Звіт

Автор: Момот Р. КІТ-119а

Дата: 25.04.2020

ЛАБОРАТОРНА РОБОТА № 8. СПИСКИ

Мета: одержати навички та закріпити знання при виконанні операцій на списках.

Індивідуальне завдання

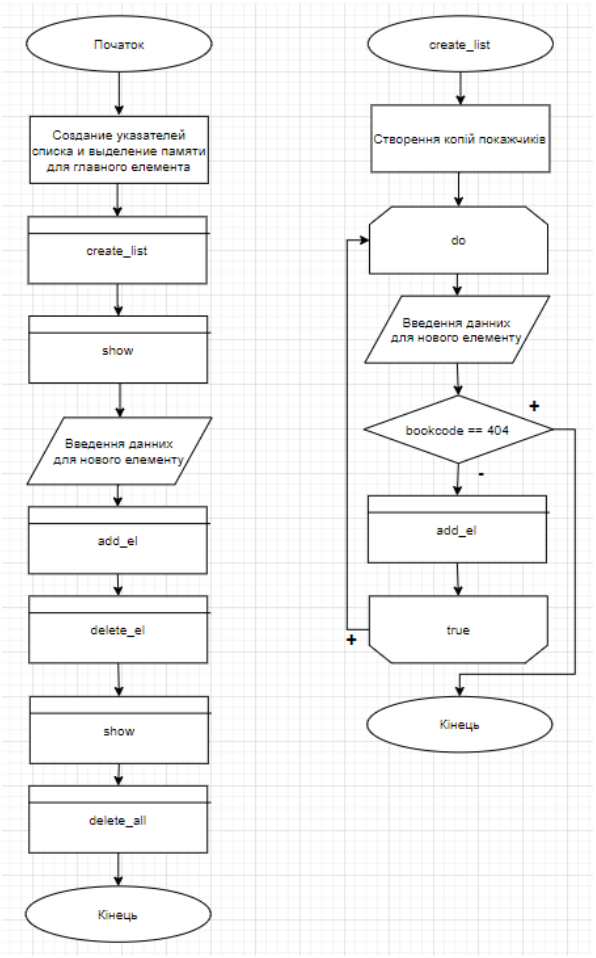
Розробити програму, що створює мультисписок. Передбачити такі функції:

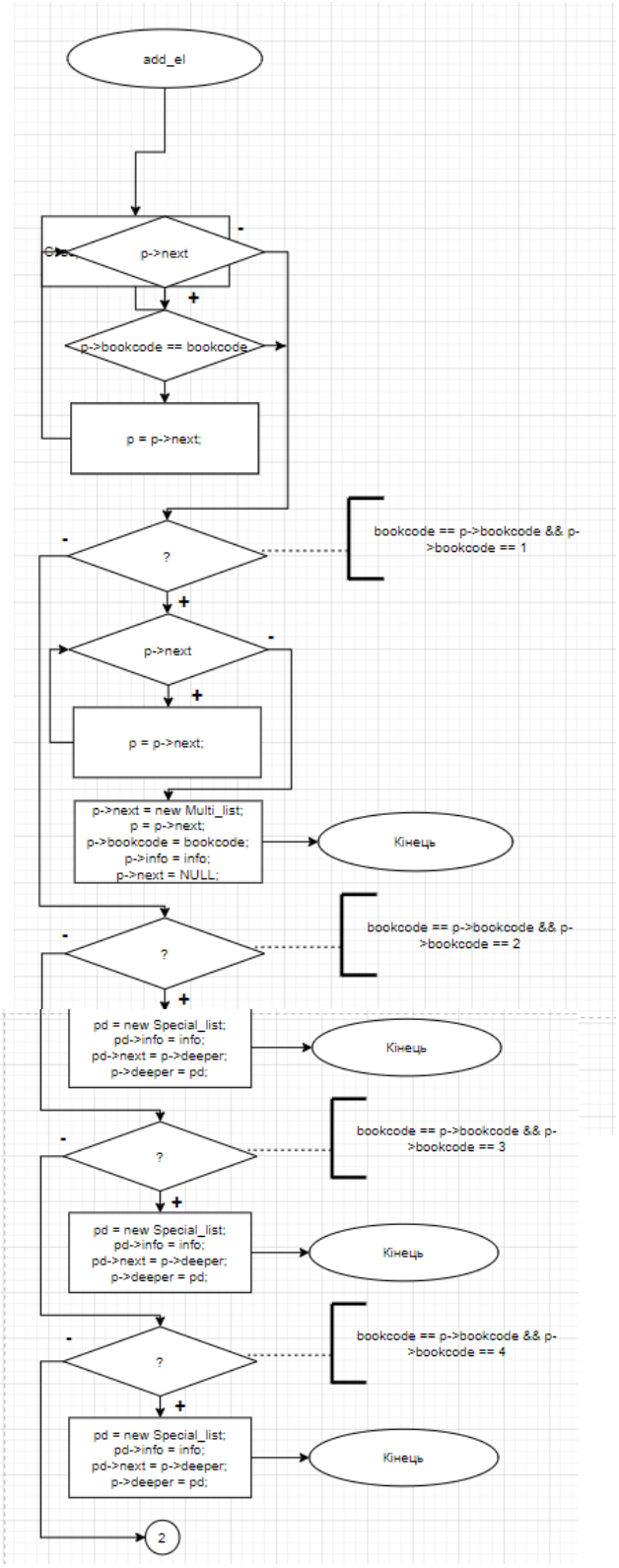
- додавання елементів у список та підсписки;
- видалення елементів з головного списку та підсписків (при видаленні елемента з одного із списків цей елемент не видаляється з пам’яті. І тільки, коли елемент не входить ні до одного із списків, він видаляється з пам’яті);
- видача вмісту списків та підсписків у консоль;
- видалення списків.

