**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

Тема: Польские нотации. Стек и очередь

|  |  |  |
| --- | --- | --- |
| Студент гр. 9894 |  | Леушкин С.М. |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2020

# **Цель работы.**

Получение практических навыков работы со стеками и очередями. Изучение обратной и прямой польской нотации. Проведение сравнительного анализа этих структур данных.

# **Основные теоретические положения.**

Стек – это частный случай однонаправленного списка, добавление элементов в который и выборка из которого выполняется с одного конца, называемого вершиной стека [1]. Другие операции со стеком не определены. Графическое представление стека представлено на рисунке 1.

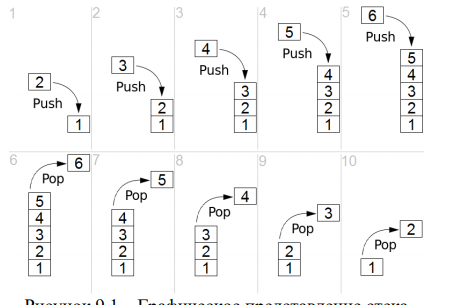


Рисунок 1 – Графическое представление стека

Основные операции над стеками:

1. чтение верхнего элемента;
2. добавление нового элемента;
3. удаление существующего элемента

Для преобразования числовых выражений используется алгоритм польской нотации. В зависимости от принципа, различают прямую и обратную польские записи. Для этого необходимо два стека – стек выхода, куда записываются числа, и стек операций, куда временно записываются операции. Каждая операция имеет свой, так называемый, вес, или приоритет, и согласно нему позволяет определить записывать операцию в стек выхода или в стек операций. Обработка строки обратной польской нотацией происходит слева направо. На рисунках 2 – 8 представлен алгоритм записи обратной польской нотации.

|  |  |
| --- | --- |
|  |  |
| Рисунок 2 – Шаг 1 | Рисунок 3 – Шаг 2 |
|  |  |
| Рисунок 4 – Шаг 3 | Рисунок 5 – Шаг 4 |
|  |  |
| Рисунок 6 – Шаг 5 | Рисунок 7 – Шаг 6 |

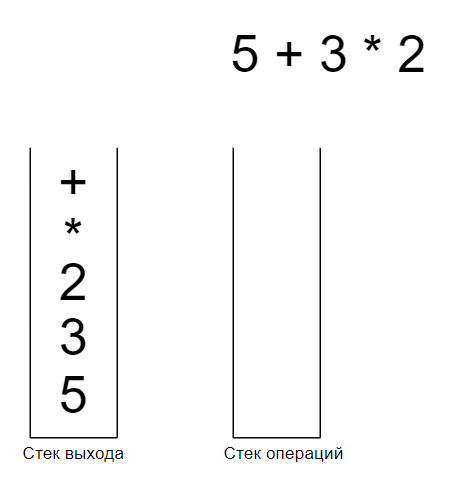


Рисунок 8 – Шаг 7

Пояснения: обрабатывая строку слева направо мы проверяем текущий символ. Если этот символ число, то мы отправляем его в стек выхода. Если этот символ операция, необходимо проверить последний символ в стеке операций. Если стек операций пуст, отправляем символ в стек операций (см. рис. 3). Если в стеке есть операция, необходимо проверить их вес (больший вес должен находится сверху стека, т.к. эта операция более приоритетная). В случае, если вес текущего символа больше последнего, то мы просто отправляем символ в стек операций (см. рис. 5). В противном случае, необходимо удалить из стека операций меньший по весу символ и отправить в стек выхода. Если веса равны, то для корректности выражения нижележащую операцию удалить и отправить в стек выхода.

По окончанию строки, все символы из стека операций по одному отправляются в стек выхода. На этом преобразование закончено.

При прямой польской нотации обработка строки происходит справа налево.

# **Постановка задачи.**

Необходимо реализовать программу, которая:

1. Реализует преобразование введенного выражения (если используется переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализован двумя способами: с клавиатуры и с файла.
2. Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польским нотациям (на выбор пользователя). Ввод выражения должен быть реализован двумя способами: с клавиатуры и с файла.
3. Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польским нотациям (на выбор пользователя). Ввод выражения должен быть реализован двумя способами: с клавиатуры и с файла.
4. Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польским нотациям (прямой и обратной). Задание и ответы к ним необходимо вывести в отдельные файлы (ответы должны быть максимально подробными).

Программа должна быть реализована двумя способами: с использованием стека и массивов. Также необходимо сравнить скорость вычисления, создания и проверки на корректность выражения в прямой и обратной польских нотациях.

Также необходимо реализовать индивидуальное задание – реализация «электронной» очереди в магазине.

# **Выполнение работы.**

Листинг программы представлен в приложении А.

Доступ к функциям программы предоставляется с помощью меню, в котором необходимо выбрать один из пунктов. Меню реализовано в функции menu(). Выбор пунктов меню реализовано в функциях count\_val() и menu\_choice(). Дополнительные подменю также реализованы для выбора нотации и для способа ввода.

Заполнение строки выполнено тремя способами:

1. С клавиатуры – пользователь самостоятельно вводит выражения, отделяя каждый символ пробелом. Способ реализован в функции fill\_str\_self().
2. Заполнение с помощью текстового файла. Способ реализован в функции fill\_str\_via\_file().

Стек реализован на односвязном списке и имеет структуру Node.

Требования в двух случаях одинаковые: между символами ставится пробел, постронних знаков быть не должно.

После ввода, строка проверяется на переменные в функции check\_literals(). Принимает на вход саму строку, и при необходимости изменяет глобальный словарь variables, добавляя новые переменные.

Разбивка строки на элементы реализована в функции Tokenizer(). Принимает на вход строку и возвращает массив отдельных элементов. Преобразование в польскую нотацию реализовано в функции polishNotation(). Принимает на вход строку, массив отдельных элементов строки, стек выхода и стек операций и поток для вывода в файл логов. Функция используется как основная для двух видов нотаций – прямой (normalPolishNotation()) и обратной (ReversePolishNotation()).

Вычисление нотации реализовано в функции Calc\_PN(). На вход она принимает стек выхода и флаг fl\_notation, который определяет выбор нотации и возвращает полученное число типа double. Функция используется как основная для двух видов подсчета – числовым выражением (Calculate\_exp()) или нотацией (Calculate\_pn()).

Проверка на корректность реализована в функции check\_exp() (для числовых выражений) и check\_pn() (для польских нотаций).

Также для функций преобразования и вычисления продублированы функции на основе массивов.

Генерация теста реализована в функции createTest(). Для этого была создана структура Test, в которой хранится массив заданий и ответы к ним. Задания записываются в текстовый файл test\_tasks.txt. Подробные ответы к ним записываются в текстовый файл test\_solution.txt.

В функции queue() реализовано индвидуальное задание. Вывод очереди реализован в функции printQueue(). На вход принимает сам массив и элемент пользователя.

## Тестирование программы.

На рисунках 9 – 15 представлены скриншоты тестирования программы.

