Московский авиационный институт

(Национальный исследовательский университет)

Институт №3:

Информатика и вычислительная техника

Кафедра 304.

Отчёт по лабораторной работе

По учебной дисциплине «Программирование»

На тему

«*Линейные списки*»

Группа: М3О-210Б-23

Выполнил:

Миронов А.Д.

Принял:

Чечиков Ю.Б.

2024 год Москва

Задание (вариант 7)

Лабораторная работа № 4 «Линейные списки»

Цель работы: изучить принципы программной реализации динамической структуры данных «линейный список».

Задание

Реализовать линейный список, состоящий из 20 элементов заданного типа. Интерфейс должен включать **для всех вариантов** следующие операции**:**

- создание списка;
- вывод на экран и/или в файл значений элементов списка с их индексами (номерами);
- удаление списка,

а также **некоторые из дополнительных операций** (согласно варианту задания):

1. Поиск в списке наличия элемента с заданным значением с получением его номера в списке (повторное вхождение одного и того же значения может быть разрешено или запрещено – см. вариант).
2. Поиск в списке элемента с максимальным значением с получением его номера в списке (повторное вхождение одного и того же значения может быть разрешено или запрещено – см. вариант).
3. Включение нового элемента в начало списка.
4. Включение нового элемента в конец списка.
5. Включение нового элемента в позицию списка с заданным в программе номером.
6. Удаление элемента из начала списка.
7. Удаление элемента из конца списка.
8. Удаление элемента из позиции списка с заданным в программе номером.

После выполнения операций включения или удаления вывести содержимое списка. Выполнение операций организовать с помощью меню.

Варианты заданий

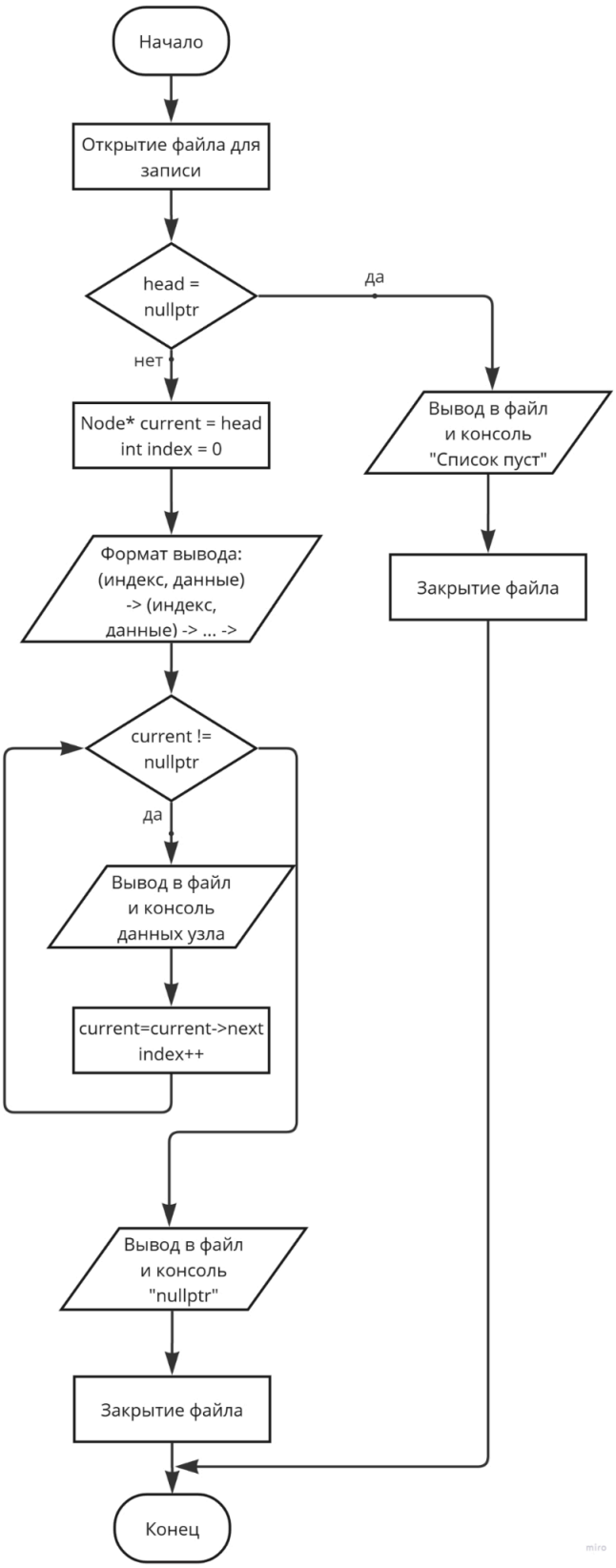
№ вар.	Вид списка	Тип элементов	Дополнительные операции
1	однонаправленный кольцевой	целочисленный	1 (запрещено), 4, 8
2	двунаправленный	символьный	2 (разрешено), 3, 6
3	двунаправленный кольцевой	целочисленный	1 (разрешено), 4, 7
4	однонаправленный	вещественный	2 (запрещено), 5, 8
5	двунаправленный	символьный	1 (запрещено), 3, 7
6	двунаправленный кольцевой	вещественный	1 (разрешено), 5, 6
7	однонаправленный	целочисленный	2 (запрещено), 4, 8
8	двунаправленный	вещественный	2 (разрешено), 5, 7
9	двунаправленный кольцевой	символьный	1 (разрешено), 3, 8
10	однонаправленный кольцевой	целочисленный	1 (запрещено), 4, 6
11	однонаправленный	целочисленный	1 (разрешено), 3, 7
12	двунаправленный	вещественный	2 (запрещено), 5, 6

