**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**По контрольной работе**

**по дисциплине «Программирование»**

Тема: **Закрепление изученного материала**

|  |  |  |
| --- | --- | --- |
| Студент гр. 9894 |  | Курдов А.А. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

**Цель работы.**

Закрепить знания о пройденном материале:

1. Типы данных, определяемые пользователем, структуры;
2. Линейные структуры данных, динамический массив и двусвязные списки;
3. Польские нотации, стек и очередь.

**Основные теоретические положения.**

Структуры представляют собой группы связанных между собой, как правило, разнотипных переменных, объединенных в единый объект, в отличие от массива, все элементы которого однотипны. В языке C++ структура является видом класса и обладает всеми его свойствами. Чаще всего ограничиваются тем, как структуры представлены в языке:

struct [имя\_типа] {

тип\_1 элемент\_1;

тип \_2 элемент\_2;

тип\_k элемент\_k;

} [ список\_описателей ];

Описание структуры начинается ключевым словом struct. Каждая входящая в структуру переменная называется членом (полем, элементом) структуры и описывается типом данных и именем. Поля структуры могут быть любого типа данных. Их количество не лимитировано. Вся эта конструкция является инструкцией языка программирования, поэтому после нее всегда должен ставиться символ «;». При описании структуры память для размещения данных не выделяется. Работать с описанной структурой можно только после того, как будет определена переменная (переменные) этого типа данных, только при этом компилятор выделит необходимую память. Для инициализации структуры значения ее элементов перечисляют в фигурных скобках в порядке их описания:

struct complex{

float real, im;

} data [2][2] = { {{1,1}, {2,2}}, {{3,3}, {4,4}} };

Все поля структурных переменных располагаются в непрерывной области памяти одно за другим. Общий объем памяти, занимаемый структурой, равен сумме размеров всех полей структуры. Для определения размера структуры следует использовать инструкцию sizeof().

Чтобы записать данные в структурную переменную, необходимо каждому полю структуры присвоить определенное значение. Для этого необходимо использовать оператор «.» («точка»):

struct Stack

{ // Cтек float arr[100]; short topIndex;

}; Stack stack; // Объявляем переменную типа Stack Stack.arr[0] = 1;

При доступе к определенному полю его следует рассматривать как обычную переменную, тип данных которой соответствует типу этого поля. Поля структур могут участвовать в качестве операндов любых выражений, допускающих использование операндов соответствующего типа данных. Копирование данных из одной структурной переменной в другую осуществляется простой операцией присваивания не зависимо от количества полей и размера структуры (это можно делать только в том случае, когда обе переменные одного и того же типа). В программировании очень часто используются такие конструкции, как массивы структур. Например, сведения о студентах некоторой учебной группы можно хранить в массиве студентов:

t\_Student Gruppa\_N [30];

Был определен 30-ти элементный массив, каждый элемент которого предназначен для хранения данных одного студента. Получение доступа к данным некоторого студента из группы *N* осуществляется обычной индексацией переменной массива. Поскольку поля структуры могут быть любого типа данных, то они могут в свою очередь быть другой структурой или массивом других структур:

struct Stud { char FN[100]; short listNumber; };

struct Group { int groupNumber; short students; Stud stud[30]; };

Но в структуре поля нельзя использовать элемент, тип которого совпадает с типом самой структуры, так как рекурсивное использование структур запрещено.

Любая структурная переменная занимает в памяти определенное положение, характеризующееся конкретным адресом.

Для работы с адресами структурных переменных (как и для простых переменных) можно использовать указатели. Указатели на структурные переменные определяются точно так же, как и для обычных переменных. Разыменование указателя (обращение к данным по адресу, хранящемуся в указателе) осуществляется так же обычным образом. Через указатели можно работать с отдельными полями структур. Для доступа к полю структуры через указатель используется оператор «→» («стрелка»), а не «точка».

Структуры можно использовать в качестве параметров функций, как и обычные переменные. Для структур поддерживаются все три механизма передачи данных – по значению, через указатели и по ссылке. Передачу структур в функции по значению необходимо использовать аккуратно:

void WriteStudent ( t\_Student S ) {

cout << "Фамилия: " << S.Fam << endl; cout << "Имя: " << S.Name << endl; cout << "Год рождения: " << S.Year << endl; if ( S.Sex ) cout << "Пол: " << "М\n"; else cout << "Пол: " << "Ж\n"; cout << "Средний балл: " << S.Grade << endl; }

Вызов такой функции сопровождается дополнительным расходом памяти для создания локальной переменной *S* и дополнительными затратами времени на физическое копирование данных из аргумента в параметр *S*. Учитывая то, что объем структур может быть очень большим, то эти дополнительные затраты вычислительных ресурсов могут быть чрезмерными. Предпочтительно использование передачи структуры по указателю или ссылке:

void WriteStudent ( t\_Student \*S ) { cout << "Фамилия: " << S -> Fam << endl; cout << "Имя: " << S -> Name << endl; cout << "Год рождения: " << S -> Year << endl;

if ( S -> Sex ) cout << "Пол: " << "М\n"; else cout << "Пол: " << "Ж\n"; cout << "Средний балл: " << S -> Grade << endl; }

Фактической передачи данных в функцию не осуществляется. Дополнительные затраты памяти для создания локальной переменной небольшие – это адрес памяти (4 байта, независимо от размера самой структуры). Вызов такой функции будет осуществляться быстрее, а расход памяти существенно меньше, чем при передачи данных по значению. Передача по ссылке по своей эффективности эквивалентна передачи данных через указатель. Однако, поскольку при передаче данных по ссылке все адресные преобразования берет на себя компилятор, существенно упрощается программирование действий со структурами. При использовании ссылочных параметров структурных типов доступ к членам структуры осуществляется обычным способом – с помощью оператора «точка». Недостатком этих способов является то, что случайные изменения значений полей структуры внутри функции отразятся на значении аргумента после окончания работы функции. Если необходимо предотвратить изменения переданных по адресу аргументов, можно при определении соответствующего параметра объявить его константой (использовать спецификатор const).

Область кода программы предназначена для хранения инструкций функций программы, обеспечивающих обработку данных. Данные в программе представляются переменными и константами. Для хранения глобальных данных предназначена область глобальных данных. Стек программы используется при вызове функций для передачи параметров и хранения локальных данных.

Одномерный однонаправленный список представляет собой совокупность отдельных элементов, каждый из которых содержит две части – информационную (𝐷𝑎𝑡𝑎) и адресную (𝑇𝑎𝑖𝑙). Информационная часть предназначена для хранения “полезных” данных и может иметь практически любой тип. Адресная часть каждого элемента содержит адрес следующего элемента списка. Для работы со списком достаточно знать только адрес первого элемента списка (𝐵𝑒𝑔). Зная адрес первого элемента списка можно последовательно получить доступ к любому другому его элементу.

Одним из недостатков односвязных списков является то, что узел (элемент списка) имеет указатель только на следующий элемент. Вернутся из текущего элемента к предыдущему явным способом невозможно.

Каждый узел двусвязного (двунаправленного) линейного списка содержит два поля указателей – на следующий и на предыдущие узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель последнего узла также содержит нулевое значение [1].

Больше всего трудностей может возникнуть именно с взаимообменом элементов двусвязного списка. Главная проблема – возможная путаница в указателях. Нужно правильно изменить указатели всех затрагиваемых узлов.

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка их которого выполняется с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел). Основные операции над стеками:

1) чтение верхнего элемента;

2) добавление нового элемента;

3) удаление существующего элемента.

Очередь – частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка – из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Говорят, что очередь реализует принцип обслуживания FIFO (первым пришел – первым ушел). В программировании очереди применяются при моделировании, диспетчеризации задач операционной системой, буферизованном вводе/выводе.

Основные операции над очередями:

1) чтение первого элемента;

2) добавление нового элемента;

3) удаление существующего элемента

Если для стека в момент добавления или удаления элемента допустимо задействование лишь его вершины, то касательно очереди эти две операции должны быть применены так, как это регламентировано в определении этой структуры данных, т. е. добавление – в конец, удаление – из начала. Выделяют два способа программной реализации очереди. Первый из них основан на базе массива, а второй на базе указателей (связного списка). Первый способ – статический, т. к. очередь представляется в виде простого статического массива, второй – динамический. Кольцевой буфер также известен, как очередь или циклический буфер и является распространенной формой очереди. Это популярный, легко реализуемый стандарт, и, хотя он представлен в виде круга, в базовом коде он является линейным. Кольцевая очередь существует как массив фиксированной длины с двумя указателями: один представляет начало очереди, а другой - хвост.

Операции с очередями работают следующим образом:

• Два указателя, называемые 𝐻𝐸𝐴𝐷 и 𝑇𝐴𝐼𝐿, используются для отслеживания первого и последнего элементов в очереди.

• При инициализации очереди значения 𝐻𝐸𝐴𝐷 и 𝑇𝐴𝐼𝐿 устанавливаются равными -1.

• При добавлении элемента постепенно увеличивается значение индекса 𝑇𝐴𝐼𝐿 и помещается новый элемент в положение, на которое указывает 𝑇𝐴𝐼𝐿.

• При снятии очереди с элемента возвращается значение, на которое указывает 𝐻𝐸𝐴𝐷, и постепенно увеличивается индекс 𝐻𝐸𝐴𝐷. 102.

• Перед постановкой в очередь проверяется, заполнена ли очередь.

• Перед снятием очереди мы проверяется, пуста ли очередь.

• При инициализации первого элемента устанавливается значение 𝐻𝐸𝐴𝐷 в 0.

• При удалении последнего элемента сбрасываются значения 𝐻𝐸𝐴𝐷 и 𝑇𝐴𝐼𝐿 в -1.

Недостатком метода является его фиксированный размер. Для очередей, где элементы должны быть добавлены и удалены в середине, а не только в начале и конце буфера, реализация в виде связанного списка является предпочтительным подходом.

Обратная польская запись – форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. Обратная польская запись имеет ряд преимуществ перед инфиксной записью при выражении алгебраических формул. Одно из которых то, что инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

Обратная польская запись отлично подходит для вычисления выражений при помощи стека. Причем сам алгоритм достаточно прост. Необходимо просто прочитать обратную польскую запись слева направо. Если встречается операнд, то его нужно поместить в стек. Если встречается оператор, нужно выполнить заданную им операцию.

