: " << endl;
	LNP.pechatiLista();

	LP.brisiLista();
	LNP.brisiLista();
}


int main() {
	EPLista Lista;

	Lista.kreirajLista(1);
	for (int i = 2; i < 7; i++)
```

```
            Lista.dodadiPosleden(i);

      Lista.pechatiLista();

      podeliSporedParnost(Lista);

      Lista.brisiLista();

      return 0;
}
```

3. Нека се дадени две единечно поврзани листи чии јазли се сортирани во растечки редослед. Да се напише функција која ќе ги спои двете листи во една листа, која треба да е сортирана во растечки редослед. Во резултантната листа одново се креираат јазли.

Забелешка: претпоставете дека во листите нема дупликати. Искористете ги структурите и функциите за работа со единечно поврзани листи од Задача 1.
За дома: Листите содржат дупликати, и секој дупликат треба да се поврзе само еднаш во резултантната листа!

```
EPLista spoiDveListi(EPLista& L1, EPLista& L2) {
      EPLista L3;
      L3.kreirajLista();

      Jazol * dv1 = L1.head;
      Jazol * dv2 = L2.head;

      while ((dv1 != NULL) && (dv2 != NULL)) {
            if (dv1->info < dv2->info) {
                  L3.dodadiPosleden(dv1->info);
                  dv1 = dv1->link;
            }
            else {
                  L3.dodadiPosleden(dv2->info);
                  dv2 = dv2->link;
            }
      }

      while (dv1 != NULL) {
            L3.dodadiPosleden(dv1->info);
            dv1 = dv1->link;
      }

      while (dv2 != NULL) {
            L3.dodadiPosleden(dv2->info);
            dv2 = dv2->link;
      }

      return L3;
};

int main() {
      EPLista L1, L2, L3;

      L1.kreirajLista(2);
      L1.brisiPos();
      L1.dodadiPrv(1);
      L1.dodadiPosleden(7);
      L1.dodadiPosleden(8);

      L2.kreirajLista(2);
```

```
        L2.dodadiPosleden(4);
        L2.dodadiPosleden(9);

        L1.pechatiLista();
        L2.pechatiLista();

        L3 = spoiDveListi(L1, L2);

        L3.pechatiLista();

        L1.brisiLista();
        L2.brisiLista();
        L3.brisiLista();

        return 0;
}
```

4. Да се напише функција која ќе ги превртува сите врски во единечно поврзана листа.

Пример:

Оригинална листа

Превртена листа


```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct Jazol
{
        int info;
        Jazol *link;
};
struct EPLista
{
        Jazol *head;
        void init();
        void kreirajLista(int el);
        void dodadiPrv(int el);
        void dodadiPosleden(int el);
        void brishiPrv();
        void brishiPosleden();
        void pechatiLista();
        void brishiLista();
};

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

```
void EPLista::kreirajLista(int el)
{
	head = new Jazol;
	head->info = el;
	head->link = NULL;
}

void EPLista::dodadiPrv(int el)
{
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->link = head;
	head = pom;
}
void EPLista::dodadiPosleden(int el)
{
	Jazol *dvizhi = head, *pom = new Jazol;
	pom->info = el;
	if (head == NULL)
		head = pom;
	else
	{
		while (dvizhi->link != NULL)
			dvizhi = dvizhi->link;
		dvizhi->link = pom;
	}
	pom->link = NULL;
}

void EPLista::brishiPrv()
{
	if (head != NULL)
	{
		if ((head)->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol *pom = head;
			head = head->link;
			delete pom;
		}
	}
}

void EPLista::brishiPosleden()
{
	if (head != NULL)
	{
		if (head->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol *pom = head, *brishi;
			while (pom->link->link != NULL)
				pom = pom->link;

			brishi = pom->link;
			pom->link = NULL;
```

```
                        delete brishi;
                }
        }
}

void EPLista::pechatiLista()
{
        cout << "Elementite vo listata se: ";
        for (Jazol *pom = head; pom != NULL; pom = pom->link)
                cout << pom->info << '\t';
        cout << endl;
}

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

void prevrti(EPLista *l1)
{
        Jazol *pom = l1->head;
        /* so pom ќе се движиме низ листата*/
        Jazol *sleden, *preth = NULL;
        /*во sleden ќе го чуваме следбеникот на pom, пред pom да го поврземе со неговиот
претходник, preth*/
        while (pom != NULL)
        {
                sleden = pom->link;
                pom->link = preth;
                preth = pom;
                pom = sleden;
        }
        l1->head = preth;
        /* сега почеток на новата листа е последниот јазол на оригиналната листа*/
}

int main()
{
        EPLista l1;
        l1.init();

        l1.dodadiPrv(1);
        l1.dodadiPosleden(7);
        l1.dodadiPosleden(8);

        l1.pechatiLista();
        prevrti(&l1);
        l1.pechatiLista();

        l1.brishiLista();
        l1.pechatiLista();

        return 0;
}
```

