**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Информационных систем**

отчет

**по лабораторной работе №3**

**по дисциплине «Программирование»**

Тема: Польская нотация. Стек и очередь

|  |  |  |
| --- | --- | --- |
| Студент гр. 0323 |  | Кольцов К.Э |
| Преподаватель |  | Глущенко А.Г. |

Санкт-Петербург

2021

**Цель работы.**

получение практических навыков работы со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных.

Необходимо написать программу, которая выполняет следующее:

1.   Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и \* с файла.

2.   Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и \* с файла.

3.   Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя). Ввод выражения должен быть реализовать двумя способами: с клавиатуры и с файла.

\*4.   Генерирует несколько (на выбор пользователя) вариантов проверочной работы по польской нотации (прямой и обратной). Задание и ответы к ним необходимо вывести в отдельные файлы (ответы должны быть максимально подробными).

Программа должна выводить и описывать все промежуточные действия.

**Основные теоретические положения.**

Обратная польская запись (нотация) – 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 них то, что инфиксные операторы имеют приоритеты, которые произвольны и нежелательны.

Рассмотрим пример: 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.

323 324 5 21 - + \*

1. Исходное состояние. Стек пуст.
2. Читаем значение 323. Это - число. Переносим в стек;
3. Читаем значение 324. Это - число. Переносим в стек;
4. Читаем значение 5. Это - число. Переносим в стек;
5. Читаем значение 21. Это - число. Переносим в стек;
6. Читаем значение -. Это оператор. Забираем числа 5 и 21 из стека. Вычитаем. Результат переносим в стек;
7. Читаем значение +. Это оператор. Забираем числа 324 и -16 из стека. Складываем. Результат отправляем в стек.
8. Читаем значение \*. Это оператор. Забираем числа 323 и 308 из стека. Умножаем. Результат отправляем в стек
9. Выражение прочитано. В стеке - одно значение - 631. Это и есть результат вычисления.

**Обработка результатов эксперимента.**

Программа выводит результат корректной обработки строки и правильно находит подстроку в предложении.

**Выводы.**

В ходе данной лабораторной работы я изучил работу стека и очереди на примере собственных структур в c++, и изучил разные виды записи математических выражений.

ПРИЛОЖЕНИЕ А

Листинг программы

#include <iostream>

#include <string>

double answer = 0;

struct Stack {

char symbol;

Stack\* tail;

};

struct NumStack {

double num;

NumStack\* tail;

};

Stack\* newStack(char);

void pushStack(Stack\*\*, char);

char popStack(Stack\*\*);

NumStack\* newStack(int);

void pushStack(NumStack\*\*, int);

double popStack(NumStack\*\*);

std::string toPostfix(std::string);

std::string truePostfix(std::string);

int main() {

setlocale(LC\_ALL, "Russian"); //для корректного вывода руских букв

std::string str = "";

char cha;

do {

std::cout << "Введите режим работы\n" \

"p: проверка и расчет обратной польской нотации\n"\

"n: проверка и расчет обычной записи\n"\

"e: выход\n";

std::cin >> cha;

switch (cha)

{

case 'p':

std::cin.ignore(INT\_MAX, '\n');

std::getline(std::cin, str);

if (truePostfix(str) != "Invalid input")

std::cout << answer << '\n';

else

std::cout << "ошибка в польской записи";

break;

case 'n':

std::cin.ignore(INT\_MAX, '\n');

std::getline(std::cin, str);

str = toPostfix(str);

std::cout << str << '\n';

if (truePostfix(str) != "Invalid input")

std::cout << answer << '\n';

else

std::cout << "ошибка в польской записи";

break;

case 'e':

str = "e";

break;

default:

break;

}

} while ((cha != 'e') && (str[0] != 'e'));

system("pause");

}

Stack\* newStack(char data) {

return new Stack{ data, NULL };

}

void pushStack(Stack\*\* sta, char data) {

if (\*sta == NULL)

\*sta = newStack(data);

else

\*sta = new Stack{ data, \*sta };

}

char popStack(Stack\*\* sta) {

char num = (\*sta)->symbol;

Stack\* del = \*sta;

\*sta = (\*sta)->tail;

delete del;

return num;

}

NumStack\* newStack(double data) {

return new NumStack{ data, NULL };

}

void pushStack(NumStack\*\* sta, double data) {

if (\*sta == NULL)

\*sta = newStack(data);

else

\*sta = new NumStack{ data, \*sta };

}

double popStack(NumStack\*\* sta) {

double num = (\*sta)->num;