**Постановка задачи.**

Необходимо реализовать программу, которая выполняет следующие действия:

1. Написать программу, генерирующую различные варианты контрольных работ по теме «Структуры данных». Контрольная работа должна включать в себя минимум 10 заданий;
2. Нужно иметь задание на добавление, удаление и изменение позиций элементов;
3. Нужно иметь задание на определение корректности введенных данных, перевод в обратную польскую нотацию и наоборот, подсчет значения выражения;
4. Нужно иметь задание на определение корректности дерева, удаление, вставку элемента в дерево;
5. Нужно иметь задание на проверку связности и поиск минимального основного дерева для графа;
6. Помимо заданий нужно создать файл с ответами на задание;
7. Создать массив структур, содержащий информацию о студентах: ФИО, номер группы, номер в списке группы, количество баллов за контрольную работу (max = 200), оценка за контрольную работу (170 – отлично, 140 – хорошо, 100 – удовлетворительно;
8. Нужно иметь возможность вносить данные о новом студенте;
9. Нужно иметь возможность удалить существующие данные о студенте или студентах;
10. Реализовать вывод данных о студентах, которые получили: отлично, хорошо, удовлетворительно, неудовлетворительно;
11. Необходимо построить модель дня экзамена. Создать двусвязный список со студентами, которые сдают экзамен по дисциплине «Программирование»;
12. Необходимо придумать и создать 3 правила построения очереди на сдачу;
13. Нужно реализовать данные об оценках, чтоб они вносились либо в динамический массив, либо в структуру из 2 пункта;
14. Реализовать модель, которая позволит отобразить весь процесс прохождения экзамена от начала до конца с отметкой времени. Начало экзамена, далее процесс прохождения самого экзамена (количество времени, которое отводится на каждого студента по вашему усмотрению), конец экзамена, проставление оценок, сдача ведомости;
15. Реализовать интерактивное меню;
16. Реализовать запись в файл и считывание из него других заданий.

Исходный код программы представлен в приложении А, Б, В, Г, Д.

**Выводы.**

Закрепил и научился организовывать структуры. Получил практические навыки работы со структурами. Умею работать со стеками и с очередями, а также получил практические навыки. Закрепил материал по обратной и прямой польской нотации и как их реализовывать. Вспомнил свойства и организацию динамических массивов и двусвязных списков. Закрепил практические навыки в работе с динамическими массивами и двусвязными списками.

Приложение А

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

// Подключение заголовочного файла с библиотеками и главное меню.

#include "Header.h"

// Подключение заголовочного файла с функциями для задания 1.

#include "Header1.h"

// Подключение заголовочного файла с функциями для задания 2.

#include "Header2.h"

// Подключение заголовочного файла с функциями для задания 1.

#include "Header3.h"

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

auto exit = 0;

do

{

const auto answer = Menu();

switch (answer)

{

// Задание 1.

case 0:

{

system("cls");

MainEnter();

break;

}

// Задание 2.

case 1:

{

system("cls");

MainEnter1();

break;

}

case 2:

{

exit = 1;

}

default:;

}

} while (exit == 0);

}

Приложение Б

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

// Директива препроцессора, предназначенная для защиты от двойного подключения заголовочного файла.

#pragma once

// Не орет на Безопасность.

#define \_CRT\_SECURE\_NO\_WARNINGS

// Подключение заголовочного файла, который позволяет работать с потоками ввода/вывода.

#include <iostream>

// Подключение заголовочного файла, для создания текстового интерфейса пользователя. Тут есть \_getch().

#include <conio.h>

// Подключение заголовочного файла, в котором содержатся прототипы функции SetConsoleCP(кодовая таблица) и SetConsoleOutputCP(кодовая таблица).

#include <windows.h>

// Подключение заголовочного файла, для использования стека.

#include <stack>

// Подключение заголовочного файла, который предоставляет интерфейс для чтения/записи данных из/в файл.

#include <fstream>

// Подключение заголовочного файла, для работы со временем.

#include <chrono>

// Подключение заголовочного файла, для работа со строками.

#include <string>

// Пространство имен, чтобы не было конфиликтов повторяющихся переменных.

using namespace std;

// Пространство имен, чтобы не было конфиликтов повторяющихся переменных.

using namespace std;

// Пространство имен, чтобы не было конфиликтов между временем.

using namespace chrono;

int Menu()

{

auto key = 0;

int code;

do {

system("cls");

key = (key + 3) % 3;

if (key == 0) cout << "-> Задание 1." << endl;

else cout << " Задание 1." << endl;

if (key == 1) cout << "-> Задание 2." << endl;

else cout << " Задание 2." << endl;

if (key == 2) cout << "-> Выход" << endl;

else cout << " Выход" << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

int Menu5()

{

auto key = 0;

int code;

do {

system("cls");

key = (key + 6) % 6;

if (key == 0) cout << "-> Создаем рандомное дерево." << endl;

else cout << " Создаем рандомное дерево." << endl;

if (key == 1) cout << "-> Вывод дерева в файл." << endl;

else cout << " Вывод дерева в файл." << endl;

if (key == 2) cout << "-> Вставка." << endl;

else cout << " Вставка." << endl;

if (key == 3) cout << "-> Удаление." << endl;

else cout << " Удаление." << endl;

if (key == 4) cout << "-> Поиск." << endl;

else cout << " Поиск." << endl;

if (key == 5) cout << "-> Назад." << endl;

else cout << " Назад." << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

Приложение В

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

// Директива препроцессора, предназначенная для защиты от двойного подключения заголовочного файла.

#pragma once

// Подключение заголовочного файла.

#include "Header.h"

#include "Header3.h"

// Рандом чисел.

void Randomize(int\* (&arr), int& N)

{

cout << "Введите размерность массива: ";

while (!(cin >> N) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

// Создаем новый

arr = new int[N];

for (int i = 0; i < N; i++)

{

arr[i] = rand() % 100;

}

}

struct Elem

{

// Данные.

int data;

Elem\* next, \* prev;

};

class List

{

// Голова, хвост

Elem\* Head, \* Tail;

// Количество элементов

int Count;

public:

// Конструктор

List();

// Конструктор копирования

List(const List&);

// Деструктор

~List();

// Получить элемент списка

Elem\* GetElem(int);

// Удалить весь список

void DelAll();

// Удаление элемента, если параметр не указывается, то функция его запрашивает

void Del(int pos = 0);

// Вставка элемента, если параметр не указывается,

// то функция его запрашивает

void Insert(int pos = 0);

// Добавление в конец списка

void AddTail(int n);

// Добавление в начало списка

void AddHead(int n);

// Печать списка

void Print();

// Ответы в файл

void PrintFile(string file);

// Ответы в файл

void PrintFileNum(int pos, string file);

// Печать определенного элемента

void Print(int pos);

};

// Конструктор.

List::List()

{

// Изначально список пуст

Head = Tail = NULL;

Count = 0;

}

// Конструктор копирования.

List::List(const List& L)

{

Head = Tail = NULL;

Count = 0;

// Голова списка, из которого копируем

Elem\* temp = L.Head;

// Пока не конец списка

while (temp != 0)

{

// Передираем данные

AddTail(temp->data);

temp = temp->next;

}

}

// Деструктор.

List::~List()

{

// Удаляем все элементы

DelAll();

}

// Добавление в начало списка.

void List::AddHead(int n)

{

// новый элемент

Elem\* temp = new Elem;

// Предыдущего нет

temp->prev = 0;

// Заполняем данные

temp->data = n;

// Следующий - бывшая голова

temp->next = Head;

// Если элементы есть?

if (Head != 0)

Head->prev = temp;

// Если элемент первый, то он одновременно и голова и хвост

if (Count == 0)

Head = Tail = temp;

else

// иначе новый элемент - головной

Head = temp;

Count++;

}

// Добавление в конец списка.

void List::AddTail(int n)

{

// Создаем новый элемент

Elem\* temp = new Elem;

// Следующего нет

temp->next = 0;

// Заполняем данные

temp->data = n;

// Предыдущий - бывший хвост

temp->prev = Tail;

// Если элементы есть?

if (Tail != 0)

Tail->next = temp;

// Если элемент первый, то он одновременно и голова и хвост

if (Count == 0)

Head = Tail = temp;

else

// иначе новый элемент - хвостовой

Tail = temp;

Count++;

}

// Вставка элемента, если параметр не указывается, то функция его запрашивает.

void List::Insert(int pos)

{

// если параметр отсутствует или равен 0, то запрашиваем его

if (pos == 0)

{

while (!(cin >> pos) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

}

// Позиция от 1 до Count?

if (pos < 1 || pos > Count + 1)

{

// Неверная позиция

cout << "!!!\n";

return;

}

// Если вставка в конец списка

if (pos == Count + 1)

{

// Вставляемые данные

int data;

cout << "Введите новый номер: ";

while (!(cin >> data) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

// Добавление в конец списка

AddTail(data);

return;

}

else if (pos == 1)

{

// Вставляемые данные

int data;

cout << "Новый номер: ";

while (!(cin >> data) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

// Добавление в начало списка

AddHead(data);

return;

}

// Счетчик

int i = 1;

// Отсчитываем от головы n - 1 элементов

Elem\* Ins = Head;

while (i < pos)

{

// Доходим до элемента,

// перед которым вставляемся

Ins = Ins->next;

i++;

}

// Доходим до элемента,

// который предшествует

Elem\* PrevIns = Ins->prev;

// Создаем новый элемент

Elem\* temp = new Elem;

// Вводим данные

cout << "Введите элемент: ";

cin >> temp->data;

// настройка связей

if (PrevIns != 0 && Count != 1)

PrevIns->next = temp;

temp->next = Ins;

temp->prev = PrevIns;

Ins->prev = temp;

Count++;

}

// Удаление элемента, если параметр не указывается, то функция его запрашивает.

void List::Del(int pos)

{

// если параметр отсутствует или равен 0, то запрашиваем его

if (pos == 0)

{

cout << "Введите: ";

while (!(cin >> pos) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

}

// Позиция от 1 до Count?

if (pos < 1 || pos > Count)

{

// Неверная позиция

cout << "!!!\n";

return;

}

// Счетчик

int i = 1;

Elem\* Del = Head;

while (i < pos)

{

// Доходим до элемента,

// который удаляется

Del = Del->next;

i++;

}

// Доходим до элемента,

// который предшествует удаляемому

Elem\* PrevDel = Del->prev;

// Доходим до элемента, который следует за удаляемым

Elem\* AfterDel = Del->next;

// Если удаляем не голову

if (PrevDel != 0 && Count != 1)

PrevDel->next = AfterDel;

// Если удаляем не хвост

if (AfterDel != 0 && Count != 1)

AfterDel->prev = PrevDel;

// Удаляются крайние?

if (pos == 1)

Head = AfterDel;

if (pos == Count)

Tail = PrevDel;

// Удаление элемента

delete Del;

Count--;

}

// Печать элемента списка.

void List::Print(int pos)

{

// Позиция от 1 до Count?

if (pos < 1 || pos > Count)

{

// Неверная позиция

cout << "!!!!!\n";

return;

}

Elem\* temp;

// Определяем с какой стороны

// быстрее двигаться

if (pos <= Count / 2)

{

// Отсчет с головы

temp = Head;

int i = 1;

while (i < pos)

{

// Двигаемся до нужного элемента

temp = temp->next;

i++;

}

}

else

{

// Отсчет с хвоста

temp = Tail;

int i = 1;

while (i <= Count - pos)

{

// Двигаемся до нужного элемента

temp = temp->prev;

i++;

}

}

// Вывод элемента

cout << pos << " Элемент: ";

cout << temp->data << endl;

}

// Печать списка.

void List::Print()

{

// Если в списке присутствуют элементы, то пробегаем по нему

// и печатаем элементы, начиная с головного

if (Count != 0)

{

Elem\* temp = Head;

cout << "( ";

while (temp->next != 0)

{

cout << temp->data << ", ";

temp = temp->next;

}

cout << temp->data << " )\n";

}

}

// Удалить весь список.

void List::DelAll()

{

// Пока остаются элементы, удаляем по одному с головы

while (Count != 0)

Del(1);

}

// Получить элемент списка.