Отчет должен содержать:

- задание на лабораторную работу, соответствующее варианту;
- схемы алгоритмов;
- текст программы и всех функций;
- результаты работы программы.

Схемы функций

PrintList



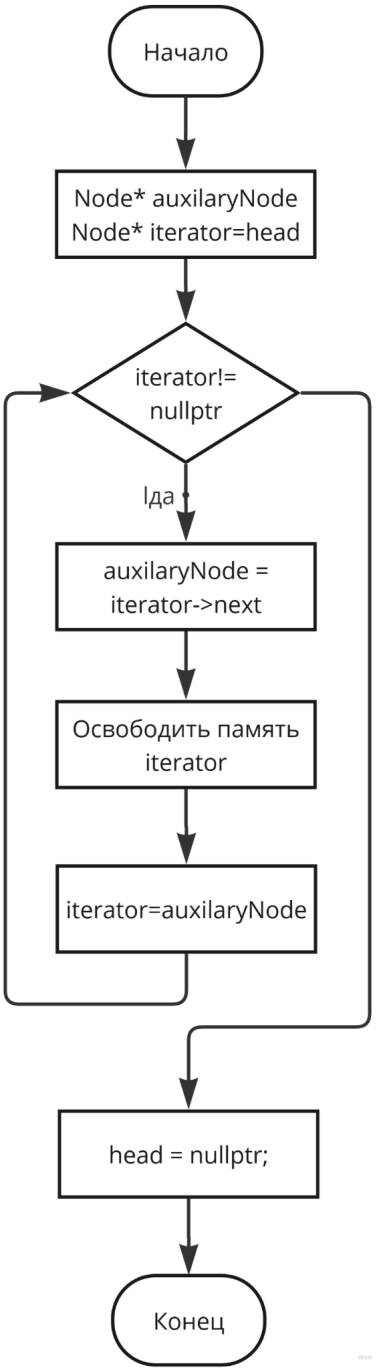
Функция выводит элементы связного списка в консоль и записывает их в указанный файл.

На вход поступают указатель на голову списка и строка с именем файла для записи.

void PrintList(Node* head, const char* filename)

(где Node - структура, представляющая узел списка, содержащая данные и указатель на следующий узел)

