Resolving Arithmetic Expressions in C#

Advanced Programming Course Work

By: Assem Nabil
Dec, 2023

Problem Description

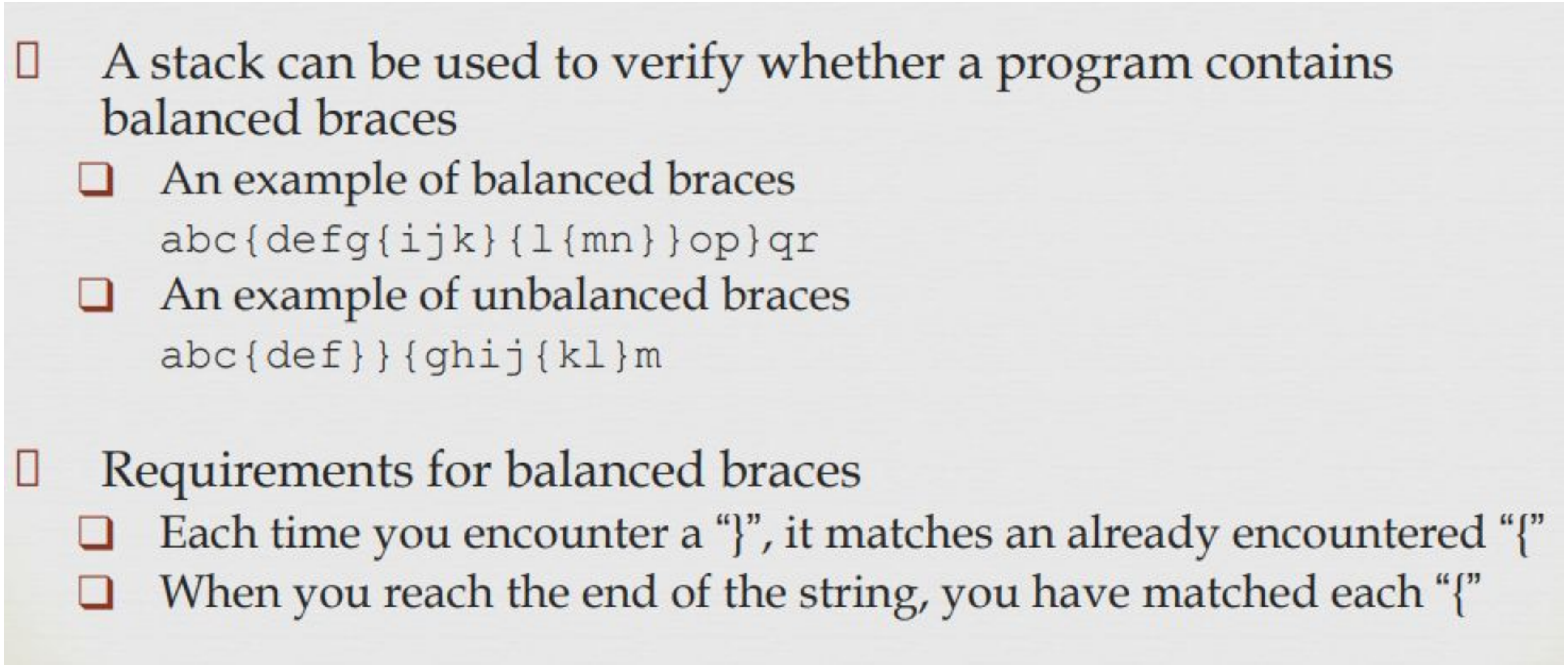
- Arithmetic expressions are used to represent mathematical operations such as addition, subtraction, multiplication, and division.
- Examples of arithmetic expressions:
  - 2 + 3 * 4
  - (5 + 2) / 3
  - x * 2 - y
  - a^2 + b^2 = c^2
- How to resolve arithmetic expression entered by user using C# programming language

Problem Solving

- Prompting the user for an arithmetic expression
- Handling user input and storing it as a string
- Implementing a **stack** data structure to manage arithmetic operations
- **Search** for unreliable character(s) in the expression, check valid expresion
- Check **balance**: Handling Parentheses using Stack
- **Split**: Breaking down the expression into individual parts (numbers and operators), Parse the expression, and find the expression **Length**
- Convert **infix** string array to **postfix** using the Stack
- **Solve** the equation through the postfix expression array and the stack
- Implement and **Interface** to manage data store
- Use Interface to save the data in a file or database