Elem\* List::GetElem(int pos)

{

Elem\* temp = Head;

// Позиция от 1 до Count?

if (pos < 1 || pos > Count)

{

// Неверная позиция

cout << " !!!\n";

return 0;

}

int i = 1;

// Ищем нужный нам элемент

while (i < pos && temp != 0)

{

temp = temp->next;

i++;

}

if (temp == 0)

return 0;

else

return temp;

}

// Ответ в файл.

void SavingData(int\* (&a), int& N, string file) // Сохранение данных

{

//Создаем поток для записи

ofstream record(file); //Открывет файл.

if (record) // если открылся

{

record << "Создание файла с ответами на задание." << endl << "Количество данных:";

record << N << endl; // записывает количество данных

for (int i = 0; i < N; i++)

{

record << a[i] << endl;

}

}

else

cout << "Ошибка открытия файла!" << endl;

record.close(); // закрываем файл

}

// Ответ в файл.

void List::PrintFile(string file)

{

//Создаем поток для записи

ofstream record(file); //Открывет файл.

if (record) // если открылся

{

record << "Создание файла с ответами на задание." << endl;

if (Count != 0)

{

Elem\* temp = Head;

record << "( ";

while (temp->next != 0)

{

record << temp->data << ", ";

temp = temp->next;

}

record << temp->data << " )\n";

}

}

else

cout << "Ошибка открытия файла!" << endl;

record.close(); // закрываем файл

}

// Ответ в файл.

void List::PrintFileNum(int pos, string file)

{

//Создаем поток для записи

ofstream record(file); //Открывет файл.

if (record) // если открылся

{

record << "Создание файла с ответами на задание." << endl;

// Позиция от 1 до Count?

if (pos < 1 || pos > Count)

{

// Неверная позиция

record << "!!!!!\n";

return;

}

Elem\* temp;

// Определяем с какой стороны

// быстрее двигаться

if (pos <= Count / 2)

{

// Отсчет с головы

temp = Head;

int i = 1;

while (i < pos)

{

// Двигаемся до нужного элемента

temp = temp->next;

i++;

}

}

else

{

// Отсчет с хвоста

temp = Tail;

int i = 1;

while (i <= Count - pos)

{

// Двигаемся до нужного элемента

temp = temp->prev;

i++;

}

}

// Вывод элемента

record << pos << " Элемент: ";

record << temp->data << endl;

}

else

cout << "Ошибка открытия файла!" << endl;

record.close(); // закрываем файл

}

// Определяет, меньшее ли преимущесто у 1-го из 2-х операторов.

boolean Advantage(char a, char b)

{

short adv1, adv2;

if (a == '\*' || a == '/')

{

adv1 = 2;

}

else if (a == '+' || a == '-')

{

adv1 = 1;

}

else

{

adv1 = 0;

}

if (b == '\*' || b == '/')

{

adv2 = 2;

}

else if (b == '+' || b == '-')

{

adv2 = 1;

}

else

{

adv2 = 0;

}

if (adv1 < adv2)

{

return true;

}

else

{

return false;

}

}

// Проверяет одинаковые ли образы символов.

boolean CheckingTheSymbol(char a, char b)

{

boolean smb1, smb2;

if (a == '\*' || a == '+' || a == '-' || a == '/')

{

smb1 = true;

}

else

{

smb1 = false;

}

if (b == '\*' || b == '+' || b == '-' || b == '/')

{

smb2 = true;

}

else

{

smb2 = false;

}

if (smb1 == smb2)

{

return true;

}

else

{

return false;

}

}

// Считывает какой тип выражения и определяет ошибки.

int ExpressionArray(char\* strinput)

{

// 0 - ошибка, 1 - простое, 2 - прямое, 3 - обратное.

int result = 0;

int i = 0;

int n = strlen(strinput);

if (n > 1)

{

// Пропускает скобки в начале.

while (strinput[i] == '(' && i != n)

{

i++;

}

// Определяет тип выражения.

char c = strinput[i];

// Если вначале идут знаки, то это прямое.

if (c == '\*' || c == '+' || c == '-' || c == '/')

{

result = 2;

}

else

{

// Ошибка, если после 1-ой скобки сразу идет вторая обратная скобка.

if (c == ')')

{

result = 0;

}

else

{

char s = strinput[n - 1];

// Если присутствуют знаки в конце, то это обратное.

if (s == '\*' || s == '+' || s == '-' || s == '/')

{

result = 3;

}

// Иначе - это простое.

else

{

result = 1;

}

}

}

// В поисках ошибок.

if (result != 0)

{

int braceopen = 0, braceclose = 0, operation = 0, operand = 0;

for (int i = 0; i < n; i++)

switch (strinput[i])

{

case '(':

{

braceopen++;

break;

}

case ')':

{

braceclose++;

break;

}

case '\*': case '+': case '-': case '/':

{

operation++;

break;

}

default:

{

operand++;

break;

}

}

// Несовпадение количества открывающих и закрывающих скобок.

if (braceopen != braceclose)

{

result = 0;

}

// Несовпадение операндов и на один больше операторов.

if (operand != (operation + 1))

{

result = 0;

}

// Подряд идущие одинаковые типы в простой записи.

if (result == 1)

{

char\* temp = new char[n];

int x = -1;

for (int i = 0; i < n; i++)

{

if (strinput[i] != '(' && strinput[i] != ')')

{

temp[++x] = strinput[i];

}

}

for (int i = 0; i < x - 1; i++)

{

if (CheckingTheSymbol(temp[i], temp[i + 1]))

{

result = 0;

}

}

}

}

}

else

{

result = 0;

}

return result;

}

// Преобразование из простой в обратную(массив).

char\* SimpleInReverseArray(char\* simple)

{

int n = strlen(simple);

char\* operand = new char[n];

int arraycount = -1;

char\* result = new char[n + 1];

int count = 0;

for (int i = 0; i < n; i++)

{

operand[i] = '\0';

}

for (int i = 0; i < n; i++)

{

// Если знак операции.

if (simple[i] == '\*' || simple[i] == '+' || simple[i] == '-' || simple[i] == '(' || simple[i] == '/')

{

if (arraycount == -1 || (Advantage(operand[arraycount], simple[i])))

{

// Если приоритет меньше, то в стек.

operand[++arraycount] = simple[i];

}

else

{

while (arraycount == -1 && !Advantage(operand[arraycount], simple[i]))

{

result[count++] = operand[arraycount];

arraycount--;

}

operand[++arraycount] = simple[i];

}

}

// Если закрывающая скобка то, содержимое стека до скобки - в строку.

else if (simple[i] == ')')

{

while (operand[arraycount] != '(')

{

result[count++] = operand[arraycount];

arraycount--;

}

arraycount--;

}

else

{

result[count++] = simple[i];

}

}

while (arraycount == -1)

{

result[count++] = operand[arraycount];

arraycount--;

}

result[count++] = '\0';

return result;

delete result;

delete operand;

}

// Рекурсивное понижение, для прямой нотации

string Recursion(string s)

{

string res;

int n = s.length();

if (s[0] == '(')

{

int pos = s.find(")");

string sub;

if (pos < s.length() - 1) sub = s.substr(pos + 2, s.length());

int nn = sub.length();

if (nn >= 3)

{

res += s.substr(pos + 1, 1)

+ Recursion(s.substr(1, pos - 1))

+ Recursion(sub);

}

else

{

res += s.substr(pos + 1, 1)

+ Recursion(s.substr(1, pos - 1))

+ sub;

}

}

else

{

if (n > 3)

{

res += s.substr(1, 1) + s.substr(0, 1) + Recursion(s.substr(2, s.length()));

}

else

{

res += s.substr(1, 1) + s.substr(0, 1) + s.substr(2, s.length());

}

}

return res;

}

// Преобразование из простой в прямую(массив).

char\* SimpleInStraightArray(char\* simple)

{

int n = strlen(simple);

char\* result = new char[n + 1];

string str = Recursion(string(simple));

for (int i = 0; i < str.length(); i++) result[i] = str[i];

result[str.length()] = '\0';

return result;

delete result;

}

// Проверка, не является ли числом.

boolean NotNumber(char ch)

{

switch (ch)

{

case '0': case '1': case '2': case '3': case '4':

case '5': case '6': case '7': case '8': case '9':

return false;

break;

default: return true;

}

}

// Калькулятор для вычисления выражения(массив).

float CalculatorArray(char\* reverse, int\* variable)

{

int len = strlen(reverse);

char\* operand = new char[len];

float result = 0;

int arraycount = -1;

for (int i = 0; i < len; i++)

{

operand[i] = '\0';

}

int v = 0;

for (int i = 0; i < len; i++)

{

if (reverse[i] == '\*' || reverse[i] == '+' || reverse[i] == '-' || reverse[i] == '/')

{

int number1 = 0;

int number2 = 0;

number1 = operand[arraycount];

arraycount--;

number2 = operand[arraycount];

arraycount--;

if (reverse[i] == '\*')

{

operand[++arraycount] = (number1 \* number2);

}

if (reverse[i] == '+')

{

operand[++arraycount] = (number1 + number2);

}

if (reverse[i] == '-')

{

operand[++arraycount] = (number1 - number2);

}

if (reverse[i] == '/')

{

operand[++arraycount] = (number1 / number2);

}

}

else

{

int number = 0;

char ch = reverse[i];

if (!NotNumber(reverse[i]))

{

number = atoi(&ch);

}

else

{

number = variable[v++];

}

operand[++arraycount] = number;

}

}

return operand[arraycount];

}

// Определяет размер файла.

int FileSize(string path)

{

ifstream file(path);

file.seekg(0, ios\_base::end);

int filesize = file.tellg();

file.close();

return filesize;

}

// Загружает выражение из файла.

void LoadFile(char\* buffer)

{

ifstream file("Expression.txt");

int filesize = FileSize("Expression.txt");

if (filesize > 0)

{

file.get(buffer, filesize + 1);

file.close();

}

else

{

cout << "Там походу файла нет!" << endl;

}

}

// Преобразование из простой в прямую.

char\* SimpleInStraight(char\* simple)

{

int n = strlen(simple);

char\* result = new char[n + 1];

string str = Recursion(string(simple));

for (int i = 0; i < str.length(); i++)

{

result[i] = str[i];

}

result[str.length()] = '\0';

return result;

delete result;

}

// Преобразование из простой в обратную, для файла с ответами

string SimpleInReverseFile(string simple)

{

stack <char> operand;

int n = simple.length();

string result;

string str = "Исходное выражение: " + simple + "\n";

int count = 0;

for (int i = 0; i < n; i++)

{

if (simple[i] == '\*' || simple[i] == '+' || simple[i] == '-' || simple[i] == '(' || simple[i] == '/')

{

if (operand.empty() || (Advantage(operand.top(), simple[i])))

// Если приоритет меньше, то в стек.

operand.push(simple[i]);

else

{

while (!operand.empty() && !Advantage(operand.top(), simple[i]))

{

str = str + "Извлекаем из стека оператор и добавляем к выражению: " + operand.top() + "\n" +

