Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторних робіт №6

**“ Рекурсивні алгоритми обробки структур даних.”**

з дисципліни "Алгоритми та структури даних ”

В-15

Виконала студентка групи СА-32

Катеринчук Тетяна Андріївна

Прийняв доцент

Щербак С.С.

*2017*

**Мета роботи:** набуття практичних навичок роботи з рекурсивними функціями.

**Завдання:**

На основі додатку з першої лабораторної роботи розробити програму для обчислення математичних виразів згідно алгоритмів з використанням рекурсивної функції та без використання рекурсивної функції. Оцінити час виконання та складність алгоритму.

**Індивідуальне завдання:**

|  |  |
| --- | --- |
| 15 |  |

**Текст програми**

Main.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Calculator.h"

#include "Interface.h"

#include <iostream>

#include <clocale>

using namespace std;

void main() {

setlocale(LC\_CTYPE, "ukr");

Interface expression = "";

bool right = true;

expression.InputExpression();

Calculate calculate = expression.OutputExpression();

cout << expression.OutputExpression() << " = ";

if (calculate.RightExpression(expression.OutputExpression())) {

calculate.TransferPolishForm();

calculate.SteckMoveExit();

cout << calculate.CalculationsPolishForm() << endl;

}

system("pause");

}

Interface.h

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Interface {

protected:

string initialExpression;

public:

Interface() {}

Interface(string f1);

string InputExpression();

string OutputExpression();

};

Interface::Interface(string f1) {

initialExpression = f1;

}

//введення виразу

string Interface::InputExpression() {

cout << "Введiть вираз: ";

cin >> initialExpression;

return initialExpression;

}

//виведення виразу

string Interface::OutputExpression() {

return initialExpression;

}

Calculator.h

#pragma once

#include "Interface.h"

#include <stack>

#include <queue>

#include <math.h>

#include <string>

//S(y = 1 - 4)(1 + y ^ 2)

//M(y = 1 - 5)(1 + y \* 6)

class Calculate :public Interface {

protected:

string initialExpression;

queue <string> exit;

queue <string> expression;

stack <string> steck;

stack <double> number;

string auxiliary;

int numberMulDiv[100];

int numberPow[100];

int numberPlMin[100];

int numberBrackets = 0;

public:

Calculate() {}

Calculate(string f1);

bool RightExpression(string initialExpression);

void TransferPolishForm();

void VerifyNumber(string element);

void VerifyPlusMinus(string element);

void VerifyMultiplicationDivision(string element);

void VerifyPower(string element);

void VerifyBrackets(string element);

void VerifyTrigonometrix(string element);

void SteckMoveExit();

double CalculationsPolishForm();

double Recursion(string newExpression, int start, int end, char symbol, char rec);

int RightTrigonometrix(int i, string initialExpression);

void RightRecursion(int i);

};

Calculate::Calculate(string f1) {

initialExpression = f1;

}

//розділення виразу на числа і знаки, запис кожного у чергу

bool Calculate::RightExpression(string initialExpression) {

for (int i = 0; i < 100; i++) {

numberPlMin[i]=0;

numberMulDiv[i] = 0;

numberPow[i] = 0;

}

for (int i = 0; i < initialExpression.size(); i++) {

if (initialExpression[i] >= 48 && initialExpression[i] <= 57) {

if (initialExpression[i + 1] >= 48 && initialExpression[i + 1] <= 57) {

auxiliary.append(initialExpression, i, 1);

}

else {

expression.push(auxiliary.append(initialExpression, i, 1));

auxiliary.clear();

}

}

else if (initialExpression[i] == '+' || initialExpression[i] == '-' || initialExpression[i] == '\*' || initialExpression[i] == '/' || initialExpression[i] == '^' || initialExpression[i] == '(' || initialExpression[i] == ')' || initialExpression[i] == '=') {

expression.push(auxiliary.append(initialExpression, i, 1));

auxiliary.clear();

}

else if (initialExpression[i] == 'S' || initialExpression[i] == 'M') {

RightRecursion(i);

return false;

}

else {

int i1 = RightTrigonometrix(i, initialExpression);

if (i < i1) {

i = i1;

}

else {

cout << "Помилка! Некоректне значення.\n ";

while (!expression.empty()) {

expression.pop();

}

return false;

}

}

}

return true;

}

//Перевірка чи є в виразі тригонометричні функції

int Calculate::RightTrigonometrix(int i, string initialExpression) {

if (initialExpression[i] == 'c'&&initialExpression[i + 1] == 'o'&&initialExpression[i + 2] == 's') {

expression.push(auxiliary.append(initialExpression, i, 3));

auxiliary.clear();

i = i + 2;

}

else if (initialExpression[i] == 's'&&initialExpression[i + 1] == 'i'&&initialExpression[i + 2] == 'n') {

expression.push(auxiliary.append(initialExpression, i, 3));

auxiliary.clear();

i = i + 2;

}

else if (initialExpression[i] == 't'&&initialExpression[i + 1] == 'g') {

expression.push(auxiliary.append(initialExpression, i, 2));

auxiliary.clear();

i = i + 1;