Red color means the functions are required in the project

Algorithms: Balancing Symbols (checking for balanced braces):

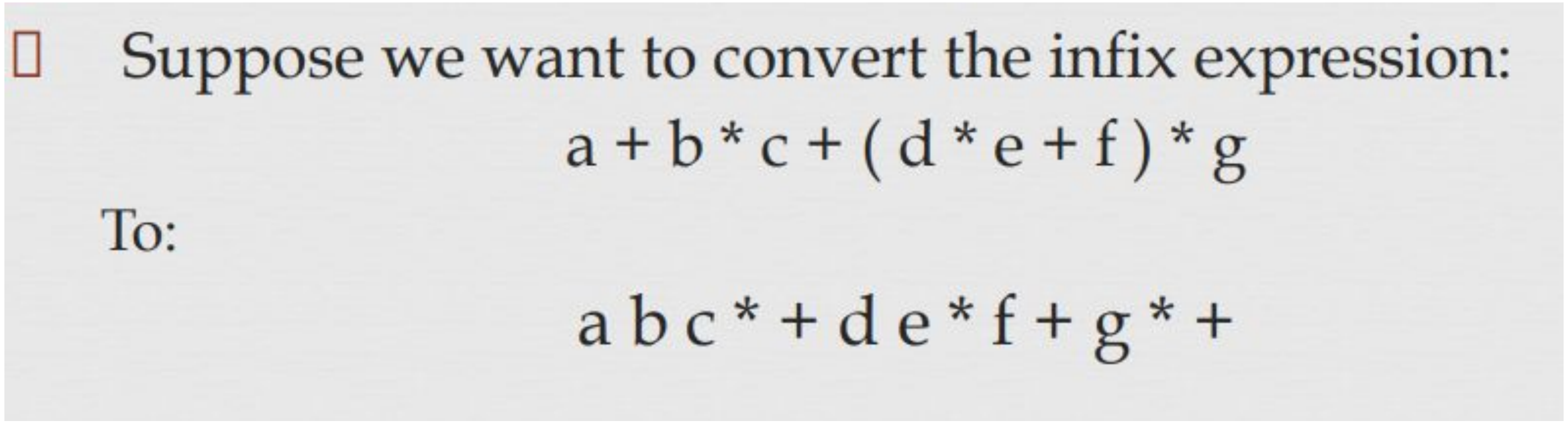


- A stack can be used to verify whether a program contains balanced braces
  - ❑ An example of balanced braces

    ```
    abc{defg{ijk}{l{mn}}op}qr
    ```
  - ❑ An example of unbalanced braces

    ```
    abc{def}}{ghij{kl}m
    ```
- Requirements for balanced braces
  - ❑ Each time you encounter a “}”, it matches an already encountered “{”
  - ❑ When you reach the end of the string, you have matched each “{”

Algorithms: Infix to Postfix Conversion:



- Suppose we want to convert the infix expression:

  a + b * c + (d * e + f) * g

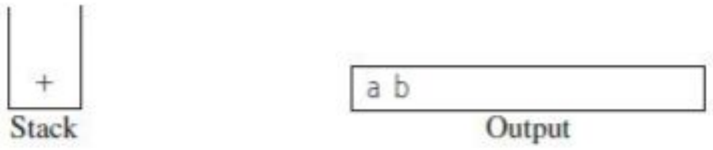
To:

  a b c * + d e * f + g * +

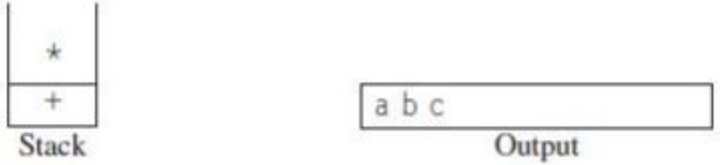
Algorithms: Infix to Postfix Conversion:

First, the symbol **a** is read, so it is passed through to the output.