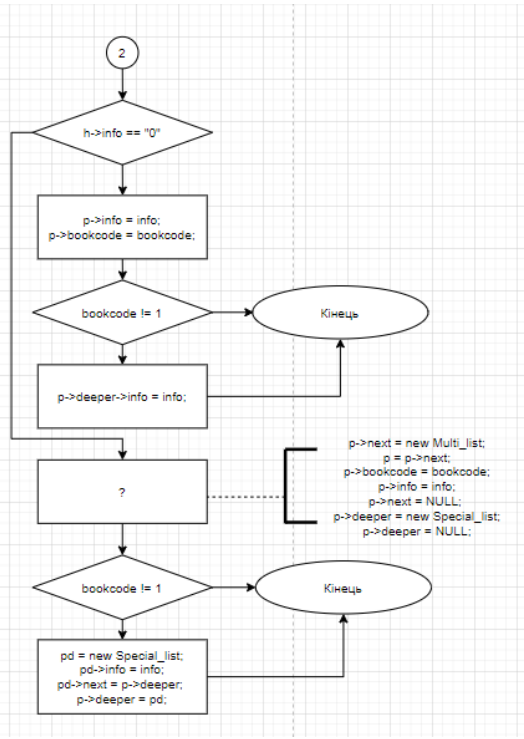
Завдання для варіантів 12-15 обрати у табл. 8.2 згідно із своїм номером у журналі групи. Перелік властивостей обрати за своїм розсудом.

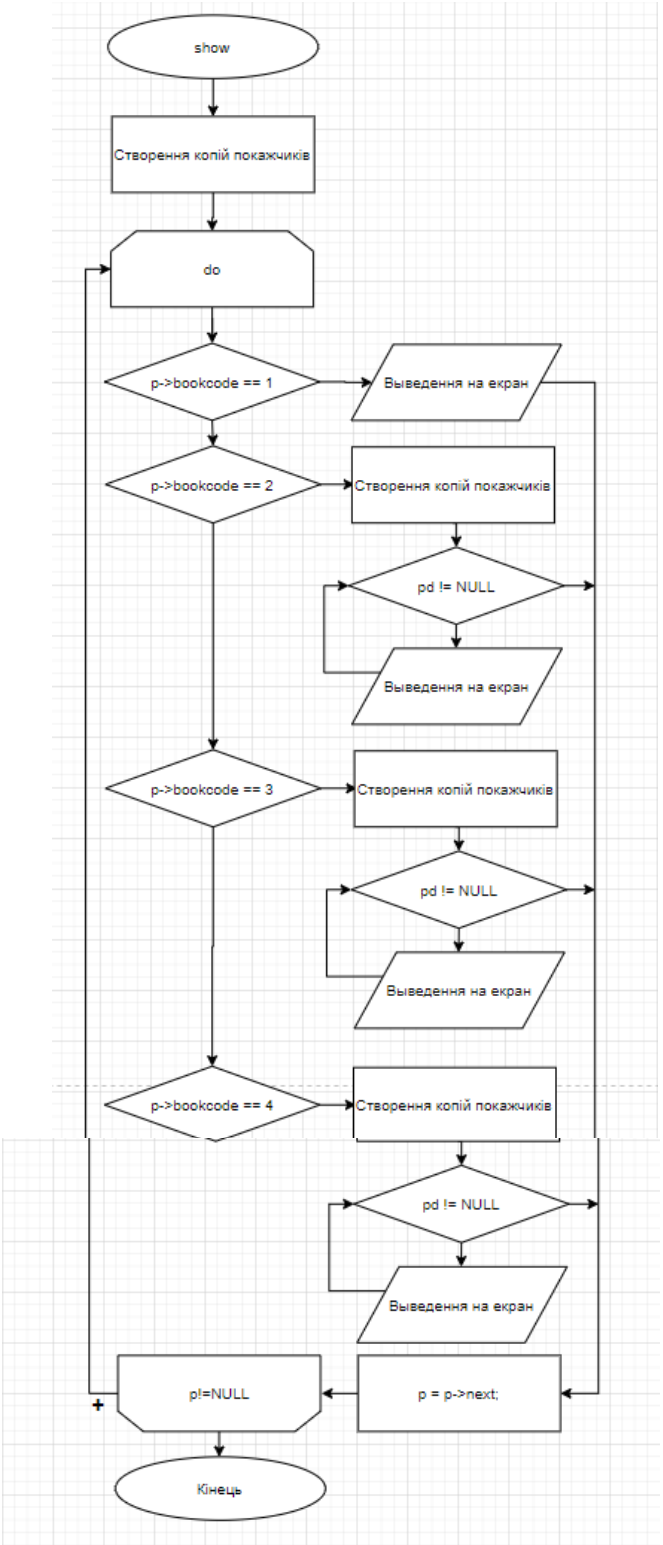
N	Об’єкт	Списки
13	Студенти інститута	1) усі студенти інститута; 2) студенти - іноземці; 3) студенти першого року навчання; 4) студенти магістри.

