Name: Mohamed Khaled Gamil Ismail

Section: 3

Seat No.: 33813

Data Structure

Assignment 1

Supported Operators:

Binary: +, -, *, /, ^

Unary: ~(unary negate), sin, cos, sqrt

Code:

```
#include <iostream>
#include <string>
#include <sstream>
#include <stack>
#include <windows.h>

using namespace std;

HANDLE  hConsole;

bool is_oper(int size, string str[], string s);
bool contain_oper(int size, string str[], string s);
void print_error();


int main()
{
	system("COLOR F0");
	hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
	string binary_oper[] = { "^","*", "/", "+", "-" };
	string uniary_oper[] = { "sin","cos", "sqrt", "~" };
	stack<string> stack;
	string expr;
	cout << "Enter postfix expression with spaces between operands and operators: " <<
endl;
	getline(cin,expr);
	istringstream is(expr);
	string term;
	while (getline(is, term, ' '))
	{
		if ((is_oper(5, binary_oper, term) || is_oper(4, uniary_oper, term)) &&
stack.empty())
		{
			print_error();
```

```
        return 1;
}
else if (!(is_oper(5, binary_oper, term) || is_oper(4, uniary_oper, term)))
{
        stack.push(term);
}
else if (is_oper(5, binary_oper, term))
{

        if (stack.empty())
        {
                print_error();
                return 1;
        }
        string op2 = stack.top();
        stack.pop();
        if (stack.empty())
        {
                print_error();
                return 1;
        }
        string op1 = stack.top();
        stack.pop();
        if (contain_oper(5, binary_oper, op1))
        {
                op1 = "(" + op1 + ")";
        }
        if (contain_oper(5, binary_oper, op2))
        {
                op2 = "(" + op2 + ")";
        }
        string newOp = op1 + " " + term + " " + op2;
        stack.push(newOp);
}

else if (is_oper(4, uniary_oper, term))
{

        if (stack.empty())
        {
                print_error();
                return 1;
        }
        string op = stack.top();
        stack.pop();
        string newOp;
        if (term == "~")
        {
                if (contain_oper(5, binary_oper, op))
                {
                        op = "(" + op + ")";
                }
                newOp = term + op;
        }
        else
        {
                newOp = term + "(" + op + ")";
        }
```

```
                        stack.push(newOp);
                }
        }
        string infix = stack.top();
        if (!(contain_oper(5, binary_oper, infix) || contain_oper(4, uniary_oper, infix)))
        {
                print_error();
                return 1;
        }
        SetConsoleTextAttribute(hConsole, 250);
        cout << "Infix expression: " << infix << endl;
        SetConsoleTextAttribute(hConsole, 240);
        return 0;
}

bool is_oper(int size, string str[], string s)
{
        for (int i = 0; i < size; i++)
        {
                if (str[i] == s) return true;
        }
        return false;
}

bool contain_oper(int size, string str[], string s)
{
        for (int i = 0; i < size; i++)
        {
                if (s.find(str[i]) != string::npos) return true;
        }
        return false;
}

void print_error()
{
        SetConsoleTextAttribute(hConsole, 252);
        cout << "Invalid postfix expression" << endl;
        SetConsoleTextAttribute(hConsole, 240);
}
```

Screenshots:

```
Enter postfix expression with spaces between operands and operators:
X y * t +
Infix expression: (X * y) + t
Press any key to continue . . .

Enter postfix expression with spaces between operands and operators:
A B * X Y - /
Infix expression: (A * B) / (X - Y)
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
4 3 + 5 * 2 -
Infix expression: ((4 + 3) * 5) - 2
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
x t y * + 2 /
Infix expression: (x + (t * y)) / 2
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
b ~ b b * 4 a * c * - sqrt + 2 a * /
Infix expression: (~b + (sqrt((b * b) - ((4 * a) * c)))) / (2 * a)
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
- A B
Invalid postfix expression
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
A - B
Invalid postfix expression
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
sin(A)
Invalid postfix expression
Press any key to continue . . .
```

```
Enter postfix expression with spaces between operands and operators:
A sin
Infix expression: sin(A)
Press any key to continue . . .
```