**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №3**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2373 |  | Чекстер А. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**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. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (последним пришел – первым ушел).

Многие языки программирования располагают встроенными средствами организации и обработки стеков. Одним из них является C++.

**stack** – контейнер, в котором добавление и удаление элементов осуществляется с одного конца, что свойственно непосредственно стеку. Использование стековых операций не требует их описания в программе, т. е. **stack** здесь предоставляет набор стандартных функций.

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.

Алгоритм:

1. Если этот символ - число (или переменная), то просто помещаем его в выходную строку.  
2. Если символ - знак операции (+, -, \*, / ), то проверяем приоритет данной операции. Операции умножения и деления имеют наивысший приоритет. Операции сложения и вычитания имеют меньший приоритет. Наименьший приоритет имеет открывающая скобка.

**Постановка задачи**

Необходимо написать программу, которая выполняет следующее:

1.   Реализует преобразование введенного выражения (если используются переменные, то пользователь должен их инициализировать).

2.   Реализует проверку на корректность простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя).

3.   Реализует вычисления простого выражения и выражения, записанного в прямой и обратной польских нотациях (на выбор пользователя).

Программа должна выводить и описывать все промежуточные действия.

**Выполнение работы.**

Код программы представлен в приложении А.

**Блок описания кода и использованных алгоритмов:**

1. 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.
2. Часто встречается операция PEEK - она получается элемент вершины стека, но не считывает его из стека. В данной работе подобный функционал не требуется.
3. 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.

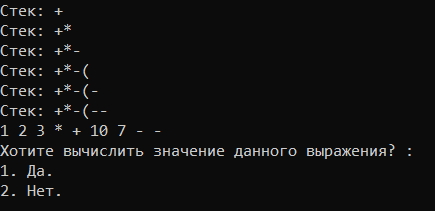
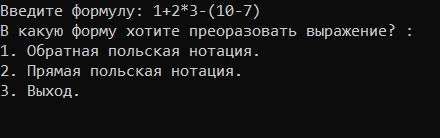
По окончанию строки, все символы из стека операций по одному отправляются в стек выхода. На этом преобразование закончено

При прямой польской нотации обработка строки происходит справа налево.

**Выводы.**

Мы научились работе со стеками и очередями; изучение обратной и прямой польской нотации; проведение сравнительного анализа этих структур данных

**Блок скриншотов работы программы**



Приложение А

рабочий код

#include <list>

#include <string>

#include <map>

#include <stack>

#include <iostream>

using namespace std;

int prior(char x)

{

if ((x == '\*') || (x == '/')) return 2;

if ((x == '+') || (x == '-')) return 1;

if ((x == '(') || (x == ')')) return 0;

}

int prior\_s(string x)

{

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

if ((x[i] == '\*') || (x[i] == '/')) return 2;

if ((x[i] == '+') || (x[i] == '-')) return 1;

if ((x[i] == '(') || (x[i] == ')')) return 0;

}

}

class Stack {

public:

string data;

int index;

Stack\* next;

Stack(string data) {

this->data = data;

this->next = NULL;

}

};

class LinkedList {

public:

Stack\* tail, \* head;

public:

LinkedList() {

head = tail = NULL;

}

Stack\* push(string data) {

Stack\* ptr = new Stack(data);

ptr->next = head;

if (tail == NULL)

tail = ptr;

head = ptr;

return ptr;

}

void pop() {

if (head == NULL) return;

Stack\* ptr = head->next;

if (ptr == NULL)

tail = NULL;

delete head;

head = ptr;

}

Stack\* getAt(int index) {

Stack\* ptr = head;

cout << head;

int n = 0;

while (n != index) {

if (ptr == NULL)

return ptr;

ptr = ptr->next;

n++;

}

return ptr;

}

string top() {

Stack\* ptr = head;

return ptr->data;

}

Stack\* operator [] (int index) {

return getAt(index);

}

bool pustota() {

if (head == NULL) return 0;

}

};

void proverka(LinkedList& stack, string operand, string& polish) {

if (operand != "(" && operand != ")") {

bool flag = false;

while (flag == false) {

if (stack.pustota() == 0 || (prior\_s(operand) > prior\_s(stack.top()))) {

stack.push(operand);

flag = true;

}

else if (prior\_s(stack.top()) >= prior\_s(operand)) {

polish += stack.top();

polish += " ";

stack.pop();

}

}

}

else if (operand == "(") {

stack.push(operand);

polish.pop\_back();

}

else if (operand == ")") {

while (stack.top() != "(") {

polish += stack.top();

polish += " ";

stack.pop();

}

stack.pop();

}

return;

}

void proverka\_p(LinkedList& stack, string operand, string& polish) {

if (operand != "(" && operand != ")") {

bool flag = false;

while (flag == false) {

if (stack.pustota() == 0 || (prior\_s(operand) > prior\_s(stack.top()))) {

stack.push(operand);

flag = true;

}

else if (prior\_s(stack.top()) >= prior\_s(operand)) {

polish += stack.top();

polish += " ";

stack.pop();

}

}

}

else if (operand == "(") {

stack.push(operand);

}

else if (operand == ")") {

while (stack.top() != "(") {

polish += stack.top();

polish += " ";

stack.pop();

}

stack.pop();

polish.pop\_back();

}

return;

