Homework 3: Functions in C++

CS16 - Winter 2021

Due:	Thursday, January 28, 2021 (11:59 PM PST)
Points:	75
Name:	Tyler_Pruitt_____
Homework buddy:	_____

- You may collaborate on this homework with **at most** one person, an optional "homework buddy."
- **Submission instructions:** All questions are to be written (either by hand or typed) *in the provided spaces* and turned in as a single PDF on Gradescope. If you submit handwritten solutions write legibly. We reserve the right to give 0 points to answers we cannot read. When you submit your answer on Gradescope, **be sure to select which portions of your answer correspond to which problem** and clearly mark on the page itself which problem you are answering. We reserve the right to give 0 points to submissions that fail to do this.

1. (15 points) Write a **definition** for a void function called `check_it()` that takes 3 integer arguments and prints "YES" if the arguments are in ascending order. Otherwise, it returns "NO".

   For example, `check_it(1, 2, 6)` returns "YES", but `check_it(6, 6, 1)` returns "NO".

```
void check_it(int a, int b, int c)
{
    if (a <= b && b <= c)
    {
        cout << "YES" << endl;
    }
    else
    {
        cout << "NO" << endl;
    }
}
```

2. (15 points) Write a **definition** for a void function called `roll()` that takes no arguments and prints a *random integer number* between 2 and 13 (inclusive) every time the function is called.

   For example, if, in a program, you do this:

   ```
   for (int k = 0; k < 5; k++) {
       roll();
   }
   ```

   You could get an output like this (note the newlines after each number). Assume your program already has included `cstdlib` and `ctime` libraries and has seeded the random number generator in the `main()` function. Just give the function definition for the answer.

   ```
   4
   2
   11
   5
   7
   ```

```
void roll()
{
    cout << rand()%12 + 2 << endl;
}
```

3. (10 points) Consider the follow code snippet:

```
int x = 10;
while (x-- >= 3) {
    cout << x << " ";
    if (!(x % 3)) {
        cout << "Buzz! ";
        if ((x % 2) == 0) {
            cout << "Fizz!";
        }
    }
    else {
        cout << "..." << endl;
    }
}
```

a. (3 points) Write what this code will print out exactly.

```
9 Buzz! 8 ...
7 ...
6 Buzz! Fizz!5 ...
4 ...
3 Buzz! 2 ...
```

b. (7 points) Explain step-by-step **why** the program prints out what it does.

First the program sets x to the value 10. Then it checks the while loop condition, since x-- decrements after the condition is checked this loop will run for x=10 up to x=3 but inside the loop this corresponds to x=9 to x=2 because x-- decrements after the while loop check condition. For each run of the while loops block of code x is printed to screen followed by a space. This explains why the numbers 9 through 2 are printed to screen followed by a space. Inside the loop body, after printing out the variable x, it will print "Buzz! " if the value of x inside the loop is divisible by 3 because divisible by 3 means that x % 3 will evaluate to zero (which means false) but there is a NOT statement here so if x % 3 is zero it will go to 1 and vice versa. Now, if the value of x is divisible by 3 it will print out "Buzz! " and then it will print "Fizz!" only if this same value is also divisible by 2. This is because if x is divisible by 2 x%2 will evaluate to zero. If the value of x, however, is not divisible by 3 then it will enter the else statement's block of code and print "..." and start a new line. Thus, since 9 is divisible by 3 "Buzz! " is printed after 9. Since 8 is not divisible by 3 "..." is printed and a new line is started. Since 6 is divisible by 3 AND divisible by 2 "Buzz! " is printed followed by "Fizz!". Since there is no extra space after "Fizz!", the next variable 5 is printed right after "Fizz!" and since 5 is not divisible by 3 "..." is printed and a new line is started. Similarly, since 4 is not divisible by 3 "..." is

printed and a new line is started. Since 3 is divisible by 3 "Buzz! " is printed and a new line is not started. Finally, the final value of x, 2, is printed and since it is not divisible by 3 "..." is printed followed by ending the line.