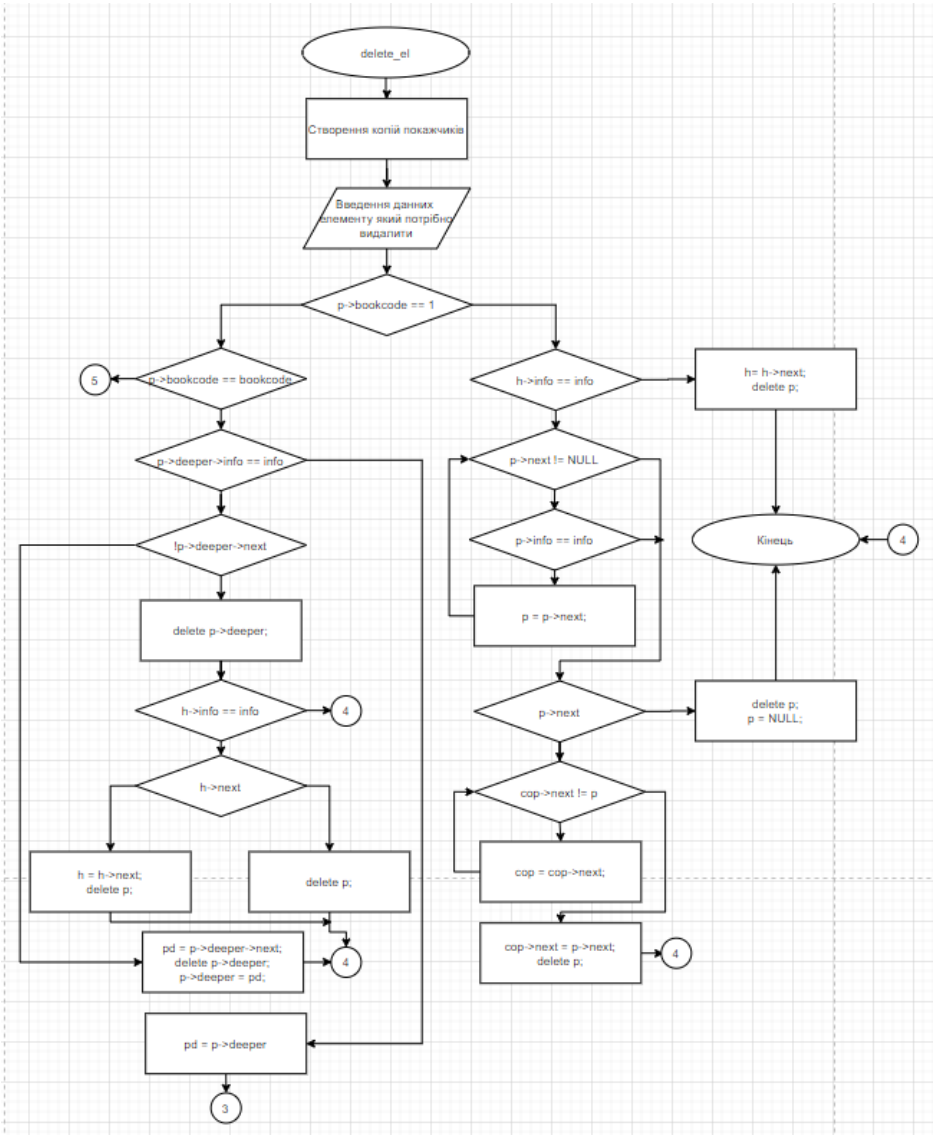
Блок-схема алгоритму програми

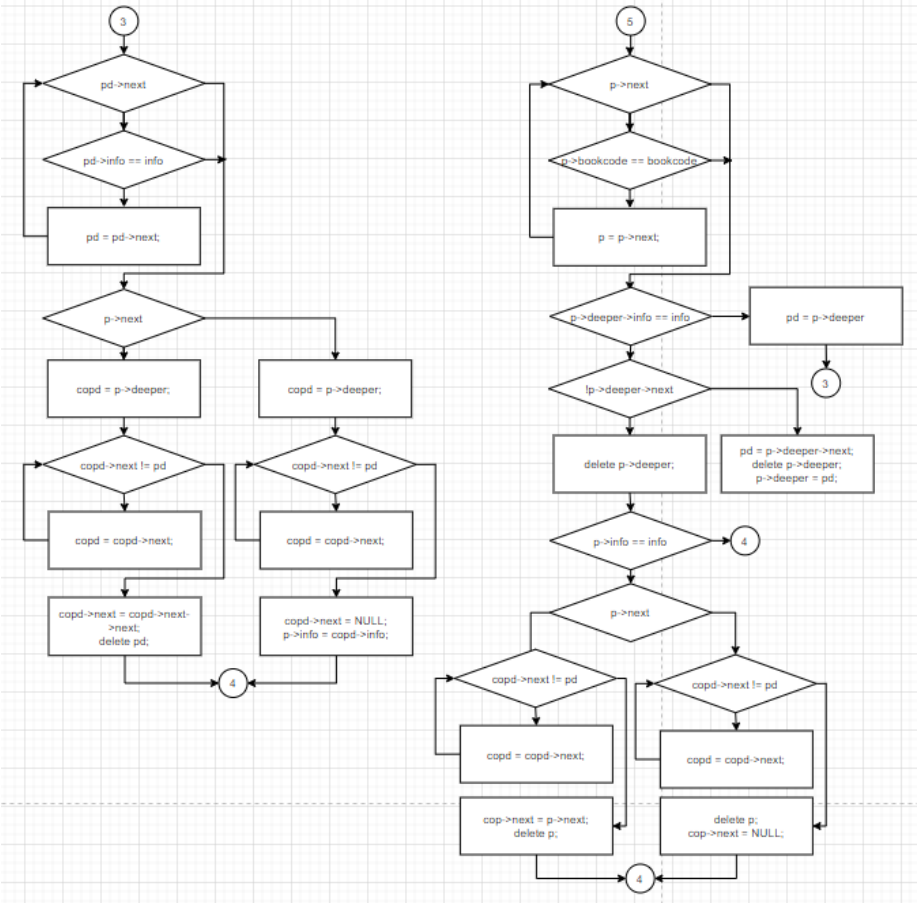


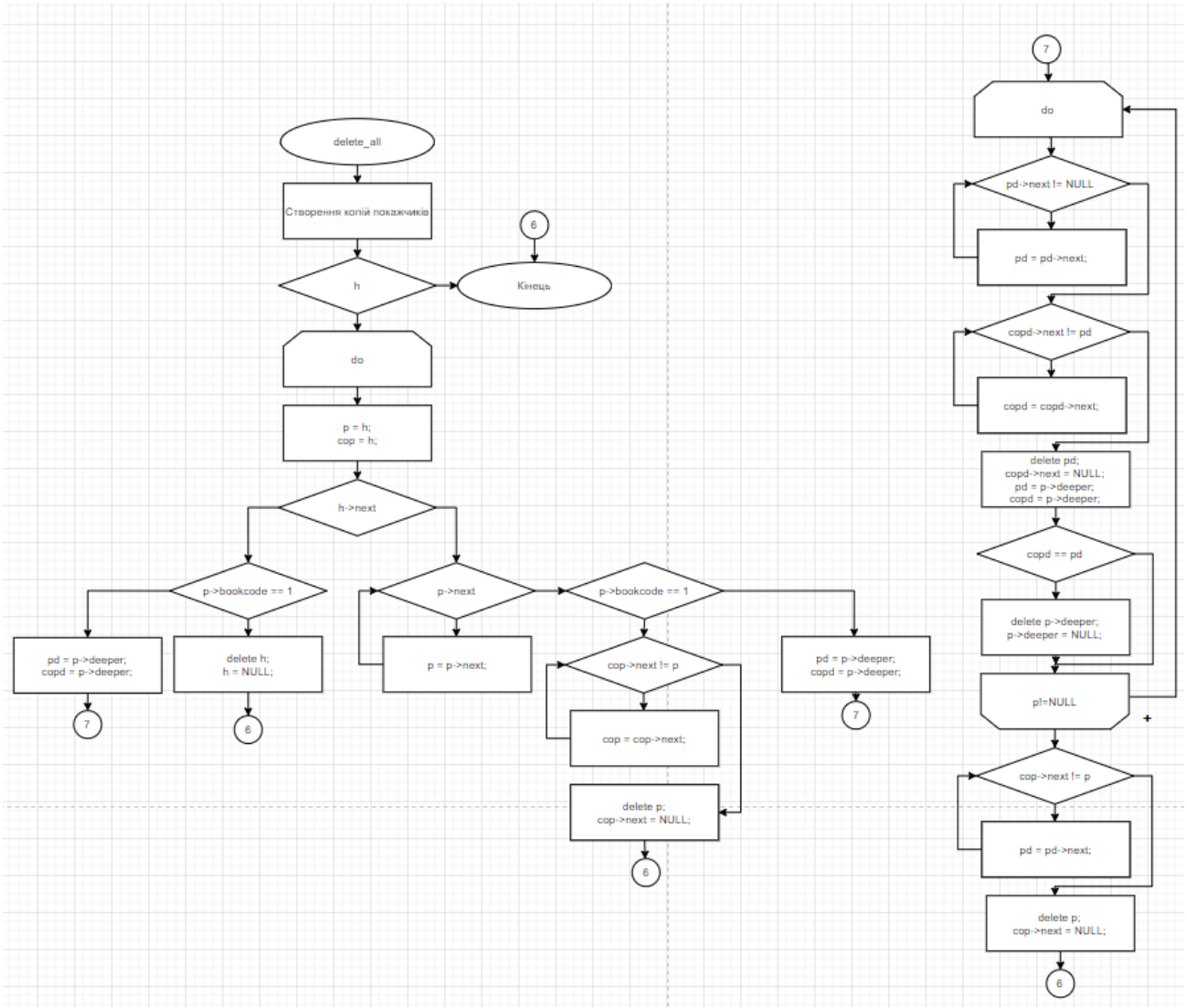



2
h->info == "0"
p->info = info;
p->bookcode = bookcode;
bookcode != 1
Кінець
p->deeper->info = info;
?
p->next = new Multi_list;
p = p->next;
p->bookcode = bookcode;
p->info = info;
p->next = NULL;
p->deeper = new Special_list;
p->deeper = NULL;
bookcode != 1
Кінець
pd = new Special_list;
pd->info = info;
pd->next = p->deeper;
p->deeper = pd;