DeleteList



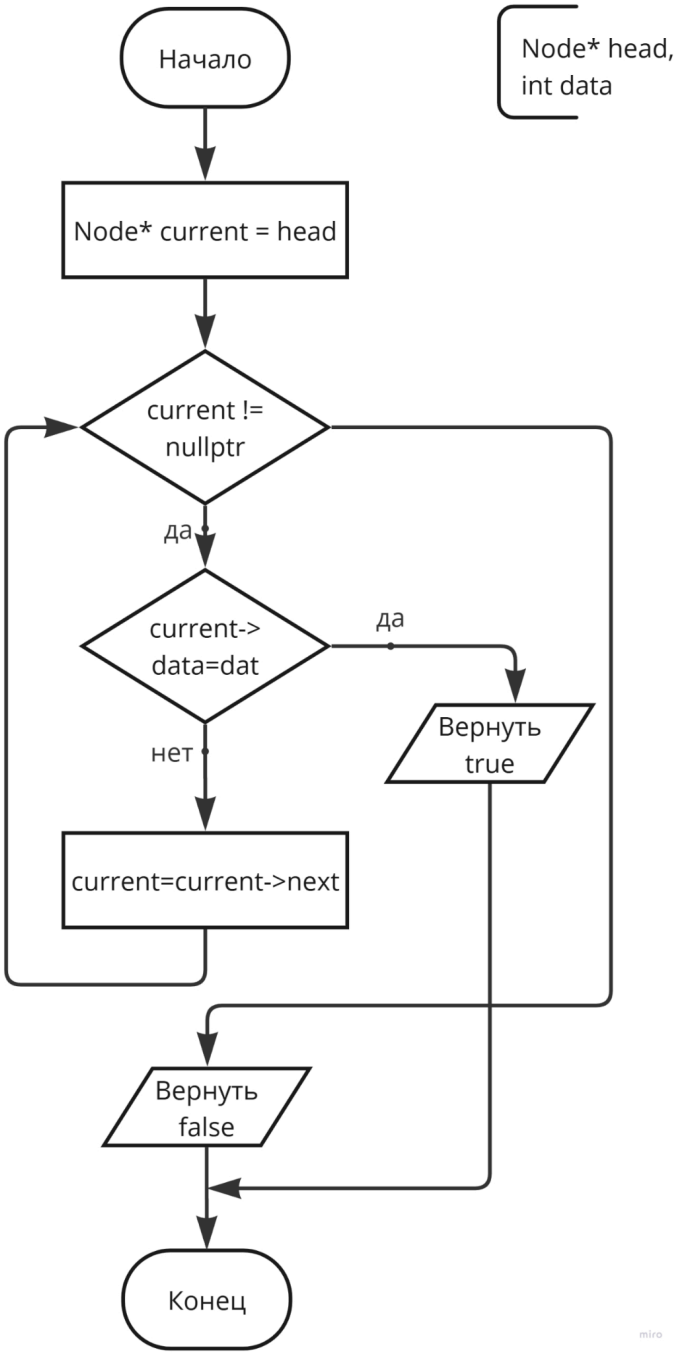
Функция удаляет все узлы связного списка и освобождает выделенную память.

На вход поступает указатель на указатель на голову списка.

void DeleteList(Node** head)

(где Node - структура, представляющая узел списка, содержащая данные и указатель на следующий узел)

Contains



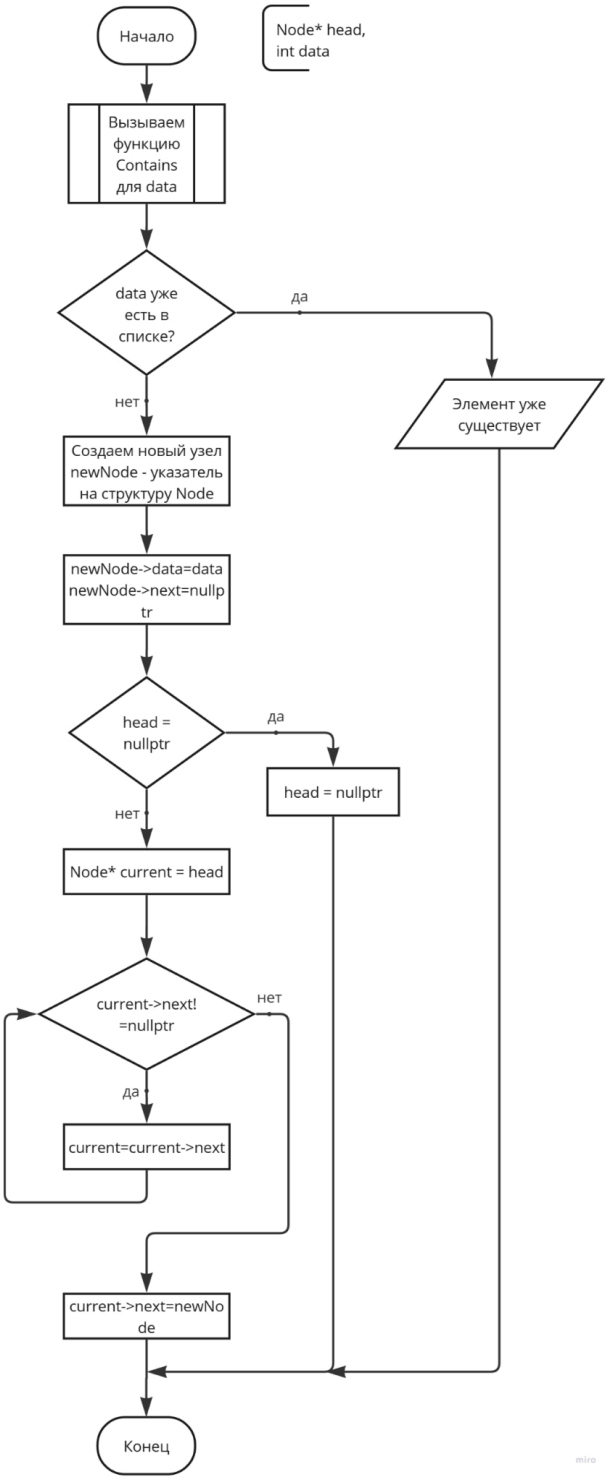
Функция проверяет, содержится ли заданное значение в связном списке.

