**Project 2 Outline**

* Accepts string which is a Postfix Input:
  + Ex. 8 2 \_1 + / 15 \*;
  + Operators: +, -, \*, /, %
  + \_ means the next int that follows is meant to be negative
  + White space will separate two integers except when it is two consecutive negative integers Ex. \_3\_4 means push -3, push -4.
  + Also an operator can be attached after an int. Ex. 4 3+ instead of 4 3 +
  + An int can also be attached after an operator for example 4 +3
* So when we split string by space, if the splitted string contains anything other than an int, than we know we have a special case.
  + 3 4 \_3\_4 +;

So here we would see \_3\_4 contains two underscores.

* + So our spitted field will either be
    - The Simple Cases
      * just ints, just an if statement checking field if all chars are ints. if just an int, this can simply be pushed into the stack
      * just an operation(will be one character long). If just an operation, we can simply use an if statement to determine which operation it is and peform it properly
      * An underscore followed by some int Ex. \_3
        + Underscore always comes before the int, so if you find an underscore, parse the int after the index of the underscore and multiply it by -1 to make it negative before pushing it onto stack.
    - The Complex Cases
      * With combined operators and operands Ex. 2\_3+ 4/5-;
      * For all the Complex Cases, one good way to solve it is to try to make the complex case a simple one, then use the same algorithms as before to solve it.
        + This can be done by first determining if we have a complex case. We know a case is complex if the splitted field is longer than 1, If it consists of non integers, and if it does not have a size of 2, in which the first char is an underscore and the second char is an integer. We might not necessarily need to check if its complex.
        + To convert all complex cases to a simple one, we just have to check each position of the string.

If the char at our current position is a integer, check to see if the char at the next position is a space or a semicolon,

if it is not a space and it is not a semicolon insert a space after our current char.

If the char at our current position is a operation, check to see if the char at the next position is a space or a semicolon,

If it is not a space and it is not a semicolon insert a space after our current char.

* + - * + This Algorithm will successfully convert all complex cases to a simple case.
* PostFix Formula
  + Push all operands onto the stack one by one
  + When you reach a operator, pop the top two operands off the stack
  + Use that operator to compute a result, push that result on top of the stack
* Exceptions
  + When incorrect entries occur, use exception handling to halt execution
    - This occurs when an invalid entry is entered (something that is not an int)\
    - Or when a division by zero is about to occur
  + Use exception handling using try, catch and throw approach.
  + Check for exceptions after the initial string input, and before each division
  + If an exception occurs print out a message and terminate the program.
  + Need Exceptions for stack overflow and stack underflow
* How this will be accomplished
  + Utilize the given stack class
  + Might need to use atoi to convert strings to ints.

Part 1 - Algorithm

1. Give me the input
   1. Ex. 8 2 \_1 + / 15 \*;
2. Check if input is correct, if it is continue, if not terminate and maybe ask user to try again
   1. first 2 chars(excluding spaces) will have to be operands or else its invalid. (A stack underflow is bound to happen)
   2. Check last char to make sure it is a semicolon, if not expression is invalid.
   3. Check each char make sure they are either (integers, spaces, or the appropriate list of operands) if anything else, the expression is invalid.
   4. Preventative Measures but not necessary
      1. Number of operators in a row has to be less than number of previous integers in a row or else a stack underflow will occur.
      2. In general number of integers has to be greater than the number of operators or else a stack underflow will occur.
      3. When we see a division operator, if the char before it was a zero a division by zero would occur. Or if we see a zero, if a division sign is after it the same error will occur.
3. After verifying that the input is valid,
4. Tokenize/Split the string input based on a 'space' delimiter.
5. Each splitted segment is called a field. Check each field to determine if that field is a simple case or a complex case.
6. If the field is a simple case, send that field to a method which will apply the simple algorithm.
   1. Check next(or first) char
      1. If char is a space move to next char until non space
      2. if char is integer push it onto the stack
      3. if char is an underscore, push the negative of the integer at the next position onto the stack
      4. if char is operator
         1. Pop the top two items off stack
         2. Compute there result
         3. Push the result onto onto the stack
      5. If char is a semicolon stop, The item on top of stack is the final result.
      6. Repeat from a.
7. If the field is not a simple case, send it to a method that will convert it from a complex case into a simple case, then send it to the simple algorithm method.

Part 2 Outline

Converting Infix to Postfix

1. Get the first linfix expression from file and store it in a string Ex. (3 + 4)
2. Tokenizing the string is not necessary to convert it to postfix, the plan is to analyze each char of the string.
3. Go to next(first) char position
   1. if char is open parenthesis (
      1. push onto stack regardless
   2. if char is an int ex. 3
      1. print it out to screen, or in our case add it to our empty vector Ex. Vector = “3”;
   3. if char is a space
      1. do nothing
   4. if char is a operator
      1. push it onto stack if it is >= what is at top of stack
   5. if char is closed parenthesis )
      1. Pop everything in the stack and append it to the vector, until you get to an open parenthesis
4. Repeat from step 3 until there are no more chars to check. Once complete vector will contain a postfix expression.

Postfix to Binary Expression Tree

This time our stack must contain pointers since eventually we will merge subtrees.

1. Start at first char
   1. If it is a operand (#), push it onto the stack.
   2. If it is a operator
      1. Create a tree Node whose char is the operator
      2. pop the top two items from stack
         1. Make the first one you pop the right child of the operator
         2. Make the second one you pop the left child of the operator