Міністерство освіти і науки України

Національний університет “Львівська політехніка”

Кафедра інформаційних систем та мереж

ЗВІТ

про виконання лабораторної роботи № 6

з дисципліни Алгоритми та структури даних

Варіант 18

Виконав студент групи СА-32

Маленчак Валентин Богданович

Прийняв доцент Щербак С.С.

*Львів 2017*

**Тема:** Рекурсивні алгоритми обробки структур даних.

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

**Завдання на роботу:** 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:**



# Хід роботи

Текст програми на мові С++:

#pragma once

#include <iostream>

#include <string>

#include <stack>

#include "UInterface.h"

using namespace std;

class Calculator

{

private:

static int priority(char operation)

{

switch (operation)

{

case '+':

case '-': return 1;

case '\*':

case '/': return 2;

case '^': return 3;

default: system("pause");

}

}

static void calculateMathFuncValue(string& str, int i, int x, UInterface myInterface)

{

int length = i + 4;

double argument = 0.0;

string arg = "\0";

while (str.at(length) != ')')

{

arg += str.at(length);

length++;

}

arg += '=';

argument = polandForm(arg, x);

switch (str.at(i))

{

case 's':

argument = sin(argument);

break;

case 'c':

argument = cos(argument);

break;

case 't':

argument = tan(argument);

break;

default:

break;

}

str.erase(i, length - i + 1);

try

{

arg = to\_string(argument);

}

catch (invalid\_argument)

{

myInterface.getID(12);

return;

}

catch (out\_of\_range)

{

myInterface.getID(13);

return;

}

str.insert(i, "(");

str.insert(i + 1, arg);

str.insert(i + 1 + arg.size(), ")");

}

static bool isOperator(char ch)

{

switch (ch)

{

case '+':

case '-':

case '\*':

case '/':

case '^':

return true;

break;

default:

return false;

break;

}

}

static void calculation(stack <double>& stk1, stack <char>& stk2)

{

double num2 = stk1.top();

stk1.pop();

double num1 = stk1.top();

stk1.pop();

switch (stk2.top())

{

case '+':

stk1.push(num1 + num2);

break;

case '-':

stk1.push(num1 - num2);

break;

case '\*':

stk1.push(num1 \* num2);

break;

case '/':

stk1.push(num1 / num2);

break;

case '^':

stk1.push(pow(num1, num2));

break;

}

stk2.pop();

}

static bool checkStr(string str, int sizestr, UInterface myInterface)

{

int openBrackets = 0, closedBrackets = 0;

int operands = -1, operators = 0;

bool p = true;

if (str.at(sizestr - 1) != '=')

{

myInterface.getID(3);

return false;

}

for (int i = 0; i < sizestr; i++)

{

switch (str.at(i))

{

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

case '0':

if (p)

if (isdigit(str.at(i)))

{

operands++;

p = false;

}

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

operators++;

p = true;

if (i == 0 || i == sizestr - 2)

{

myInterface.getID(9);

return false;

}

break;

case '(':

if (i > 0)

{

if (isdigit(str.at(i - 1)) && str.at(i - 1) != 'n' && str.at(i - 1) != 's' || isOperator(str.at(i + 1)))

{

myInterface.getID(14);

return false;

}

}

openBrackets++;

break;

case ')':

if (i == 1)

{

myInterface.getID(16);

return false;

}

closedBrackets++;

if (str.at(i - 1) == '(')

{

myInterface.getID(11);

return false;

}

if (isOperator(str.at(i - 1)))

{

myInterface.getID(15);

return false;

}

break;

case '=':

if (i != sizestr - 1)

{

myInterface.getID(10);

return false;

}

break;

case 'x':

if (p)

if (!isdigit(str.at(i + 1)))

{

operands++;

p = false;

}

else;

else

{

myInterface.getID(19);

return false;

}

break;

case 's':

if (str.at(i + 1) == 'i' && str.at(i + 2) == 'n' && str.at(i + 3) == '(')

i += 2;

else

{

myInterface.getID(19);

return false;

}

break;

case 'c':

if (str.at(i + 1) == 'o' && str.at(i + 2) == 's' && str.at(i + 3) == '(')

i += 2;

else

{

myInterface.getID(19);

return false;

}

break;

case 't':

if (str.at(i + 1) == 'a' && str.at(i + 2) == 'n' && str.at(i + 3) == '(')

i += 2;

else

{

myInterface.getID(19);

return false;