show
Створення копій покажчиків
do
p->bookcode == 1
Виведення на екран
p->bookcode == 2
Створення копій покажчиків
pd != NULL
Виведення на екран
p->bookcode == 3
Створення копій покажчиків
pd != NULL
Виведення на екран
p->bookcode == 4
Створення копій покажчиків
pd != NULL
Виведення на екран
p!=NULL
p = p->next;
Кінець


delete_el
Створення копій покажчиків
Введення данних елементу який потрібно видалити
p->bookcode == 1
p->bookcode == bookcode
5
h->info == info
h= h->next;
delete p;
p->deeper->info == info
p->next != NULL
Кінець
4
!p->deeper->next
p->info == info
delete p->deeper;
p = p->next;
h->info == info
4
p->next
delete p;
p = NULL;
h->next
cop->next != p
h = h->next;
delete p;
delete p;
cop = cop->next;
pd = p->deeper->next;
delete p->deeper;
p->deeper = pd;
4
cop->next = p->next;
delete p;
4
pd = p->deeper
3


3
pd->next
pd->info == info
pd = pd->next;
p->next
copd = p->deeper;
copd = p->deeper;
copd->next != pd
copd->next != pd
copd = copd->next;
copd = copd->next;
copd->next = copd->next->next;
delete pd;
copd->next = NULL;
p->info = copd->info;
4
5
p->next
p->bookcode == bookcode
p = p->next;
p->deeper->info == info
pd = p->deeper
!p->deeper->next
3
delete p->deeper;
pd = p->deeper->next;
delete p->deeper;
p->deeper = pd;
p->info == info
4
p->next
copd->next != pd
copd->next != pd
copd = copd->next;
copd = copd->next;
cop->next = p->next;
delete p;
delete p;
cop->next = NULL;
4


delete_all
Створення копій покажчиків
h
Кінець
6
do
p = h;
cop = h;
h->next
p->bookcode == 1
pd = p->deeper;
copd = p->deeper;
delete h;
h = NULL;
7
6
p->next
p = p->next;
p->bookcode == 1
cop->next != p
cop = cop->next;
delete p;
cop->next = NULL;
6
pd = p->deeper;
copd = p->deeper;
7
7
do
pd->next != NULL
pd = pd->next;
copd->next != pd
copd = copd->next;
delete pd;
copd->next = NULL;
pd = p->deeper;
copd = p->deeper;
copd == pd
delete p->deeper;
p->deeper = NULL;
p!=NULL
cop->next != p
pd = pd->next;
delete p;
cop->next = NULL;
6