NumStack\* del = \*sta;

\*sta = (\*sta)->tail;

delete del;

return num;

}

std::string toPostfix(std::string str) {

std::string postfix = "";

char\* point = &str[0];

Stack\* sta = NULL;

bool lastNum = false, lastBracket = false, lastSymbol = false;

while (\*point != 0) {

switch (\*point)

{

case '0':

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

if (lastNum)

return "Invalid input";

if (lastBracket) {

while ((sta != NULL) && (sta->symbol != '(') && (sta->symbol != '-') && (sta->symbol != '+')) {

postfix += ' ';

postfix += popStack(&sta);

}

pushStack(&sta, '\*');

}

if ((postfix != "") && (postfix != "-"))

postfix += " ";

postfix += \*point;

while ((\*(point + 1) > '0') && (\*(point + 1) < '9')) {

point++;

postfix += \*point;

}

lastNum = true;

lastBracket = false;

lastSymbol = false;

break;

case '+':

if (postfix == "")

return "Invalid input";

case '-':

if (lastSymbol)

return "Invalid input";

if (postfix == "")

postfix = "-";

else {

while ((sta != NULL) && (sta->symbol != '(')) {

postfix += ' ';

postfix += popStack(&sta);

}

pushStack(&sta, \*point);

}

lastNum = false;

lastBracket = false;

lastSymbol = true;

break;

case '\*':

case '/':

if (lastSymbol)

return "Invalid input";

if (postfix == "")

return "Invalid input";

else {

while ((sta != NULL) && (sta->symbol != '(') && (sta->symbol != '-') && (sta->symbol != '+')) {

postfix += ' ';

postfix += popStack(&sta);

}

pushStack(&sta, \*point);

}

lastNum = false;

lastBracket = false;

lastSymbol = true;

break;

case '(':

if (lastNum || lastBracket) {

while ((sta != NULL) && (sta->symbol != '(') && (sta->symbol != '-') && (sta->symbol != '+')) {

postfix += ' ';

postfix += popStack(&sta);

}

pushStack(&sta, '\*');

}

pushStack(&sta, \*point);

lastNum = false;

lastBracket = false;

lastSymbol = false;

break;

case ')':

if (lastSymbol)

return "Invalid input";

if (sta->symbol == '(')

return "Invalid input";

while (sta->symbol != '(') {

postfix += ' ';

postfix += popStack(&sta);

if (sta == NULL) {

std::cout << postfix << "\n";

return "Invalid input";

}

}

popStack(&sta);

lastNum = false;

lastBracket = true;

lastSymbol = false;

break;

case ' ':

break;

default:

return "Invalid input";

}

point++;

}

while (sta != NULL) {

postfix += ' ';

postfix += popStack(&sta);

}

return postfix;

}

std::string truePostfix(std::string postfix) {

NumStack\* numbers = NULL;

char\* point = &postfix[0];

double num, a, b;

bool minus;

while (\*point != 0) {

switch (\*point)

{

case '0':

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

num = \*point - '0';

minus = false;

if ((point - &postfix[0] == 1) && (postfix[0] == '-'))

minus = true;

while ((\*(point + 1) > '0') && (\*(point + 1) < '9')) {

point++;

num = 10 \* num + \*point - '0';

}

if (minus)

num = -num;

pushStack(&numbers, num);

break;

case '+':

if ((numbers != NULL) && (numbers->tail != NULL)) {

b = popStack(&numbers);

a = popStack(&numbers);

pushStack(&numbers, a + b);

}

else

return "Invalid input";

break;

case '-':

if ((numbers != NULL) && (numbers->tail != NULL)) {

b = popStack(&numbers);

a = popStack(&numbers);

pushStack(&numbers, a - b);

}

else

return "Invalid input";

break;

case '\*':

if ((numbers != NULL) && (numbers->tail != NULL)) {

b = popStack(&numbers);

a = popStack(&numbers);

pushStack(&numbers, a \* b);

}

else

return "Invalid input";

break;

case '/':

if ((numbers != NULL) && (numbers->tail != NULL)) {

b = popStack(&numbers);

a = popStack(&numbers);

if (b == 0)

return "Invalid input";

pushStack(&numbers, a / b);

}

else

return "Invalid input";

break;

case ' ':

break;

default:

return "Invalid input";

}

point++;

}

if (numbers == NULL)

return "Invalid input";

else {

if (numbers->tail != NULL)

return "Invalid input";

answer = popStack(&numbers);

return "";

}

}