На вход поступают указатель на голову списка и значение, которое необходимо найти.

bool Contains(Node* head, int data)

(где Node - структура, представляющая узел списка, содержащая данные и указатель на следующий узел)

InsertAtTail



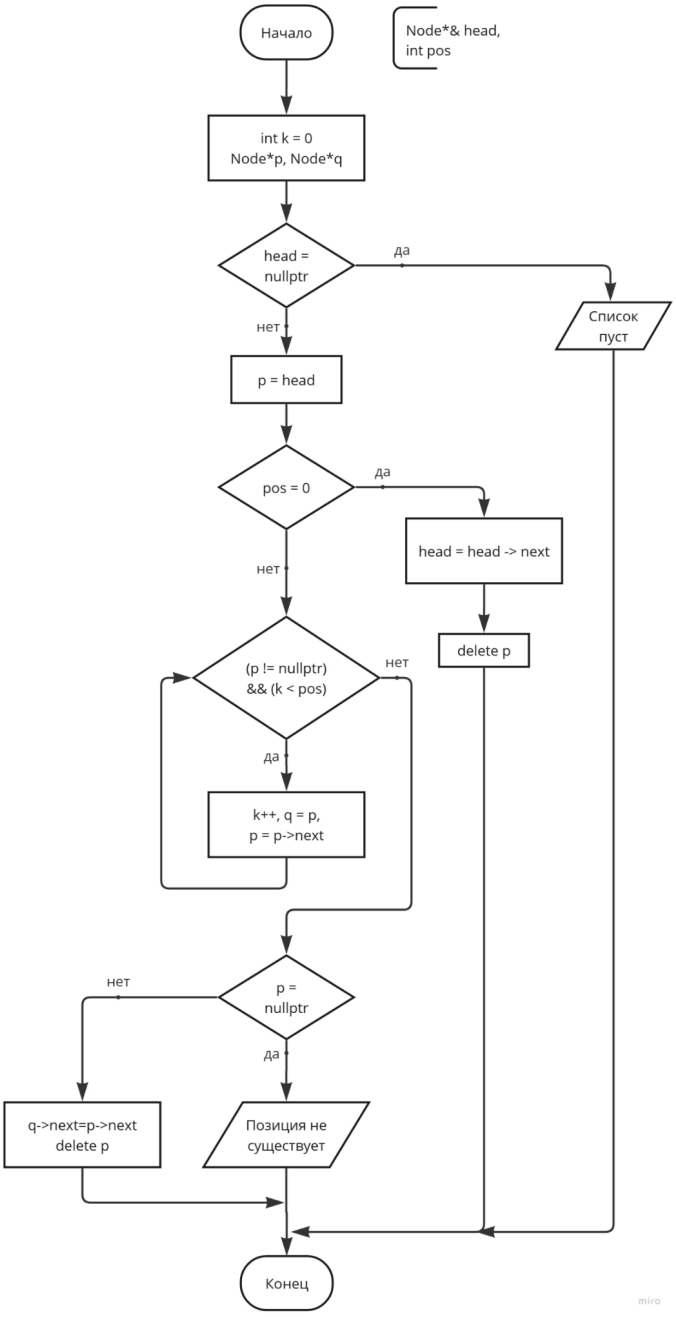
Функция для вставки элемента в конец связного списка.