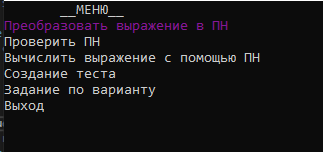


Рисунок 9 – Скриншот меню программы

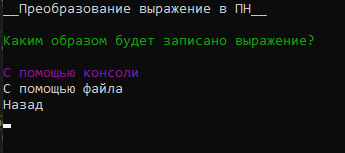


Рисунок 10 – Скриншот подменю преобразования выражения

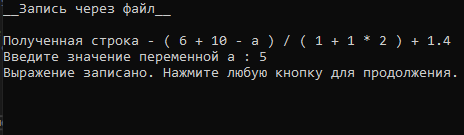


Рисунок 11 – Скриншот выбора пункта «С помощью файла»

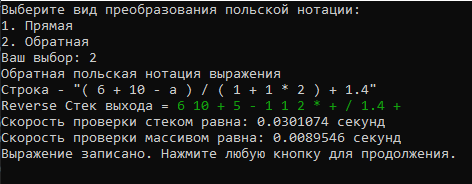


Рисунок 12 – Скриншот преобразования выражения в польскую нотацию

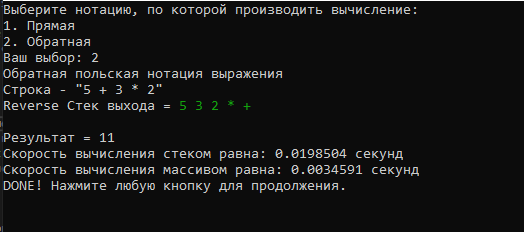


Рисунок 13 – Скриншот вычисления числового выражения

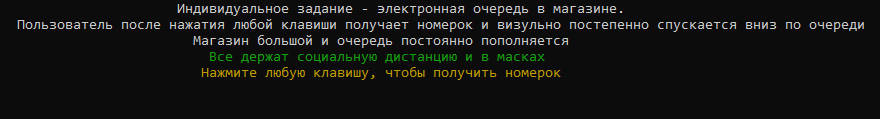


Рисунок 14 – Скриншот пояснения к индвидуальному заданию

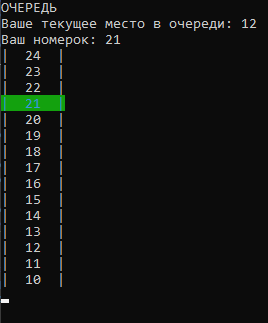


Рисунок 15 – Скриншот выполнения индвидиуального задания

Также было проведено сравнение скорости двусвязного списка и динамического массива при одинаковой размерности. Заполнение значений выполнялось автоматически. Сравнение происходило в создании структуры данных, поиска, вставки и удаления элемента (см. табл. 1).

Таблица 1 – Сравнение скорости двусвязного списка и динамического массива

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Выражение | Стек | | | Массив | | |
| Преобразование, сек | Вычисление, сек | Проверка, сек | Преобразование, сек | Вычисление, сек | Проверка, сек |
| 1 |  | 0,017 | 0,019 | 0,018 | 0,002 | 0,004 | 0,004 |
| 2 |  | 0,019 | 0,035 | 0,026 | 0,006 | 0,009 | 0,008 |
| 3 |  | 0,033 | 0,036 | 0,032 | 0,009 | 0,013 | 0,012 |
| 4 |  | 0,032 | 0,047 | 0,046 | 0,011 | 0,044 | 0,024 |
| 5 |  | 0,026 | 0,040 | 0,038 | 0,010 | 0,016 | 0,016 |

Из представленных графиков (см. рис. 16 – 18) видно, что реалиазция польской нотации на массиве обрабатывается немного быстрее по сравнению с реализацией на стеке. Можно предположить, что это связано с тем, что во время обработки стеком дополнительно сохраняются логи подробного решения, что немного затрудняет обработку.

Рисунок 16 – График зависимости скорости от сложности выражения во время преобразования выражения

Рисунок 17 – График зависимости скорости от сложности выражения во время вычисления выражения

Рисунок 18 – График зависимости скорости от сложности выражения во время проверки выражения на корректность

# **Выводы.**

Были освоены теоретические знания и закреплены практические навыки по работе с стеками и массивами на примере изучения польских нотаций. Было проведено исследование по практическому применению стеков и массивов. Приведённые графики показывают, что использование стеков или массивов для обработки числовых выражений не играет особой роли, потому как скорость обработки у них относительно одинакова.

# **Список использованных источников.**

1. Основы алгоритмизации и программирования на языке C++. Методические указания к практическим работам / сост.: А.Г. Глущенко. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2020. 126с.

Приложение А

ЛИСТИНГ ПРОГРАММЫ

//// Лабораторная работа №3 - Польская нотация

#include <iostream>

#include <sstream>

#include <Windows.h>

#include <string>

#include <vector>

#include <map>

#include <conio.h>

#include <fstream>

#include <iomanip>

#include <ctime>

#include <chrono>

using namespace std;

string input\_str;

string check\_str;

string calc\_notation\_file = "notation.txt";

string input\_file = "inputStr.txt";

string check\_file\_pn = "checkPN.txt";

string check\_file\_exp = "checkexp.txt";

bool fl\_test = FALSE;

bool back = FALSE;

map <string, string> variables;

map <string, int> weight\_op = { {"+", 1 },

{"-", 1 },

{"\*", 2 },

{"/", 2 },

{"(", 0 },

{")", 0 } };

map <string, int> check\_op = { {"+", 1 },

{"|", 1 },

{"\*", 2 },

{"/", 2 } };

int m\_count = 0;

int subm\_conv\_count = 0;

int subm\_conv\_pn\_count = 0;

int subm\_calc\_pn\_count = 0;

int subm\_check\_pn\_count = 0;

void SetColor(int text, int bg) { //Функция смены цвета, взятая из Интернета

HANDLE hStdOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hStdOut, (WORD)((bg << 4) | text));

}

////////////////// PROTOTYPES /////////////////////

typedef struct Node;

bool getCalcNotation();

void menu(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_convert\_exp(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_calc\_pn(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_check\_pn(string& str, Node\*& outStack, Node\*& opStack);

void menu\_choice(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_convert\_exp\_choice(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_calc\_pn\_choice(string& str, Node\*& outStack, Node\*& opStack);

void submenu\_check\_pn\_choice(string& str, Node\*& outStack, Node\*& opStack);

void check\_literals(string& str, Node\*& outStack, Node\*& opStack);

void fill\_str\_self(string& str, Node\*& outStack, Node\*& opStack);

void fill\_str\_via\_file(string& str, Node\*& outStack, Node\*& opStack, string& filename);

void logFile(ofstream& fout, string str, const string& token, Node\* Stack, string object, string Action);

////////////////// PROTOTYPES /////////////////////

double str2doub(std::string str)

{

double d;

std::stringstream iss(str);

iss >> d;

return d;

}

int checkdigit() {

while (true) {

int value;

cin >> value;

if (cin.fail()) {

cin.clear();

cin.ignore(32767, '\n');

cout << "Недопустимое заданное число. Введите число правильно" << '\n';

}

else {

if (value >= 0) {

cin.ignore(32767, '\n');

return value;

}

else {

cin.clear();

cin.ignore(32767, '\n');

cout << "Недопустимое заданное число. Введите число правильно" << '\n';

}

}

}

}

typedef struct exercise {

string task;

string notation\_answer;

double answer;

};

typedef struct Test {

vector<exercise> Task;

};

////////////////// STACK /////////////////////

typedef struct Node {

string value;

struct Node\* next;

} Node;

void push(Node\*\* head, string data) {

void \*ex\_raw = malloc(sizeof(Node));

Node\* tmp = new(ex\_raw) Node;

tmp->value = data;

tmp->next = (\*head);

(\*head) = tmp;

}

string pop(Node\*\* head) {

Node\* prev = NULL;

string val;

if (head == NULL) {

exit(-1);

}

prev = (\*head);

val = prev->value;

(\*head) = (\*head)->next;

free(prev);

return val;

}

Node\* getLast(Node\* head) {

if (head == NULL) {

return NULL;

}

while (head->next) {

head = head->next;

}

return head;

}

void pushBack(Node\* head, string value) {

Node\* last = getLast(head);

void\* ex\_raw = malloc(sizeof(Node));

Node\* tmp = new(ex\_raw) Node;

tmp->value = value;

tmp->next = NULL;

last->next = tmp;

}

Node\* getLastButOne(Node\* head) {

if (head == NULL) {

exit(-2);

}

if (head->next == NULL) {

return NULL;

}

while (head->next->next) {

head = head->next;

}

return head;

}

string popBack(Node\*\* head) {

string value;

Node\* lastbn = NULL;

//Получили NULL

if (!head) {

exit(-1);

}

//Список пуст

if (!(\*head)) {

exit(-1);

}

lastbn = getLastButOne(\*head);

value = lastbn->next->value;

