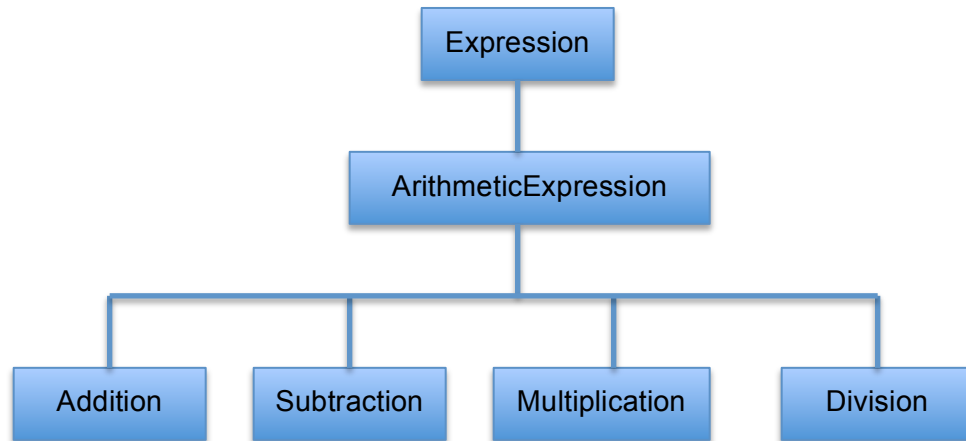


# SfwrEng/CompSci 2S03 Fall 2015 Homework 6

In a **group of 3**, you are to implement a calculator in C++ that is capable of evaluating basic arithmetic (+, -, \*, /) expressions. You may work in a different group of 3 from HW4. The system shall take as input a string of arithmetic expression, e.g (1 \* 2 / (4 + 5))/9. It will (1) check if the input is valid, (2) parse the input into expression tree, and (3) evaluate the expression.

## Class-Hierarchy



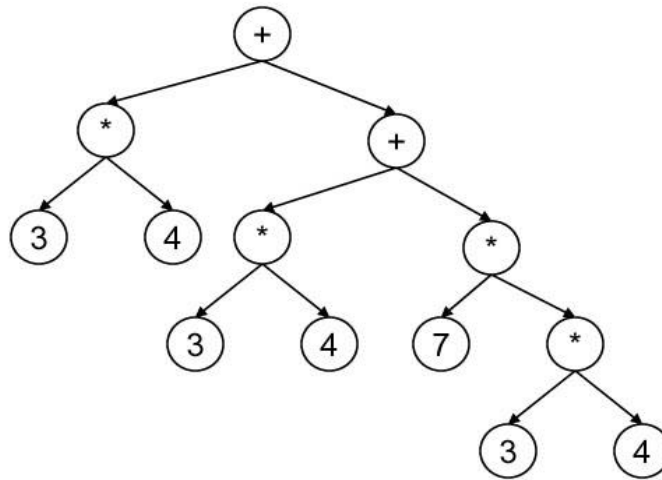
```
class Expression{
    virtual string evaluate(); // evaluate expression and return string
                                representation of the result.
    virtual void print(); // prints expression
}
class ArithmeticExpression : public Expression{
    Expression *left;
    Expression *right;
    string evaluate () { //evaluate left expression and right expression
    void print(); // prints expression
        ...
    }
    float convert (string s){ // Converts a string (as would be returned by
evaluate) to a float
    }
}
//Add two expression
class Add: public ArithmeticExpression{
    string evaluate(){ ... }
    void print(){ ... } // (left `+` right)
}
//Subtract two expression
class Subtract: public ArithmeticExpression{
    string evaluate(){ ... }
    void print(){ ... } // (left `-` right)
}
```

```

//Multiply two expression
class Multiply: public ArithmeticExpression{
    string evaluate(){ ... }
    void print(){ ... } // (left `*` right)
}
//Divide two expression
class Divide: public ArithmeticExpression{
    string evaluate(){ ... }
    void print(){ ... } // (left `/` right)
}

```

**Expression Tree** of expression  $(3*4) + ((3*4) + (7* (3*4)))$  is the following



This expression tree should be constructed by dynamically allocating appropriate objects of Expression classes.