Текст програми

```
#include <iostream>
#include <locale>
#include <string>
#include <iomanip>
#include <exception>
using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::getline;
using std::setw;
using std::exception;

typedef struct List
{
	string info;
	List* next;
}List;

typedef struct MultiList
{
	int studcode;
	string info;

	MultiList* next;
	List* deeper;
}MultiList;

void AddStudent(MultiList*, List*, string, int);
void CreateList(MultiList*);
MultiList* DeleteStudent(MultiList*);
void DeleteList(MultiList*);
void OutputData(MultiList*);

int main()
{
	setlocale(LC_ALL, "Russian");
	MultiList* head = new MultiList;
	head->deeper = new List;
	head->next = NULL;
	head->deeper->next = NULL;
	head->info = "0";
	string data, data2;
	int studcode;

	CreateList(head);
	OutputData(head);

	cout << "\nДобавление нового елемента списка\n\nВведите код студента\n";
	cout << "1) Обычный студент\n2) Студенты-иностранцы\n3) Студенты первокурсники\n";
	cout << "4) Магистры\n======================\n";
	cout << "Ваш выбор: ";
	cin >> studcode;
	cout << "Введите имя студента: ";
	cin >> data;
	getline(cin, data2);
	data = data + data2;

	AddStudent(head, head->deeper, data, studcode);
	OutputData(head);
	head = DeleteStudent(head);
	OutputData(head);
	DeleteList(head);
```

```
	return 0;
}

void OutputData(MultiList* h)
{
	MultiList* p = h;
	List* pDeeper = h->deeper;
	int i = 0;
	cout << endl;
	do
	{
		i = 0;
		if (p->studcode == 1)
		{
			cout << "Обычный студент " << p->info << setw(20) << "|| Код: " << p-
>studcode << endl;
		}

		else if (p->studcode == 2)
		{
			cout << "\nИностранцы" << endl;
			pDeeper = p->deeper;
			while (pDeeper != NULL)
			{

				if (i == 0)
				{
					cout << "Студент " << p->info << setw(20) << "|| Код: " << p-
>studcode << endl;
					i = 1;
				}
				cout << "  Студент: " << pDeeper->info << endl;
				pDeeper = pDeeper->next;

			}
		}
		else if (p->studcode == 3)
		{
			cout << "\nПервокурсники" << endl;
			pDeeper = p->deeper;
			while (pDeeper != NULL)
			{
				if (i == 0)
				{
					cout << "Студент " << p->info << setw(20) << "|| Код: " << p-
>studcode << endl;
					i = 1;
				}
				cout << "  Студент " << pDeeper->info << endl;
				pDeeper = pDeeper->next;

			}
		}
		else if (p->studcode == 4)
		{
			cout << "\nМагистры" << endl;
			pDeeper = p->deeper;
			while (pDeeper != NULL)
			{
				if (i == 0)
				{
					cout << "Студент " << p->info << setw(20) << "|| Код: " << p-
>studcode << endl;
					i = 1;
				}
```

```
                        cout << "  Студент " << pDeeper->info << endl;
                        pDeeper = pDeeper->next;
                    }
                }
                p = p->next;
        } while (p != NULL);
}
void DeleteList(MultiList* h)
{
        MultiList* p = h, * pCopy;
        List* pDeeper, * dCopy;

        if (h)
                do
                {
                        p = h;
                        pCopy = h;
                        if (h->next)
                        {
                                while (p->next) p = p->next;
                                if (p->studcode == 1)
                                {
                                        while (pCopy->next != p) pCopy = pCopy->next;
                                        delete p;
                                        pCopy->next = NULL;
                                }
                                else
                                {
                                        pDeeper = p->deeper;
                                        dCopy = p->deeper;
                                        do
                                        {
                                                if (dCopy == pDeeper)
                                                {
                                                        delete p->deeper;
                                                        p->deeper = NULL;
                                                }
                                                else
                                                {
                                                        pDeeper = p->deeper;
                                                        dCopy = p->deeper;
                                                        while (pDeeper->next != NULL)
                                                        {
                                                                pDeeper = pDeeper->next;
                                                        }
                                                        while (dCopy->next != pDeeper)
                                                                dCopy = dCopy->next;
                                                        delete pDeeper;
                                                        dCopy->next = NULL;
                                                        pDeeper = p->deeper;
                                                        dCopy = p->deeper;
                                                }
                                        } while (p->deeper);
                                        while (pCopy->next != p) pCopy = pCopy->next;
                                        delete p;
                                        pCopy->next = NULL;
                                }
                        }
                        else
                        {
                                if (p->studcode == 1)
                                {
                                        delete h;
                                        h = NULL;
                                }
                                else
```

```
                        {
                                pDeeper = p->deeper;
                                dCopy = p->deeper;
                                do
                                {
                                        if (dCopy == pDeeper)
                                        {
                                                delete p->deeper;
                                                p->deeper = NULL;
                                        }
                                        else
                                        {
                                                pDeeper = p->deeper;
                                                dCopy = p->deeper;
                                                while (pDeeper->next != NULL)
                                                {
                                                        pDeeper = pDeeper->next;
                                                }
                                                while (dCopy->next != pDeeper)
                                                        dCopy = dCopy->next;
                                                delete pDeeper;
                                                dCopy->next = NULL;
                                                pDeeper = p->deeper;
                                                dCopy = p->deeper;
                                        }
                                } while (p->deeper);
                                delete h;
                                h = NULL;
                        }
                }
        } while (h);
}
MultiList* DeleteStudent(MultiList* h)
{
        MultiList* p = h, * pCopy = h;
        List* pDeeper = h->deeper, * dCopy;
        string data, data2;
        int studcode;

        cout << "Введите код списка из которого хотите удалить студента: ";
        cin >> studcode;
        cout << "Введите имя студента которого хотите удалить: ";
        cin >> data;
        getline(cin, data2);
        data = data + data2;

        if (studcode == 1)
        {
                if (h->info == data)
                {