На вход поступают указатели на голову и хвост списка, а также значение, которое необходимо добавить.

void InsertAtTail(Node** head, Node** tail, int data)

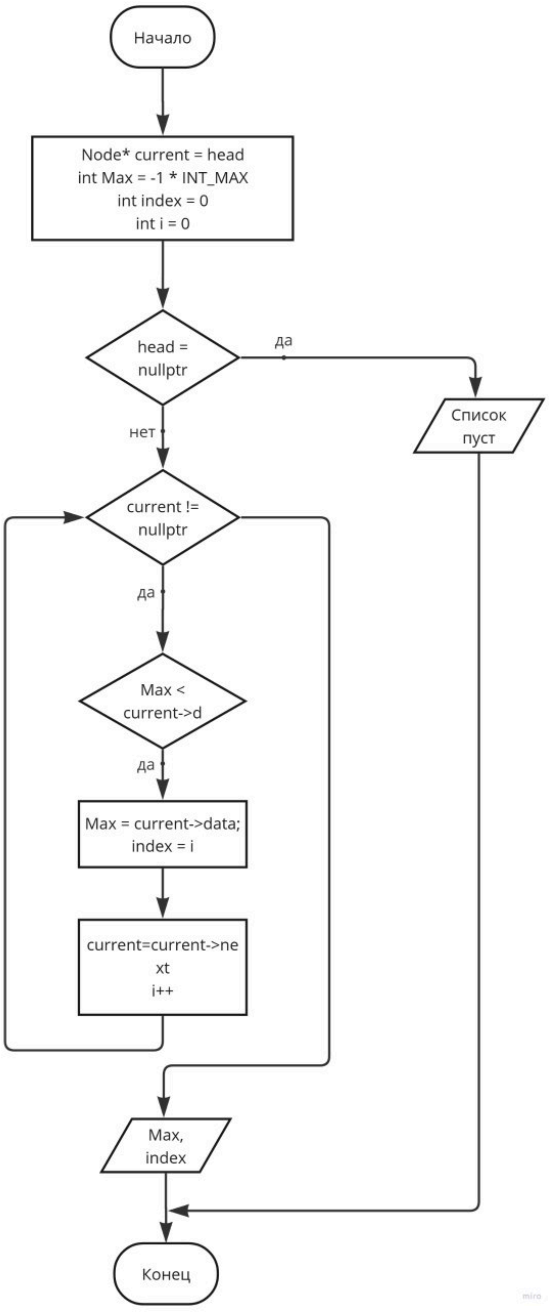
(где Node - структура, представляющая узел списка, содержащая данные и указатель на следующий узел)

DeleteAtPos



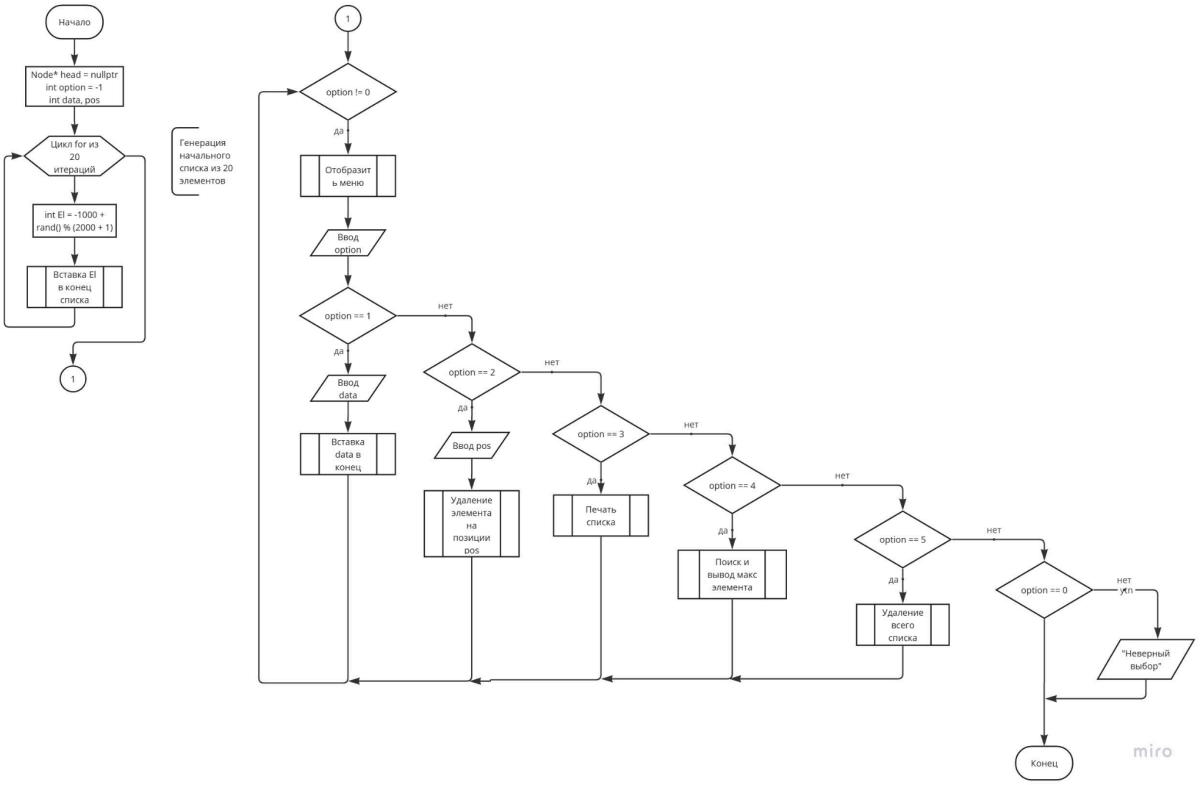
Функция для удаления элемента на заданной позиции в связном списке. На вход поступают указатели на голову и хвост списка, а также позиция элемента, который необходимо удалить. void DeleteAtPos(Node** head, Node** tail, int pos) (где Node - структура, представляющая узел списка, содержащая данные и указатели на следующий узел). Функция обрабатывает различные случаи, включая удаление первого элемента, последнего элемента и элементов из середины списка, а также учитывает ситуации с пустым списком и не существующими позициями.