### Main method

You should create a file named HWK6\_MACID.cpp which contains a main function that does the following:

- Reads a string from the standard input
- Parses that string into an Expression tree. Expression objects should be created dynamically. If an input string is invalid (e.g.,  $+2*3-$ ), only print "Expression is not well formed".
- When the input string is successfully parsed, the program should evaluate the resulting expression by calling `Expression::evaluate()` of the **top** Expression object. This should recursively evaluate the entire expression.
- Prints the original string by calling the `Expression::print()` of the **top** Expression object, then an equal sign, followed by the final result of the evaluation. Again, printing should be accomplished recursively.
- Deallocates all dynamically created objects appropriately. If your program exhibits memory leaks, you will lose points.

- Will repeat this process until the string '#' is received, at which point execution will terminate.

## Requirements

- Evaluation should follow the correct order of operations (left to right). For Arithmetic expressions the order is BEDMAS.
- Keep results rounded to 2 decimal points when applicable
- You should follow the class hierarchy (names, inheritance, member functions, etc). You should not change the signature of member functions. However, you can add new private member variables and functions.

## Example

Please enter an expression:  $1 + 2 + 3 + 4$

$1 + 2 + 3 + 4 = 10$

Please enter an expression:  $1 + 2 / 3 + 4$

$1 + 2 / 3 + 4 = 5.67$

Please enter an expression:  $(1 + 2) / 3 + 4$

$(1+2) / 3 + 4 = 5$

Please enter an expression:  $1 + 2 / * 3 + 4$

Expression is not well formed

Please enter an expression: #

## Submission

**\*\*\*Only one submission per group\*\*\***

All files are to be submitted using the Avenue system. Please ensure you submit all files with the correct names, as described below:

- Upload a **ZIP** file named HWK6\_MACID.zip
  - NOTE: the folder zipped must also be named HWK6\_MACID, where MACID is the MACID of the submitting member.

The ZIP file must contain the following .java source files:

- HWK6\_MACID.cpp (contains the main function)
- Expression.cpp
- Expression.h
- ArithmeticExpression.cpp
- ArithmeticExpression.h

- Addition.cpp
  - Addition.h
  - Subtraction.cpp
  - Subtraction.h
  - Multiplication.cpp
  - Multiplication.h
  - Division.cpp
  - Division.h
- If you don't follow the above naming convention, you will receive no credit.
  - If your program does not compile, you will receive 0. Your files must be submitted **by 8:30am Dec. 12, 2015**, on Avenue to Learn.

## Documentation

Your program must be commented properly: each section of the code as well as each line.

## Format

The following preamble **must** be written at the top of each .cpp source file:

```
/*
 * Name: [Full name of (no nicknames or chosen names) of each group member]
 * MacID: [MacIDs of all group members]
 * Student Number: [Student number of all group members]
 * Description: [This is an informative excerpt about this file.]
 */
```

**Failure in meeting this format and file naming convention will result in 0 credit.**

## Extra credit

If '@' is entered, then the last expression should be re-evaluated by incrementing all numbers in the expression. For example, if the last entered expression was  $(3+4)*9$ , then upon entering '@',  $(4+5)*10$  will be evaluated. To this end, you need to implement the following:

- `Expression *NewExp = new Expression(OldExp);`
- `NewExp -> increment();`
- `NewExp -> evaluate();`

The first line should clone the entire old expression tree in the copy constructor, where OldExp is the last entered expression. Then, `increment()` will increment all numbers in the expression and `evaluate()` will re-evaluate the new expression. Of course, one can enter '@' multiple times.