4. (5 points) Explain the difference between these 2 snippets of code.

```
for (int i = 0; i < 10; i++) {
    cout << i;
}
```

```
int i;
for (i = 0; i < 10; i++) {
    cout << i;
}
```

These two snippets of code have the same output but they are accomplished in two different ways. The first body of code is declaring and initializing a local variable i inside the for loop and incrementing this variable for each run of the loop. The second body of code is first declaring the integer variable i outside of the for loop so that it is not a local variable to the for loop, and then it is using a for loop to initialize this variable to the value zero and then is incrementing this variables for each run of the loop. The main difference here is that the first body of code is treating i as a local integer variable to the for loop and is not accessible outside of the for loop, whereas the second body of code is treating i not as a local variable to the for loop but as a regular variable that is accessible outside of the for loop. Although their output is the same, these two snippets of code are not entirely equivalent.

5. (6 points) Consider the code below.

```
int a = 7, b = 9;
cout << "Here is ";
while (a++ % b != 0) {
    cout << a << " ";
    b += 2;
    a -= 2;
}
cout << endl;
```

a. (2 points) Write what this code will print exactly.

```
Here is 8 7 6 5 4 3 2
```

b. (4 points) How is the value of variable `a` changing? Show your work!

The variable a is initialized to the value 7. Then, the while loop condition is checked. After the while loop condition is checked, a is incremented by one because there is a++ inside the loop condition. Inside the loop, a is then decremented by two because of the a -= 2 statement. Altogether, the while loop body decrements a by one because it first increments a by one from the while loop condition and then it decrements a by two because of the

final decrement statement. For example, in the very beginning a is initialized to 7 and then the while condition is checked. Since 7 % 9 is not zero, the while loop body is executed and first a is incremented by 1 to 8. Then 8 is printed to screen. Then a is decremented down by 2 to 6. Now the while loop condition is checked again and this process keeps repeating until a % b is zero.

6. (15 points) Consider the following `main()` function.

```
int main() {
    int x = 10, y = 20, z = 30;
    shift(x, y, z);
    cout << x << " " << y << " " << z << endl;
    return 0;
}
```

The **body** of the `shift()` function is as follows.

```
{
    int temp;
    temp = var1;
    var1 = var2;
    var2 = var3;
    var3 = temp;
}
```

What will this program print for each of the following function declarations for `shift()`. **Explain why!**

a. (5 points) `void shift(int var1, int var2, int &var3);`

This program will print "10 20 10". This is because in the declaration and definition of the shift() function var1 and var2 are being passed in by call-by-value meaning that this function will not change var1 or var2 outside of the function itself. However, var3 is being passed through the function shift() by call-by-reference so any modifications to var3 inside shift() will also affect the variable outside of the scope of the shift() function. This means that x and y do not change by z changes to x from the function definition so what is printed is "10 20 10".

b. (5 points) `void shift(int &var1, int &var2, int var3);`

This program will print "20 30 30". This is because var1 and var2 are being passed through shift() by call-by-reference so changes from the shift() function will continue to exist for var1 and var2 outside of the scope of shift(), whereas var3 is being passed through shift() by call-by-value so changes to var3 inside of shift() will not affect the var3 outside of the function scope. Thus, x is assigned to y, y is assigned to z, and z does not change. Thus the program prints "20 30 30".

c. (5 points) `void shift(int &var1, int &var2, int &var3);`

The program will print "20 30 10". This is because all arguments of the shift() function: var1, var2, and var3, are being passed through shift() by call-by-reference so changes that occur to var1, var2, var3 inside of the shift() function continue to exist outside of body of shift(). This means that var1 is assigned to var2, var2 to var3, and var3 to var1. When x, y, and z are passed to shift(), x is assigned to y, y to z, and z to x. Thus, the program will print "20 30 10".

7. (6 points) We talked about 3 concepts related to programmer-defined functions: (1) function declaration, (2) function definition, and (3) function call.

```
1  #include <iostream>
2  using namespace std;
3  bool isDivisibleBy(int a, int b);
4
5  int main() {
6      cout << "15 divisible by 5? " << isDivisibleBy(15, 5) << endl;
7      return 0;
8  }
9
10 bool isDivisibleBy(int a, int b) {
11     return (a % b == 0);
12 }
```

a. (2 points) List the line number(s) for the function declaration of isDivisibleBy().

line 3

b. (2 points) List the line number(s) for the function definition of isDivisibleBy().

lines 10, 11, 12

c. (2 points) List the line number(s) for the function calls of isDivisibleBy().

line 6

8. (3 points) What is a flag in a program and what use is it? (Read Chapter 3.)

A flag is a variable that is modified to indicate that a certain event has occurred. Flags are used to end/terminate loops and change the control flow. A common technique of doing this is the exit-on-a-flag technique.