MaxElement



Функция для нахождения максимального элемента в связном списке. На вход поступает указатель на голову списка: void MaxElement(Node* head) (где Node - структура, представляющая узел списка, содержащая данные и указатель на следующий узел). Функция сначала проверяет, пуст ли список, и выводит соответствующее сообщение в случае, если он пуст. Затем она инициализирует переменную для хранения максимального значения, начиная с первого узла, и проходит по всем узлам списка, сравнивая значения и обновляя максимальный элемент при необходимости. В конце функция выводит найденное максимальное значение.

Схема главной функции



Код программы

```
/*****************************************************************************************
*                КУРС КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ                                           *
*-----------------------------------------------------------------------------------------*
* Project Type  : Win32 Console Application                                               *
* Project Name  : lab2                                                                    *
* File Name     : lab4.cpp                                                                *
* Programmer(s) : Миронов А.Д., Фомин В.А.,  М3О-210Б-23                                  *
* Modified By   :                                                                         *
* Created       : 06/10/24                                                                *
* Last Revision :                                                                         *
* Comment(s)    :                                                                         *
*****************************************************************************************/

// Вариант 7 - целочисленный однонаправленный

#include <iostream> // Подключаем библиотеку стандартного ввода-вывода
#include <fstream>  // Подключаем библиотеку для работы с файлами
#include <limits.h> // Подключаем библиотеку для использования INT_MAX

using namespace std;

const char FNAME[30] = "lists.txt"; // Имя файла для вывода списка

// Структура для узла списка
struct Node {
   int data;       // Данные узла
   Node* next;     // Указатель на следующий узел
};

// Объявления функций
void PrintList(Node*& head, const char* filename);       // Печать списка
(файл + консоль)
void DeleteList(Node*& head);                            // Удаление списка
void InsertAtTail(Node*& head, int data);                // Вставка в конец
bool Contains(Node*& head, int data);                    // Проверка на
наличие элемента
void DeleteAtPos(Node*& head, int pos);                  // Удаление элемента
на позиции
void MaxElement(Node*& head);                            // Поиск
максимального элемента
void ShowMenu();                                         // Показ меню

int main() {
   srand(time(NULL)); // Инициализируем генератор случайных чисел текущим
временем

   Node* head = nullptr; // Указатель на голову списка
   int option = -1;      // Переменная для выбора операции
   int data, pos;        // Переменные для данных и позиции

   // Генерация списка из 20 случайных элементов (как положительных, так и
отрицательных)
   for (int i = 0; i < 20; i++) {
       int El = -1000 + rand() % (2000 + 1); // Генерация случайного числа в
диапазоне [-1000, 1000]
```

```
        InsertAtTail(head, El); // Вставка сгенерированного элемента в конец
списка
    }

    // Основной цикл программы
    while (option != 0) {
        ShowMenu(); // Отображение меню
        cout << "Введите номер операции: ";
        cin >> option; // Ввод выбора операции

        switch (option) {
            case 1:
                cout << "Введите элемент для вставки в конец списка: ";
                cin >> data; // Ввод элемента для вставки
                InsertAtTail(head, data); // Вставка элемента в конец списка
                break;
            case 2:
                cout << "Введите позицию удаляемого элемента в списке: ";
                cin >> pos; // Ввод позиции для удаления
                DeleteAtPos(head, pos); // Удаление элемента на заданной
позиции
                break;
            case 3:
                PrintList(head, FNAME); // Печать списка
                break;
            case 4: // Поиск максимального элемента
                MaxElement(head); // Поиск и вывод максимального элемента
                break;
            case 5: // Удаление списка
                DeleteList(head); // Удаление всего списка
                cout << "Список удален." << endl;
                break;
            case 0: // Выход
                cout << "Выход из программы..." << endl;
                break;
            default:
                cout << "Неверный выбор. Пожалуйста, попробуйте снова." <<
endl; // Обработка неверного выбора
        }
    }

    return 0; // Завершение программы
}

// Функция для печати списка
void PrintList(Node*& head, const char* filename) {
    ofstream file(filename, ofstream::out); // Открытие файла для записи

