**Programming I**

**Lab #8**

Objectives:

* Demonstrate understanding of:
  + Loops with multiple termination conditions
  + Value-returning functions

**Preparation:**

Step 1: Download the file Lab08\_Tasks.zip from bboard. (see Assignments->Labs->Lab08). To download the file, click right on the link and select Save Target As.... Then, navigate to the Lab folder you created earlier. Save Lab08\_Tasks.zip in this folder and unzip it to this folder.

**Task 1**

**For tasks 1 through 3, use your cannon project from the previous lab (lab 6). Make the following modifications to this code.**

We need to enhance the program to detect when the cannon ball hits the building or the ground. In this phase, you will make the animation terminate when the ball hits the ground (we will ignore the building for now).

To detect a ground hit:

* In the program, after you calculate the position of the ball, compare the newly calculated position of the ball to the position of the ground. Note that the **x** position of the ball is irrelevant for purposes of this comparison. When the **y** position is *>=* the **y** position of the ground (recall that the y dimension is “upside down”), a hit has occurred and the animation loop should terminate[[1]](#footnote-1). The position of the ground is given by the constant GROUND\_Y in the code.
* You should structure your logic so that the ball is drawn in the location where you detected the hit (i.e., make sure you don’t erase the final resting place of the ball). You may notice that, depending on the speed and angle, the ball goes “into” the ground before a hit is detected: that is to be expected since we are only calculating the position of the ball at discrete points in time; you don’t need to “fix” that problem.
* You may use **break** to break out of the loop when you detect the ground hit.

**Task 2**

Extend the cannon code to break out of the loop when the ball strikes the building. You need to determine the appropriate conditional statement that will handle all scenarios of the ball hitting the building. Test your program to make sure that it works when the ball strikes the side or top of the building. Make sure that if the ball flies *over* the building that your program doesn’t treat it as a building hit.

Some useful constants provided in the source code are:

BLDG\_LEFT // x coordinate of left side of building

BLDG\_RIGHT // x coordinate of right side of building

BLDG\_TOP // y coordinate of top of building

**Task 3**

When we learned how to use **break**, we also learned that it is generally considered good practice to avoid its use. Thus, go back and improve your code from tasks 1 and 2 so that the loop termination condition is only at the “top” of the loop.

***Hint:***

This is a great place to use a **bool** variable and an && in the loop, as illustrated below:

bool animate = true;

// would work equally well with a ‘for’ loop

while ( time <= TIME\_LIMIT && animate)

{

....

// set animate = false when ball hits the building or

// ground

// make other code adjustments as necessary

}

**Task 4**

You have completed the cannon project , so close your old workspace and open the Lab08\_tasks workspace that you downloaded for this lab.

Make the **roman** project the startup project. Create a function **letterToInt** with the following interface:

**int letterToInt( char letter )**

This function should accept a character variable. This variable represents *one* symbol in a roman numeral. Your function should return the integer equivalent (e.g., if you pass the function an 'X', it would return the number 10). The function should return -1 if the letter passed to it is not a valid letter in a roman numeral.

**Additional Requirements:**

* Use a switch statement in your function (see chapter 4 for examples).
* Your function should contain only one **return** statement.

Write a main() function to test your function.

The valid roman numerals and their meanings are:

I - 1

V - 5

X - 10

L - 50

C - 100

M - 1000

NOTE: This function does *not* need to handle a complete roman number such XIV; it only handles *one* digit.

**Task 5**

Make the **table** project the startup project. Write a function with the following interface:

**void multiplyTable(int num)**

This function should display the multiplication table for values from 1...num. For example, if the function is passed 10 when it is called, it should display the following:

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

3 6 9 12 15 18 21 24 27 30

4 8 12 16 20 24 28 32 36 40

5 10 15 20 25 30 35 40 45 50

6 12 18 24 30 36 42 48 54 60

7 14 21 28 35 42 49 56 63 70

8 16 24 32 40 48 56 64 72 80

9 18 27 36 45 54 63 72 81 90

10 20 30 40 50 60 70 80 90 100

The first row shows the product of 1\*1, 1\*2, 1\*3......1\*10. The second row shows the product of 2\*1, 2\*2, 2\*3.....2\*10, and so on, for 1..10 in each dimension.

As another example, if the function is passed 5, it would print the following:

1 2 3 4 5

2 4 6 8 10

3 6 9 12 15

4 8 12 16 20

5 10 15 20 25

Write a main() to test your function. Main should ask the user for an input value (in the range 1..15), then call **multiplyTable** with that value.

Hints:

* Used nested **for** loops to print the table.
* The maximum parameter value that your function will need to handle is 15.
* The alignment is achieved by displaying each number in a field width of 4 characters.

1. Notice we are comparing the center of the ball rather than the surface of the ball. This is close enough. [↑](#footnote-ref-1)