+ is read and pushed onto the stack. Next **b** is read and passed through to the output. The state of affairs at this juncture is as follows:

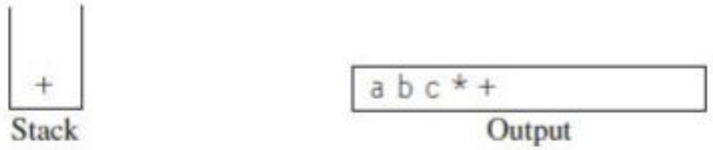


Next, a * is read. The top entry on the operator stack has lower precedence than *, so nothing is output and * is put on the stack. Next, **c** is read and output. Thus far, we have

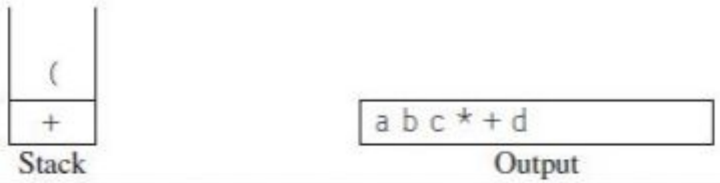


Algorithms: Infix to Postfix Conversion:

The next symbol is a +. Checking the stack, we find that we will pop a * and place it on the output; pop the other +, which is not of *lower* but equal priority, on the stack; and then push the +.

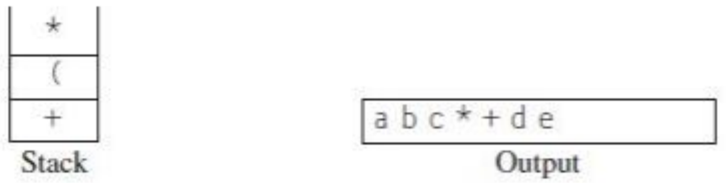


The next symbol read is a (. Being of highest precedence, this is placed on the stack. Then d is read and output.

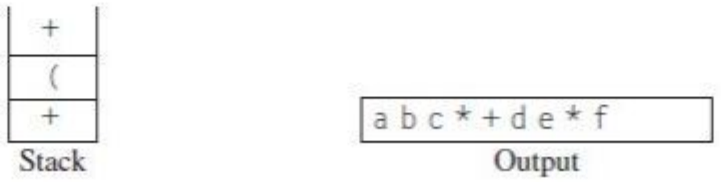


Algorithms: Infix to Postfix Conversion:

We continue by reading a `*`. Since open parentheses do not get removed except when a closed parenthesis is being processed, there is no output. Next, `e` is read and output.

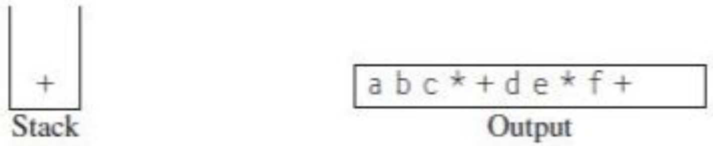


The next symbol read is a `+`. We pop and output `*` and then push `+`. Then we read and output `f`.

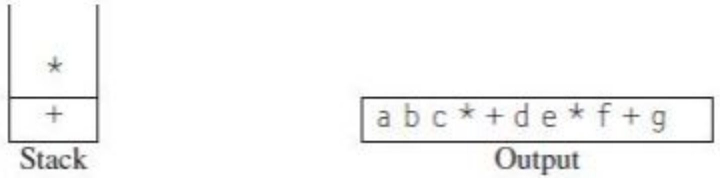


Algorithms: Infix to Postfix Conversion:

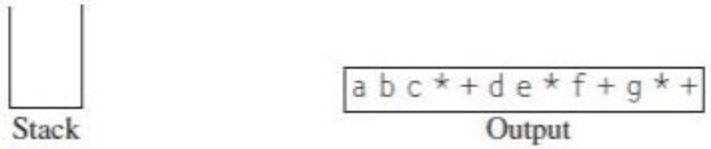
Now we read a), so the stack is emptied back to the (. We output a +.



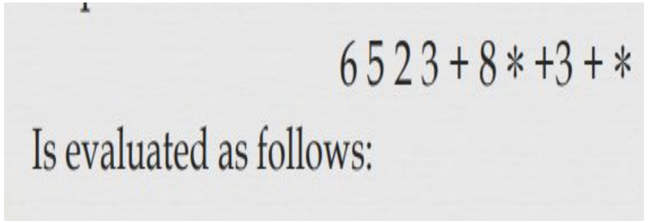
We read a * next; it is pushed onto the stack. Then g is read and output.



The input is now empty, so we pop and output symbols from the stack until it is empty.



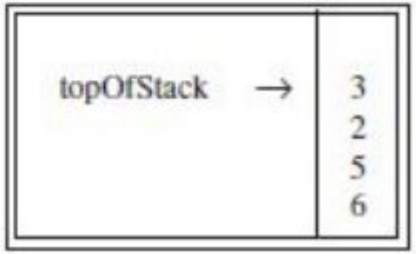
Algorithms: Calculate Postfix Expressions:



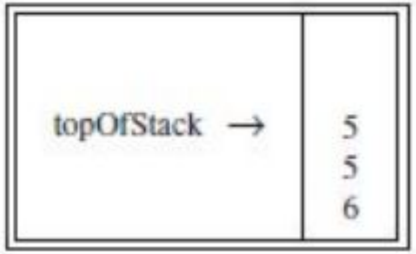
6 5 2 3 + 8 * +3 + *

Is evaluated as follows:

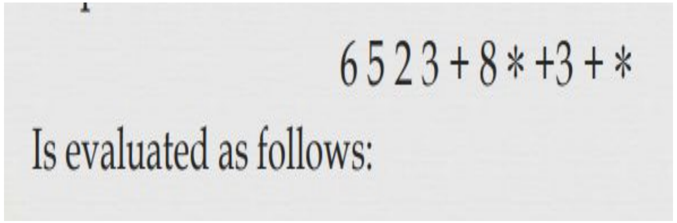
The first four symbols are placed on the stack. The resulting stack is



Next, a '+' is read, so 3 and 2 are popped from the stack, and their sum, 5, is pushed.



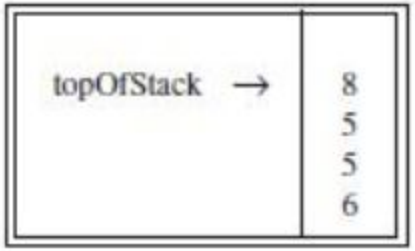
Algorithms: Calculate Postfix Expressions:



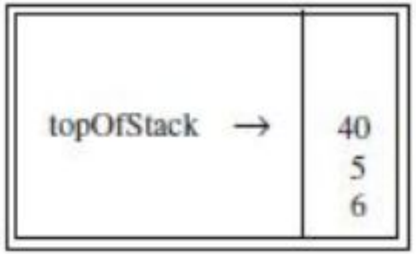
6 5 2 3 + 8 * +3 + *

Is evaluated as follows:

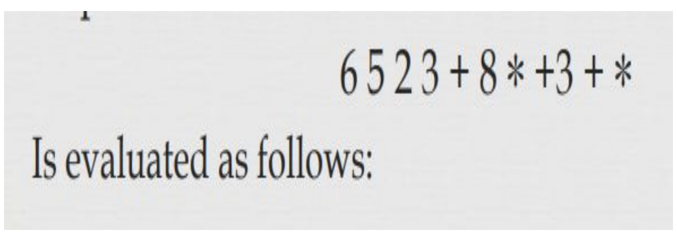
Next, 8 is pushed.



Now a '*' is seen, so 8 and 5 are popped, and $5 * 8 = 40$ is pushed.



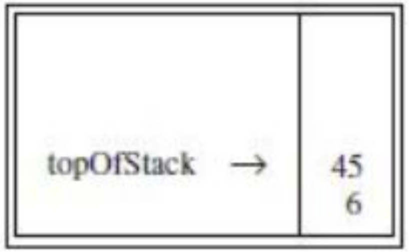
Algorithms: Calculate Postfix Expressions:



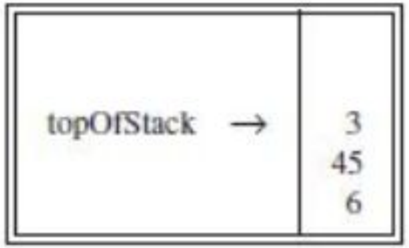
6 5 2 3 + 8 * +3 + *

Is evaluated as follows:

Next, a '+' is seen, so 40 and 5 are popped, and 5 + 40 = 45 is pushed.

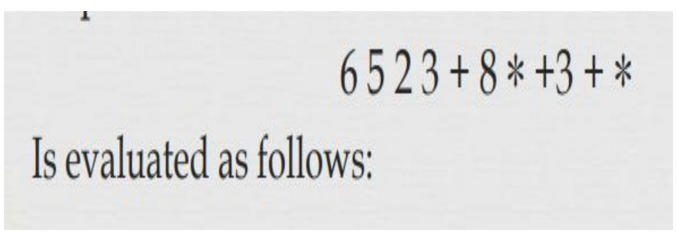


Now, 3 is pushed.



Next, '+' pops 3 and 45 and pushes 45 + 3 = 48.

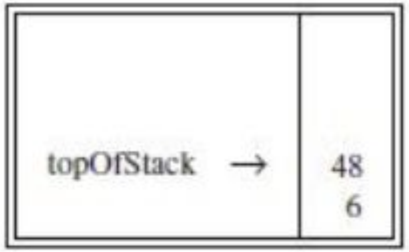
Algorithms: Calculate Postfix Expressions:



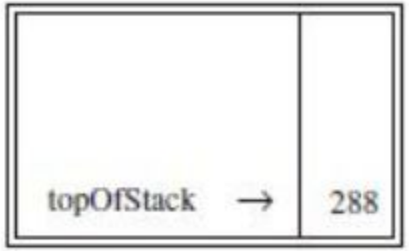
6 5 2 3 + 8 * +3 + *

Is evaluated as follows:

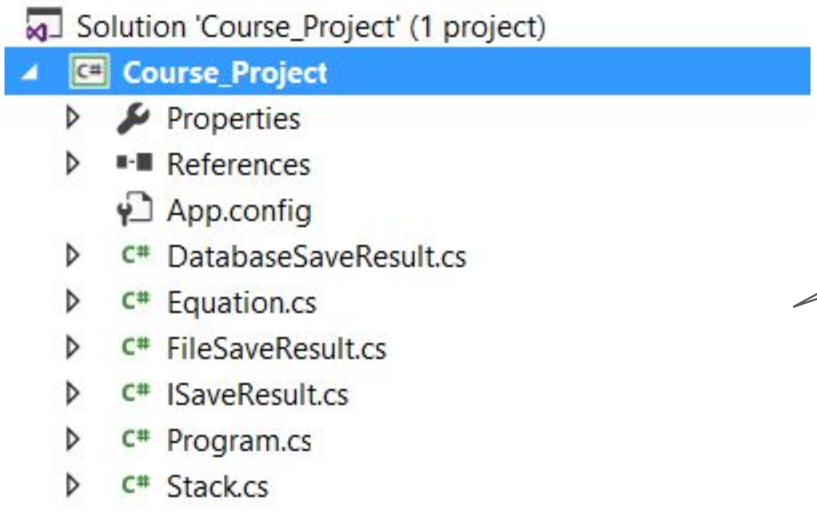
Next, '+' pops 3 and 45 and pushes 45 + 3 = 48.



Finally, a '*' is seen and 48 and 6 are popped; the result, 6 * 48 = 288, is pushed.



Implementation: Solution



Solution 'Course_Project' (1 project)
- Course_Project
  - Properties
  - References
  - App.config
  - DatabaseSaveResult.cs
  - Equation.cs
  - FileSaveResult.cs
  - ISaveResult.cs
  - Program.cs
  - Stack.cs

Five Classes & One Interface

Implementation: Stack Class

- Methods
  - public void Push(string x)
  - public string Pop()
  - public string Top()
  - public bool IsEmpty()
  - public int Count()
  - public bool Search(string x)

Eight Methods

Implementation: Equation Class

- Property
  - public int ExpressionLength
- Methods
  - public bool CheckValidity(string infix) // Check if the mathematical equation is valid
  - public bool CheckBalance(string equ) //Check the equation's brackets balance
  - public string[] Split(string infix) //Split mathematical expression. Parsing
  - public string[] Infix2Postfix(string infix) //convert infix expression to postfix
  - public string CalculatePostfix(string[] equ, int elmNo) //solve the equation
  - public void SaveResult(ISaveResult sr, string infix, string result) //Store the result

One Property &
Six Methods

Implementation: ISaveResult Interface

- Methods
  - void AddEntry(string infixExp, string result)

Only one Methods to be implemented by other classes used the interface

Implementation: FileSaveResult : ISaveResult

- Methods
  - void AddEntry(string infixExp, string result)

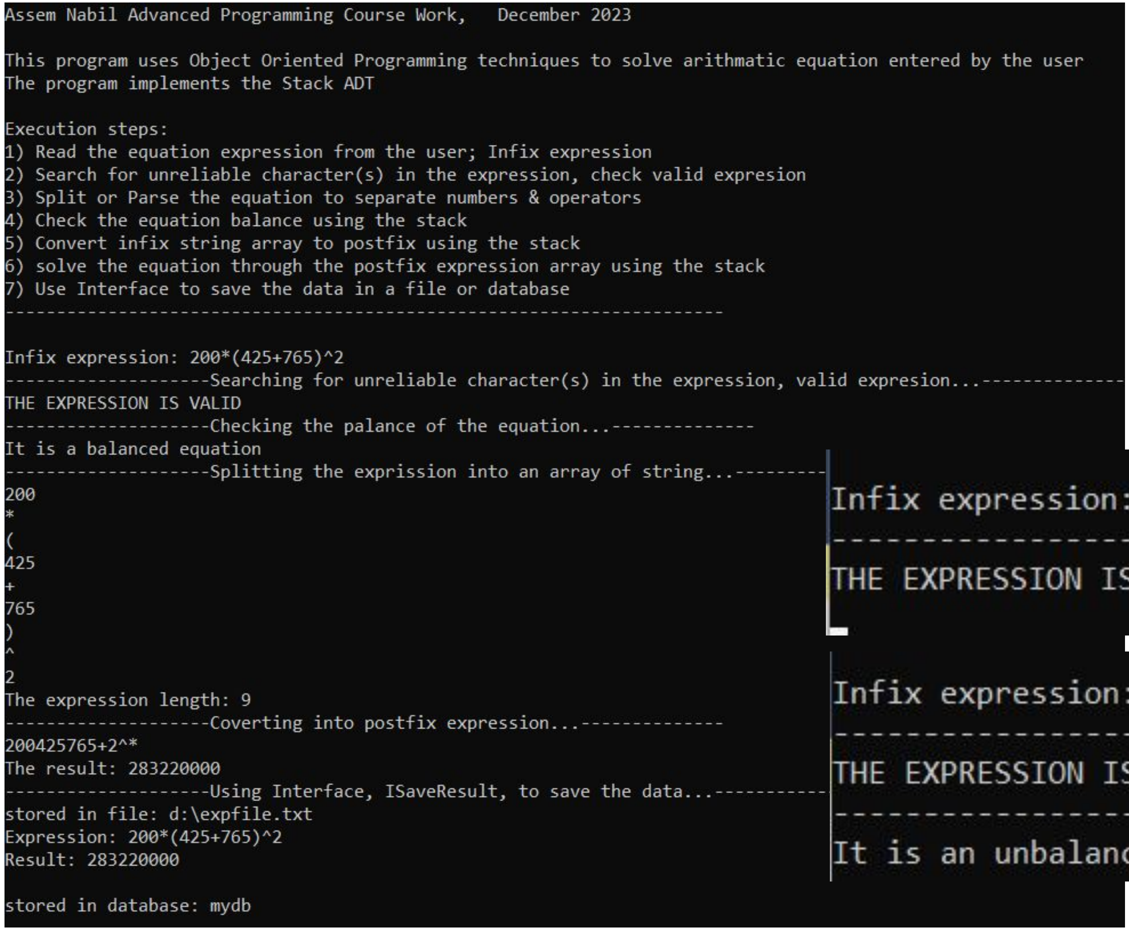
The class implement the interface method to save the data in a text file

Implementation: DatabaseSaveResult : ISaveResult

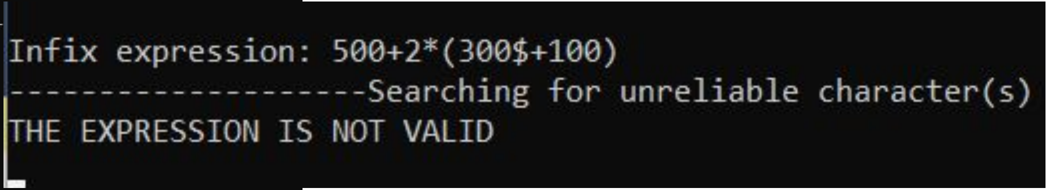
- Methods
  - void AddEntry(string infixExp, string result)