+"Выражение: " + result + "\n";

result += operand.top();

operand.pop();

}

str = str + "Добавляем оператор в стек: " + simple[i] + "\n Выражение: " + result + "\n";

operand.push(simple[i]);

}

}

// Если закрывающая скобка, то содержимое стека до скобки - в строку.

else if (simple[i] == ')')

{

while (operand.top() != '(')

{

str = str + "Извлекаем из стека символы до скобки: " + operand.top() + "\n";

result += operand.top();

operand.pop();

}

operand.pop();

str = str + "Выражение: " + result + "\n";

}

else

{

result += simple[i];

str = str + "Добавляем операнд в выражение: " + simple[i] + "\n Выражение: " + result + "\n";

}

}

while (!operand.empty())

{

str = str + "Пока стека не пуст добавлем его содержимое в выражение: " + operand.top()

+ "\n Выражение: " + result + "\n";

result[count++] = operand.top();

operand.pop();

}

return str + " Ответ: " + result;

system("simple.txt");

}

// Рекурсивный спуск, прямой нотации для файла с ответами.

string Recursionanswer(string s, string\* p)

{

string res;

int n = s.length();

if (s[0] == '(')

{

\*p = \*p + "Найдена открывающаяся скобка \n";

int pos = s.find(")");

string sub;

if (pos < s.length() - 1) sub = s.substr(pos + 2, s.length());

int nn = sub.length();

if (nn >= 3)

{

\*p = \*p + "Вызываем рекурсию выражения в скобках: " + s.substr(1, pos - 1) + "\n"

+ "и рекурсию всего остального выражения: " + sub;

res += s.substr(pos + 1, 1)

+ Recursionanswer(s.substr(1, pos - 1), p)

+ Recursionanswer(sub, p);

}

else

{

string rek = Recursionanswer(s.substr(1, pos - 1), p);

\*p = \*p + "Добавляем в выражение подстроку: " + s.substr(pos + 1, 1) + "\n"

+ "Добавляем в выражение рекурсию: " + s.substr(1, pos - 1) + "\n"

+ "Добавляем в выражение оставшуюся подстроку: " + sub + "\n";

res += s.substr(pos + 1, 1)

+ rek

+ sub;

}

}

else

{

if (n > 3)

{

string rek = Recursionanswer(s.substr(2, s.length()), p);

\*p = \*p + "Скобок нет, добавляем в выражение два оператора: " + s.substr(1, 1) + s.substr(0, 1) +

"\n Вызываем рекурсию оствшегося выражения: " + s.substr(2, s.length()) + "\n";

res += s.substr(1, 1) + s.substr(0, 1) + rek;

}

else

{

\*p = \*p + "Добавляем в выражение оставшиеся 3 символа в нужной последовательности \n";

res += s.substr(1, 1) + s.substr(0, 1) + s.substr(2, s.length());

}

}

return res;

}

// Преобразование из простой в прямую, для файла с ответами.

string SimpleInStraightFile(string simple)

{

string str;

string\* s = &str;

string result = Recursionanswer(simple, s);

return "Исходное выржанеие: " + simple + "\n" + str + "Ответ: " + result + "\n";

}

// Преобразование из простой в обратную.

char\* SimpleInReverse(char\* simple)

{

stack <char> operand;

int n = strlen(simple);

char\* result = new char[n + 1];

int count = 0;

for (int i = 0; i < n; i++)

{

if (simple[i] == '\*' || simple[i] == '+' || simple[i] == '-' || simple[i] == '(' || simple[i] == '/')

{

if (operand.empty() || (Advantage(operand.top(), simple[i])))

{

operand.push(simple[i]);

}

else

{

while (!operand.empty() && !Advantage(operand.top(), simple[i]))

{

result[count++] = operand.top();

operand.pop();

}

operand.push(simple[i]);

}

}

else if (simple[i] == ')')

{

while (operand.top() != '(')

{

result[count++] = operand.top();

operand.pop();

}

operand.pop();

}

// Если операнд, то помещаем в строку.

else

{

result[count++] = simple[i];

}

}

while (!operand.empty())

{

result[count++] = operand.top();

operand.pop();

}

result[count++] = '\0';

return result;

delete result;

}

// Преобразование из обратной в простую.

char\* ReverseInSimple(char\* reverse)

{

int n = strlen(reverse);

char\* result = new char[n \* 2];

int count = 0;

stack <char> operand;

for (int i = 0; i < n; i++)

{

if (reverse[i] == '\*' || reverse[i] == '+' || reverse[i] == '-' || reverse[i] == '/')

{

char temp = operand.top();

operand.pop();

if (!operand.empty())

{

result[count++] = operand.top();

operand.pop();

}

result[count++] = reverse[i];

result[count++] = temp;

if (i + 1 < n)

{

// Cкобки - если следующий символ знак, то все что ранее - в скобки.

char next = reverse[i + 1];

if (!(next == '\*' || next == '+'

|| next == '-' || next == '/'))

{

char\* tempchar = new char[count + 1];

for (int x = 0; x < count; x++) tempchar[x] = result[x];

result[0] = '(';

for (int x = 0; x < count; x++) result[x + 1] = tempchar[x];

count++;

result[count++] = ')';

}

}

}

else operand.push(reverse[i]);

}

result[count++] = '\0';

return result;

delete result;

}

// Преобразование из прямой в простую.

char\* StraightInSimple(char\* reverse)

{

int n = strlen(reverse);

char\* result = new char[n \* 2];

int count = 0;

char bracket = 0;

stack <char> operand;

for (int i = 0; i < n; i++)

{

if (reverse[i] == '\*' || reverse[i] == '+' || reverse[i] == '-' || reverse[i] == '/')

{

operand.push(reverse[i]);

}

else

{

result[count++] = reverse[i];

if (bracket != 0)

{

result[count++] = bracket;

bracket = 0;

}

if (!operand.empty())

{

result[count++] = operand.top();

operand.pop();

// Cкобки, если в стеке есть еще оператор и приоритет больше.

if (!operand.empty() && Advantage(result[count - 1], operand.top()))

{

char\* tempchar = new char[count + 1];

for (int x = 0; x < count; x++)

{

tempchar[x] = result[x];

}

result[0] = '(';

for (int x = 0; x < count; x++)

{

result[x + 1] = tempchar[x];

}

count++;

bracket = ')';

}

}

}

}

result[count++] = '\0';

return result;

delete result;

}

// Определяет тип выражения и ищет ошибки.

int CheckExpression(char\* strinput)

{

// 0 - ошибка, 1 - простое, 2 - прямое, 3 - обратное.

int result = 0;

int i = 0;

int n = strlen(strinput);

if (n > 1)

{

// Пропускает скобки в начале.

while (strinput[i] == '(' && i != n)

i++;

// Определяет тип выражения.

char c = strinput[i];

// Если вначале идут знаки, то это прямое.

if (c == '\*' || c == '+' || c == '-' || c == '/')

{

result = 2;

}

else

{

// Ошибка, если после '(' сразу идет ')'.

if (c == ')')

{

result = 0;

}

else

{

char s = strinput[n - 1];

// Если знаки в конце, то это обратное.

if (s == '\*' || s == '+' || s == '-' || s == '/')

{

result = 3;

}

// По остаточному принципу - это простое.

else

{

result = 1;

}

}

}

// Поиск ошибок.

if (result != 0)

{

int braceopen = 0, braceclose = 0, operation = 0, operand = 0;

for (int i = 0; i < n; i++)

switch (strinput[i])

{

case '(':

braceopen++;

break;

case ')':

braceclose++;

break;

case '\*': case '+': case '-': case '/':

operation++;

break;

default:

operand++;

break;

}

// Несовпадение количества открывающих и закрывающих скобок.

if (braceopen != braceclose)

{

result = 0;

}

// Несовпадение операндов и на один больше операторов.

if (operand != (operation + 1))

{

result = 0;

}

// Подряд идущие одинаковые типы в простой записи.

if (result == 1)

{

char\* temp = new char[n];

int x = -1;

for (int i = 0; i < n; i++)

{

if (strinput[i] != '(' && strinput[i] != ')')

{

temp[++x] = strinput[i];

}

}

for (int i = 0; i < x - 1; i++)

{

if (CheckingTheSymbol(temp[i], temp[i + 1]))

{

result = 0;

}

}

}

}

}

else

{

result = 0;

}

return result;

}

// Выделяет имя переменной под номером p.

char SelectVariable(char\* c, int p)

{

char ch = 0;

int len = strlen(c);

int i = 0;

int count = 0;

while (i < len && count != p)

{

if (!(c[i] == '\*' || c[i] == '+' || c[i] == '-' || c[i] == '/' || c[i] == '(' || c[i] == ')'))

{

// Разбил на два условия для читабельности.

if (NotNumber(c[i]))

{

count++;

if (count == p)

{

ch = c[i];

}

}

}

i++;

}

return ch;

}

// Проверка на дурака. Можно ввести только цифры и знак минус.

int CheckFool()

{

char ch;

string str;

do

{

ch = \_getch();

switch (ch)

{

case '-': case '0': case '1': case '2': case '3': case '4':

case '5': case '6': case '7': case '8': case '9':

if ((ch == '-' && str == "") || ch != '-')

{

str += ch;

cout << ch;

}

break;

}

} while (ch != 13 || str == "");

return stoi(str);

}

// Калькулятор - вычисляет выражение.

float Calculator(char\* reverse, int\* variable)

{

stack <float> operand;

int len = strlen(reverse);

float result = 0;

int v = 0;

for (int i = 0; i < len; i++)

{

if (reverse[i] == '\*' || reverse[i] == '+' || reverse[i] == '-' || reverse[i] == '/')

{

int number1 = 0;

int number2 = 0;

number1 = operand.top();

operand.pop();

number2 = operand.top();

operand.pop();

if (reverse[i] == '\*')

{

operand.push(number1 \* number2);

}

if (reverse[i] == '+')

{

operand.push(number1 + number2);

}

if (reverse[i] == '-')

{

operand.push(number1 - number2);

}

if (reverse[i] == '/')

{

operand.push(number1 / number2);

}

}

else

{

int number = 0;

char ch = reverse[i];

if (!NotNumber(reverse[i]))

{

number = atoi(&ch);

}

else

{

number = variable[v++];

}

operand.push(number);

}

}

return operand.top();

}

// Вывод польской прямой нотации.

void OutputStraight(char\* expression)

{

cout << "Польская прямая: " << endl;

int n = strlen(expression);

char\* straight = new char[n + 1];

straight = SimpleInStraightArray(expression);

straight = SimpleInStraight(expression);

int i = 0;

while (straight[i] != '\0') cout << straight[i++];

cout << endl;

}

// Вывод польской обратной нотации.

char\* OutputReverse(char\* expression)

{

cout << "Польская обратная: " << endl;

int n = strlen(expression);

char\* reverse = new char[n + 1];

reverse = SimpleInReverseArray(expression);

reverse = SimpleInReverse(expression);

int i = 0;

while (reverse[i] != '\0')

{

cout << reverse[i++];

}

cout << endl;

return reverse;

}

// Обработка выражения.

void ProcessingExpression(char\* expression)

