МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет

Практическая работа №5

Дисциплина Структуры и алгоритмы обработки данных

Тема. Структуры хранения линейных структур данных. Однонаправленный динамический список.

Выполнил студент Черномуров С. А.

Фамилия И.О.

Группа ИКБО-13-21

Номер группы

Москва 2022

Ответы на вопросы:

1) Расскажите о трех уровнях представления данных в программной системе.

Существует три уровня представления данных:

- Внешний (пользовательский);
- Логический;
- Физический.

Внешний уровень (уровень задачи):
Как видит данные пользователь будущей программной системы.
Определяет наиболее полезную и удобную для конкретного пользователя форму представления подмножества данных, необходимых для выполнения стоящих перед пользователем задач.
К таким данным может относиться текст, структурированная информация в виде таблиц, графика, шаблон представления информации в документе в виде разметки страницы и т.д.
На внешнем уровне структура данных описывается как абстрактный объект (АТД): данные как видим их в задаче и набор действий (операций над данными).
Логический:
Как реализует программист данные задачи.
Этот уровень можно назвать уровнем программиста. Логическое представление совокупности элементов данных с отображением отношений между ними в программе – в структуры данных языка программирования. На этом уровне архитектуры определяются сущности предметной области, их атрибуты и связи, накладываемые на данные ограничения. Основной акцент в представлении информации на логическом уровне делается на семантике (т.е. смыслового значении) данных. Логический уровень не рассматривает вопросы физического хранения информации.
Физический:

Характеризует данные в виде, воспринимаемом операционной системой, т.е. физическое хранение данных. На этом уровне описывается физическая реализация данных - это биты, байты, сведения о размещении записей в файлах и таблицах, сведения о сжатии данных, выбранных методах их шифрования и т.д. Необходимость изучения этого способа представления заключается в непосредственном рассмотрении способов размещения данных при реализации проектных решений.

2) Что определяет тип данных?

Тип данных определяет одновременно:

- Формат внутреннего представления
- Множество значений
- Набор операций, допустимых над этими значениями

3) Что определяет структура данных?

Структура данных определяет способ хранения и организации данных, облегчающий доступ к этим данным и их модификацию.

4) Расскажите о структурах хранения данных в компьютерных технологиях.

СХД – это представление определенной структуры данных в памяти компьютера. Одна и та же структура данных может быть представлена в памяти различными структурами хранения, так иерархическая структура данных может быть реализована либо на векторе (массиве), либо на сети (дереве из узлов). Различные структуры данных могут быть представлены одной и той же структурой хранения. Структура хранения, выбранная для представления структуры данных, должна с одной стороны сохранить отношения, между элементами данных, с другой обеспечить наиболее простое и эффективное выполнение операций над структурой в целом (создание, удаление) и отдельными элементами (включение, исключение, доступ).

5) Дайте определение линейной структуре данных.

Линейная структура представляют список элементов, упорядоченных по положению. Доступ к элементу прямой по номеру

6) Дайте определение структуре данных линейный список.

Структура линейный список — это список элементов упорядоченных по местоположению a1 , a2 , a3 ,….ap…..an , над которыми допустимы операции:

- вставить элемент в позицию i;
- удалить элемент из позиции i;
- получить доступ к элементу в позиции i.

7) Дайте определение структуре данных стек.

Стек – линейный список, в котором операции вставки, удаления, доступа выполняются над элементом в вершине списка. Дисциплина обслуживания стека называется LIFO (Last Input First Output).

8) Дайте определение структуре данных очередь.

Очередь – линейный список, в котором операция вставки (добавления) выполняется в конец списка, удаление из вершины, доступ к элементу в вершине. Дисциплина обслуживания очереди FIFO (First Input First Output).

9) Чем стек отличается от структуры данных линейный список?

В стеке операции вставки, удаления, доступа допустимы только над элементом в вершине списка. В структуре данных линейный список эти операции допустимо выполнять независимо от позиции элемента.

10) Какой из видов линейных списков лучше использовать, если нужно введенную последовательность вывести наоборот?

Для вывода введённой последовательности, наоборот, лучше использовать линейную структуру Дек, так как в ней доступ к элементам осуществляется как с начала, так и с конца.

11) Определите сложность алгоритма операции вставки элемента в i-ую позицию: а) массива; б) линейного списка.

а) Сложность вставки в массив O(n);

б) Сложность вставки в структуру данных линейный список O(1).