//Если в списке один элемент

if (lastbn == NULL) {

free(\*head);

\*head = NULL;

}

else {

free(lastbn->next);

lastbn->next = NULL;

}

return value;

}

void deleteList(Node\*\* head) {

Node\* prev = NULL;

while ((\*head)->next) {

prev = (\*head);

(\*head) = (\*head)->next;

free(prev);

}

free(\*head);

}

void printStack(const Node\* head, ofstream& fout, bool console) {

while (head) {

if(!console)

fout << head->value << " ";

else {

cout << head->value << " ";

fout << head->value << " ";

}

head = head->next;

}

if(!console)

fout << endl;

else {

cout << endl;

fout << endl;

}

return;

}

int sizeStack(Node\* head) {

int count = 0;

while (head) {

count++;

head = head->next;

}

return count;

}

Node\* reverseStack(Node\* outStack)

{

Node\* tmp = NULL;

int size\_stack = sizeStack(outStack);

if (outStack) {

if (size\_stack >= 2)

push(&tmp, popBack(&outStack));

else

push(&tmp, pop(&outStack));

size\_stack--;

}

else

exit(-1);

while (outStack)

{

if (size\_stack != 1)

pushBack(tmp, popBack(&outStack));

else

pushBack(tmp, pop(&outStack));

size\_stack--;

}

return tmp;

}

void addStackElem(Node\* &Stack , const string& token) {

if (!Stack)

push(&Stack, token);

else

pushBack(Stack, token);

}

////////////////// STACK /////////////////////

////////////////// QUEUE /////////////////////

void printQueue(const vector<int>& Queue, int my\_index) {

system("cls");

cout << "ОЧЕРЕДЬ" << endl;

cout << "Ваше текущее место в очереди: " << Queue.size() - my\_index << endl;

cout << "Ваш номерок: " << Queue[my\_index] << endl;

for (int i = 0; i <= Queue.size() - 1; i++) {

if (my\_index == i) {

SetColor(3, 2); cout << "| " << setw(3) << right << Queue[i] << setw(3) << right << " |" << endl;

SetColor(7, 0);

}

else

cout << "| " << setw(3) << right << Queue[i] << setw(3) << right << " |" << endl;

}

}

void fill\_queue(vector<int>& Queue, int& value) {

value = rand() % 50;

for (int i = Queue.size() - 1; i >= 0; i--) {

Queue[i] = value++;

}

}

void step\_forward(vector<int>& Queue, int& value) {

Queue.pop\_back();

Queue.insert(Queue.begin(), value++);

}

void queue() {

system("cls");

int count = rand() % 20;

int value;

vector<int> Queue;

while (count == 0) {

count = rand() % 20;

}

Queue.resize(count);

fill\_queue(Queue, value);

cout << setw(80) << right << "Индивидуальное задание - электронная очередь в магазине." << endl;

cout << setw(110) << "Пользователь после нажатия любой клавиши получает номерок и визульно постепенно спускается вниз по очереди" << endl;

cout << setw(73) << "Магазин большой и очередь постоянно пополняется" << endl;

SetColor(2, 0);

cout << setw(70) << "Все держат социальную дистанцию и в масках" << endl;

SetColor(7, 0);

SetColor(6, 0);

cout << setw(72) << "Нажмите любую клавишу, чтобы получить номерок" << endl;

SetColor(7, 0);

\_getch();

for (int i = 0; i <= Queue.size() - 1; i++) {

step\_forward(Queue, value);

printQueue(Queue, i);

Sleep(3000);

}

cout << "Теперь вы можете оплатить покупку!" << endl;

system("pause");

}

////////////////// QUEUE /////////////////////

/////////////////// OPZ ///////////////////

vector<string> Tokenizer(const string& text) {

// <- 1 2 3 4 + 3 -2

// -> [1,2,3,4,+,3,-2]

vector<string> tokens{};

istringstream input(text);

for (string token; getline(input, token, ' '); /\*nothing to do\*/) {

tokens.push\_back(token);

}

return tokens;

}

bool IsNumber(const string& str) {

try {

auto res = std::stod(str); // we need only exception or no exception

}

catch (const std::invalid\_argument& ex) {

return FALSE;

}

return TRUE;

}

bool checkWeight(const string& str, Node\* head, const map <string, int> & map\_) {

if (map\_.find(str)->second > map\_.find((getLast(head)->value))->second)

return TRUE;

else

return FALSE;

}

bool checkWeight\_vec(const string& token, vector<string>& opArr, const map <string, int>& map\_) {

if (map\_.find(token)->second > map\_.find(opArr[opArr.size() - 1])->second)

return TRUE;

else

return FALSE;

}

bool IsOperation(const string& str, map <string, int> map\_) {

auto res = map\_.find(str);

if (res == map\_.end())

return FALSE;

else

return TRUE;

}

bool IsLiteral(const string& str) {

auto res = variables.find(str);

if (res == variables.end())

return FALSE;

else

return TRUE;

}

double Action(double & value1, double &value2, string& operation) {

switch (operation[0]) {

case '+':

return value1 + value2;

break;

case '-':

return value1 - value2;

break;

case '\*':

return value1 \* value2;

break;

case '/':

return value1 / value2;

break;

default:

break;

}

}

void reverseTokens(vector <string>& str\_tokens) {

vector<string> temp = str\_tokens;

for (int i = 0; i <= str\_tokens.size() - 1; i++)

{

if (temp[str\_tokens.size() - 1 - i] == "(") {

str\_tokens[i] = ")";

continue;

}

if (temp[str\_tokens.size() - 1 - i] == ")") {

str\_tokens[i] = "(";

continue;

}

str\_tokens[i] = temp[str\_tokens.size() - 1 - i];

}

}

void polishNotation(string& inpt\_str, vector<string>& str\_tokens, Node\*&outStack, Node\* &opStack, ofstream& fout) {

string current\_str;

back = FALSE;

for (const auto& token : str\_tokens) {

logFile(fout, inpt\_str, token, outStack, "token", "curr");

if (IsNumber(token)) {

addStackElem(outStack, token);

logFile(fout, inpt\_str, token, outStack, "token", "add");

logFile(fout, inpt\_str, token, outStack, "stack", "curr");

continue;

}

if (IsOperation(token, weight\_op)) {

if (token == "(") {

addStackElem(opStack, token);

logFile(fout, inpt\_str, token, opStack, "token", "add");

logFile(fout, inpt\_str, token, opStack, "stack", "curr");

continue;

}

if (token == ")") {

while (opStack && current\_str != "(") {

if (sizeStack(opStack) >= 2)

current\_str = popBack(&opStack);

else

current\_str = pop(&opStack);

logFile(fout, inpt\_str, current\_str, opStack, "stack", "del");

addStackElem(outStack, current\_str);

logFile(fout, inpt\_str, current\_str, outStack, "token", "add");

logFile(fout, inpt\_str, current\_str, outStack, "stack", "curr");

current\_str = getLast(opStack)->value;

}

current\_str.clear();

if (!opStack) {

cerr << "[ERROR] character ( is not found. Try another one" << endl;

fout << "[ERROR] character ( is not found. Try another one" << endl;

back = TRUE;

\_getch();

return;

}

if (sizeStack(opStack) >= 2)

popBack(&opStack);

else

pop(&opStack);

logFile(fout, inpt\_str, "(", opStack, "stack", "del");

continue;

}

if (!opStack) {

push(&opStack, token);

logFile(fout, inpt\_str, token, opStack, "token", "add");

logFile(fout, inpt\_str, token, opStack, "stack", "curr");

continue;

}

if (checkWeight(token, opStack, weight\_op)) {

// Если вес текущего больше или равен предыдущему

pushBack(opStack, token);

logFile(fout, inpt\_str, token, opStack, "token", "add");

logFile(fout, inpt\_str, token, opStack, "stack", "curr");