{

int n = strlen(expression);

char\* simple = new char[n \* 2];

char\* reverse = new char[n \* 2];

for (int i = 0; i < n; i++) cout << expression[i];

cout << endl;

cout << "Тип записи: " << endl;

int check;

check = CheckExpression(expression);

check = ExpressionArray(expression);

if (check == 0)

{

cout << "Некорректное выражение" << endl;

}

if (check == 1)

{

cout << "Простое выражение" << endl;

}

if (check == 2)

{

cout << "Прямая польская запись" << endl;

}

if (check == 3)

{

cout << "Обратная польская" << endl;

}

// Простая.

if (check == 1)

{

simple = expression;

reverse = OutputReverse(expression);

OutputStraight(expression);

}

// Прямая.

if (check == 2)

{

cout << "Простое выражение: " << endl;

simple = StraightInSimple(expression);

int i = 0;

while (simple[i] != '\0') cout << simple[i++];

cout << endl;

reverse = OutputReverse(simple);

}

// Обратная.

if (check == 3)

{

cout << "Простая запись: ";

simple = ReverseInSimple(expression);

reverse = expression;

int i = 0;

while (simple[i] != '\0') cout << simple[i++];

cout << endl;

OutputStraight(simple);

}

cout << endl;

if (check != 0)

{

int i = 1;

int\* variable = new int[n];

while (SelectVariable(simple, i) != 0)

{

cout << "Введите значение переменной " << SelectVariable(simple, i) << " : ";

variable[i - 1] = CheckFool();

cout << endl;

i++;

}

float calc;

calc = CalculatorArray(reverse, variable);

calc = Calculator(reverse, variable);

cout << "Результат вычислений: " << endl;

cout << calc << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

// Обработка файла.

void ProcessingFile()

{

cout << "Выражение из файла: " << endl;

int n = FileSize("Expression.txt");

char\* expression = new char[n + 1];

LoadFile(expression);

ProcessingExpression(expression);

}

// Меняет номер оператора на его символ.

string ChangesOperator(int n)

{

switch (n)

{

case 0: return "+"; break;

case 1: return "-"; break;

case 2: return "\*"; break;

case 3: return "/"; break;

}

}

// Меняет номер переменной её буквенным обозначением.

string ChangesVariable(int n)

{

switch (n)

{

case 0: return "X"; break;

case 1: return "Y"; break;

case 2: return "Z"; break;

case 3: return "A"; break;

case 4: return "B"; break;

case 5: return "C"; break;

case 6: return "D"; break;

case 7: return "E"; break;

case 8: return "F"; break;

case 9: return "G"; break;

case 10: return "H"; break;

case 11: return "I"; break;

case 12: return "J"; break;

}

}

// Создание тест файла с файлом ответов.

void Test(int n)

{

for (int i = 1; i <= n; i++)

{

ofstream file("test.txt", ios\_base::app);

ofstream answer("answers.txt", ios\_base::app);

int varcount = rand() % 5 + 2;

int len = rand() % 10 + 10;

string s = "";

string sk1 = "", sk2 = "";

int skobka = 0;

for (int i = 0; i < len; i++)

{

string oper = ChangesOperator(rand() % 4);

string var = ChangesVariable(rand() % varcount);

if (i != len - 1 && skobka == 0 && (oper == "+" || oper == "-"))

{

if (rand() % 2) skobka = 2;

}

if (skobka == 2)

{

sk1 = "(";

skobka = 1;

}

else if (skobka == 1)

{

if ((i == len - 1) || rand() % 2)

{

sk2 = ")";

skobka = 0;

}

}

s += sk1 + var + sk2;

if (i != len - 1) s += oper;

sk1 = "";

sk2 = "";

}

file << "Задание № " << i << " Простая запись: ";

file << s << endl;

answer << "Решение задания № " << i << " Обратная нотация. \n" << SimpleInReverseFile(s) << endl;

string str = SimpleInStraightFile(s);

answer << "Решение задания № " << i << " Прямая нотация. \n" << str << endl;

file.close();

answer.close();

}

cout << "Файл test.txt сохранен" << endl;

cout << "Файл answer.txt сохранен" << endl;

}

// Меню 1 задания.

int MenuEnter() { //вывод меню, управляемого стрелками

auto key = 0;

int code;

do {

system("cls");

key = (key + 4) % 4;

if (key == 0) cout << "-> Двусвязные" << endl;

else cout << " Двусвязные" << endl;

if (key == 1) cout << "-> Польские нотации" << endl;

else cout << " Польские нотации" << endl;

if (key == 2) cout << "-> Дерево" << endl;

else cout << " Дерево" << endl;

if (key == 3) cout << "-> Главное меню" << endl;

else cout << " Главное меню" << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

// Меню 1 задания.

int MenuEnter2()

{

auto key = 0;

int code;

do {

system("cls");

key = (key + 5) % 5;

if (key == 0) cout << "-> Формирование двусвязного списка с помощью рандомных чисел." << endl;

else cout << " Формирование двусвязного списка с помощью рандомных чисел." << endl;

if (key == 1) cout << "-> Вставка." << endl;

else cout << " Вставка." << endl;

if (key == 2) cout << "-> Удаление по индексу." << endl;

else cout << " Удаление по индексу." << endl;

if (key == 3) cout << "-> Получение по индексу." << endl;

else cout << " Получение по индексу." << endl;

if (key == 4) cout << "-> Назад" << endl;

else cout << " Назад" << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

// Меню 1 задания.

int MenuEnter3()

{

auto key = 0;

int code;

do {

system("cls");

key = (key + 4) % 4;

if (key == 0) cout << "-> Ввод из файла." << endl;

else cout << " Ввод из файла." << endl;

if (key == 1) cout << "-> Ввод вручную." << endl;

else cout << " Ввод вручную." << endl;

if (key == 2) cout << "-> Тест, вывод в файл с ответами." << endl;

else cout << " Тест, вывод в файл с ответами." << endl;

if (key == 3) cout << "-> Назад" << endl;

else cout << " Назад" << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

// Функция работы меню 1 задания.

void MainEnter()

{

List L;

int n = 0;

int\* a = new int[n];

auto exit = 0;

do {

const auto answer = MenuEnter();

switch (answer)

{

case 0:

{

auto exit = 0;

do {

const auto answer = MenuEnter2();

switch (answer)

{

case 0:

{

system("cls");

Randomize(a, n);

for (int i = 0; i < n; i++)

{

L.AddHead(a[i]);

SavingData(a, n, "file.txt");

}

// Распечатка списка

cout << "List L:\n";

L.Print();

cout << endl;

system("file.txt");

system("cls");

break;

}

case 1:

{

cout << "Введите местечко куда его вставить: ";

// Вставка элемента в список

L.Insert();

// Распечатка списка

cout << "List L:\n";

L.Print();

L.PrintFile("file.txt");

system("file.txt");

system("cls");

break;

}

case 2:

{

L.Del();

L.Print();

L.PrintFile("file.txt");

system("file.txt");

break;

}

case 3:

{

cout << "Введите номер элемента:" << endl;

while (!(cin >> n) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

L.Print(n);

L.PrintFileNum(n, "file.txt");

system("file.txt");

system("cls");

break;

}

case 4:

{

exit = 1;

}

default:;

}

} while (exit == 0);

break;

}

// Нотации

case 1:

{

auto exit = 0;

do {

const auto answer = MenuEnter3();

switch (answer)

{

case 0:

{

system("cls");

ProcessingFile();

system("pause");

system("cls");

break;

}

case 1:

{

cout << "\t\t";

system("cls");

cout << "Введите выражения вручную: ";

char expression[300];

cin >> expression;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "Ваше выражение:" << endl;

ProcessingExpression(expression);

system("pause");

system("cls");

break;

}

case 2:

{

system("cls");

cout << "Введите количество необходимых проверок: ";

int t;

cin >> t;

Test(t);

system("answers.txt");

system("test.txt");

system("pause");

system("cls");

break;

}

case 3:

{

exit = 1;

}

default:;

}

} while (exit == 0);

break;

}

// Дерево

case 2:

{

auto exit = 0;

ofstream in\_file;

int enter, NodeN = 0;

Node\* origin = nullptr;

do

{

const auto answer = Menu5();

switch (answer)

{

// Рандом

case 0:

{

system("cls");

cout << "Введите количество элементов дерева: ";

cin >> NodeN;

if ((NodeN <= 0) || (NodeN > 300))

{

break;

}

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

RandomTree(NodeN, origin, 300);

EquallyTree(origin);

cout << endl;

system("Pause");

system("cls");

break;

}

// Вывод

case 1:

{

in\_file.open("tree.txt");

TreeOutput(in\_file, origin, NodeN, 1);

in\_file.close();

system("tree.txt");

system("cls");

break;

}

// Вставка

case 2:

{

system("cls");

cout << "Значение узла: ";

int val = 0;

cin >> val;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

if (ApendTree(origin, val, NodeN))

{

system("pause");

system("cls");

}

else

{

cout << "Узел с таким значением уже существует.\n";

system("pause");

system("cls");

}

break;

}

// Удаление

case 3:

{

system("cls");

cout << "Узел со значением, чтобы удалить: ";

int val = 0;

cin >> val;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

if (DeleteFromTree(origin, val))

{

NodeN--;

system("pause");

system("cls");

}

else

{

cout << "Узел со значением " << val << " не найден\n";

system("pause");

system("cls");

}

break;

}

// Поиск

case 4:

{

system("cls");

cout << "Найти узел со значением: ";

int val = 0;

cin >> val;

cin.clear();

cin.ignore(cin.rdbuf()->in\_avail());

Node\* search = LookNode(origin, val);

if (search->data != val)

{

cout << "Узел с таким значением не найден.\n";

system("pause");

system("cls");

}

else

{

cout << "Узел найден.\n";

system("pause");

system("cls");

}

break;

}

case 5:

{

exit = 1;

}

default:;

}

} while (exit == 0);

break;

}

// Выход

case 3:

{

exit = 1;

}

default:;

}

} while (exit == 0);

}

Приложение Г

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

// Директива препроцессора, предназначенная для защиты от двойного подключения заголовочного файла.

#pragma once

// Подключение заголовочного файла.