12) Определите сложность алгоритма операции удаления элемента из i-ой позиции: а) массива; б) линейного однонаправленного списка.

а) Сложность удаления элемента из массива O(n) ;

б) Сложность удаления элемента из структуры данных линейный список O(n).

13) В чем суть трюка Вирта при выполнении операции удаления элемента из линейного однонаправленного списка?

Трюк Вирта заключается в том, чтобы указатель на удаляемый элемент перенаправить на элемент, следующий за удаляемым.

14) Определите структуру узла однонаправленного списка.

Односвязный список состоит из узлов. Узел содержит информационное поле и указатель на следующий узел.

15) Реализуйте алгоритм вывода линейного однонаправленного списка на языке C++.

```
void list :: print() {
	if (is_empty()) return;
	Node* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2)<<pointer->data << " ";
		pointer = pointer->next;
	}
}
```

16) Приведите фрагмент кода программы на языке C++ выполнения операции перемещения последнего элемента в начало списка.

```
void lasttofirst(MyList*& L)
{
   MyList* LastOne = nullptr;
   MyList* FirstOne = L;
   while (FirstOne->next) {
      LastOne = FirstOne;
      FirstOne = FirstOne->next;
   }
   LastOne->next = NULL;
   FirstOne->next = L;
   L = FirstOne;
```

```
}
```

17) Какой из действий лишнее в следующем фрагменте кода? Куда вставляется новый узел?

```
struct Node
{
	int info;
	Node*next;
};
typedef Node *List;
List* L=new List;
void insertToList(List LL, int x)
{
	List q=new Node; q->info=x; q->next=0;
	if (LL==nullptr)
		LL->next=q;
	else
		q->next=LL->next;
		LL->next=q;
}
```

Лишним действием является условный оператор (if … else):

оператор if и его блок необходимо удалить, а блок else оставить. Новый узел будет вставляться после узла, который передан.

Задание

Вариант №23

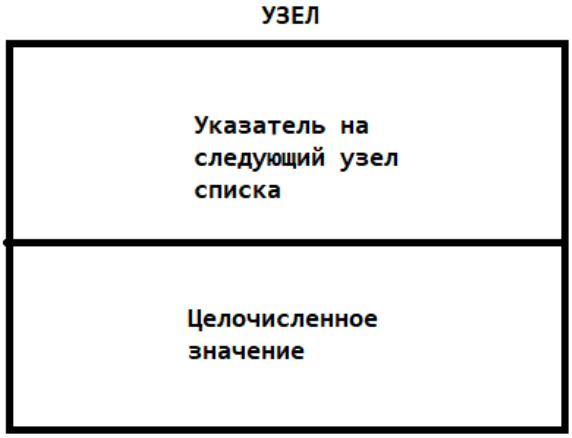
Даны два списка: список L из i узлов {L0, L1, …Li} и список M из j узлов {M0,M1,…Mj}при этом j>=i.

1) Разработать функцию, которая генерирует значения узлам списка. Список M заполняет i значениями (1<=i<=10) в диапазоне от 1 до 10, а список L j значениями (1<=j<=20) в диапазоне от 1 до 20.

2) Объединить два списка L и M в список LM с узлами {L0, M0, L1, M1,…,Lj, Mj } узлы которого содержат пары значений.

3) Удалить из списка M j-i последних узла.

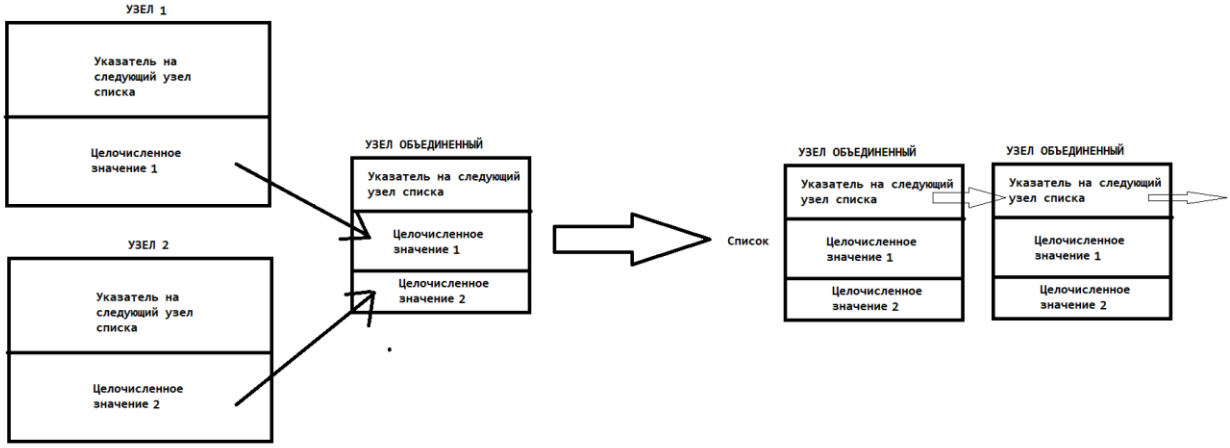
1. Постановка задачи
   1.1. Реализовать программу решения задачи варианта по использованию линейного однонаправленного списка.
   1.2. Дано. Списки L и M, длина L<=длина M
   Результат. Заполненные списки L и M, взаимодействие со списками в зависимости от выбранного функционала.
2. Структура узла однонаправленного списка:



3. Графическое представление операций в соответствии с заданиями варианта:
   3.1. Генерирование узлов списка:

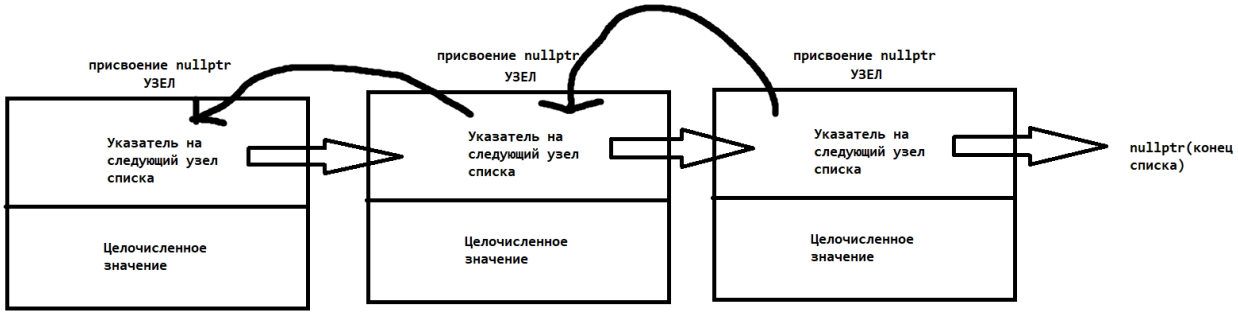
В список вставляется узел, указатель на следующий элемент в этом узле становится nullptr (т.к на данный момент вставленный узел является последним). Если список до вставки был пустой, то указатель на созданный узел присваивается указателю на первый элемент списка first. Целочисленное значение определяется случайным образом.

3.2. Объединение двух списков в один список:



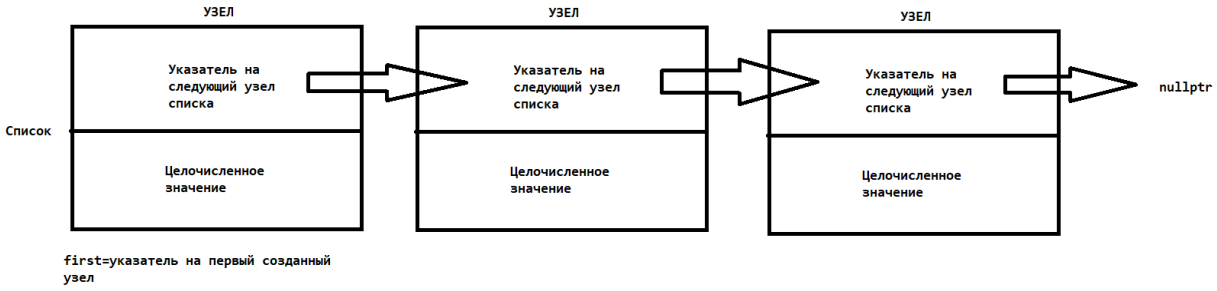
Целочисленные значения узла 1 и узла 2 передаются в соответствующие поля узла объединенного, после чего узел объединенный вставляется в третий список (работа с указателем происходит также, как и с генерированием узлов списка)

3.3. Удаление из списка некоторого количества последних узлов:



С конца списка (реализовано через перегрузку оператора []) и до определенного значения происходит присваивание указателю на следующий узел значения nullptr, что приводит к тому, что к «обнуленному» элементу более нельзя обратиться.