}

else {

// Если вес текущего меньше предыдущего

logFile(fout, inpt\_str, getLast(opStack)->value, opStack, "stack", "del");

if (sizeStack(opStack) >= 2)// если стэк операций больше двух

pushBack(outStack, popBack(&opStack));

else

pushBack(outStack, pop(&opStack));

addStackElem(opStack, token);

logFile(fout, inpt\_str, token, opStack, "token", "add");

logFile(fout, inpt\_str, token, opStack, "stack", "curr");

}

continue;

}

if (IsLiteral(token)) {

addStackElem(outStack, variables.find(token)->second);

logFile(fout, inpt\_str, variables.find(token)->second, outStack, "token", "add");

logFile(fout, inpt\_str, variables.find(token)->second, outStack, "stack", "curr");

continue;

}

cerr << "[ERROR] unexpected token " << token << endl;

fout << "[ERROR] unexpected token " << token << endl;

back = TRUE;

\_getch();

return;

}

fout << "Строка пройдена." << endl;

fout << "Стек выхода = "; printStack(outStack, fout, FALSE); // FALSE - file output

fout << "Стек переменных = "; printStack(opStack, fout, FALSE); // FALSE - file output

fout << "Очищаем стек переменных, заполняя стек выхода" << endl;

while (opStack) {

if (sizeStack(opStack) >= 2)// если стэк операций больше двух

pushBack(outStack, popBack(&opStack));

else

pushBack(outStack, pop(&opStack));

}

fout << "Стек выхода = "; printStack(outStack, fout, FALSE); // FALSE - file output

fout << "Стек переменных = "; printStack(opStack, fout, FALSE); // FALSE - file output

fout << endl;

}

void polishNotation\_vec(string& inpt\_str, vector<string>& str\_tokens, vector<string>& outArr, vector<string>& opArr) {

string current\_str;

back = FALSE;

for (const auto& token : str\_tokens) {

if (IsNumber(token)) {

outArr.push\_back(token);

continue;

}

if (IsOperation(token, weight\_op)) {

if (token == "(") {

opArr.push\_back(token);

continue;

}

if (token == ")") {

while (opArr.size() == 0 && current\_str != "(") {

current\_str = opArr[opArr.size() - 1];

opArr.pop\_back();

outArr.push\_back(current\_str);

current\_str = opArr[opArr.size() - 1];

}

current\_str.clear();

if (opArr.size() == 0) {

cerr << "[ERROR] character ( is not found. Try another one" << endl;

\_getch();

back = TRUE;

return;

}

opArr.pop\_back();

continue;

}

if (opArr.size() == 0) {

opArr.push\_back(token);

continue;

}

if (checkWeight\_vec(token, opArr, weight\_op)) {

// Если вес текущего больше или равен предыдущему

opArr.push\_back(token);

}

else {

// Если вес текущего меньше предыдущего

outArr.push\_back(opArr[opArr.size() - 1]);

opArr.pop\_back();

opArr.push\_back(token);

}

continue;

}

if (IsLiteral(token)) {

outArr.push\_back(variables.find(token)->second);

continue;

}

cerr << "[ERROR] unexpected token " << token << endl;

\_getch();

back = TRUE;

return;

}

while (opArr.size() != 0) {

outArr.push\_back(opArr[opArr.size() - 1]);

opArr.pop\_back();

}

}

void normalPolishNotation(string& inpt\_str, Node\* &outStack, Node\* &opStack) {

outStack = NULL;

opStack = NULL;

vector<string> str\_tokens;

ofstream fout;

if (fl\_test)

fout.open("test\_solution.txt", ios::app);

else

fout.open("notation\_solution.txt", ios::app);

fout << endl << endl << "Преобразование выражения в нормальную польскую нотацию" << endl;

fout << "Строка, над которой происходит преобразование - " << input\_str << endl;

if (!fl\_test)

cout << "Строка - \"" << inpt\_str << "\"" << endl;

str\_tokens = Tokenizer(inpt\_str);

reverseTokens(str\_tokens);

polishNotation(inpt\_str, str\_tokens, outStack, opStack, fout);

if (back)

menu(inpt\_str, outStack, opStack);

outStack = reverseStack(outStack);

if (!fl\_test) {

cout << "Normal Стек выхода = ";

SetColor(2, 0);

printStack(outStack, fout, TRUE); // TRUE - console output

SetColor(7, 0);

}

fout.close();

}

void normalPolishNotation\_vec(string& inpt\_str, vector<string>& outArr, vector<string>& opArr) {

vector<string> str\_tokens;

str\_tokens = Tokenizer(inpt\_str);

reverseTokens(str\_tokens);

polishNotation\_vec(inpt\_str, str\_tokens, outArr, opArr);

if (back)

return;

reverseTokens(outArr);

}

void reversePolishNotation(string& inpt\_str, Node\*& outStack, Node\*& opStack) {

outStack = NULL;

opStack = NULL;

vector<string> str\_tokens;

ofstream fout;

if (fl\_test)

fout.open("test\_solution.txt", ios::app);

else

fout.open("notation\_solution.txt", ios::app);

fout << endl << endl << "Преобразование выражения в обратную польскую нотацию" << endl;

fout << "Строка, над которой происходит преобразование - " << input\_str << endl;

if (!fl\_test)

cout << "Строка - \"" << inpt\_str << "\"" << endl;

str\_tokens = Tokenizer(inpt\_str);

polishNotation(inpt\_str, str\_tokens, outStack, opStack, fout);

if (back)

menu(inpt\_str, outStack, opStack);

if (!fl\_test) {

cout << "Reverse Стек выхода = ";

SetColor(2, 0);

printStack(outStack, fout, TRUE); // TRUE - console output

SetColor(7, 0);

}

fout.close();

}

void reversePolishNotation\_vec(string& inpt\_str, vector<string>& outArr, vector<string>& opArr) {

vector<string> str\_tokens;

ofstream fout;

str\_tokens = Tokenizer(inpt\_str);

polishNotation\_vec(inpt\_str, str\_tokens, outArr, opArr);

}

/////////////////// OPZ ///////////////////

void printOut(vector<double>& res, ofstream& fout) {

for (const auto& token : res) {

fout << token << " ";

}

fout << endl;

}

double Calc\_PN(Node\* &outStack, bool fl\_notation) { // fl\_notation : FALSE - Reverse notation; TRUE - normal notation

back = FALSE;

vector<double> res;

double value1 = 0;

double value2 = 0;

string operation;

ofstream fout;

if (fl\_test)

fout.open("test\_solution.txt", ios::app);

else

fout.open("notation\_solution.txt", ios::app);

fout << endl << "Вычисление выражения" << endl; printStack(outStack, fout, FALSE); // FALSE - file output

if (fl\_notation) { // if normal polish notation

outStack = reverseStack(outStack);

fout << "Перевернем стек для удобства вычисления" << endl; printStack(outStack, fout, FALSE); // FALSE - file output

}

while (outStack) {

logFile(fout,"",outStack->value,outStack,"token", "curr");

if (IsNumber(outStack->value)) {

logFile(fout, "", outStack->value, outStack, "stack", "del");

res.push\_back(str2doub(pop(&outStack)));

logFile(fout, "", outStack->value, outStack, "stack", "curr");

fout << "Массив выхода: "; printOut(res, fout);

continue;

}

if (IsOperation(outStack->value, weight\_op)) {

if (res.size() >= 2) {

logFile(fout, "", outStack->value, outStack, "stack", "del");

operation = pop(&outStack);

if (fl\_notation) { // if normal polish notation

value1 = res[res.size() - 1];

res.pop\_back();

value2 = res[res.size() - 1];

res.pop\_back();

res.push\_back(Action(value1, value2, operation));

}

else {

value2 = res[res.size() - 1];

res.pop\_back();

value1 = res[res.size() - 1];

res.pop\_back();

res.push\_back(Action(value1, value2, operation));

}

fout << "Выполняем действие" << operation << " получаем результат: "; printOut(res, fout);

continue;