#include "Header.h"

// Структура ФИО.

struct Initial

{

string surname,

name,

patronymic;

};

// Структура номера группы.

struct NumberGroup

{

short NomerGrup;

};

// Структура номера студента в списке группы.

struct NumberListOfGroup

{

short NomerStudenta;

};

// Структура количества баллов.

struct AssessmentsStudents

{

short Bal;

};

// Структура отметки внесения или изменения данных.

struct Time

{

time\_t day, mon, year, hour, min;

};

// Структура данных.

struct Data

{

Initial \_initial;

NumberGroup \_numbergroup;

NumberListOfGroup \_numberlistofgroup;

AssessmentsStudents \_assesmentsstudents;

Time \_time;

};

// Создаем новую запись о студенте.

void DataEntry(Data\* (&d), int& n) // (&d) - Сылка на массив для выделения новой памяти // int& - Сылка на количество элементов

{

cout << "Введите количество студентов для новой записи данных: ";

while (!(cin >> n) || (cin.peek() != '\n')) // cin.peek - проверяет, достигнут ли конец потока входного файла

{

cin.clear();

while (cin.get() != '\n'); //проверяет, является ли строка cin.get() пустой строкой

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

//выделяем память

d = new Data[n];

for (int i = 0; i < n; i++)

{

time\_t now;

struct tm nowLocal;

now = time(NULL);

nowLocal = \*localtime(&now);

d[i].\_time.day = nowLocal.tm\_mday;

d[i].\_time.mon = nowLocal.tm\_mon + 1;

d[i].\_time.year = nowLocal.tm\_year + 1900;

d[i].\_time.hour = nowLocal.tm\_hour;

d[i].\_time.min = nowLocal.tm\_min;

cout << "Введите ФИО: ";

while (!(cin >> d[i].\_initial.surname) || (d[i].\_initial.surname <= "А") || (d[i].\_initial.surname >= "Я") && (d[i].\_initial.surname >= "а") || (d[i].\_initial.surname >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[i].\_initial.name) || (d[i].\_initial.name <= "А") || (d[i].\_initial.name >= "Я") && (d[i].\_initial.name >= "а") || (d[i].\_initial.name >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[i].\_initial.patronymic) || (d[i].\_initial.patronymic <= "А") || (d[i].\_initial.patronymic >= "Я") && (d[i].\_initial.patronymic >= "а") || (d[i].\_initial.patronymic >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер группы: ";

while (!(cin >> d[i].\_numbergroup.NomerGrup) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер студента в списке группы: ";

while (!(cin >> d[i].\_numberlistofgroup.NomerStudenta) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите количество баллов: ";

while (!(cin >> d[i].\_assesmentsstudents.Bal) || (d[i].\_assesmentsstudents.Bal > 200))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз. Оценивание( max = 200):" << endl;

}

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

// Выводим.

void Print(Data\* d, int n)

{

for (int i = 0; i < n; i++)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: "

<< d[i].\_initial.surname << endl

<< d[i].\_initial.name << endl

<< d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

if (d[i].\_assesmentsstudents.Bal >= 170)

{

cout << "Оценка: \"Отлично\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 170 && d[i].\_assesmentsstudents.Bal >= 140)

{

cout << "Оценка: \"Хорошо\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 140 && d[i].\_assesmentsstudents.Bal >= 100)

{

cout << "Оценка: \"Удовлетворительно\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 100)

{

cout << "Оценка: \"Неудовлетворительно\"" << endl;

}

cout << "Время: "

<< d[i].\_time.day << "."

<< d[i].\_time.mon << "."

<< d[i].\_time.year << "г."

<< d[i].\_time.hour << ":"

<< d[i].\_time.min << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

// Изменяем данные о студенте.

void DataChange(Data\* (&d), int n)

{

int \_n;

cout << "Введите номер данных студента (От 1 до " << n << "): ";

while (!(cin >> \_n) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

system("cls");

if (\_n > 0)

{

\_n--;// Так как индексация идет от нуля в С++ а у нас от 1

time\_t now;

struct tm nowLocal;

now = time(NULL);

nowLocal = \*localtime(&now);

d[\_n].\_time.day = nowLocal.tm\_mday;

d[\_n].\_time.mon = nowLocal.tm\_mon + 1;

d[\_n].\_time.year = nowLocal.tm\_year + 1900;

d[\_n].\_time.hour = nowLocal.tm\_hour;

d[\_n].\_time.min = nowLocal.tm\_min;

cout << "Введите ФИО: ";

while (!(cin >> d[\_n].\_initial.surname) || (d[\_n].\_initial.surname <= "А") || (d[\_n].\_initial.surname >= "Я") && (d[\_n].\_initial.surname >= "а") || (d[\_n].\_initial.surname >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[\_n].\_initial.name) || (d[\_n].\_initial.name <= "А") || (d[\_n].\_initial.name >= "Я") && (d[\_n].\_initial.name >= "а") || (d[\_n].\_initial.name >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[\_n].\_initial.patronymic) || (d[\_n].\_initial.patronymic <= "А") || (d[\_n].\_initial.patronymic >= "Я") && (d[\_n].\_initial.patronymic >= "а") || (d[\_n].\_initial.patronymic >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер группы: ";

while (!(cin >> d[\_n].\_numbergroup.NomerGrup) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер студента в списке группы: ";

while (!(cin >> d[\_n].\_numberlistofgroup.NomerStudenta) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Количество баллов: ";

while (!(cin >> d[\_n].\_assesmentsstudents.Bal) || (d[\_n].\_assesmentsstudents.Bal > 200))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз. Оценивание(max = 200):" << endl;

}

cout << "Время: "

<< d[\_n].\_time.day << " " << d[\_n].\_time.mon << " " << d[\_n].\_time.year << " " << d[\_n].\_time.hour << " " << d[\_n].\_time.min << endl;

system("cls");

cout << "Данные изменены!" << endl;

}

else

cout << "Номер введен не верно!" << endl;

}

// Сохраняем.

void SavingData(Data\* d, int n, string students)

{

//Создаем поток для записи

ofstream record(students); //Открывет файл.

if (record) // если открылся

{

record << n << endl; // записывает количество данных

for (int i = 0; i < n; i++)

{

record << d[i].\_initial.surname << endl;;

record << d[i].\_initial.name << endl;;

record << d[i].\_initial.patronymic << endl;;

record << d[i].\_numbergroup.NomerGrup << endl;;

record << d[i].\_numberlistofgroup.NomerStudenta << endl;;

record << d[i].\_assesmentsstudents.Bal << endl;

if (i < n - 1)//если элемент не последний то отступ на следующую строчку

record << d[i].\_time.day << d[i].\_time.mon << d[i].\_time.year << d[i].\_time.hour << d[i].\_time.min << endl;

else // иначе без отступа

record << d[i].\_time.day << d[i].\_time.mon << d[i].\_time.year << d[i].\_time.hour << d[i].\_time.min;

}

}

else

cout << "Ошибка открытия файла!" << endl;

record.close(); // закрываем файл

}

// Время изменений.

void Times(Data\* d, int n)

{

int enter = 0;

int T;

cout << "Введите день ";

while (!(cin >> T) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

for (int i = 0; i < n; i++) {

if (d[i].\_time.day == T)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: " << d[i].\_initial.surname << d[i].\_initial.name << d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

if (d[i].\_assesmentsstudents.Bal >= 170)

{

cout << "Оценка: \"Отлично\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 170 && d[i].\_assesmentsstudents.Bal >= 140)

{

cout << "Оценка: \"Хорошо\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 140 && d[i].\_assesmentsstudents.Bal >= 100)

{

cout << "Оценка: \"Удовлетворительно\"" << endl;

}

if (d[i].\_assesmentsstudents.Bal < 100)

{

cout << "Оценка: \"Неудовлетворительно\"" << endl;

}

cout << "Время: "

<< d[i].\_time.day << "." << d[i].\_time.mon << "." << d[i].\_time.year << "г." << " " << d[i].\_time.hour << ":" << d[i].\_time.min << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

else

{

cout << "Студента под номером " << i + 1 << " в этот день не записан!" << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

}

}

// Копирует весь масив.

void Copy(Data\* (&d\_n), Data\* (&d\_o), int n)

{

for (int i = 0; i < n; i++)

{

d\_n[i] = d\_o[i]; //присваивание даных массиву

}

}

// Добавление студента.

void AddDate(Data\* (&d), int& n)

{

//временный массив данных

Data\* buf;

//выделяем данные

buf = new Data[n];

//сохраняем в временный массив

Copy(buf, d, n);

//выделяем новую память на 1 больше. Так как добавили

n++;

d = new Data[n];

Copy(d, buf, --n);// возвращаем данные которые были в buf в наш основной массив

time\_t now;

struct tm nowLocal;

now = time(NULL);

nowLocal = \*localtime(&now);

d[n].\_time.day = nowLocal.tm\_mday;

d[n].\_time.mon = nowLocal.tm\_mon + 1;

d[n].\_time.year = nowLocal.tm\_year + 1900;

d[n].\_time.hour = nowLocal.tm\_hour;

d[n].\_time.min = nowLocal.tm\_min;

cout << "Введите ФИО: ";

while (!(cin >> d[n].\_initial.surname) || (d[n].\_initial.surname <= "А") || (d[n].\_initial.surname >= "Я") && (d[n].\_initial.surname >= "а") || (d[n].\_initial.surname >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[n].\_initial.name) || (d[n].\_initial.name <= "А") || (d[n].\_initial.name >= "Я") && (d[n].\_initial.name >= "а") || (d[n].\_initial.name >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

while (!(cin >> d[n].\_initial.patronymic) || (d[n].\_initial.patronymic <= "А") || (d[n].\_initial.patronymic >= "Я") && (d[n].\_initial.patronymic >= "а") || (d[n].\_initial.patronymic >= "я"))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер группы: ";

while (!(cin >> d[n].\_numbergroup.NomerGrup) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите номер студента в списке группы: ";

while (!(cin >> d[n].\_numberlistofgroup.NomerStudenta) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

cout << "Введите количество баллов: " << endl;

while (!(cin >> d[n].\_assesmentsstudents.Bal) || (d[n].\_assesmentsstudents.Bal > 200))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз. Оценивание(max = 200):" << endl;

}

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

delete[]buf;

}

// Копирует один элемент массива.

void Copy(Data& d\_n, Data& d\_o)

{

d\_n.\_initial.surname = d\_o.\_initial.surname;

d\_n.\_initial.name = d\_o.\_initial.name;

d\_n.\_initial.patronymic = d\_o.\_initial.patronymic;

}

// Сортировка по алфавиту.

void Sorting(Data\* d, int n)

{

// временная переменная

Data buf;

// сортировка методом пузырька

for (int i = 0; i < n;i++)

{

for (int j = i + 1; j < n; j++)

{

//условие сортировки

if (d[i].\_initial.surname > d[j].\_initial.surname)

{

Copy(buf, d[j]); // данные d[j] помещаем в buf

Copy(d[j], d[i]); // меняем местами

Copy(d[i], buf); // данные bud помещаем в d[i]

}

if (d[i].\_initial.name > d[j].\_initial.name)

{

Copy(buf, d[j]);

Copy(d[j], d[i]);

Copy(d[i], buf);

}

if (d[i].\_initial.patronymic > d[j].\_initial.patronymic)

{

Copy(buf, d[j]);

Copy(d[j], d[i]);

Copy(d[i], buf);

}

}

}

}

// Удаление данных студента.

void Delete(Data\* (&d), int& n)

{

int \_n;

cout << "Представлены данные студенты (от 1 до " << n << " ) выберите номер для удаления:";

while (!(cin >> \_n) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

\_n--;

//Проверка что ввели правильное значение

if (\_n >= 0 && \_n < n)

{

//Временный масив

Data\* buf = new Data[n];

Copy(buf, d, n); //скопировали в этот временный масив данные

//Выделяем новую память

--n; // так как удаляем нужно в основном массиве выделить память на 1 меньше

d = new Data[n];// данные которые были в массиве удалятся для этого и нужен был временный массив buf

int q = 0;

//Запоминаем данные кроме ненужного

for (int i = 0; i <= n; i++)