4. Графическое представление структуры данных:



5. Декомпозиция

5.1. Список подзадач:

1) Проверка наличия узлов в списке
2) Присоединение узла в конец списка
3) Присоединение узла в конец списка LM
4) Вывод списка на экран
5) Вывод списка LM на экран
6) Генерация списка определенной длины
7) Обращение к узлу списка по индексу (Перегрузка оператора [])
8) Удаление узлов из конца списка M
9) Создание узлов списка LM

5.1.1. **Проверка наличия узлов в списке:**

Предусловие. Список.

Постусловие. true – если список пустой, false – если список не пустой.

```
bool is_empty();
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список с указателем first=nullptr	true
2	Список с указателем first!=nullptr	false

```
bool list2::is_empty() {
	if (first == nullptr) return true;
	else return false;
}
```

5.1.2. Присоединение узла в конец списка:

Предусловие. Список list, _data – числовое значение, которое будет записано в узел.

Постусловие. Список с присоединенным в конец узлом.

```
void push_back(int _data);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Пустой список, _data=5, size=0	Список, first=указатель на первый узел, first->data=5, size=1
2	Список с указателем first!=nullptr, _data=10, size=3	Список, first=указатель на первый узел, последний узел списка->data=10, size=4

```
void list :: push_back(int _data) {
	Node* pointer = new Node(_data);
	if (is_empty()) {
		first = pointer;
		size++;
	}
	else {
		Node* temp = first;
		while (temp != nullptr) {
			if (temp->next == nullptr) { temp->next = pointer; break; }
			else temp = temp->next;
		}
		temp = new Node(_data);
		size++;
	}

}
```

5.1.3. Присоединение узла в конец списка LM:

Предусловие. Список list, _dataL – числовое значение, которое будет записано в поле узла dataL, _dataM – числовое значение, которое будет записано в поле узла dataM.

Постусловие. Список с присоединенным в конец узлом.

```
void push_back(int _dataL, int _dataM);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Пустой список, _dataL=5, _dataM=10	Список, first=указатель на первый узел, first->dataL=5, first->dataM=10
2	Список с указателем first!=nullptr, _dataL=10, _dataM=0	Список, first=указатель на первый узел, последний узел списка->dataL=10, dataM=0

```
void list2::push_back(int _dataL, int _dataM) { //

		//cout << l[i]->data << " " << m[i]->data<<endl;
		Node2* pointer = new Node2(_dataL, _dataM);
		if (is_empty()) {
			first = pointer;
		}
		else {
			Node2* temp = first;
			while (temp != nullptr) {
				if (temp->next == nullptr) { temp->next = pointer; break; }
				else temp = temp->next;
			}
			temp = new Node2(_dataL, _dataM);
		}
}
```

5.1.4. Вывод списка на экран:

Предусловие. Список list.

Постусловие. Список list, выведенный на экран.

```
void print();
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список из 5 узлов	Выведенный на экран список из 5 узлов

2	Пустой список	void

```
void list :: print() {
	if (is_empty()) return;
	Node* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2)<<pointer->data << " ";
		pointer = pointer->next;
	}
```

5.1.5. Вывод списка LM на экран:

Предусловие. Список list2.

Постусловие. Список list2, выведенный на экран.

```
void print();
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список из 5 узлов	Выведенный на экран список из 5 узлов
2	Пустой список	void

```
void list2::print() { //

	if (is_empty()) return;
	Node2* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2) << pointer->dataL << " " << pointer->dataM << 
"\n";
		pointer = pointer->next;
	}

}
```

5.1.6. Генерация списка определенной длины:

Предусловие. len – длина генерируемого списка.

Постусловие. Сгенерированный список list длины len.

```
void generate(int len);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	len=5	Список из 5 узлов, size=5
2	len=1	Список из 1 узла, size=1

```
void list::generate(int len) {

	for (int i = 0; i < len; i++) {
		int value = rand() % 100;
		this->push_back(value);
	}
}
```

5.1.7. Обращение к узлу списка по индексу:

Предусловие. index – индекс узла, указатель на который нужно вернуть.

Постусловие. Указатель на index-овый узел списка.

```
Node* operator[](const int index);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	index = 3, size = 5	Указатель на 3-й узел списка
2	Index = 5, size = 4	nullptr

```
Node* list::operator[] (const int index) { //
	if (is_empty()) return nullptr;
	Node* pointer = first;
	for (int i = 0; i < index; i++) {
		pointer = pointer->next;
		if (!pointer) return nullptr;
	}
	return pointer;
}
```

5.1.8. Удаление узлов из конца списка M:

Предусловие. Список L, Список M.

Постусловие. Список M, в котором удалены последние (длина M – длина L) узлов.