}

break;

default:

myInterface.getID(5);

return false;

break;

}

}

if (operands != operators)

{

myInterface.getID(4);

return false;

}

if (openBrackets != closedBrackets)

{

myInterface.getID(6);

return false;

}

return true;

}

static bool pushNumberToStack(stack <double>& stk1, string& tmp, UInterface myInterface)

{

try

{

stk1.push(stod(tmp));

tmp = "\0";

}

catch (invalid\_argument)

{

myInterface.getID(12);

return false;

}

catch (out\_of\_range)

{

myInterface.getID(13);

return false;

}

return true;

}

static double polandForm(string str, int x)

{

stack <double> stk1;

stack <char> stk2;

string tmp = "\0";

string xValue = to\_string(x);

int dynamicBrackets = 0;

unsigned int sizeStr = str.size(), sizeStk2;

UInterface myInterface;

if (!checkStr(str, sizeStr, myInterface))

{

system("pause");

exit(-1);

}

for (unsigned int i = 0; i < sizeStr; i++)

{

switch (str.at(i))

{

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

case '0':

case ',':

tmp += str.at(i);

break;

break;

case '+':

case '-':

case '\*':

case '/':

case '^':

sizeStk2 = stk2.size();

if (!tmp.empty())

if (!pushNumberToStack(stk1, tmp, myInterface))

{

system("pause");

exit(-1);

}

if ((str.at(i) == '-' && i == 0))

{

tmp += str.at(i);

break;

}

if(str.at(i) == '-' && str.at(i - 1) == '(')

{

tmp += str.at(i);

break;

}

if (stk2.size() - dynamicBrackets == 0)

stk2.push(str.at(i));

else

{

for (unsigned int j = sizeStk2; j > 0; j--)

{

if (stk2.top() == '(') break;

if (priority(stk2.top()) >= priority(str.at(i)))

calculation(stk1, stk2);

}

stk2.push(str.at(i));

}

break;

case '(':

dynamicBrackets++;

stk2.push(str.at(i));

break;

case ')':

if (!tmp.empty())

if (!pushNumberToStack(stk1, tmp, myInterface))

{

system("pause");

exit(-1);

}

while (stk2.top() != '(')

calculation(stk1, stk2);

stk2.pop();

dynamicBrackets--;

break;

case '=':

if (!tmp.empty())

if (!pushNumberToStack(stk1, tmp, myInterface))

{

system("pause");

exit(-1);

}

break;

case 'x':

str.erase(i, i + 1);

str.insert(i, xValue);

sizeStr = str.size();

i = i + xValue.size() - 1;

tmp += xValue;

break;

case 's':

case 'c':

case 't':

calculateMathFuncValue(str, i, x, myInterface);

sizeStr = str.size();

i--;

break;

default:

break;

}

}

while (stk1.size() != 1 && !stk2.empty())

calculation(stk1, stk2);

return stk1.top();

}

public:

double recursionSum(string str, double& result, int x, UInterface myInterface)

{

if (x == 0)

return 0.0;

result += polandForm(str, x);

x--;

return result + recursionSum(str, result, x, myInterface);

}

double recursionProd(string str, double& result, int x, UInterface myInterface)

{

if (x == 0)

return 1.0;

result \*= polandForm(str, x);

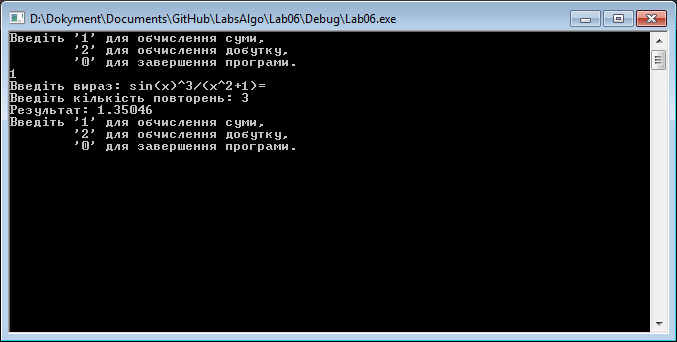
x--;

return recursionProd(str, result, x, myInterface);

}

};

Результати комп'ютерної реалізації:



**Висновок:** я набув практичних навичок роботи з рекурсивними функціями. У лабораторній роботі була удосканалений калькулятор, що працює на основі зворотнього польського запису, шляхом добавлення в нього тригонометричних функцій.