}

cerr << endl << "[ERROR] Выражение записано некорректно" << endl;

fout << endl << "[ERROR] Выражение записано некорректно" << endl;

cout << "Попробуйте снова." << endl;

fout << "Попробуйте снова." << endl;

back = TRUE;

\_getch();

return 0;

}

if (IsLiteral(outStack->value)) {

res.push\_back(str2doub(variables.find(outStack->value)->second));

logFile(fout, "", variables.find(outStack->value)->second, outStack, "token", "add");

logFile(fout, "", variables.find(outStack->value)->second, outStack, "stack", "curr");

pop(&outStack);

continue;

}

cerr << endl << "[ERROR] unexpected token " << outStack->value << endl;

fout << endl << "[ERROR] unexpected token " << outStack->value << endl;

cout << "Попробуйте снова." << endl;

fout << "Попробуйте снова." << endl;

back = TRUE;

\_getch();

return 0;

}

fout << "Вычисление закончено. Результат вычисления - " << res[0] << endl;

fout.close();

return res[0];

}

double Calc\_PN\_vec(vector <string>& outArr, bool fl\_notation) {

back = FALSE;

vector<double> res;

double value1 = 0;

double value2 = 0;

string operation;

if (fl\_notation) { // if normal polish notation

reverseTokens(outArr);

}

while (outArr.size() != 0) {

if (IsNumber(outArr[0])) {

res.push\_back(str2doub(outArr[outArr.size() - 1]));

outArr.erase(outArr.begin());

continue;

}

if (IsOperation(outArr[0], weight\_op)) {

if (res.size() >= 2) {

operation = outArr[0];

outArr.erase(outArr.begin());

if (fl\_notation) { // if normal polish notation

value1 = res[res.size() - 1];

res.pop\_back();

value2 = res[res.size() - 1];

res.pop\_back();

res.push\_back(Action(value1, value2, operation));

}

else {

value2 = res[res.size() - 1];

res.pop\_back();

value1 = res[res.size() - 1];

res.pop\_back();

res.push\_back(Action(value1, value2, operation));

}

continue;

}

cerr << endl << "[ERROR] Выражение записано некорректно" << endl;

cout << "Попробуйте снова." << endl;

back = TRUE;

\_getch();

return 0;

}

if (IsLiteral(outArr[0])) {

res.push\_back(str2doub(variables.find(outArr[0])->second));

outArr.erase(outArr.begin());

continue;

}

cerr << endl << "[ERROR] unexpected token " << outArr[0] << endl;

cout << "Попробуйте снова." << endl;

back = TRUE;

\_getch();

return 0;

}

if (res.size() == 1)

return res[0];

else {

cerr << endl << "[ERROR] Выражение записано некорректно" << endl;

cout << "Попробуйте снова." << endl;

back = TRUE;

\_getch();

return 0;

}

}

void convert\_exp(string& str, Node\* &outStack, Node\* &opStack) {

back = FALSE;

system("cls");

chrono::steady\_clock sc;

vector<string> outArr;

vector<string> opArr;

cout << "Выберите вид преобразования польской нотации:" << endl;

cout << "1. Прямая" << endl;

cout << "2. Обратная" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1': {

auto start = sc.now(); // устанавливаем начало отсчета времени события

cout << "Прямая польская нотация выражения" << endl;

normalPolishNotation(str, outStack, opStack);

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость проверки стеком равна: " << time\_span.count() << " секунд" << endl;

auto start2 = sc.now(); // устанавливаем начало отсчета времени события

normalPolishNotation\_vec(str, outArr, opArr);

auto end2 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span2 = static\_cast<chrono::duration<double>>(end2 - start2); // высчитываем время, затраченное на событие

cout << "Скорость проверки массивом равна: " << time\_span2.count() << " секунд" << endl;

break;

}

case '2': {

auto start1 = sc.now(); // устанавливаем начало отсчета времени события

cout << "Обратная польская нотация выражения" << endl;

reversePolishNotation(str, outStack, opStack);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

cout << "Скорость проверки стеком равна: " << time\_span1.count() << " секунд" << endl;

auto start3 = sc.now(); // устанавливаем начало отсчета времени события

reversePolishNotation\_vec(str, outArr, opArr);

auto end3 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span3 = static\_cast<chrono::duration<double>>(end3 - start3); // высчитываем время, затраченное на событие

cout << "Скорость проверки массивом равна: " << time\_span3.count() << " секунд" << endl;

break;

}

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

convert\_exp(str, outStack, opStack);

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

convert\_exp(str, outStack, opStack);

}

cout << "Выражение записано. Нажмите любую кнопку для продолжения.";

\_getch();

variables.clear();

menu(str, outStack, opStack);

}

void check\_exp(string& str, Node\*& outStack, Node\*& opStack) {

chrono::steady\_clock sc;

vector<string> outArr;

vector<string> opArr;

vector<string> str\_tokens;

string temp1, temp2;

system("cls");

bool notation;

outStack = NULL;

opStack = NULL;

cout << "Выберите способ ввода выражения:" << endl;

cout << "1. Через консоль" << endl;

cout << "2. С помощью файла" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1':

fill\_str\_self(str, outStack, opStack);

break;

case '2':

fill\_str\_via\_file(str, outStack, opStack, check\_file\_exp);

break;

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

check\_exp(str, outStack, opStack);

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

check\_exp(str, outStack, opStack);

}

notation = getCalcNotation();

auto start = sc.now(); // устанавливаем начало отсчета времени события

if (str.size() > 1) {

temp1 += str[0];

temp2 += str[str.size() - 1];

if (IsOperation(temp1, check\_op) || IsOperation(temp2, check\_op)) {

cerr << endl << "[ERROR] Выражение записано некорректно" << endl;

cout << "Попробуйте снова." << endl;

\_getch();

menu(str, outStack, opStack);

}

}

if (notation)

normalPolishNotation\_vec(str, outArr, opArr);

else

reversePolishNotation\_vec(str, outArr, opArr);

if (back)

menu(str, outStack, opStack);

Calc\_PN\_vec(outArr, notation);

if (back)

menu(str, outStack, opStack);

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

auto start1 = sc.now(); // устанавливаем начало отсчета времени события

if (notation)

normalPolishNotation(str, outStack, opStack);

else

reversePolishNotation(str, outStack, opStack);

Calc\_PN(outStack, notation);

if (back)

menu(str, outStack, opStack);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

SetColor(2, 0);

cout << endl << "Выражение корректно." << endl;

SetColor(7, 0);

cout << "Скорость проверки стеком равна: " << time\_span1.count() << " секунд" << endl;

cout << "Скорость проверки массивом равна: " << time\_span.count() << " секунд" << endl;

cout << "Нажмите любую кнопку для продолжения.";

\_getch();

variables.clear();

menu(str, outStack, opStack);

}

void check\_PN(string& str, Node\*& outStack, Node\*& opStack) {

chrono::steady\_clock sc;

vector<string> outArr;

vector<string> opArr;

system("cls");

bool notation = getCalcNotation();

system("cls");

outStack = NULL;

opStack = NULL;

vector <string> str\_tokens;

cout << "Выберите способ ввода выражения:" << endl;

cout << "1. Через консоль" << endl;

cout << "2. С помощью файла" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1':

fill\_str\_self(str, outStack, opStack);

break;

case '2':

fill\_str\_via\_file(str, outStack, opStack, check\_file\_pn);

break;

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

check\_PN(str, outStack, opStack);

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

check\_PN(str, outStack, opStack);

}

auto start1 = sc.now(); // устанавливаем начало отсчета времени события

str\_tokens = Tokenizer(str);

Calc\_PN\_vec(str\_tokens, notation);

if (back)

menu(str, outStack, opStack);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

auto start = sc.now(); // устанавливаем начало отсчета времени события

str\_tokens = Tokenizer(str);

for (const auto& token : str\_tokens) {

addStackElem(outStack, token);

}

Calc\_PN(outStack, notation);

if (back)

menu(str, outStack, opStack);

SetColor(2, 0);

cout << "Выражение корректно." << endl;