5. Треба да се состави игра во која победник е оној кој собрал повеќе поени помеѓу двајца играчи. Поените за секое поле се запишани во единечно поврзана листа од која секој јазол содржи цел број (поени). Во главната функција дадено е пополнување на единечно поврзаната листа со цели броеви. Исто така во главната функција дадени се внесени и други две листи единечно пополнети листи (по една за секој од двајцата играчи) и претставуваат чекори со кои ќе се движат по патеката. За секој играч се знае име, презиме, поени (на почеток секој играч има по 0 поени) и

листа со чекори (која се пополнува во главната функција). Да се креира структурата играч. После внесените информации за играчите и патеката (главната листа) повикана е функција за реализација на играта, која треба да се напише. Играта се ализира во функција igra(), која како аргументи ги добива двајцата играчи и пополнетата единечно поврзана листа (главната, односно патеката). Прво игра едниот играч се' додека не ја заврши играта, а потоа игра вториот играч, исто така се' додека не ја заврши играта. После играта кога и двајцата играчи имаат собрано одредени поени, односно после реализацијата на функцијата igra(), во главната функција се печатат името, презимето и поените на победникот. Доколку и двајцата играчи имаат ист број на поени, се печатат истите информации и за двајцата играчи.
Играта се одвива на тој начин што од листата на секој играч се чита информацијата во првиот јазол и колку што изнесува тој број, толку чекори оди напред низ листата патека. На пример, ако првиот јазол од листата на првиот играч содржи информација цел број 3, се движи три чекори во листата и застанува на третиот јазол после тековниот, потоа следниот јазол ако содржи информација цел број 2, играчот се движи уште два чекори напред во низата итн. Играта завршува за тој играч, ако играчот ја измине цела своја листа или во моментот кога јазолот од неговата листа содржи број кој е поголем од преостанатите јазли во листата патека (на пример, ако тековниот јазол од неговата листа содржи информација цел број 4, а во листата патека се наоѓа на претпоследен јазол). Притоа, на секое поле (јазол) на кој ќе застане играчот, содржи цел број што во играта претставуваат поени и се собираат за време на целата игра, се додека не заврши. Кога играчот ќе започнува со играта стои на првиот јазол и тие поени не влегуваат во вкупниот број на поени. Потоа, на ист начин започнува да игра и вториот играч.
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 и печатење на елементите од листата.

Пример:
main_list: |6| → |5| → |2| → |10| → |3| → |0| → |7| → |11|

Igrac1: Paul Kalk
list1: |2| → |1| → |2| → |1|
прв чекор: 2; поени = 0 + 2 = 2
втор чекор: 1; поени = 2 + 10 = 12
трет чекор: 2; поени = 12 + 0 = 12
четврт чекор: 1; the end! Поени = 12 + 7 = 19

Igrac2: Jamie Jones
list2: |3| → |4| → |2| → |5|
прв чекор: 3; поени = 0 + 10 = 10
втор чекор: 4; поени = 10 + 11 = 21
трет чекор: 2; game over! Поени = 21

Pobednik e Jamie Jones so 21 poeni!

Решение:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct Jazol
{
	int info;
	Jazol* link;
};
struct EPLista
{
```

```
	Jazol* head;
	void init();
	void kreirajLista(int el);
	void dodadiPrv(int el);
	void dodadiPosleden(int el);
	void brishiPrv();
	void brishiPosleden();
	void pechatiLista();
	void brishiLista();
};
struct igrac
{
	string ime;
	string prezime;
	int poeni;
	EPLista l;
};
void EPLista::init()
{
	head = NULL;
}
void EPLista::kreirajLista(int el)
{
	head = new Jazol;
	head->info = el;
	head->link = NULL;
}
void EPLista::dodadiPrv(int el)
{
	Jazol* pom = new Jazol;
	pom->info = el;
	pom->link = head;
	head = pom;
}
void EPLista::dodadiPosleden(int el)
{
	Jazol* dvizhi = head, * pom = new Jazol;
	pom->info = el;
	if (head == NULL)
		head = pom;
	else
	{
		while (dvizhi->link != NULL)
			dvizhi = dvizhi->link;
		dvizhi->link = pom;
	}
	pom->link = NULL;
}
void EPLista::brishiPrv()
{
	if (head != NULL)
	{
		if ((head)->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol* pom = head;
			head = head->link;
			delete pom;
		}
	}
}
void EPLista::brishiPosleden()
{
	if (head != NULL)
```