```
void delete_M(list l, list m);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Длина L = 4, Длина M = 7	Список M без последних 3-х узлов
2	Длина L = 5, Длина M = 5	Список M без изменений

```
void delete_M(list l, list m) { //
	for (int i = m.size - 1; i >= l.size-1; i--)
		m[i]->next = nullptr;
}
```

5.1.9. Создание узлов списка LM:

Предусловие. Список L, Список M.

Постусловие. Список LM, в котором каждый узел имеет пару элементов L[i], M[i].

```
void match(list l, list m);
```

Тест функции:

Номер теста	Исходные данные	Ожидаемый результат
1	Список L, список M	Список LM, узлы которого содержат соответствующие значения списков L, M

```
void list2::match(list l, list m) {
	for (int i = 0; i < max(l.size,m.size); i++) {
		if (i >= l.size) push_back(-1, m[i]->data);
		else
		if (i >= m.size) push_back(l[i]->data, -1);
		else push_back(l[i]->data, m[i]->data);
	}
}
```

Тестирование программы

```
Лабораторная работа №5 ИКБО-13-21 Черномуров С.А. Вариант 23

Введите количество элементов в списке L:
5

Введите количество элементов в списке M:
7

Сгенерированный список L: 41 67 34  0 69

Сгенерированный список M: 24 78 58 62 64  5 45
Выберите задание:
1) Создать и вывести список LM
2) Удалить из списка M последние (длина списка M - длина списка L) элементов
0) Закончить программу
```

```
Сгенерированный список LM:
41 24
67 78
34 58
 0 62
69 64
-1 5
-1 45
```

```
Введите количество элементов в списке L:
4

Введите количество элементов в списке M:
8

Сгенерированный список L: 81 27 61 91

Сгенерированный список M: 95 42 27 36 91  4  2 53
Выберите задание:
1) Создать и вывести список LM
2) Удалить из списка M последние (длина списка M - длина списка L) элементов
0) Закончить программу
2
```

```
Список M без удаленных элементов: 95 42 27 36
```

```
Лабораторная работа №5 ИКБО-13-21 Черномуров С.А. Вариант 23

Введите количество элементов в списке L:
5

Введите количество элементов в списке M:
5

Сгенерированный список L: 13  3 33 82  4

Сгенерированный список M: 11  3 74 43 76
Выберите задание:
1) Создать и вывести список LM
2) Удалить из списка M последние (длина списка M - длина списка L) элементов
0) Закончить программу
1
```

```
Сгенерированный список LM:
13 11
 3 3
33 74
82 43
 4 76
```

```
Лабораторная работа №5 ИКБО-13-21 Черномуров С.А. Вариант 23

Введите количество элементов в списке L:
5

Введите количество элементов в списке M:
5

Сгенерированный список L: 34 78 79 88 27

Сгенерированный список M: 28 91 37 80 73
Выберите задание:
1) Создать и вывести список LM
2) Удалить из списка M последние (длина списка M - длина списка L) элементов
0) Закончить программу
2
```

```
Список M без удаленных элементов: 28 91 37 80 73
```

Полный код программы на языке C++

Файл functions.h (содержит структуры для создания списков, а также функционал списков)

```
#pragma once


struct Node {
	Node* next = nullptr;
```

```
        int data;
        Node(int _data) : data(_data), next(nullptr) {}
};



struct list {
        Node* first;
        int size=0;
        list() : first(nullptr) {}

        bool is_empty();

        void push_back(int _data);
        void print();


        void generate(int len);
        Node* operator[](const int index);
};
////////////////////////////////////////////////////////////////////////////////////////
/

struct Node2 {
        Node2* next = nullptr;
        int dataL;
        int dataM;
        Node2(int _dataL, int _dataM) : dataL(_dataL), dataM(_dataM), next(nullptr) {}
};

struct list2 {
        Node2* first;

        list2() : first(nullptr) {}

        bool is_empty();
        void push_back(int _dataL, int _dataM);
        void print();
        void match(list l, list m);
};

void delete_M(list l, list m);
```