                        h = h->next;
                        delete p;
                }
                else
                {
                        while (p->next != NULL)
                        {
                                if (p->info == data) break;
                                p = p->next;
                        }
                        if (p->next)
                        {
                                while (pCopy->next != p) pCopy = pCopy->next;
                                pCopy->next = p->next;
                                delete p;
                        }
```

```
                else
                {
                        delete p;
                        p = NULL;
                }

        }
}
else
{
        if (p->studcode == studcode)
        {
                if (p->deeper->info == data)
                {
                        if (!p->deeper->next)
                        {
                                delete p->deeper;
                                if (p->info == data)
                                {
                                        if (h->next)
                                        {
                                                h = h->next;
                                                delete p;
                                        }
                                        else delete p;
                                }
                        }
                        else
                        {
                                pDeeper = p->deeper->next;
                                delete p->deeper;
                                p->deeper = pDeeper;
                        }
                }
                else
                {
                        pDeeper = p->deeper;
                        while (pDeeper->next)
                        {
                                if (pDeeper->info == data) break;
                                pDeeper = pDeeper->next;
                        }
                        if (pDeeper->next)
                        {
                                dCopy = p->deeper;
                                while (dCopy->next != pDeeper) dCopy = dCopy->next;
                                dCopy->next = dCopy->next->next;
                                delete pDeeper;
                        }
                        else
                        {
                                dCopy = p->deeper;
                                while (dCopy->next != pDeeper) dCopy = dCopy->next;
                                delete pDeeper;
                                dCopy->next = NULL;
                                p->info = dCopy->info;
                        }

                }
        }
        else
        {
                while (p->next)
                {
                        if (p->studcode == studcode) break;
                        p = p->next;
```

```
                        }
                        if (p->deeper->info == data)
                        {
                                if (!p->deeper->next)
                                {
                                        delete p->deeper;
                                        if (p->info == data)
                                        {
                                                if (p->next)
                                                {
                                                        while (pCopy->next != p) pCopy = pCopy->next;
                                                        pCopy->next = p->next;
                                                        delete p;
                                                }
                                                else
                                                {
                                                        while (pCopy->next != p) pCopy = pCopy->next;
                                                        delete p;
                                                        pCopy->next = NULL;
                                                }
                                        }
                                }
                                else
                                {
                                        pDeeper = p->deeper->next;
                                        delete p->deeper;
                                        p->deeper = pDeeper;
                                }
                        }
                        else
                        {
                                pDeeper = p->deeper;
                                while (pDeeper->next)
                                {
                                        if (pDeeper->info == data) break;
                                        pDeeper = pDeeper->next;
                                }
                                if (p->next)
                                {
                                        dCopy = p->deeper;
                                        while (dCopy->next != pDeeper) dCopy = dCopy->next;
                                        dCopy->next = dCopy->next->next;
                                        delete pDeeper;
                                }
                                else
                                {
                                        dCopy = p->deeper;
                                        while (dCopy->next != pDeeper) dCopy = dCopy->next;
                                        delete pDeeper;
                                        dCopy->next = NULL;
                                        p->info = dCopy->info;
                                }
                        }
                }
        }
        return h;
}
void CreateList(MultiList* h)
{
        List* pDeeper = h->deeper;
        string data, data2;
        int choise;

        do
        {
                try
```

```
		{
			cout << "\nВведите код студента\n1) Обычный студент\n";
			cout << "2) Иностранец\n3) Первокурсник\n";
			cout << "4) Магистр\n5) Закончить заполнение списка\n";
			cout << "=======================\nВаш выбор: ";
			cin >> choise;

			if (choise == 5) break;
			else if (choise >= 6 || choise <= 0) throw exception("Ошибка. Вы ввели
неверный номер действия.");

			cout << endl << "Введите имя студента: ";
			cin >> data;
			getline(cin, data2);
			data = data + data2;

			AddStudent(h, pDeeper, data, choise);
		}
		catch (const std::exception& ex)
		{
			cout << ex.what() << endl;
		}

	} while (true);

}
void AddStudent(MultiList* h, List* pDeeper, string info, int studcode)
{

	MultiList* p = h;

	while (p->next)
	{
		if (p->studcode == studcode) break;
		p = p->next;
	}
	if (studcode == p->studcode && p->studcode == 1)
	{
		while (p->next) p = p->next;
		p->next = new MultiList;
		p = p->next;
		p->studcode = studcode;
		p->info = info;
		p->next = NULL;
	}
	else if (studcode == p->studcode && p->studcode == 2)
	{
		pDeeper = new List;
		pDeeper->info = info;
		pDeeper->next = p->deeper;
		p->deeper = pDeeper;
	}
	else if (studcode == p->studcode && p->studcode == 3)
	{
		pDeeper = new List;
		pDeeper->info = info;
		pDeeper->next = p->deeper;
		p->deeper = pDeeper;
	}
	else if (studcode == p->studcode && p->studcode == 4)
	{
		pDeeper = new List;
		pDeeper->info = info;
		pDeeper->next = p->deeper;
		p->deeper = pDeeper;
	}
```

```
else
{
	if (h->info == "0")
	{
		p->info = info;
		p->studcode = studcode;
		if (studcode != 1) p->deeper->info = info;
	}
	else
	{
		p->next = new MultiList;
		p = p->next;
		p->studcode = studcode;
		p->info = info;
		p->next = NULL;
		p->deeper = new List;
		p->deeper = NULL;

		if (studcode != 1)
		{
			pDeeper = new List;
			pDeeper->info = info;
			pDeeper->next = p->deeper;
			p->deeper = pDeeper;
		}
	}
}
```

```
}
```

