Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Реализация программы для преобразования выражения в инфиксной записи с целыми числами в ПОЛИЗ и его вычисление»

Выполнила:

Студент(ка) 1 курса 7 группы

Подшиваленко Диана Игоревна

Проверил:

Белодед Николай Иванович

2024, Минск

В данном реферате представлена программа, которая на вход получает корректное выражение в целых числах в инфиксной форме, проверяет правильность скобочной последовательности, преобразует его в выражение, представленное в обратной польской записи(ПОЛИЗ). Далее производится вычисление.

Для преобразования и вычисления используется стек.

#include <iostream>

#include <string>

#include <Windows.h>

#include <stack>

using namespace std;

#define division\_by\_zero "На ноль делить нельзя\n"

struct Mystack {

char num;

Mystack\* next = nullptr;

};

struct Mystack2 {

int num;

Mystack2\* next = nullptr;

};

void push(Mystack2\*&, int);

void pop(Mystack2\*&);

int top(Mystack2\*&);

bool empty(Mystack2\*&);

int size(Mystack2\*);

void push(Mystack2\*& st, int n) {

Mystack2\* newst = new Mystack2;

newst->num = n;

newst->next = st;

st = newst;

}

void pop(Mystack2\*& st) {

if (st == NULL) {

exit(-1);

}

else {

Mystack2\* newst = st;

st = st->next;

delete newst;

}

}

int top(Mystack2\*& st) {

if (st == NULL) {

return 0;

}

else {

return st->num;

}

}

bool empty(Mystack2\*& st) {

return (st == NULL);

}

int size(Mystack2\* st) {

long long int s = 0;

while (st) {

s++;

st = st->next;

}

return s;

}

void push(Mystack\*&, char);

void pop(Mystack\*&);

char top(Mystack\*&);

bool empty(Mystack\*&);

int size(Mystack\*);

void push(Mystack\*& st, char n) {

Mystack\* newst = new Mystack;

newst->num = n;

newst->next = st;

st = newst;

}

void pop(Mystack\*& st) {

if (st == NULL) {

exit(-1);

}

else {

Mystack\* newst = st;

st = st->next;

delete newst;

}

}

char top(Mystack\*& st) {

if (st == NULL) {

return 0;

}

else {

return st->num;

}

}

bool empty(Mystack\*& st) {

return (st == NULL);

}

int size(Mystack\* st) {

long long int s = 0;

while (st) {

s++;

st = st->next;

}

return s;

}

bool checkbrackets(string expression) {

Mystack \*operators = nullptr;

for (char c : expression) {

if (c == '(') {

push(operators, c);

}

else if (c == ')') {

if (empty(operators)) {

return 0;

}

pop(operators);

}

}

return 1;

}

bool isoperator(char sym) {

return sym == '+' || sym == '-' || sym == '/' || sym == '\*';

}

int operator\_priority(char sym) {

if (sym == '/' || sym == '\*') {

return 2;

}

else if (sym == '+' || sym == '-') {

return 1;

}

return 0;

}

string getnum(string expr, int& i) {

string buf = "";

for (; i < expr.length(); i++) {

if (expr[i] >= '0' && expr[i] <= '9') {

buf += expr[i];

}

else {

i--;

break;

}

}

return buf;

}

string getRPN(string expr) {

string rpn = "";

Mystack\* operators = nullptr;

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

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

continue;

}

else if (expr[i] >= '0' && expr[i] <= '9') {

rpn += getnum(expr, i);

rpn += ' ';

}

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

push(operators, expr[i]);

}

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

while (!empty(operators) && top(operators) != '(') {

rpn += top(operators);

rpn += ' ';

pop(operators);

}

pop(operators);

}

else if (isoperator(expr[i])) {

while (!empty(operators) && isoperator(top(operators)) && operator\_priority(top(operators)) >= operator\_priority(expr[i])) {

rpn += top(operators);

rpn += ' ';

pop(operators);

}

push(operators, expr[i]);

}

}

while (!empty(operators)) {

rpn += top(operators);

rpn += ' ';

pop(operators);

}

return rpn;

}

int devide(int a, int b) {

if (b == 0) {

throw division\_by\_zero;

}

return a / b;

}

int calculateRPN(string rpn) {

Mystack2 \*numbers = nullptr;

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

if (isoperator(rpn[i])) {

int b = top(numbers);

pop(numbers);

int a = top(numbers);

pop(numbers);

switch (rpn[i]) {

case '+': {

push(numbers, a + b);

break;

}

case '-': {

push(numbers, a - b);

break;

}

case '\*': {

push(numbers, a \* b);

break;

}

case '/': {

int c;

try {

c = devide(a, b);

}

catch (const char\* e) {

cout << division\_by\_zero;

exit(0);

}

push(numbers, c);

break;

}

}

}

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

continue;

}

else {

push(numbers, stoi(getnum(rpn, i)));

}

}

return top(numbers);

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

string expression;

cout << "Введите выражение:\n";

getline(cin, expression);

if (!checkbrackets(expression)) {

cout << "Неверная скобочная последовательность\n";

return 0;

}

cout << "Верная скобочная последовательность\n";

string rpn = getRPN(expression);

cout << rpn << '\n';

cout << "Результат выполнения:\n" << calculateRPN(rpn) << '\n';

}

Результат выполнения:

