CSE4014 Data Structures and Algorithms I

Project#1

Postfix Expression Calculator

```
C:\Users\ksafa\CLionProjects\Data-Proj1\cmake-build-debug\Data_Proj1.exe
Infix Expression: A+B-C
Postfix Expression: AB+C-

Infix Expression: (A+B)*C
Postfix Expression: AB+C*

Infix Expression: (A+B)*(C-D)
Postfix Expression: AB+CD-*

Infix Expression: A+((B+C)*(E-F)-G)/(H-I)
Postfix Expression: ABC+EF-*G-HI-/+

Infix Expression: A+B*(C+D)-E/F*G+H
Postfix Expression: ABCD+*+EF/G*-H+

Infix Expression: 2+4-1
Postfix Expression: 24+1-
Result of Postfix Expression: 5

Infix Expression: (6+3)*2
Postfix Expression: 63+2*
Result of Postfix Expression: 18


Process finished with exit code 0
```

Kerem Safa Dirican

1800002205

int evaluatePostfix(char *exp)

This function takes a char array as a parameter and returns an integer.

`stackType<char> stack(MAX_VALUE);` Creating a stack.

`for (int i = 0; exp[i]; ++i)` Function returns the last element of the stack after this loop.

`if (isalpha(exp[i])) { return -1; }` Cheking is there any letter in the array. If it encounter a letter, returns -1.

`if (isdigit(exp[i])) { stack.push(exp[i] - '0'); }` Finding the numbers and pushing them to the stack.

`else` In this case, function popping 2 value from the stack and do the operation according to the operator and push the result to the stack..

bool infixToPostfix::precedence(char opr1, char opr2)

This function determines the precedence between two operators. If *opr1* is '*' or '/' it returns 1. Also it returns 1 if *opr2* is '+' or '-'. Otherwise it returns 0.

void infixToPostfix::convertToPostfix()

This function converts an infix expression to a postfix expression.

`stackType<char> stack(MAX_VALUE);` Creating a local stack.

Loop 1.) `for (char i : infx)` In this loop, function takes actions according to the conditions.

There are total 5 conditions.

1. `if (i == '(')` **Open Parenthesis**
2. `else if (i == ')')` **Close Parenthesis**
3. `else if (i == '+' || i == '-')` **Addition**
4. `else if (i == '*' || i == '/')` **Multiplication**
5. `else` **Letter or Number**

1.) Open Parenthesis

Always push the character to the stack.

2.) Close Parenthesis

Pop everything to the pfx until see the '('. After that pop '(' from the stack.

3.) Addition

If the stack is empty or the top item of the stack is '(' push the operator to the stack. If not, pop the stack to pfx. Due to the change of the *stack.top* program should check if the stack is empty. If not, the program should check the *stack.top* again before pushing the operator to the stack. If the *stack.top* is '+' or '-', pop the stack to pfx again. Using only the *precedence* function for this process is unsafe. It can cause errors if the *stack.top* will be open parenthesis after popping the stack.

4.) Multiplication

Almost the same as **addition**, if the stack is empty or *stack.top* isn't '*' or '/', push operator to the stack. If it's pop the stack to pfx and check *stack.top* again. If the *stack.top* is '*' or '/', pop the stack to pfx before pushing the operator to the stack.

5.) Letter or Number

Always push to the stack.

Loop 2.) `while (!stack.isEmptyStack())` This loop pop all remaining operators and operands from the stack to pfx.

`int main( )`

Reading values from a text file with *ifstream* and store them in *infx* with *getline* function. Copying, calculation, and printing are handling by member functions. For evaluating the postfix expression, *getPfx()* value copying to *pfx* and `char *exp = &pfx[0]` using for sending parameters of current pfx to *evaluatePostfix* function. If *evaluatePostfix* returns a value other than -1, the program also prints Result of Postfix Expression after printing the Infix Expression and Postfix Expression.