```
	{
		if (head->link == NULL)
		{
			delete head;
			head = NULL;
		}
		else
		{
			Jazol* pom = head, * brishi;
			while (pom->link->link != NULL)
				pom = pom->link;
			brishi = pom->link;
			pom->link = NULL;
			delete brishi;
		}
	}
}
void EPLista::pechatiLista()
{
	cout << "Elementite vo listata se: ";
	for (Jazol* pom = head; pom != NULL; pom = pom->link)
		cout << pom->info << '\t';
	cout << endl;
}
void EPLista::brishiLista()
{
	while (head != NULL)
		brishiPosleden();
}
void igra(igrac& i1, igrac& i2, EPLista l)
{
	Jazol* pom = l.head;
	Jazol* pom1 = i1.l.head;
	Jazol* pom2 = i2.l.head;
	int value;
	bool prodolzi = 1;

	//za igrac 1
	while (prodolzi && pom1 != NULL)
	{
		value = pom1->info;
		//value--;
		while (value && pom->link != NULL)
		{
			pom = pom->link;
			value--;
		}
		if (!value)
		{
			i1.poeni += pom->info;
			cout << "Prv igrac: " << i1.poeni << endl;
		}
		else
		{
			prodolzi = 0;
			cout << "Game over!" << endl;
		}
		pom1 = pom1->link;
	}
	if (prodolzi)
		cout << "The end" << endl;

	//za igrac 2
	prodolzi = 1;
	pom = l.head;
	while (prodolzi && pom2 != NULL)
	{
		value = pom2->info;
```

```
                //value--;
                while (value && pom->link != NULL)
                {
                        pom = pom->link;
                        value--;
                }
                if (!value)
                {
                        i2.poeni += pom->info;
                        cout << "Vtor igrac: " << i2.poeni << endl;
                }
                else
                {
                        prodolzi = 0;
                        cout << "Game over!" << endl;
                }
                pom2 = pom2->link;
        }
        if (prodolzi)
                cout << "The end" << endl;
}

int main()
{
        EPLista l;
        l.init();
        l.kreirajLista(6);
        l.dodadiPosleden(5);
        l.dodadiPosleden(2);
        l.dodadiPosleden(10);
        l.dodadiPosleden(3);
        l.dodadiPosleden(0);
        l.dodadiPosleden(7);
        l.dodadiPosleden(11);
        igrac i1;
        igrac i2;
        i1.ime = "Paul";
        i1.prezime = "Kalk";
        i1.poeni = 0;
        i2.ime = "Jamie";
        i2.prezime = "Jones";
        i2.poeni = 0;

        i1.l.init();
        i1.l.kreirajLista(2);
        i1.l.dodadiPosleden(1);
        i1.l.dodadiPosleden(2);
        i1.l.dodadiPosleden(1);
        i1.l.pechatiLista();
        i2.l.init();
        i2.l.kreirajLista(3);
        i2.l.dodadiPosleden(4);
        i2.l.dodadiPosleden(2);
        i2.l.dodadiPosleden(5);
        i2.l.pechatiLista();
        igra(i1, i2, l);
        if (i1.poeni > i2.poeni)
                cout << "Pobednik e " << i1.ime << " " << i1.prezime << " so " << i1.poeni <<
"poeni";
        if (i2.poeni > i1.poeni)
                cout << "Pobednik e " << i2.ime << " " << i2.prezime << " so " << i2.poeni <<
"poeni";
        if (i1.poeni == i2.poeni)
        {
                cout << "Dvajcata igraci imaat ist broj na poeni: " << i1.ime << " " <<
i1.prezime << " so " << i1.poeni << "poeni";
                cout << " i " << i2.ime << " " << i2.prezime << " so " << i2.poeni << "poeni";
        }
```