SetColor(7, 0);

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость проверки стеком равна: " << time\_span.count() << " секунд" << endl;

cout << "Скорость проверки массивом равна: " << time\_span1.count() << " секунд" << endl;

cout << "Нажмите любую кнопку для продолжения.";

\_getch();

variables.clear();

menu(str, outStack, opStack);

}

bool getCalcNotation() {

system("cls");

cout << "Выберите нотацию, по которой производить вычисление:" << endl;

cout << "1. Прямая " << endl;

cout << "2. Обратная" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1':

cout << "Прямая польская нотация выражения" << endl;

return TRUE;

case '2':

cout << "Обратная польская нотация выражения" << endl;

return FALSE;

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

getCalcNotation();

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

getCalcNotation();

}

}

void Calculate\_exp(string& str, Node\*& outStack, Node\*& opStack) {

chrono::steady\_clock sc;

variables.clear();

vector <string> outArr;

vector <string> opArr;

string temp1, temp2;

system("cls");

bool notation;

outStack = NULL;

opStack = NULL;

cout << "Выберите способ ввода выражения:" << endl;

cout << "1. Через консоль" << endl;

cout << "2. С помощью файла" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1':

fill\_str\_self(str, outStack, opStack);

break;

case '2':

fill\_str\_via\_file(str, outStack, opStack, input\_file);

break;

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

Calculate\_exp(str, outStack, opStack);

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

Calculate\_exp(str, outStack, opStack);

}

notation = getCalcNotation();

auto start1 = sc.now(); // устанавливаем начало отсчета времени события

if (str.size() > 1) {

temp1 += str[0];

temp2 += str[str.size() - 1];

if (IsOperation(temp1, check\_op) || IsOperation(temp2, check\_op)) {

cerr << endl << "[ERROR] Выражение записано некорректно" << endl;

cout << "Попробуйте снова." << endl;

\_getch();

menu(str, outStack, opStack);

}

}

if (notation)

normalPolishNotation\_vec(str, outArr, opArr);

else

reversePolishNotation\_vec(str, outArr, opArr);

if (back)

menu(str, outStack, opStack);

Calc\_PN\_vec(outArr, notation);

if (back)

menu(str, outStack, opStack);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

auto start = sc.now(); // устанавливаем начало отсчета времени события

if (notation)

normalPolishNotation(str, outStack, opStack);

else

reversePolishNotation(str, outStack, opStack);

double res = Calc\_PN(outStack, notation);

if (back)

menu(str, outStack, opStack);

cout << endl << "Результат = " << res << endl;

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость вычисления стеком равна: " << time\_span.count() << " секунд" << endl;

cout << "Скорость вычисления массивом равна: " << time\_span1.count() << " секунд" << endl;

cout << "DONE! Нажмите любую кнопку для продолжения.";

\_getch();

variables.clear();

menu(str, outStack, opStack);

}

void Calculate\_pn(string& str, Node\*& outStack, Node\*& opStack) {

chrono::steady\_clock sc;

variables.clear();

system("cls");

bool notation = getCalcNotation();

system("cls");

outStack = NULL;

opStack = NULL;

vector <string> str\_tokens;

cout << "Выберите способ ввода выражения:" << endl;

cout << "1. Через консоль" << endl;

cout << "2. С помощью файла" << endl;

cout << "Ваш выбор: ";

char value[256]; // переменная, которая хранит выбранное значение

cin >> value; // считываем выбранное значение

if (strlen(value) == 1) { // проверяем количество введенных символов. Если много, то просим ввести еще раз, иначе проверям дальше

switch (value[0]) { // проверям, взяв первый символ переменной value

case '1':

fill\_str\_self(str, outStack, opStack);

break;

case '2':

fill\_str\_via\_file(str, outStack, opStack, calc\_notation\_file);

break;

default: // если число не подходит ни к одному из

cout << "Число введено неверно. Введите заново" << endl;

Calculate\_pn(str, outStack, opStack);

}

}

else { // если введено символов больше необходимого

cout << "Необходимо ввести один символ. Попробуйте ввести заново" << endl;

Calculate\_pn(str, outStack, opStack);

}

auto start1 = sc.now(); // устанавливаем начало отсчета времени события

str\_tokens = Tokenizer(str);

Calc\_PN\_vec(str\_tokens, notation);

if (back)

menu(str, outStack, opStack);

auto end1 = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span1 = static\_cast<chrono::duration<double>>(end1 - start1); // высчитываем время, затраченное на событие

auto start = sc.now(); // устанавливаем начало отсчета времени события

str\_tokens = Tokenizer(str);

for (const auto& token : str\_tokens) {

addStackElem(outStack, token);

}

double res = Calc\_PN(outStack, notation);

if (back)

menu(str, outStack, opStack);

cout << endl << "Результат = " << res << endl;

auto end = sc.now(); // устанавливаем конец отсчета времени события

auto time\_span = static\_cast<chrono::duration<double>>(end - start); // высчитываем время, затраченное на событие

cout << "Скорость вычисления стеком равна: " << time\_span.count() << " секунд" << endl;

cout << "Скорость вычисления массивом равна: " << time\_span1.count() << " секунд" << endl;

cout << "DONE! Нажмите любую кнопку для продолжения.";

\_getch();

variables.clear();

menu(str, outStack, opStack);

}

/////////////////// INPUT /////////////////////

void fill\_str\_self(string& str, Node\*& outStack, Node\*& opStack) {

system("cls");

cout.width(15);

cout << right << "\_\_Запись через консоль\_\_" << endl << endl;

str.clear();

cin.clear();

//cin.ignore();

cout << "Введите строку выражения (Каждый символ отделяйте пробелом): ";

getline(cin, str);

if (str.size() == 0) {

cout << "Выражение записано некорректно. Попробуйте снова" << endl;

fill\_str\_self(str, outStack, opStack);

return;

}

cout << "Полученная строка - " << str << endl;

check\_literals(str, outStack, opStack);

cout << "Выражение записано. Нажмите любую кнопку для продолжения." << endl;

\_getch();

}

void fill\_str\_via\_file(string& str, Node\*& outStack, Node\*& opStack, string& filename) {

system("cls");

cout.width(15);

cout << right << "\_\_Запись через файл\_\_" << endl;

str.clear();

string temp;

ifstream fin;

fin.open(filename);

if (fin.is\_open()) {

while (!fin.eof()) {

getline(fin, temp);

str += temp;

}

}

else {

cout << "Файл не удалось открыть. Закройте программу и создайте файл \"" << filename <<"\" в корневой папке программы и попробуйте снова." << endl;

cout << "Нажмите любую клавишу, чтобы закрыть программу" << endl;

\_getch();

exit(3);

}

if (str.size() == 0) {

cout << "Файл пуст. Закройте программу и заполните файл \"" << filename << "\" в корневой папке программы данными и попробуйте снова." << endl;

cout << "Нажмите любую клавишу, чтобы закрыть программу" << endl;

\_getch();

exit(3);

}

cout << endl << "Полученная строка - " << str << endl;

check\_literals(str, outStack, opStack);

cout << "Выражение записано. Нажмите любую кнопку для продолжения." << endl;

fin.close();

\_getch();

}

void add\_variable(char str) {

string value;

string temp\_str;

bool fl\_stop = FALSE;

temp\_str += str;

if (variables.find(temp\_str) != variables.end())

return;

temp\_str.clear();

while (!fl\_stop) {

fl\_stop = TRUE;

cout << "Введите значение переменной " << str << " : ";

cin >> value;

for (int i = 0; i <= value.size() - 1; i++) {

if (isalpha(value[i])) {

fl\_stop = FALSE;

cout << "Значение введено некорректно, попробуйте снова" << endl;

break;

}

}

}

temp\_str += str; // map не принимает один символ за стринг, поэтому добавляю к временному string

variables.insert(pair<string, string>(temp\_str, value));

temp\_str.clear();