}

else if (initialExpression[i] == 'c'&&initialExpression[i + 1] == 't'&&initialExpression[i + 2] == 'g') {

expression.push(auxiliary.append(initialExpression, i, 3));

auxiliary.clear();

i = i + 2;

}

else if (initialExpression[i] == 'a'&&initialExpression[i + 1] == 's'&&initialExpression[i + 2] == 'i'&&initialExpression[i + 3] == 'n') {

expression.push(auxiliary.append(initialExpression, i, 4));

auxiliary.clear();

i = i + 3;

}

else if (initialExpression[i] == 'a'&&initialExpression[i + 1] == 'c'&&initialExpression[i + 2] == 'o'&&initialExpression[i + 3] == 's') {

expression.push(auxiliary.append(initialExpression, i, 4));

auxiliary.clear();

i = i + 3;

}

else if (initialExpression[i] == 'a'&&initialExpression[i + 1] == 't'&&initialExpression[i + 2] == 'g') {

expression.push(auxiliary.append(initialExpression, i, 3));

auxiliary.clear();

i = i + 2;

}

else if (initialExpression[i] == 'a'&&initialExpression[i + 1] == 'c'&&initialExpression[i + 2] == 't'&&initialExpression[i + 3] == 'g') {

expression.push(auxiliary.append(initialExpression, i, 4));

auxiliary.clear();

i = i + 3;

}

else if (initialExpression[i] == 'l'&&initialExpression[i + 1] == 'n') {

expression.push(auxiliary.append(initialExpression, i, 2));

auxiliary.clear();

i = i + 1;

}

else if (initialExpression[i] == 'l'&&initialExpression[i + 1] == 'g') {

expression.push(auxiliary.append(initialExpression, i, 2));

auxiliary.clear();

i = i + 1;

}

return i;

}

//Підготовка до рекурсії, зчитання з якого значення по яке рахувати, перезапис вираз в нову змінну

void Calculate::RightRecursion(int i) {

char rec = initialExpression[i];

char symbol;

int start;

int end;

string newExpression;

i = i + 2;

bool audit = true;

symbol = initialExpression[i];

while (initialExpression[i] != ')')

{

if (initialExpression[i] >= 48 & initialExpression[i] <= 57) {

if (initialExpression[i + 1] >= 48 && initialExpression[i + 1] <= 57) {

auxiliary.append(initialExpression, i, 1);

}

else {

auxiliary.append(initialExpression, i, 1);

if (audit) {

start = atoi(auxiliary.c\_str());

audit = false;

}

else {

end = atoi(auxiliary.c\_str());

}

auxiliary.clear();

}

}

i++;

}

i = i + 3;

int nE = 0;

for (; initialExpression.size() > i + nE; nE++);

newExpression.append(initialExpression, i - 1, nE);

i = i + nE + 1;

cout << (Recursion(newExpression, start, end, symbol, rec)) << endl;

}

//Обрахунок рекурсії

double Calculate::Recursion(string newExpression, int start, int end, char symbol, char rec) {

double resultRecursion;

if (rec == 'S') {

resultRecursion = 0;

}

if (rec == 'M') {

resultRecursion = 1;

}

for (; start <= end; start++) {

string stepExpression = newExpression;

for (int i = 0; i < stepExpression.size(); i++) {

if (stepExpression[i] == symbol) {

string right, left;

left.append(stepExpression, 0, i);

right.append(stepExpression, i + 1, stepExpression.size());

stepExpression.clear();

stepExpression = left + to\_string(start) + right;

}

}

RightExpression(stepExpression);

TransferPolishForm();

SteckMoveExit();

if (rec == 'S') {

resultRecursion = resultRecursion + CalculationsPolishForm();

}

if (rec == 'M') {

resultRecursion = resultRecursion \* CalculationsPolishForm();

}

}

return resultRecursion;

}

//перевід виразу в польський зворотній запис

void Calculate::TransferPolishForm() {

while (!expression.empty()) {

VerifyNumber(expression.front());

VerifyPlusMinus(expression.front());

VerifyMultiplicationDivision(expression.front());

VerifyPower(expression.front());

VerifyTrigonometrix(expression.front());

VerifyBrackets(expression.front());

if (!expression.empty())

expression.pop();

}

}

//перевірка чи елемент черги число

void Calculate::VerifyNumber(string element) {

if (element[0] >= 48 && element[0] <= 57) {

exit.push(element);

if (numberPow[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

numberPow[numberBrackets] = 0;

}

}

}

//перевірка чи елемент черги знак плюс чи мінус

void Calculate::VerifyPlusMinus(string element) {

if (element == "+" || element == "-") {

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

if (numberPlMin[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberPlMin[numberBrackets] = 0;

}

else

numberPlMin[numberBrackets]++;

steck.push(element);