    if (head == nullptr) { // Проверка на пустой список
        cout << "Список пуст" << endl;
        file << "Список пуст" << endl;
        file.close(); // Закрытие файла
        return;
    }

    Node* current = head; // Указатель на текущий узел
    int index = 0; // Индекс текущего узла
    cout << "Формат вывода: (индекс, данные) -> (индекс, данные) -> ... ->
nullptr" << endl;
```

```
    while (current != nullptr) { // Проход по всем узлам списка
        cout << "(" << index << ", " << current->data << ")" << " -> "; //
Вывод данных узла в консоль
        file << "(" << index << ", " << current->data << ")" << " -> "; //
Запись данных уз ла в файл
        current = current->next; // Переход к следующему узлу
        index++; // Увеличение индекса
    }
    cout << "nullptr" << endl; // Завершение вывода списка
    file << "nullptr" << endl; // Запись завершения списка в файл
    file.close(); // Закрытие файла
}

// Функция для удаления списка
void DeleteList(Node*& head) {
    Node* auxilaryNode; // Временный указатель для хранения следующего узла
    Node* iterator = head; // Указатель для прохода по списку
    while (iterator) { // Проход по всем узлам списка
        auxilaryNode = iterator->next; // Сохранение следующего узла
        delete iterator; // Удаление текущего узла
        iterator = auxilaryNode; // Переход к следующему узлу
    }
    head = nullptr; // Обнуление указателя на голову списка
}

// Функция для проверки наличия элемента в списке
bool Contains(Node*& head, int data) {
    Node* current = head; // Указатель на текущий узел
    while (current != nullptr) { // Проход по всем узлам списка
        if (current->data == data) { // Проверка на совпадение данных
            return true; // Элемент найден
        }
        current = current->next; // Переход к следующему узлу
    }
    return false; // Элемент не найден
}

// Функция для вставки элемента в конец списка
void InsertAtTail(Node*& head, int data) {
    if (Contains(head, data)) { // Проверка на наличие дубликата
        cout << "Элемент " << data << " уже существует в списке. Пропускаем
вставку." << endl;
        return; // Не добавляем дубликат
    }

    Node* newNode = new Node; // Создание нового узла
    newNode->data = data; // Заполнение данных узла
    newNode->next = nullptr; // Указатель на следующий узел равен nullptr

    if (head == nullptr) { // Если список пуст
        head = newNode; // Новый узел становится головой списка
        return;
    }

    Node* current = head; // Указатель для прохода по списку

    while (current->next) { // Поиск последнего узла
        current = current->next; // Переход к следующему узлу
    }
```

```
    current->next = newNode; // Присоединение нового узла в конец списка
}

// Функция для удаления элемента на заданной позиции
void DeleteAtPos(Node*& head, int pos) {
    int k = 0; // Счетчик для отслеживания позиции
    Node* p; // Указатель на текущий узел
    Node* q; // Указатель на предыдущий узел
    if (head == nullptr) { // Проверка на пустой список
        cout << "Список пуст" << endl;
        return;
    }
    p = head; // Начинаем с головы списка
    if (pos == 0) { // Если удаляем первый элемент
        head = head->next; // Обновляем голову списка
        delete p; // Удаляем первый узел
        return;
    } else {
        while ((p != nullptr) && (k < pos)) { // Поиск узла на заданной позиции
            k++;
            q = p; // Сохраняем предыдущий узел
            p = p->next; // Переход к следующему узлу
        }
    }

    if (p == nullptr) { // Если позиция не существует
        cout << "Позиция не существует" << endl;
    } else {
        q->next = p->next; // Пропускаем узел p
        delete p; // Удаляем узел p
    }
}

// Функция для поиска максимального элемента в списке
void MaxElement(Node*& head) {
    Node* current = head; // Указатель на текущий узел
    int Max = -1 * INT_MAX; // Инициализация максимального элемента
    int index = 0; // Индекс максимального элемента
    int i = 0; // Счетчик для индексации

    if (head == nullptr) { // Проверка на пустой список
        cout << "Список пуст" << endl;
        return;
    }

    while (current != nullptr) { // Проход по всем узлам списка
        if (Max < current->data) { // Если текущий элемент больше максимального
            Max = current->data; // Обновляем максимальный элемент
            index = i; // Сохраняем индекс максимального элемента
        }
        current = current->next; // Переход к следующему узлу
        i++; // Увеличение индекса
    }

    cout << "Максимальный элемент: " << Max << "\t\tИндекс: " << index <<
endl; // Вывод максимального элемента и его индекса
}

// Функция для отображения меню
```

```
void ShowMenu() {
    cout << "\nМеню:" << endl;
    cout << "1. Вставить элемент в конец" << endl;
    cout << "2. Удалить элемент по позиции" << endl;
    cout << "3. Печать списка" << endl;
    cout << "4. Найти максимальный элемент" << endl;
    cout << "5. Удалить весь список" << endl;
    cout << "0. Выход" << endl;
}
```