Файл functions.cpp (содержит тела функций для списков L, M)

```
#include "functions.h"
#include <iostream>
#include <iomanip>
#include "random"
using namespace std;

bool list :: is_empty() { //
        if (first == nullptr) return true;
        else return false;
}



void list :: push_back(int _data) { //
        Node* pointer = new Node(_data);
        if (is_empty()) {
                first = pointer;
```

```
		size++;
	}
	else {
		Node* temp = first;
		while (temp != nullptr) {
			if (temp->next == nullptr) { temp->next = pointer; break; }
			else temp = temp->next;
		}
		temp = new Node(_data);
		size++;
	}

}


void list :: print() { //
	if (is_empty()) return;
	Node* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2)<<pointer->data << " ";
		pointer = pointer->next;
	}
}

void list::generate(int len) { //

	for (int i = 0; i < len; i++) {
		int value = rand() % 100;
		this->push_back(value);
	}
}

Node* list::operator[] (const int index) { //
	if (is_empty()) return nullptr;
	Node* pointer = first;
	for (int i = 0; i < index; i++) {
		pointer = pointer->next;
		if (!pointer) return nullptr;
	}
	return pointer;
}

void delete_M(list l, list m) { //
	for (int i = m.size - 1; i >= l.size-1; i--)
		m[i]->next = nullptr;
}
```

Файл functions2.cpp (содержит тела функций для объединенного списка LM)

```
#include "functions.h"
#include <iostream>
#include <iomanip>
#include "random"
using namespace std;

bool list2::is_empty() {
	if (first == nullptr) return true;
	else return false;
}
```

```
void list2::push_back(int _dataL, int _dataM) { //

		//cout << l[i]->data << " " << m[i]->data<<endl;
		Node2* pointer = new Node2(_dataL, _dataM);
		if (is_empty()) {
			first = pointer;
		}
		else {
			Node2* temp = first;
			while (temp != nullptr) {
				if (temp->next == nullptr) { temp->next = pointer; break; }
				else temp = temp->next;
			}
			temp = new Node2(_dataL, _dataM);
		}
}

void list2::match(list l, list m) {
	for (int i = 0; i < max(l.size,m.size); i++) {
		if (i >= l.size) push_back(-1, m[i]->data);
		else
		if (i >= m.size) push_back(l[i]->data, -1);
		else push_back(l[i]->data, m[i]->data);
	}
}

void list2::print() { //

	if (is_empty()) return;
	Node2* pointer = first;
	while (pointer != nullptr) {
		cout << setw(2) << pointer->dataL << " " << pointer->dataM << "\n";
		pointer = pointer->next;
	}

}
```

Файл main.cpp (основной алгоритм программы)

```
#include <iostream>
#include "functions.h"
#include "random"
using namespace std;

int main()
{
	srand(time(NULL));
	setlocale(LC_ALL, "");

	cout << "Лабораторная работа №5 ИКБО-13-21 Черномуров С.А. Вариант 23" << endl
<< endl;

	cout << "Введите количество элементов в списке L:\n";
	int len1;
	cin >> len1;

	cout << "\nВведите количество элементов в списке M:\n";
	int len2;
	cin >> len2;

	list l, m;
```

```
	cout << "\nСгенерированный список L: ";
	l.generate(len1);
	l.print();
	cout << "\n";

	cout << "\nСгенерированный список M: ";
	m.generate(len2);
	m.print();
	cout << "\n";

	cout << "Выберите задание:\n1) Создать и вывести список LM\n2) Удалить из списка
M последние (длина списка M - длина списка L) элементов\n0) Закончить программу\n";
	int choice1;

	do {
		cin >> choice1;

		if (choice1 != 1 && choice1 != 2 && choice1 != 0) cout << "Введено
неверное значение, попробуйте снова.\n";
	} while (choice1 != 1 && choice1 != 2 && choice1 != 0);

	system("cls");

	switch (choice1) {

	case 1: {
		cout << "Сгенерированный список LM:\n";
		list2 lm;
		lm.match(l, m);
		lm.print();
		cout << "\n";
		break; }

	case 2: {
		cout << "Список M без удаленных элементов: ";
		delete_M(l, m);
		m.print();
		cout << "\n\n";
		break; }

	case 0:
		return 0;
	}
	main();
}
```

ВЫВОД

В ходе работы было выполнено задание в соответствии с поставленным вариантом . Были получены знания и практические навыки управления динамическим однонаправленным списком, реализации многоэлементных структур данных, работы с функциями и их параметрами.