The class implement the interface method to save the data in a database

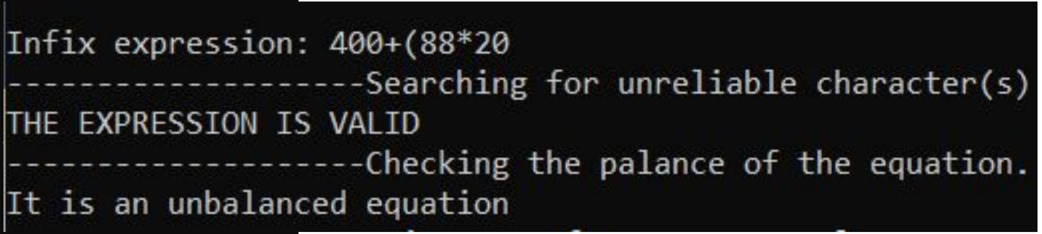
Testing and Validation



```
Assem Nabil Advanced Programming Course Work,   December 2023

This program uses Object Oriented Programming techniques to solve arithmatic equation entered by the user
The program implements the Stack ADT

Execution steps:
1) Read the equation expression from the user; Infix expression
2) Search for unreliable character(s) in the expression, check valid expresion
3) Split or Parse the equation to separate numbers & operators
4) Check the equation balance using the stack
5) Convert infix string array to postfix using the stack
6) solve the equation through the postfix expression array using the stack
7) Use Interface to save the data in a file or database
----------------------------------------------------------------------

Infix expression: 200*(425+765)^2
--------------------Searching for unreliable character(s) in the expression, valid expresion...-------------
THE EXPRESSION IS VALID
--------------------Checking the palance of the equation...--------------
It is a balanced equation
--------------------Splitting the exprission into an array of string...--------
200
*
(
425
+
765
)
^
2
The expression length: 9
--------------------Coverting into postfix expression...--------------
200425765+2^*
The result: 283220000
--------------------Using Interface, ISaveResult, to save the data...----------
stored in file: d:\expfile.txt
Expression: 200*(425+765)^2
Result: 283220000

stored in database: mydb
```



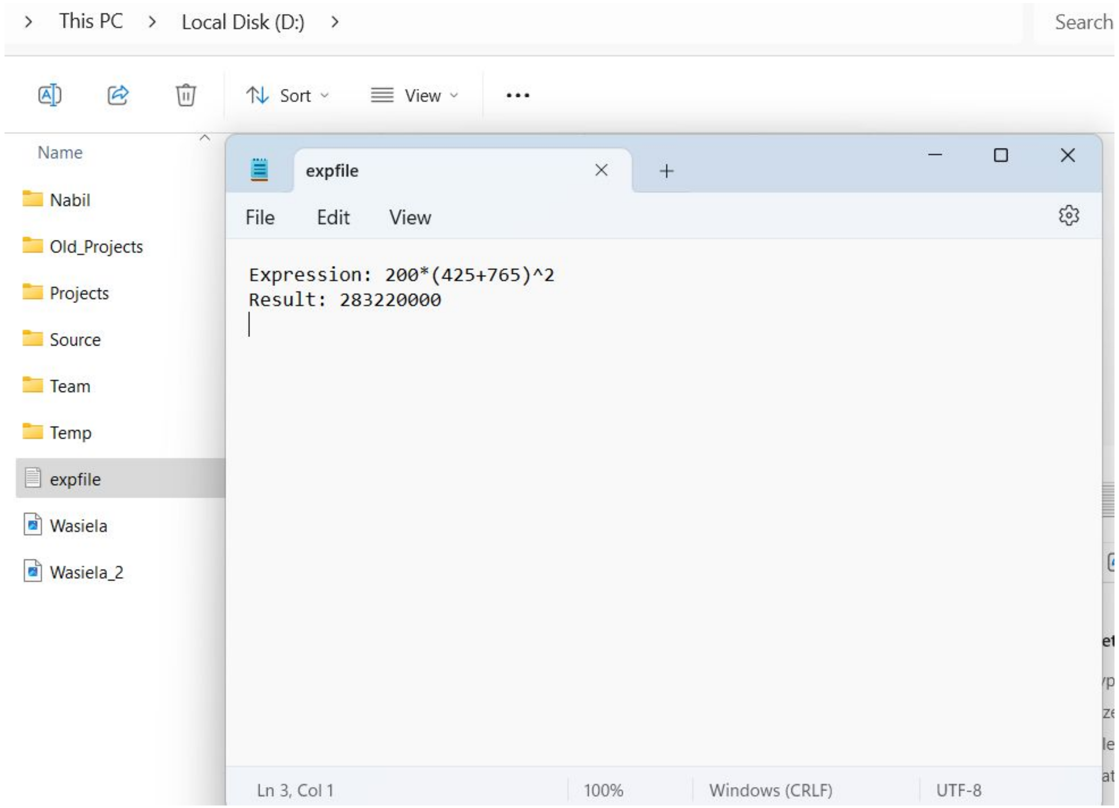
```
Infix expression: 500+2*(300$+100)
--------------------Searching for unreliable character(s)
THE EXPRESSION IS NOT VALID
```



```
Infix expression: 400+(88*20
--------------------Searching for unreliable character(s)
THE EXPRESSION IS VALID
--------------------Checking the palance of the equation.
It is an unbalanced equation
```