```
	i1.l.brishiLista();
	i2.l.brishiLista();
	l.brishiLista();
	return 0;
}
```

6. По испитот по ПСП потребно е да се напише функција со која ќе се издвојат студентите кои освоиле поголем број на поени од просекот на целата група. Студентите се елементи на единечно поврана листа и притоа за секој студент (инфо поле) во листата се чува информација за индексот на студентот и поените кои ги освоил. Функцијата од оваа единечно поврзана листа треба да го одреди просекот на групата, потоа од листата студенти да ги издвои студентите чии поени се повисоки од просекот на групата и да ги запише во нова единечно поврзана листа, која ќе биде проследена во главната функција за печатење.

Пример:
Студенти:
|12_2020, 92| -> |25_2020, 53| -> |18_2020, 10| -> |156_2020, 100| -> |93_2020, 71| -> |43_2020, 43| -> |20_2020, 25|
Просек: 56
Издвоени студенти: |12_2020, 92| -> |156_2020, 100| -> |93_2020, 71|

Решение:

```
#include <iostream>
#include <stdlib.h>
using namespace std;

struct Student {
	string indeks;
	int poeni;
};

struct Jazol {
	Student info;
	Jazol *link;
};

struct EPLista {
	Jazol *head;
	void init(){head = NULL;};
	void kreirajLista(Student el);
	void dodadiPrv(Student el);
	void dodadiPosleden(Student el);
	void brishiPrv();
	void brishiPosleden();
	void pechatiLista();
	void brishiLista();
};

void EPLista::kreirajLista(Student el) {
	head = new Jazol;
	head->info = el;
	head->link = NULL;
}

void EPLista::dodadiPrv(Student el) {
	Jazol *pom = new Jazol;
	pom->info = el;
	pom->link = head;
	head = pom;
}
```

```
void EPLista::dodadiPosleden(Student el) {
	Jazol *dvizhi = head, *pom = new Jazol;
	pom->info = el;
	if (head == NULL)
		head = pom;
	else {
		while (dvizhi->link != NULL)
			dvizhi = dvizhi->link;
		dvizhi->link = pom;
	}
	pom->link = NULL;
}

void EPLista::brishiPrv() {
	if (head != NULL) {
		if ((head)->link == NULL) {
			delete head;
			head = NULL;
		} else {
			Jazol *pom = head;
			head = head->link;
			delete pom;
		}
	}
}

void EPLista::brishiPosleden() {
	if (head != NULL) {
		if (head->link == NULL) {
			delete head;
			head = NULL;
		} else {
			Jazol *pom = head, *brishi;
			while (pom->link->link != NULL)
				pom = pom->link;

			brishi = pom->link;
			pom->link = NULL;
			delete brishi;
		}
	}
}

void EPLista::pechatiLista() {
	cout << "Elementite vo listata se: " << endl;
	for (Jazol *pom = head; pom != NULL; pom = pom->link)
		cout << pom->info.indeks << '  ' << pom->info.poeni << endl;
	cout << endl;
}

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

void selektirajStudenti(EPLista l1, EPLista *l2) {
	Jazol *pom = l1.head;
	int sum = 0, count = 0, prosek;
	while (pom != NULL) {
		sum += pom->info.poeni;
		count += 1;
		pom = pom->link;
	}
	prosek = sum / count;
	cout << prosek << endl;

	pom = l1.head;
	while (pom != NULL) {
```

```
                if (pom->info.poeni > prosek) {
                        l2->dodadiPrv(pom->info);
                }
                pom = pom->link;
        }
}

int main() {
        EPLista l1, l2;
        Student s;

        l1.init();
        l2.init();

        s.indeks = "12_2020";
        s.poeni = 92;
        l1.dodadiPrv(s);
        s.indeks = "25_2020";
        s.poeni = 53;
        l1.dodadiPrv(s);
        s.indeks = "18_2020";
        s.poeni = 10;
        l1.dodadiPrv(s);
        s.indeks = "156_2020";
        s.poeni = 100;
        l1.dodadiPrv(s);
        s.indeks = "93_2020";
        s.poeni = 71;
        l1.dodadiPrv(s);
        s.indeks = "43_2020";
        s.poeni = 43;
        l1.dodadiPrv(s);
        s.indeks = "20_2020";
        s.poeni = 25;
        l1.dodadiPrv(s);

        selektirajStudenti(l1, &l2);

        l1.pechatiLista();
        l2.pechatiLista();

        l1.brishiLista();
        l2.brishiLista();

        return 0;
}
```

Дополнување: Задача 4, второ решение

```
void prevrti(EPLista *l1)
{
        Jazol *pom2 = NULL;
        Jazol *pom = l1->head;
        while (pom != NULL)
        {
                l1->head = l1->head->link;
                pom->link = pom2;
                pom2 = pom;
                pom = l1->head;
        }
        l1->head = pom2;
}
```