Результаты работы

Меню:

```
Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции:
```

Печать изначального списка, заполненного случайно сгенерированными целыми числами с индексами элементов

```
Введите номер операции: 3
 Формат вывода: (индекс, данные) -> (индекс, данные) -> ... -> nullptr
(0, 853) -> (1, -692) -> (2, -123) -> (3, -515) -> (4, -521) -> (5, -596) -> (6, 706) -> (7, -955) -> (8, -251) -> (9, 7
60) -> (10, 122) -> (11, 711) -> (12, -298) -> (13, -536) -> (14, -924) -> (15, -424) -> (16, 615) -> (17, -869) -> (18,
 563) -> (19, -461) -> nullptr
```

Вставка элементов в конец списка

```
5. Удалить весь список
0. Выход
Введите номер операции: 1
 Введите элемент для вставки в конец списка: 11111

Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции: 1
 Введите элемент для вставки в конец списка: 22222

Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции: 3
 Формат вывода: (индекс, данные) -> (индекс, данные) -> ... -> nullptr
(0, 853) -> (1, -692) -> (2, -123) -> (3, -515) -> (4, -521) -> (5, -596) -> (6, 706) -> (7, -955) -> (8, -251) -> (9, 7
60) -> (10, 122) -> (11, 711) -> (12, -298) -> (13, -536) -> (14, -924) -> (15, -424) -> (16, 615) -> (17, -869) -> (18,
 563) -> (19, -461) -> (20, 11111) -> (21, 22222) -> nullptr
```

Удаление элемента на позиции 0

```
Введите номер операции: 2
 Введите позицию удаляемого элемента в списке: 0

Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции: 3
 Формат вывода: (индекс, данные) -> (индекс, данные) -> ... -> nullptr
(0, -692) -> (1, -123) -> (2, -515) -> (3, -521) -> (4, -596) -> (5, 706) -> (6, -955) -> (7, -251) -> (8, 760) -> (9, 1
22) -> (10, 711) -> (11, -298) -> (12, -536) -> (13, -924) -> (14, -424) -> (15, 615) -> (16, -869) -> (17, 563) -> (18,
 -461) -> (19, 11111) -> (20, 22222) -> nullptr
```

Удаление элемента на позиции 10

```
Введите номер операции: 2
 Введите позицию удаляемого элемента в списке: 10

Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции: 3
 Формат вывода: (индекс, данные) -> (индекс, данные) -> ... -> nullptr
(0, -692) -> (1, -123) -> (2, -515) -> (3, -521) -> (4, -596) -> (5, 706) -> (6, -955) -> (7, -251) -> (8, 760) -> (9, 1
22) -> (10, -298) -> (11, -536) -> (12, -924) -> (13, -424) -> (14, 615) -> (15, -869) -> (16, 563) -> (17, -461) -> (18
, 11111) -> (19, 22222) -> nullptr
```

Поиск максимального элемента в списке

```
Введите номер операции: 4
 Максимальный элемент: 22222                Индекс: 19
```

Попытка вставить новый узел с уже существующим значением

```
Введите номер операции: 1
 Введите элемент для вставки в конец списка: 22222
 Элемент 22222 уже существует в списке. Пропускаем вс
тавку.
```

Удаление всего списка

```
Введите номер операции: 5
 Список удален.

Меню:
1. Вставить элемент в конец
2. Удалить элемент по позиции
3. Печать списка
4. Найти максимальный элемент
5. Удалить весь список
0. Выход
Введите номер операции: 3
 Список пуст
```