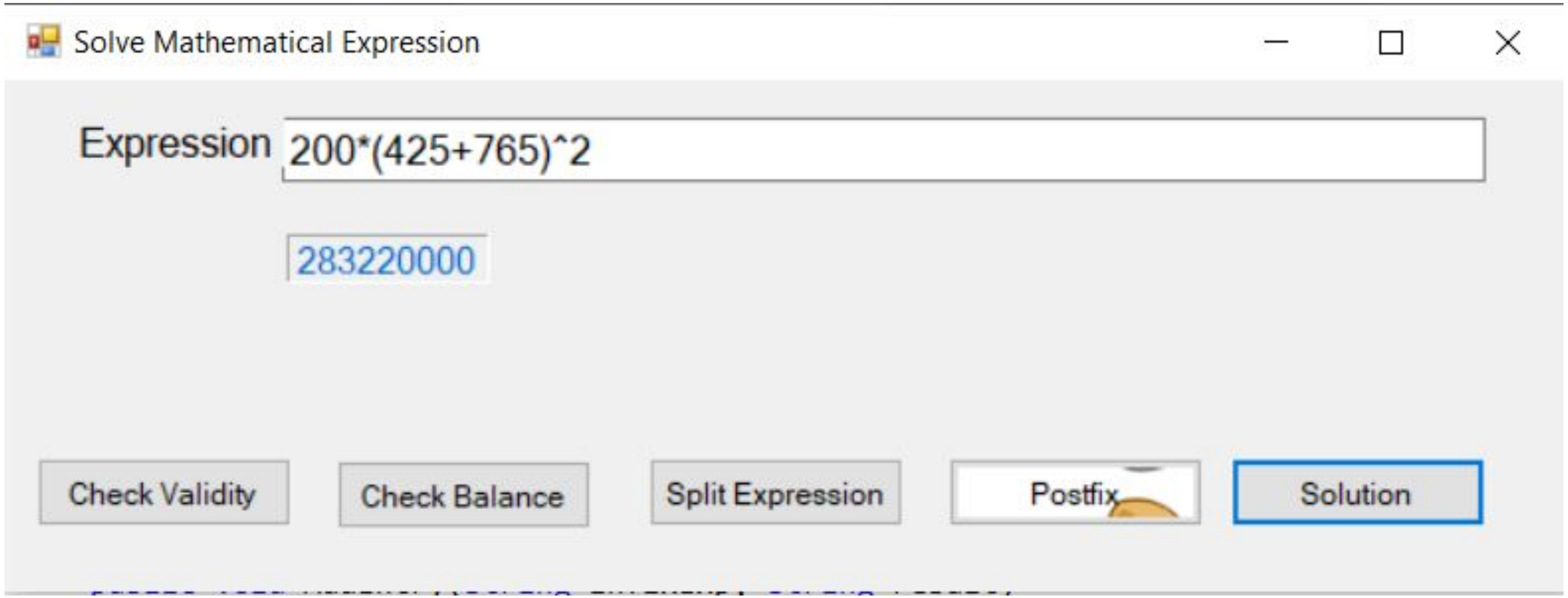
Testing and Validation

The output in the text file



```
Expression: 200*(425+765)^2
Result: 283220000
```

User Interface



```
200*(425+765)^2
```

```
283220000
```