Результати роботи програми

```
Иностранцы
Студент John            || Код: 2
  Студент: John
Обычный студент Dmitry           || Код: 1
Обычный студент ivan            || Код: 1

Первокурсники
Студент Sergiy           || Код: 3
  Студент Jim
  Студент Sergiy

Магистры
Студент Peter           || Код: 4
  Студент Peter

Добавление нового елемента списка

Введите код студента
1) Обычный студент
2) Студенты-иностранцы
3) Студенты первокурсники
4) Магистры
=======================
Ваш выбор: 2
Введите имя студента: Vladimir


Иностранцы
Студент John            || Код: 2
  Студент: Vladimir
  Студент: John
Обычный студент Dmitry           || Код: 1
Обычный студент ivan            || Код: 1

Первокурсники
Студент Sergiy           || Код: 3
  Студент Jim
  Студент Sergiy

Магистры
Студент Peter           || Код: 4
  Студент Peter
Введите код списка из которого хотите удалить студента: 1
Введите имя студента которого хотите удалить: Dmitry


Иностранцы
Студент John            || Код: 2
  Студент: Vladimir
  Студент: John
Обычный студент ivan            || Код: 1

Первокурсники
Студент Sergiy           || Код: 3
  Студент Jim
  Студент Sergiy

Магистры
Студент Peter           || Код: 4
```

Висновок

У результаті роботи програми було розроблено програму, яка забезпечує роботу з мультісписком. Реалізовані такі функції: додавання елементів, видалення елементів, видача вмісту списку.

Відповіді на питання

1. Які відмінності списка від масива?

- У списку кожен елемент містить покажчик на наступний елемент (іноді і на інші);
- Робота зі списками складніше в реалізації;
- На поля свіязей списку потрібна додаткова пам'ять;
- Розмір обмежений тільки кількістю пам'яті;
- Робота зі списком відбувається по «ланцюжку», в той час як в масиві доступ до елементів відбувається за індексом, що в деяких випадках набагато швидше.

2. Які бувають списки?

Списки бувають:

- Зв'язні лінійні списки;
- Мультисписки (ортогональні списки);
- Нелінійні розгалужені списки;
- Двійкове дерево.

3. Які операції є характерними для лінійних списків.

- Перебір елементів списку;
- Вставка елемента в будь-яке місце списку;
- Видалення елемента з будь-якого місця списку;
- Перестановка елементів у списку;
- Копіювання частини списку;
- Злиття двох списків.

4. Які операції є характерними для нелінійних списків.

- Додавання вузла;
- Видалення вузла;
- Обход списка;

5. Якими характеристиками описуються нелінійні розгалужені списки?

Розгалужені списки описуються трьома характеристиками: **порядком**, **глибиною** і **довжиною**.

Порядок – послідовність елементів у списку.

Глибина – це максимальний рівень, який приписують елементам усередині списку або всередині будь-якого підсписку в списку.

Довжина – це число елементів першого рівня в списку.

6. Які характерні риси має двійкове дерево?

- є один вузол, на який не посилається ні який інший вузол. Цей вузол називається корінь або батьківський вузол;
- є вузли, які не посилаються ні на які інші вузли – це листи.

Інші вузли – вузли розгалуження. Корінь і вузли розгалуження посилаються на ліве і праве піддерева.

Оскільки дерево – по суті рекурсивна структура, то і усі фунции роботи з деревами – рекурсивні.

7. Що таке мультисписок?

Мультисписок (або ортогональний список) - це структура, кожен елемент якої входить більш ніж в один список одночасно і має відповідне числу списків кількість полів зв'язку.

8. Чим відрізняється кільцевий список від простого лінійного списка?

Кільцевій список є різновидом лінійного списку. При цьому в однозв'язному списку покажчик останнього елемента повинен вказувати на перший елемент. При роботі з такими списками кілька спрощуються деякі процедури, що виконуються над списком.

9. Які особливості видалення елементів з мультисписка?

Виняток елемента з якого-небудь одного списку ще не означає необхідності видалення елемента з пам'яті, так як елемент може залишатися в складі інших списків. Пам'ять повинна звільнятися тільки в тому випадку, коли елемент вже не входить ні в один з приватних списків мультісписка.

10. Які особливості видалення елементів із списка списків (нелінійного списку)?

При видаленні елемента з головного списку видаляється і пов’язаний з ним підсписок.

11. Чим визначається глибина нелінійного списка?

Глибина списку визначається максимальним значенням рівня серед рівнів всіх атомів списку.

12. Чим визначається довжина нелінійного списка?

Довжина нелінійного списку визначається кількістю елементів 1-го рівня у цьому списку.

13. Що описує порядок елінійного списка?

Порядок описує послідовність, у якій елементи стоять у списку.

14. Які мови програмування призначені виключно для обробки списків?

Сімейство мов програмування LISP є найбільш розвиненим і поширеним сімейством мов обробки списків. Наприклад, Lisp, MacLisp, PSL, Scheme, NIL та інші.

15. Побудуйте нелінійний список для арифметичного виразу *(a+b)*(c-d) /e*

a+b
c-d
(a+b)*(c-d)
(a+b)*(c-d) / e

16. Побудуйте нелінійний список для арифметичного виразу *d + f / (a+b)*c*

a+b
f / (a+b)
f / (a+b) * c
d + f / (a+b) * c

17. Назвіть відомі методи контроля вільної динамічної пам’яті.

Методи обліку вільної пам'яті можуть грунтуватися на:

- принципі бітової карти;
- принципі списків вільних блоків.

18. Назвіть алгоритми виділення вільної динамічної пам’яті.

Відомі дві основні дисципліни, які зводяться до принципів

- «найкращий»;
- «перший відповідний».

19. Які відомі методи звільнення динамічної пам’яті?

- «Прибирання сміття»;
- Звільнення будь-якого блоку, як тільки він перестає використовуватися;
- Метод відновлення раніше зарезервованої пам'яті.

20. Наведіть опис алгоритма «самий підходящий» для виділення по запиту динамічної пам’яті.

З дисципліни «найкращий» виділяється той вільну ділянку, розмір якого дорівнює запрошенням або перевищує його на мінімальну величину.

21. Наведіть опис алгоритма «перший підходящий» для виділення по запиту динамічної пам’яті.

З дисципліни «перший відповідний» виділяється перший же знайдений вільна ділянка, розмір якого не менше запитаного.

22. Що таке «діри» в пам’яті і якими вони бувають?

«Діри» - ділянки пам'яті, які не можуть бути використані. Розрізняються діри **внутрішні** і **зовнішні**.

Внутрішня діра - невживана частина виділеного блоку, вона виникає, коли розмір виділеного блоку більше запитаного. Внутрішні діри характерні для виділення пам'яті блоками фіксованою довжини.

Зовнішня діра - вільний блок, який в принципі міг би бути виділений, але розмір його занадто малий для задоволення запиту. Зовнішні діри характерні для виділення блоками змінної довжини.