}

void check\_literals(string& str, Node\*& outStack, Node\*& opStack) {

string temp;

for (int i = 0; i <= str.size() - 1; i++) {

if (isalpha(str[i])) {

if (i == 0) {

if (str[i + 1] == ' ') {

add\_variable(str[i]);

continue;

}

}

if (i >= 1 and i != str.size() - 1) {

if (str[i - 1] == ' ' and str[i + 1]) {

add\_variable(str[i]);

continue;

}

}

if (i == str.size() - 1) {

if (str[i - 1] == ' ') {

add\_variable(str[i]);

continue;

}

}

cerr << "[ERROR] unexpected token " << str[i] << endl;

exit(-1);

}

}

}

/////////////////// EXPRESSION /////////////////////

////////////////// TEST //////////////////////

string getOperation() {

vector<string> operators{ "+","\*","/","-" };

return operators[rand() % (operators.size())];

}

string getNumeric(char op) {

string value;

string temp;

vector<string> numbers{ "1","2","3","4","5","6","7","8","9","0" };

bool rank = rand() % 2;

bool stop = TRUE;

int count = 0;

int rang;

rank ? rang = 1 : rang = 2;

do {

if (op == '/' && stop) {

temp = numbers[rand() % numbers.size()];

while (temp == "0") {

temp = numbers[rand() % numbers.size()];

}

value += temp;

stop = FALSE;

}

else {

value += numbers[rand() % numbers.size()];

}

count++;

} while (count != rang);

return value;

}

void createTask(string& task) {

bool variant = FALSE;

task += getNumeric(' ');

for (int i = 1; i <= 13; i++) {

if (i % 2 == 0) {

variant ? task += getNumeric(task[task.size() - 1 - 1]): task +=getOperation();

variant = !variant;

}

else

task += " ";

}

}

void createTest(string& str, Node\*& outStack, Node\*& opStack) {

system("cls");

fl\_test = TRUE;

int count\_of\_ex;

int index = 1;

Test test;

bool notation = FALSE;

ofstream fout;

fout.open("test\_tasks.txt", ios::app);

cout << "Введите количество заданий: ";

count\_of\_ex = checkdigit();

test.Task.resize(count\_of\_ex);

fout << "ПРОВЕРОЧНАЯ РАБОТА НА ПРЕОБРАЗОВАНИЕ ВЫРАЖЕНИЮ В ПОЛЬСКУЮ НОТАЦИЮ" << endl;

fout << "Количество заданий в работе: " << count\_of\_ex << endl;

for (auto& ex : test.Task) {

createTask(ex.task);

fout << index << ". Перевести выражение " << ex.task;

if (notation) {

fout << " в нормальную польскую нотацию." << endl;

normalPolishNotation(ex.task, outStack, opStack);

}

else {

fout << " в обратную польскую нотацию." << endl;

reversePolishNotation(ex.task, outStack, opStack);

}

ex.answer = Calc\_PN(outStack, notation);

index++;

notation = !notation;

}

cout << "Тестовые задания созданы. Задания находятся в файле test\_task.txt. Побдробое решение находится в файле test\_solution.txt" << endl;

cout << "Нажмите любую кнопку для продолжения.";

fl\_test = FALSE;

fout.close();

\_getch();

menu(str, outStack, opStack);

}

////////////////// TEST //////////////////////

////////////////// MENU ////////////////////////

void submenu\_check\_pn\_conf\_val(string& str, Node\*& outStack, Node\*& opStack) {

switch (subm\_check\_pn\_count)

{

case 0:

check\_exp(str, outStack, opStack);

break;

case 1:

check\_PN(str, outStack, opStack);

break;

case 2:

menu(str, outStack, opStack);

break;

default:

break;

}

}

void submenu\_calc\_pn\_conf\_val(string& str, Node\*& outStack, Node\*& opStack) {

switch (subm\_calc\_pn\_count)

{

case 0:

Calculate\_exp(str, outStack, opStack);

break;

case 1:

Calculate\_pn(str, outStack, opStack);

break;

case 2:

menu(str, outStack, opStack);

break;

default:

break;

}

}

void submenu\_convert\_exp\_conf\_val(string& str, Node\*& outStack, Node\*& opStack) {

switch (subm\_conv\_count) {

case 0:

fill\_str\_self(input\_str, outStack, opStack);

break;

case 1:

fill\_str\_via\_file(input\_str, outStack, opStack, input\_file);

break;

case 2:

menu(str, outStack, opStack);

break;

default:

break;

}

convert\_exp(str, outStack, opStack);

}

void conf\_val(string& str, Node\*& outStack, Node\*& opStack) {

switch (m\_count) {

case 0:

submenu\_convert\_exp(str, outStack, opStack);

break;

case 1:

submenu\_check\_pn(str, outStack, opStack);

break;

case 2:

submenu\_calc\_pn(str, outStack, opStack);

break;

case 3:

createTest(str, outStack, opStack);

break;

case 4:

queue();

break;

case 5:

SetColor(0, 0);

exit(0);

break;

default:

break;

}

menu(str, outStack, opStack);

}

void submenu\_check\_pn\_choice(string& str, Node\*& outStack, Node\*& opStack) {

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

subm\_check\_pn\_count--;

if (subm\_check\_pn\_count < 0) subm\_check\_pn\_count = 0;

submenu\_check\_pn(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_check\_pn\_count++;

if (subm\_check\_pn\_count > 2) subm\_check\_pn\_count = 2;

submenu\_check\_pn(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_check\_pn\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_check\_pn\_choice(str, outStack, opStack);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_check\_pn\_count--;

if (subm\_check\_pn\_count < 0) subm\_check\_pn\_count = 0;

submenu\_check\_pn(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_check\_pn\_count++;

if (subm\_check\_pn\_count > 2) subm\_check\_pn\_count = 2;

submenu\_check\_pn(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_check\_pn\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_check\_pn\_choice(str, outStack, opStack);

}

}

void submenu\_calc\_pn\_choice(string& str, Node\*& outStack, Node\*& opStack) {

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

subm\_calc\_pn\_count--;

if (subm\_calc\_pn\_count < 0) subm\_calc\_pn\_count = 0;

submenu\_calc\_pn(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_calc\_pn\_count++;

if (subm\_calc\_pn\_count > 2) subm\_calc\_pn\_count = 2;

submenu\_calc\_pn(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_calc\_pn\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_calc\_pn\_choice(str, outStack, opStack);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_calc\_pn\_count--;

if (subm\_calc\_pn\_count < 0) subm\_calc\_pn\_count = 0;

submenu\_calc\_pn(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_calc\_pn\_count++;

if (subm\_calc\_pn\_count > 2) subm\_calc\_pn\_count = 2;

submenu\_calc\_pn(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_calc\_pn\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_calc\_pn\_choice(str, outStack, opStack);

}

}

void submenu\_convert\_exp\_choice(string& str, Node\*& outStack, Node\*& opStack) {

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

subm\_conv\_count--;

if (subm\_conv\_count < 0) subm\_conv\_count = 0;

submenu\_convert\_exp(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_conv\_count++;

if (subm\_conv\_count > 2) subm\_conv\_count = 2;

submenu\_convert\_exp(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_convert\_exp\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_convert\_exp\_choice(str, outStack, opStack);

}

}

switch (k1) {

case 0x48: // стрелка вверх

subm\_conv\_count--;

if (subm\_conv\_count < 0) subm\_conv\_count = 0;

submenu\_convert\_exp(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

subm\_conv\_count++;

if (subm\_conv\_count > 2) subm\_conv\_count = 2;

submenu\_convert\_exp(str, outStack, opStack);

break;

case 0xD: // подтвердить

submenu\_convert\_exp\_conf\_val(str, outStack, opStack);

break;

default:

submenu\_convert\_exp\_choice(str, outStack, opStack);