{

if (i != n) // есди элемент не является тем который мы хотим удалить

{

d[q] = buf[i]; // сохраняем данные из буфера в d[q]

++q;

}

}

system("cls");

delete[]buf;

cout << "Данные удалены!" << endl;

}

else

cout << "Номер введен не верно!" << endl;

}

// Вывод данных о успеваемости студентов.

void Assesment(Data\* d, int n)

{

int enter = 0;

cout << "Выберите пункт: " << endl;

cout << "1.Студенты, которые учатся на \"Отлично\"." << endl

<< "2.Студенты, которые учатся на \"Хорошо\"." << endl

<< "3.Студенты, которые учатся на \"Удовлетворительно\"." << endl

<< "4.Студенты, которые учатся на \"Неудовлетворительно\"." << endl;

while (!(cin >> enter) || (cin.peek() != '\n'))

{

cin.clear();

while (cin.get() != '\n');

cout << "Ошибка ввода. Попробуйте ещё раз:" << endl;

}

for (int i = 0; i < n; i++)

{

if (enter == 1)

{

if (d[i].\_assesmentsstudents.Bal >= 170)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: " << d[i].\_initial.surname << d[i].\_initial.name << d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

}

else

{

cout << "Студентов нет!" << endl;

}

}

if (enter == 2)

{

if (d[i].\_assesmentsstudents.Bal < 170 && d[i].\_assesmentsstudents.Bal >= 140)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: " << d[i].\_initial.surname << d[i].\_initial.name << d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

}

else

{

cout << "Студентов нет!" << endl;

}

}

if (enter == 3)

{

if (d[i].\_assesmentsstudents.Bal < 140 && d[i].\_assesmentsstudents.Bal >= 100)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: " << d[i].\_initial.surname << d[i].\_initial.name << d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

}

else

{

cout << "Студентов нет!" << endl;

}

}

if (enter == 4)

{

if (d[i].\_assesmentsstudents.Bal < 100)

{

cout << "Данные №" << i + 1 << ":" << endl;

cout << "ФИО: " << d[i].\_initial.surname << d[i].\_initial.name << d[i].\_initial.patronymic << endl;

cout << "Номер группы: "

<< d[i].\_numbergroup.NomerGrup << endl;

cout << "Номер студента в списке группы: "

<< d[i].\_numberlistofgroup.NomerStudenta << endl;

cout << "Количество баллов: "

<< d[i].\_assesmentsstudents.Bal << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

break;

}

else

{

cout << "Студентов нет!" << endl;

}

}

}

}

// Меню 2 задания.

int MenuEnter4() { //вывод меню, управляемого стрелками

auto key = 0;

int code;

do {

system("cls");

key = (key + 8) % 8;

if (key == 0) cout << "-> Создание новой записи о студенте." << endl;

else cout << " Создание новой записи о студенте." << endl;

if (key == 1) cout << "-> Внесение изменений в уже имеющуюся запись." << endl;

else cout << " Внесение изменений в уже имеющуюся запись." << endl;

if (key == 2) cout << "-> Вывод всех данных о студентах." << endl;

else cout << " Вывод всех данных о студентах." << endl;

if (key == 3) cout << "-> Вывод данных студентов по времени" << endl;

else cout << " Вывод данных студентов по времени" << endl;

if (key == 4) cout << "-> Добавить нового студента" << endl;

else cout << " Добавить нового студента" << endl;

if (key == 5) cout << "-> Удаление данных студента" << endl;

else cout << " Удаление данных студента" << endl;

if (key == 6) cout << "-> Успеваемость студентов" << endl;

else cout << " Успеваемость студентов" << endl;

if (key == 7) cout << "-> Главное меню" << endl;

else cout << " Главное меню" << endl;

code = \_getch();

if ((code == 115) || (code == 119))

{

if (code == 115) key++;

if (code == 119) key--;

}

else

if (code == 224) {

code = \_getch();

if (code == 80) key++;

if (code == 72) key--;

}

} while (code != 13);

system("cls");

return key;

}

// Функция работы с меню 2 задания.

void MainEnter1()

{

List L;

int n = 0;

int\* a = new int[n];

auto exit = 0;

int \_actions, amountOfData = 0; //Переменная которая указывает количество данных

//Массив данных

Data\* d = new Data[amountOfData]; //указатель который указывает на массив

do {

const auto answer = MenuEnter4();

switch (answer)

{

case 0:

{

system("cls"); //Очистка консоли

DataEntry(d, amountOfData); //создаем новую запись о студент(-е, ах)(Массив, Количество данных)

SavingData(d, amountOfData, "students.txt"); //сохранение в файл students

if (amountOfData != 0)

{

Print(d, amountOfData); //вывод

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause"); // Задержка консоли

system("cls"); // очистка консоли

break; // Ломаем

}

case 1:

{

system("cls");

if (amountOfData != 0)

{

DataChange(d, amountOfData); //изменение данных о студенте

SavingData(d, amountOfData, "students.txt"); // сохранение в файл students

}

else

cout << "О записей ни слуху ни духу!" << endl;

Print(d, amountOfData); // вывод

system("Pause");

system("cls");

break;

}

case 2:

{

system("cls");

if (amountOfData != 0)

{

Sorting(d, amountOfData); // сортируем по алфавиту

Print(d, amountOfData); // выводим

SavingData(d, amountOfData, "students.txt");

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause");

system("cls");

break;

}

case 3:

{

system("cls");

if (amountOfData != 0)

{

Times(d, amountOfData);

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause"); // Задержка консоли

system("cls");

break;

}

case 4:

{

system("cls");

if (amountOfData != 0)

{

AddDate(d, amountOfData); // Добавить студента

amountOfData++;

SavingData(d, amountOfData, "students.txt"); // сохранение

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause"); // Задержка консоли

system("cls");

break;

}

case 5:

{

system("cls");

if (amountOfData != 0)

{

Delete(d, amountOfData);

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause"); // Задержка консоли

system("cls");

break;

}

case 6:

{

system("cls");

if (amountOfData != 0)

{

Assesment(d, amountOfData);

}

else

cout << "О записей ни слуху ни духу!" << endl;

system("Pause"); // Задержка консоли

system("cls");

break;

}

case 7:

{

exit = 1;

}

default:;

}

} while (exit == 0);

}

Приложение Д

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

#pragma once

#include "Header.h"

// Узел.

struct Node

{

int data;

int ratio = 0;

Node\* l = nullptr;

Node\* r = nullptr;

};

// Шаблон класса.

template <class T>

// Лист.

struct list

{

T data;

list\* next = nullptr;

};

// Поиск узлов.

Node\* LookNode(Node\* present, int val)

{

if (present == nullptr)

return nullptr;

if (val > present->data)

{

if (present->r != nullptr)

present = LookNode(present->r, val);

}

else if (val < present->data)

{

if (present->l != nullptr)

present = LookNode(present->l, val);

}

return present;

}

// Поиск с прошлым.

Node\* LookNode(Node\* present, Node\*& last, int val)

{

if (present == nullptr)

return nullptr;

if (val > present->data)

{

if (present->r != nullptr)

{

last = present;

present = LookNode(present->r, last, val);

}

}

else if (val < present->data)

{

if (present->l != nullptr)

{

last = present;

present = LookNode(present->l, last, val);

}

}

return present;

}

// Проходимся прямо по дереву.

void StraightTree(Node\* origin)

{

cout << origin->data << " ";

if (origin->l != nullptr)

StraightTree(origin->l);

if (origin->r != nullptr)

StraightTree(origin->r);

}

// Проходимся обратно по дереву.

void BackTree(Node\* origin)

{

if (origin->l != nullptr)

BackTree(origin->l);

if (origin->r != nullptr)

BackTree(origin->r);

cout << origin->data << " ";

}

// Проходимся так же одинаково.

void EquallyTree(Node\* origin)

{

if (origin->l != nullptr)

EquallyTree(origin->l);

cout << origin->data << " ";

if (origin->r != nullptr)

EquallyTree(origin->r);

}

// С предыдущими.

Node\* LookNode(Node\* present, list <Node\*>\*& head, int val)

{

if (present == nullptr)

return nullptr;

list<Node\*>\* newl;

if (val > present->data)

{

head->data = present;

if (present->r != nullptr)

{

newl = new list<Node\*>;

newl->next = head;

head = newl;

present = LookNode(present->r, head, val);

}

}

else if (val < present->data)

{

head->data = present;

if (present->l != nullptr)

{

newl = new list<Node\*>;

newl->next = head;

head = newl;

present = LookNode(present->l, head, val);

}

}

return present;

}

// Максимальная длина.

int MaxLenght(Node\* origin)

{

int lenght = 1, l = 0, r = 0;

if (origin->l != nullptr)

l = MaxLenght(origin->l);

if (origin->r != nullptr)

r = MaxLenght(origin->r);

if (l >= r)

lenght += l;

else

lenght += r;

return lenght;

}

// Преобразование коэффициентов.

void TransformationCoefficients(Node\* origin)

{

if (origin != nullptr)

{

int maxL = 0, maxR = 0;

if (origin->l != nullptr)

{

maxL = MaxLenght(origin->l);

TransformationCoefficients(origin->l);

}

if (origin->r != nullptr) {

maxR = MaxLenght(origin->r);

TransformationCoefficients(origin->r);

}

origin->ratio = maxR - maxL;

}

}

// Маленький поворот направо.

void LittleTreeRight(Node\* present, Node\*& origin)

{

Node\* last = nullptr, \* q;

LookNode(origin, last, present->data);

if (present->r == nullptr)

{

if (present->l->r == nullptr)

{

q = present;

present = present->l;

q->l = nullptr;

present->r = q;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

else

{

q = present;

present = present->l->r;

present->l = q->l;

q->l->r = nullptr;

q->l = nullptr;

present->r = q;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

}

else

{

q = present->l->r;

present->l->r = present;

Node\* q2 = present->l;

present->l = q;

present = q2;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

TransformationCoefficients(origin);

}

// Маленький поворот налево.

void LittleTreeLeft(Node\* present, Node\*& origin)

{

Node\* last = nullptr, \* q;

LookNode(origin, last, present->data);

if (present->l == nullptr)

{

if (present->r->l == nullptr)

{

q = present;

present = present->r;

q->r = nullptr;

present->l = q;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

else

{

q = present;

present = present->r->l;

present->r = q->r;

q->r->l = nullptr;

q->r = nullptr;

present->l = q;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

}

else

{

q = present->r->l;

present->r->l = present;

Node\* q2 = present->r;

present->r = q;

present = q2;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

}

TransformationCoefficients(origin);

}

// Большой поворот направо.

void BigTreeRight(Node\* present, Node\*& origin)

{

Node\* last = nullptr, \* q, \* q2;

LookNode(origin, last, present->data);

q = present->l;

present->l = q->r->r;

q2 = q->r;

q->r = q->r->l;

q2->l = q;

q2->r = present;

present = q2;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

TransformationCoefficients(origin);

}

// Большой поворот налево.

void BigTreeLeft(Node\* present, Node\*& origin)

{

Node\* last = nullptr, \* q, \* q2;