}

}

//перевірка чи елемент черги знак множення або віднімання

void Calculate::VerifyMultiplicationDivision(string element) {

if (element == "\*" || element == "/") {

if (numberMulDiv[numberBrackets] > 0) {

exit.push(steck.top());

steck.pop();

numberMulDiv[numberBrackets] = 0;

}

steck.push(element);

numberMulDiv[numberBrackets]++;

}

}

//перевірка чи елемент черги дужки

void Calculate::VerifyBrackets(string element) {

if (element == "(") {

numberBrackets++;

steck.push(element);

expression.pop();

while (!expression.empty()) {

VerifyNumber(expression.front());

VerifyPlusMinus(expression.front());

VerifyMultiplicationDivision(expression.front());

VerifyPower(expression.front());

VerifyBrackets(expression.front());

if (expression.front() == ")") {

numberBrackets--;

break;

}

expression.pop();

}

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

exit.push(steck.top());

steck.pop();

}

steck.pop();

}

}

//перевірка чи елемент черги тригонометричне значення

void Calculate::VerifyTrigonometrix(string element) {

if (element == "cos" || element == "sin" || element == "tg" || element == "ctg" || element == "acos" || element == "asin" || element == "atg" || element == "actg" || element == "ln" || element == "lg") {

numberBrackets++;

steck.push(element);

expression.pop();

steck.push(expression.front());

expression.pop();

while (!expression.empty()) {

VerifyNumber(expression.front());

VerifyPlusMinus(expression.front());

VerifyMultiplicationDivision(expression.front());

VerifyPower(expression.front());

VerifyBrackets(expression.front());

VerifyTrigonometrix(expression.front());

if (expression.front() == ")") {

numberBrackets--;

break;

}

expression.pop();

}

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

exit.push(steck.top());

steck.pop();

}

steck.pop();

exit.push(steck.top());

steck.pop();

}

}

//перевірка чи елемент черги знак степінь

void Calculate::VerifyPower(string element) {

if (element == "^") {

steck.push(element);

numberPow[numberBrackets]++;

}

}

//переміщення всіх елементів з стеку в чергу

void Calculate::SteckMoveExit() {

while (!steck.empty()) {

exit.push(steck.top());

steck.pop();

}

}

//обчислення польського виразу

double Calculate::CalculationsPolishForm() {

double calculations = 0;

while (!exit.empty()) {

auxiliary = exit.front();

if (auxiliary[0] >= 48 && auxiliary[0] <= 57) {

number.push(atoi(auxiliary.c\_str()));

}

else if (auxiliary == "cos" || auxiliary == "sin" || auxiliary == "tg" || auxiliary == "ctg" || auxiliary == "acos" || auxiliary == "asin" || auxiliary == "atg" || auxiliary == "actg" || auxiliary == "ln" || auxiliary == "lg") {

double element = number.top();

number.pop();

if (auxiliary == "cos") {

number.push(cos(element));

}

if (auxiliary == "sin") {

number.push(sin(element));

}

if (auxiliary == "tg") {

number.push(tan(element));

}

if (auxiliary == "ctg") {

number.push(1 / tan(element));

}

if (auxiliary == "acos") {

number.push(acos(element));

}

if (auxiliary == "asin") {

number.push(asin(element));

}

if (auxiliary == "atg") {

number.push(atan(element));

}

if (auxiliary == "actg") {

number.push(1 / atan(element));

}

if (auxiliary == "ln") {

number.push(log(element));

}

if (auxiliary == "lg") {

number.push(log10(element));

}

}

else {

double right = number.top();

number.pop();

double left = number.top();

number.pop();

if (auxiliary == "+") {

number.push(left + right);

}

if (auxiliary == "-") {

number.push(left - right);

}

if (auxiliary == "\*") {

number.push(left \* right);

}

if (auxiliary == "/") {

number.push(left / right);

}

if (auxiliary == "^") {

number.push(pow(left, right));

}

}

calculations = number.top();

exit.pop();

}

auxiliary.clear();

return calculations;

}

**Результат програми**

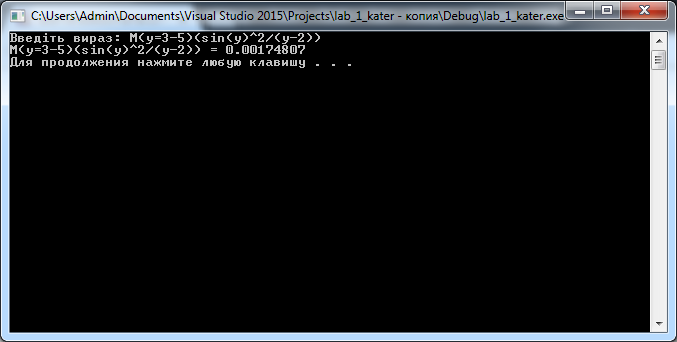


Рис.1 Реалізація заданої рекурсії

**Висновок:** При виконання даної лабораторної роботи я набула навичок з рекурсивними функціями. Вдосконалила свою програму калькулятор з першої лабораторної: додала тригонометричні функції та універсальне визначення суми чи множення ряду за допомогою рекурсивних функцій.