}

void symbol\_check(string& symbol, string& polish) {

int xyz;

try {

int num = stoi(symbol);

polish += symbol;

polish += " ";

}

catch (...) {

cout << symbol << " = ";

cin >> xyz;

polish += to\_string(xyz);

polish += " ";

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

string symbol\_p;

string str;

string formula;

string pformula;

LinkedList pchisla;

string psymbol;

cout << "Введите формулу: ";

cin >> formula;

int choose;

string schet;

Menu: {

cout << "В какую форму хотите преоразовать выражение? :\n"

<< "1. Обратная польская нотация.\n"

<< "2. Прямая польская нотация.\n"

<< "3. Выход.\n";

}

cin >> choose;

switch (choose) {

case 1: {

system("cls");

int position = 0;

string symbol;

string polish;

string ctek;

LinkedList stack;

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

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

symbol = formula.substr(position, i - position);

if (!(symbol == "+" || symbol == "-" || symbol == "\*" || symbol == "/" || symbol == "(" || symbol == ")")) {

if ("a" <= symbol && symbol <= "z")

symbol\_check(symbol, polish);

else {

polish += symbol;

polish += " ";

}

}

string operand = formula.substr(i, 1);

proverka(stack, operand, polish);

position = i + 1;

ctek += stack.top();

cout << "Стек: " << ctek << endl;

}

}

symbol = formula.substr(position);

polish += symbol;

while (stack.pustota() != 0) {

polish += stack.top();

polish += " ";

stack.pop();

}

cout << polish << endl;

schet = polish;

Menu1: {

cout << "Хотите вычислить значение данного выражения? :\n"

<< "1. Да.\n"

<< "2. Нет.\n";

}

int choose1;

cin >> choose1;

switch (choose1) {

case 1: {

LinkedList chisla;

string symbol;

int count = 0;

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

if (schet[i] == ' ') {

symbol = schet.substr(count, i - count);

if (!(symbol == "+" || symbol == "-" || symbol == "\*" || symbol == "/")) {

chisla.push(symbol);

}

else {

int num1 = stoi(chisla.top());

chisla.pop();

int num2 = stoi(chisla.top());

chisla.pop();

if (symbol == "\*") {

int num = num1 \* num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "/") {

auto num = num1 / num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "+") {

int num = num1 + num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "-") {

int num = num2 - num1;

symbol = std::to\_string(num);

chisla.push(symbol);

}

}

count = i + 1;

}

}

int rez = stoi(chisla.top());

cout << "Результат = " << rez << endl;

system("pause");

goto Menu;

}

case 2: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

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

system("pause");

goto Menu1;

}

}

goto Menu;

}

case 2: {

system("cls");

int position\_p = 0;

LinkedList stack\_p;

pformula = formula;

reverse(pformula.begin(), pformula.end());

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

if (pformula[i] == ')') {

pformula[i] = '(';

}

else if (pformula[i] == '(') {

pformula[i] = ')';

}

}

string ctekp;

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

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

symbol\_p = pformula.substr(position\_p, i - position\_p);

//cout << symbol << endl;

if (!(symbol\_p == "+" || symbol\_p == "-" || symbol\_p == "\*" || symbol\_p == "/" || symbol\_p == "(" || symbol\_p == ")")) {

if ("a" <= symbol\_p && symbol\_p <= "z")

symbol\_check(symbol\_p, str);

else {

str += symbol\_p;

str += " ";

}

}

string operand\_p = pformula.substr(i, 1);

proverka\_p(stack\_p, operand\_p, str);

position\_p = i + 1;

if (stack\_p.pustota() != 0) {

ctekp += stack\_p.top();

cout << "Стек: " << ctekp << endl;

}

}

}

symbol\_p = pformula.substr(position\_p);

str += symbol\_p;

str += " ";

while (stack\_p.pustota() != 0) {

str += stack\_p.top();

stack\_p.pop();

}

reverse(str.begin(), str.end());

schet = str;

cout << str << endl;

Menu2: {

cout << "Хотите вычислить значение данного выражения? :\n"

<< "1. Да.\n"

<< "2. Нет.\n";

}

int choose1;

cin >> choose1;

switch (choose1) {

case 1: {

system("cls");

reverse(schet.begin(), schet.end());

LinkedList chisla;

string symbol;

cout << schet << '\n';

schet += ' ';

int count = 0;

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

if (schet[i] == ' ') {

symbol = schet.substr(count, i - count);

if (!(symbol == "+" || symbol == "-" || symbol == "\*" || symbol == "/")) {

reverse(symbol.begin(), symbol.end());

chisla.push(symbol);

}

else {

int num1 = stoi(chisla.top());

chisla.pop();

int num2 = stoi(chisla.top());

chisla.pop();

if (symbol == "+") {

int num = num1 + num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "-") {

int num = num1 - num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "/") {

auto num = num1 / num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

else if (symbol == "\*") {

int num = num1 \* num2;

symbol = std::to\_string(num);

chisla.push(symbol);

}

}

count = i + 1;

}

}

string result;

result = chisla.top();

reverse(result.begin(), result.end());

int rez = stoi(result);

cout << "Результат = " << rez << endl;

system("pause");

goto Menu;

}

case 2: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

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

system("pause");

goto Menu2;

}

}

}

case 3: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

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

system("pause");

goto Menu;

}

}

}}