LookNode(origin, last, present->data);

q = present->r;

present->r = q->l->l;

q2 = q->l;

q->l = q->l->r;

q2->r = q;

q2->l = present;

present = q2;

if (last != nullptr)

{

if (present->data > last->data)

last->r = present;

else

last->l = present;

}

else

origin = present;

TransformationCoefficients(origin);

}

// Добавить в дерево.

bool ApendTree(Node\*& origin, int val, int& N)

{

// Коэффициенты баланса

const int min = -1, max = 1;

list<Node\*>\* head, \* q;

head = new list<Node\*>;

Node\* present = LookNode(origin, head, val);

if (present == nullptr)

{

delete head;

return 0;

}

if (val == present->data)

{

do

{

q = head->next;

delete head;

head = q;

} while (head != nullptr);

return 0;

}

N++;

Node\* newG;

q = new list <Node\*>;

if (val > present->data)

{

newG = new Node;

present->r = newG;

newG->data = val;

q->data = newG;

}

if (val < present->data) {

newG = new Node;

present->l = newG;

newG->data = val;

q->data = newG;

}

q->next = head;

head = q;

// Пересчет коэффициентов.

do {

if (head->data->data > head->next->data->data)

head->next->data->ratio++;

else

head->next->data->ratio--;

head = head->next;

} while (!((head->data->ratio == 0) || (head->data->ratio > max) || (head->data->ratio < min) || (head->next == nullptr)));

if (head->data->ratio < min)

{

if (head->data->r == nullptr || val <= head->data->l->data)

LittleTreeRight(head->data, origin);

else

BigTreeRight(head->data, origin);

}

else if (head->data->ratio > max) {

if (head->data->l == nullptr || val >= head->data->r->data)

LittleTreeLeft(head->data, origin);

else

BigTreeLeft(head->data, origin);

}

head = q;

while (head != nullptr)

{

q = head;

head = head->next;

delete q;

}

return 1;

}

// Баланс.

Node\* Balancе(Node\* origin) {

// Коэффициенты баланса

const int min = -1, max = 1;

Node\* p = nullptr;

if (origin->ratio < min or origin->ratio > max)

return origin;

if (origin->l != nullptr)

p = Balancе(origin->l);

if (p != nullptr)

return p;

if (origin->r != nullptr)

p = Balancе(origin->r);

return p;

}

// Балансировка деревьев.

void TreeBalancing(Node\*& origin) {

// Коэффициенты баланса

const int min = -1, max = 1;

do {

if (origin == nullptr)

break;

TransformationCoefficients(origin);

Node\* p = Balancе(origin);

if (p == nullptr)

break;

if (p->ratio < min)

{

int l = 0, r = 0;

if ((p->l != nullptr) && (p->l->l != nullptr))

l = MaxLenght(p->l->l);

if ((p->l != nullptr) && (p->l->r != nullptr))

r = MaxLenght(p->l->r);

if ((p->r == nullptr) || (l >= r))

LittleTreeRight(p, origin);

else

BigTreeRight(p, origin);

}

else if (p->ratio > max)

{

int l = 0, r = 0;

if ((p->r != nullptr) && (p->r->r != nullptr))

r = MaxLenght(p->r->r);

if ((p->r != nullptr) && (p->r->l != nullptr))

l = MaxLenght(p->r->l);

if ((p->l == nullptr) || (r >= l))

LittleTreeLeft(p, origin);

else

BigTreeLeft(p, origin);

}

} while (1);

}

// Удалить из дерева.

bool DeleteFromTree(Node\*& origin, int val)

{

if (origin == nullptr)

return 0;

Node\* last = nullptr;

Node\* present = LookNode(origin, last, val);

if (val != present->data)

return 0;

if ((present->l != nullptr) && (present->r != nullptr))

{

Node\* p = present->l;

while (p->r != nullptr)

p = p->r;

if (last != nullptr) {

if (last->data < present->data)

last->r = present->l;

else

last->l = present->l;

}

else

origin = present->l;

p->r = present->r;

delete present;

}

else if (present->l != nullptr)

{

if (last != nullptr) {

if (last->data < present->data)

last->r = present->l;

else

last->l = present->l;

}

else

origin = present->l;

delete present;

}

else if (present->r != nullptr)

{

if (last != nullptr)

{

if (last->data < present->data)

last->r = present->r;

else

last->l = present->r;

}

else

origin = present->r;

delete present;

}

else

{

if (last != nullptr)

{

if (last->data < present->data)

last->r = nullptr;

else

last->l = nullptr;

}

else

origin = nullptr;

delete present;

}

TreeBalancing(origin);

return 1;

}

// Случайное дерево.

void RandomTree(int N, Node\*& origin, int diapason)

{

origin = new Node;

origin->data = rand() % diapason - diapason / 2;

for (int i = 1; i < N;)

ApendTree(origin, rand() % diapason - diapason / 2, i);

}

// Удаление дерева.

void DeleteTree(Node\* origin) {

if (origin->l != nullptr)

DeleteTree(origin->l);

if (origin->r != nullptr)

DeleteTree(origin->r);

delete origin;

}

// Int в char.

char\* IntToChar(int i) {

char\* str = new char[15];

bool mn = 0;

if (i < 0) {

i = abs(i);

mn = 1;

}

else if (i == 0)

{

str[0] = (unsigned char)(48);

str[1] = '\0';

return str;

}

str[0] = '\0';

while (i > 0) {

for (int i = 14; i > 0; i--)

str[i] = str[i - 1];

str[0] = (unsigned char)(48 + i % 10);

i = i / 10;

}

if (mn) {

for (int i = 14; i > 0; i--)

str[i] = str[i - 1];

str[0] = '-';

}

return str;

}

// Матрица сдвига вправо.

void RightMatrix(int\*\* matrix, int i, int j, int N)

{

int breadth = N \* 3 + 2;

int nine = -999999, nine\_1 = -999998;

for (int q = i; q >= 0; q--) {

for (int k = breadth - 1; k > j; k--)

matrix[q][k] = matrix[q][k - 1];

if (matrix[q][j] != nine)

matrix[q][j] = nine\_1;

}

}

// Вывод дерева.

void TreeOutput(ofstream& file, const Node\* origin, int N, bool in\_file)

{

const int min = -1, max = 1;

int nine = -999999, nine\_1 = -999998;

if (origin != nullptr)

{

int breadth = N \* 3 + 2;

if (in\_file)

file << "Дерево" << ".\n";

list<const Node\*>\* head, \* present, \* newt;

head = new list<const Node\*>;

head->data = origin;

int\*\* matrix;

int\* startPoint = new int[N / 2];

int i = 0, j = 0, startTail = 1;

matrix = new int\* [N \* 2];

for (int i = 0; i < N \* 2; i++)

{

matrix[i] = new int[breadth];

for (int j = 0; j < breadth; j++)

matrix[i][j] = nine;

}

startPoint[0] = 1;

while (head != nullptr) {

j = 0;

present = head;

while (present != nullptr)

{

matrix[i][startPoint[j]] = present->data->data;

if (present->data->l != nullptr)

{

for (int q = 0; q < 3; q++)

{

RightMatrix(matrix, i + 1, startPoint[j], N);

for (int q = j + 1; q < startTail; q++)

startPoint[q]++;

matrix[i + 1][startPoint[j]] = nine\_1;

}

startTail++;

for (int i = startTail; i > j; i--)

startPoint[i] = startPoint[i - 1];

j++;

startPoint[j] += 3;

}

if (present->data->r != nullptr)

{

for (int q = 0; q < 3; q++)

{

RightMatrix(matrix, i + 1, startPoint[j] + 1, N);

for (int q = j + 1; q < startTail; q++)

startPoint[q]++;

matrix[i + 1][startPoint[j] + 1] = nine\_1;

}

startPoint[j] += 3;

j++;

}

else

{

for (int i = j; i < startTail; i++)

startPoint[i] = startPoint[i + 1];

startTail--;

}

present = present->next;

}

present = head;

while (present != nullptr)

{

bool b = 0;

if (present->data->r != nullptr)

{

newt = new list<const Node\*>;

newt->next = present->next;

present->next = newt;

newt->data = present->data->r;

b = 1;

}

if (present->data->l != nullptr)

{

present->data = present->data->l;

present = present->next;

}

else

{

if (present == head)

{

head = present->next;

delete present;

present = head;

}

else

{

newt = present;

present = head;

while (present->next != newt)

present = present->next;

present->next = newt->next;

delete newt;

present = present->next;

}

}

if (b)

present = present->next;

}

i += 2;

}

for (int q = 0; q < i - 1; q++)

{

if (q % 2 == 0)

{

for (j = 0; j < breadth; j++)

{

if (matrix[q][j] > nine\_1)

{

if (in\_file)

file << "|";

else

cout << "|";

j++;

while (matrix[q][j] > nine\_1&& j < breadth)

{

j++;

if (in\_file)

file << " ";

else

cout << " ";

}

j--;

}

else if (in\_file)

file << " ";

else

cout << " ";

}

if (in\_file)

file << endl;

else

cout << endl;

if (in\_file)

{

for (j = 0; j < breadth; j++)

{

if (matrix[q][j] > nine\_1)

{

file << (unsigned char)149;

j++;

while (matrix[q][j] > nine\_1&& j < breadth) {

j++;

file << " ";

}

j--;

}

else

file << " ";

}

file << endl;

}

}

bool b = 0;

for (j = 0; j < breadth; j++)

{

if (!b && matrix[q][j] == nine\_1 && (matrix[q + 1][j] != nine || matrix[q - 1][j - 1] != nine))

b = 1;

if (b && matrix[q][j] == nine)

b = 0;

if (matrix[q][j] == nine || matrix[q][j] == nine\_1 && !b)

{

matrix[q][j] = nine;

if (in\_file)

file << " ";

else

cout << " ";

}

else if (matrix[q][j] == nine\_1)

{

if (matrix[q + 1][j] != nine && matrix[q][j - 1] == nine)

if (in\_file)

file << ",";

else

cout << ",";

else if (matrix[q + 1][j] != nine)

if (in\_file)

file << ".";

else

cout << ".";

else if (in\_file)

file << "-";

else

cout << "-";

}

else {

if (in\_file)

file << matrix[q][j];

else

cout << matrix[q][j];

char\* str = IntToChar(matrix[q][j]);

j += strlen(str) - 1;

delete[] str;

}

}

if (in\_file)

file << endl;

else

cout << endl;

}

for (i = 0; i < N \* 2; i++)

delete[] matrix[i];

delete[] matrix;

}

}

// Поиск отрицательного узла.

int MinusLookNode(Node\* origin)

{

int val = 0;

if (origin->data < 0)

return origin->data;

if (origin->l != nullptr)

val = MinusLookNode(origin->l);

return val;

}

// Поиск.

int StrangeSearch(Node\*& origin) {

int val = 0;

if (origin->data % 2 != 0)

return origin->data;

if (origin->l != nullptr)

val = StrangeSearch(origin->l);

if (val != 0)

return val;

if (origin->r != nullptr)

val = StrangeSearch(origin->r);

return val;

}

**Список использованных источников.**

1. Глущенко А.Г. Основы алгоритмизации и программирования. Методические указания к практическим работам : учеб. пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2020. 126 с.