}

}

void menu\_choice(string& str, Node\*& outStack, Node\*& opStack) { // в зависимости от стрелок изменяем меню

int k1;

k1 = \_getch(); // получаем символ стрелки без вывода знака

if (k1 == 0xE0) { // если стрелки

switch (k1) {

case 0x48: // стрелка вверх

m\_count--;

if (m\_count < 0) m\_count = 0;

menu(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

m\_count++;

if (m\_count > 5) m\_count = 5;

menu(str, outStack, opStack);

break;

case 0xD: // подтвердить

conf\_val(str, outStack, opStack);

break;

default:

menu\_choice(str, outStack, opStack);

}

}

switch (k1) {

case 0x48: // стрелка вверх

m\_count--;

if (m\_count < 0) m\_count = 0;

menu(str, outStack, opStack);

break;

case 0x50: // стрелка вниз

m\_count++;

if (m\_count > 5) m\_count = 5;

menu(str, outStack, opStack);

break;

case 0xD: // подтвердить

conf\_val(str, outStack, opStack);

break;

default:

menu\_choice(str, outStack, opStack);

}

}

void submenu\_check\_pn(string& str, Node\*& outStack, Node\*& opStack) {

system("cls");

if (subm\_check\_pn\_count == 0) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

SetColor(5, 0);

cout << "Числовое выражение" << endl;

SetColor(7, 0);

cout << "Польская нотация" << endl;

cout << "Назад" << endl;

submenu\_check\_pn\_choice(str, outStack, opStack);

}

if (subm\_check\_pn\_count == 1) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

cout << "Числовое выражение" << endl;

SetColor(5, 0);

cout << "Польская нотация" << endl;

SetColor(7, 0);

cout << "Назад" << endl;

submenu\_check\_pn\_choice(str, outStack, opStack);

}

if (subm\_check\_pn\_count == 2) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

cout << "Числовое выражение" << endl;

cout << "Польская нотация" << endl;

SetColor(5, 0);

cout << "Назад" << endl;

SetColor(7, 0);

submenu\_check\_pn\_choice(str, outStack, opStack);

}

}

void submenu\_calc\_pn(string& str, Node\*& outStack, Node\*& opStack) {

system("cls");

if (subm\_calc\_pn\_count == 0) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

SetColor(5, 0);

cout << "Числовое выражение" << endl;

SetColor(7, 0);

cout << "Польская нотация" << endl;

cout << "Назад" << endl;

submenu\_calc\_pn\_choice(str, outStack, opStack);

}

if (subm\_calc\_pn\_count == 1) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

cout << "Числовое выражение" << endl;

SetColor(5, 0);

cout << "Польская нотация" << endl;

SetColor(7, 0);

cout << "Назад" << endl;

submenu\_calc\_pn\_choice(str, outStack, opStack);

}

if (subm\_calc\_pn\_count == 2) {

cout.width(15);

cout << right << "\_\_Вычисление выражения\_\_" << endl << endl;

SetColor(2, 0);

cout << "Что будет в качестве выражения?" << endl << endl;

SetColor(7, 0);

cout << "Числовое выражение" << endl;

cout << "Польская нотация" << endl;

SetColor(5, 0);

cout << "Назад" << endl;

SetColor(7, 0);

submenu\_calc\_pn\_choice(str, outStack, opStack);

}

}

void submenu\_convert\_exp(string& str, Node\*& outStack, Node\*& opStack) {

system("cls");

if (subm\_conv\_count == 0) {

cout.width(15);

cout << right << "\_\_Преобразование выражение в ПН\_\_" << endl << endl;

SetColor(2, 0);

cout << "Каким образом будет записано выражение?" << endl << endl;

SetColor(7, 0);

SetColor(5, 0);

cout << "С помощью консоли" << endl;

SetColor(7, 0);

cout << "С помощью файла" << endl;

cout << "Назад" << endl;

submenu\_convert\_exp\_choice(str, outStack, opStack);

}

if (subm\_conv\_count == 1) {

cout.width(15);

cout << right << "\_\_Преобразование выражение в ПН\_\_" << endl << endl;

SetColor(2, 0);

cout << "Каким образом будет записано выражение?" << endl << endl;

SetColor(7, 0);

cout << "С помощью консоли" << endl;

SetColor(5, 0);

cout << "С помощью файла" << endl;

SetColor(7, 0);

cout << "Назад" << endl;

submenu\_convert\_exp\_choice(str, outStack, opStack);

}

if (subm\_conv\_count == 2) {

cout.width(15);

cout << right << "\_\_Преобразование выражение в ПН\_\_" << endl << endl;

SetColor(2, 0);

cout << "Каким образом будет записано выражение?" << endl << endl;

SetColor(7, 0);

cout << "С помощью консоли" << endl;

cout << "С помощью файла" << endl;

SetColor(5, 0);

cout << "Назад" << endl;

SetColor(7, 0);

submenu\_convert\_exp\_choice(str, outStack, opStack);

}

}

void menu(string& str, Node\*& outStack, Node\*& opStack) { // меню

system("cls");

if (m\_count == 0) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

SetColor(5, 0);

cout << "Преобразовать выражение в ПН" << endl;

SetColor(7, 0);

cout << "Проверить ПН" << endl;

cout << "Вычислить выражение с помощью ПН" << endl;

cout << "Создание теста" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(str, outStack, opStack);

}

if (m\_count == 1) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Преобразовать выражение в ПН" << endl;

SetColor(5, 0);

cout << "Проверить ПН" << endl;

SetColor(7, 0);

cout << "Вычислить выражение с помощью ПН" << endl;

cout << "Создание теста" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(str, outStack, opStack);

}

if (m\_count == 2) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Преобразовать выражение в ПН" << endl;

cout << "Проверить ПН" << endl;

SetColor(5, 0);

cout << "Вычислить выражение с помощью ПН" << endl;

SetColor(7, 0);

cout << "Создание теста" << endl;

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(str, outStack, opStack);

}

if (m\_count == 3) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Преобразовать выражение в ПН" << endl;

cout << "Проверить ПН" << endl;

cout << "Вычислить выражение с помощью ПН" << endl;

SetColor(5, 0);

cout << "Создание теста" << endl;

SetColor(7, 0);

cout << "Задание по варианту" << endl;

cout << "Выход" << endl;

menu\_choice(str, outStack, opStack);

}

if (m\_count == 4) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Преобразовать выражение в ПН" << endl;

cout << "Проверить ПН" << endl;

cout << "Вычислить выражение с помощью ПН" << endl;

cout << "Создание теста" << endl;

SetColor(5, 0);

cout << "Задание по варианту" << endl;

SetColor(7, 0);

cout << "Выход" << endl;

menu\_choice(str, outStack, opStack);

}

if (m\_count == 5) {

cout.width(15);

cout << right << "\_\_МЕНЮ\_\_" << endl;

cout << "Преобразовать выражение в ПН" << endl;

cout << "Проверить ПН" << endl;

cout << "Вычислить выражение с помощью ПН" << endl;

cout << "Создание теста" << endl;

cout << "Задание по варианту" << endl;

SetColor(5, 0);

cout << "Выход" << endl;

SetColor(7, 0);

menu\_choice(str, outStack, opStack);

}

}

////////////////// MENU ////////////////////////

void logFile(ofstream& fout, string str, const string& token, Node\* Stack, string object, string Action) {

if (object == "str") {

if (Action == "add") {

fout << "Строка, над которой происходит преобразование - " << str << endl;

}

}

if (object == "token") {

if (Action == "curr") {

fout << "Текущий элемент - " << token << endl;

return;

}

if (Action == "add") {

fout << "Добавление элемента " << token << endl;

return;

}

}

if (object == "stack") {

if (Action == "curr") {

fout << "Текущий стек = ";

printStack(Stack, fout, FALSE); // FALSE - file output

return;

}

if (Action == "del") {

fout << "Удаление текущего элемента " << token << " из стека " << endl;

return;

}

}

}

int main() {

setlocale(0, "");

Node\* outStack = NULL;

Node\* opStack = NULL;

srand(time(NULL));

menu(input\_str, outStack, opStack);

system("pause");

return 0;

}