Assignment 3

Instructions: Write a makefile that has targets B, C, D and E, corresponding to each of the problems below. For each problem, write as many source files as you like and name them what you like, but make sure that the makefile correctly compiles each problem. For problem A, submit two files, A.c and A.cpp. You do not need to include problem A in your makefile. Tar and zip your solutions and submit them via http://www.cs.uoregon.edu/Classes/12F/cis330/turnin.php before midnight GMT on Saturday, Oct 20.

A) Write two programs that print a table of the first 50 multiples of 3, with 5 on each line, lining up on the right edge. Your output should look like the following:

```
    3   6   9   12  15
  18  21  24  27  30
  33  36  39  42  45
  48  51  54  57  60
  63  66  69  72  75
  78  81  84  87  90
  93  96  99 102 105
108 111 114 117 120
123 126 129 132 135
138 141 144 147 150
```

Write two versions, one in C using printf and one in C++ using the stream operators.

B) Write a C++ program that reads numbers from the cin stream until it reads a zero. For each number, your program should output one of the following messages on a line in standard out:

- The number is probably prime
- The number is divisible by two
- The number is divisible by three
- The number is divisible by five
- The number is not an Euler prime

Recall that \( n \) is an Euler prime (or pseudoprime) if \( p^{(n-1)/2} \equiv \pm 1 \) (mod \( n \)) for some prime \( p \). For each parameter \( n \), your program should test \( p = 2, 3 \) and 5. \( n \) is an Euler prime, if it is also prime, otherwise, it is an Euler pseudoprime. Most Euler numbers are prime.

Your program should not make use of the imprecise C pow() function. Rather, recall that we can use the following identity to quickly calculate powers:

\[
a^b = \begin{cases} 
(a^{b/2})^2 & \text{b even} \\
ab(a^{b-1}) & \text{b odd}
\end{cases}
\]

C) Write a C++ program that creates an unbalanced binary search tree structure. Your program should read positive numbers from standard in until it reads a zero. Then, your program should output the tree structure. The first number read will be the root of the tree, and subsequent numbers should be put in the correct place. That is sufficient since your tree is not self-balancing. Recall that if we add 5 to the binary search tree on the left, it will look like the tree on the right.
Your output should represent a pre-order walk with null values included, contain each node value on a single line and represent each change in depth by prepending a tab to the line. Each node should be represented by the following data structure:

```c
struct treenode{
  struct treenode* left;
  struct treenode* right;
  int value;
};
```

Hint: Use a typedef so you don’t have to type `struct` all the time.

<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Sample output</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 7 1 5 4</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>

D) The following BNF grammar describes a prefix arithmetic problem on natural numbers.

```plaintext
E ::= unsignedInt space | operator E E
```

The production `unsignedInt` is given by the regular expression `[0-9]+` and `operator` is given by the regular expression `[-+*/^]`. Note that this definition requires a trailing space character.

Write a C++ program that reads from the standard in stream and prints the postfix version of the input to the standard output stream. Thus, if the input is `+ 2 3 4`, the output would be `2 3 4 +`. In postfix, there should be a space after each number, but no space after each operator. Hint: throw out your old code and parse the input into a tree structure, and then do a post-order walk, similar to problem 3’s preorder walk.

Your program should continue to read lines until it encounters a line containing only zero. Your program should stop when it reads a line containing only zero.

E) Write a C++ program uses the state pattern to simulate the Deschutes Hall Pop Machine. The input can consist of lines containing one of the following:

- quarter
- dollar
- pepsi
- diet
- dew
- mist
- pepper
- water
- exit

The output of your program may consist of lines containing the following output:

- accept quarter
- accept dollar
- reject quarter
- reject dollar
- dispense quarter
- dispense pepsi
- dispense diet
- dispense dew
- dispense mist
- dispense pepper
- dispense water
- shutting down

The input represents the customer either depositing a quarter or a dollar or pressing a product button, and the output represents the machine’s action for each input. Bottled water costs $1.00. Every other product costs $1.50. The machine will reject any money input once it has reached $1.50. When a product is input, if the user has input at least the cost, the machine will first dispense the product, then it will dispense enough quarters to refund any change owed the customer (in that order!) “Exit” in the input means that the machine is unplugged. It will not refund any money when it is unplugged. Your program should represent the FSM on the following page. Each state represents how much money is in the machine, with the negative meaning the machine has that amount of money, but owes it to the customer. The green arrows represent the machine accepting a quarter; the orange arrows represent the machine accepting a dollar; The red arrows represent the machine dispensing some number of quarters; the blue arrow representing the machine dispensing a water, and the black arrows represent the machine dispensing some product other than water. Rejecting a quarter or dollar is a self-loop to the same state, and not shown in the FSM. Pushing a product button without enough money in the machine is also a self-loop, but generates no output.

Each state in the FSM should be represented in your program by a function, which takes the user input, possibly prints some output, and sets a global variable to a function pointer to the next state. Do not use classes in your C++ code. Here is some code to get you started:
#include <iostream>
#include <string>
using namespace std;

typedef void (*state)(string); //typedef for state pointer
void have0(string);
void have25(string);
void have50(string);
void have75(string);
void have100(string);
// more function declarations here
void owe25(string);
// more function declarations here
state current_state = have0;

int main(int argc, char** argv) {
    string input;
    //get input
    while (input != "exit") {
        current_state(input);
        //get input
    }
}
<table>
<thead>
<tr>
<th>Sample Input</th>
<th>Internal State</th>
<th>Sample Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>quarter</td>
<td>25¢</td>
<td>accept quarter</td>
</tr>
<tr>
<td>pepper</td>
<td>25¢</td>
<td>accept dollar</td>
</tr>
<tr>
<td>dollar</td>
<td>$1.25</td>
<td>accept dollar</td>
</tr>
<tr>
<td>pepper</td>
<td>$1.25</td>
<td>dispense pepper</td>
</tr>
<tr>
<td>dollar</td>
<td>$2.25</td>
<td>dispense quarter</td>
</tr>
<tr>
<td>pepper</td>
<td>-$1.25</td>
<td>dispense quarter</td>
</tr>
<tr>
<td>dollar</td>
<td>0¢</td>
<td>dispense quarter</td>
</tr>
<tr>
<td>exit</td>
<td>$1</td>
<td>shutting down</td>
</tr>
</tbody>
</table>