CSI 1430
Week 4 Resource

Colin Burdine

2/8/21

Major Topics:

1. Conditional Branching (Continued)
2. Boolean Values
3. Logical Operators
4. Programming Example Problem

Keywords: *branching, boolean values, logic*

Reminder: Group tutoring sessions for this course will be held online every Monday from 7:00pm to 8:00pm CT. For more information on how to sign up for these sessions, go to: https://www.baylor.edu/tutoring.

1 Notes on Major Topics:

1.1 Conditional Branching (Continued)

Last week we discussed some arithmetic operators and ways of performing mathematical operations in C++. This week, we are going to be exploring some concepts that are crucial to writing well-structured, logical code. In particular, we will be focusing on conditional branching and logic. Last week, we wrote the following program to tell us if a number was divisible by seven. To accomplish this, we use the *modulus* operator (%) to find the remainder of an integer when it is divided by 7:

```
1  #include<iostream>
2
3  using namespace std;
4
5  int main(){
6      int number;
7
8      // ask the user for a number:
9      cout << "Enter a number: ";
10
11     // read the number from the console:
12     cin >> number;
13
14     // Here, we print whether or not the number is divisible by 7.
15     // To do this, we check if the remainder of the number divided
16     // by 7 (obtained through the % operator) is equal to 0:
17     int remainder = number % 7;
18     if(remainder == 0){
19         cout << "Your number is divisible by seven." << endl;
20     } else {
21         cout << "Your number is not divisible by seven." << endl;
22     }
23     return 0;
24 }
```

Note that when this program is executed, there are two possible results that the program could output; either the number entered by the user is divisible by seven, or it is not. This is an example of an *if-else* branch, in which we either execute the first block (line 19) if the the condition in parentheses (following the `if` on line 18) evaluates to be true, or we execute the second block (line 21) otherwise. This act of specifying behaviors in a program that are predicated on different conditions is referred to as *branching*. There are several different kinds of conditional branching blocks that you can use in C++. I have illustrated some of the more common branching patterns in the code snippets below:

- **"if" branch:**

```
1 if ( <condition A> ){
2     // executes if condition A is true
3 }
```

- **"if"-"else" branch**

```
1 if ( <condition A> ){
2     // executes if condition A is true
3 } else {
4     // executes if condition A is false
5 }
```

- **"if"-"else if"-"else" branch:**

```
if ( <condition A> ){
    // executes if condition A is true
} else if ( <condition B> ){
    // executes if condition A is false, but condition B is true
} else {
    // executes if both conditions A and B are false
}
```

In the three code snippets above, it is worth noting that each of the blocks in the set of branches above represents an outcome that is *completely independent* of all other blocks; that is, only a single block will be executed when the program runs.

1.2 Boolean Variables

You may be wondering by now *What kinds of code segments can I put inside the parentheses following an* `if` *statement?*. The answer is, of course, Boolean variables! Last week, we said that we can declare Boolean variables using the `bool` type specifier in our code. Essentially, Boolean variables can hold either a `true` or `false` value. We can set their values using the `true` or `false` keywords in C++. The code snippet below shows how we can use boolean values for conditional branching:

```

bool someCondition = true;

if(someCondition){
    // executes if someCondition is true (this is the case here)
} else {
    // executes if someCondition is false (this is not the case here)
}
```

1.3 Logical Operators

It's quite handy that we can express a branching condition as a Boolean (`true` or `false`) value. In fact, we can perform logical operations on boolean values, much like we can perform arithmetic on numerical values. These operations are referred to as *logical operators* and are summarized in the table below:

Operator	Name	Effect
`a && b`	Logical AND	Evaluates to true only if `a` and `b` are true
`a || b`	Logical OR	Evaluates to true if `a` or `b` (or both) are true
`!a`	Logical NOT	Evaluates to true if `a` is false (or false if `a` is true)

1.4 Programming Example Problem

To illustrate how these operators can be used to together with Boolean variables, we will consider the following problem in which we extend the functionality of the "divisibility by seven" program we created above:

In mathematics, there is a result called the Sieve of Eratosthenes, by which we can be certain that any integer from 10 to 100 (inclusive) is prime if is it ***not*** *divisible by any prime number less than 10 (i.e. by 2, 3, 5, or 7). Otherwise, if at least one of those primes divides it, the number is not prime. Write a program that checks is a number entered by the user is prime using this method. If the number entered is outside the range from 2 to 100 (inclusive), then print that the number is out of range.*

My solution to this problem is on the next page. You can look at my answer if you are stuck, but I would strongly encourage you to use the empty space below to sketch out a design first, and then proceed to write your solution in C++. This exercise is intended to check your understanding of the concepts from this week and last week.

```cpp
/*
 *  Your correct header comments should go here :)
*/

#include <iostream>

using namespace std;

int main(){

    // This variable stores the number that the user enters:
    int number;

    // these variables store whether our value divides
    // one of the primes less than 10:
    bool numDivides2, numDivides3, numDivides5, numDivides7;

    // prompt user for a number:
    cout << "Enter a number from 10 to 100: ";
    cin >> number;

    // determine if the number is in the proper range:
    if ( 10 <= number && number <= 100 ){

        // determine if the number divides 2, 3, 5, or 7 by storing
        // the divisibility test results as bool (Boolean) types:
        numDivides2 = ((number % 2) == 0);
        numDivides3 = ((number % 3) == 0);
        numDivides5 = ((number % 5) == 0);
        numDivides7 = ((number % 7) == 0);

        // if at least one of the primes divides the number
        // then it is not prime:
        if ( numDivides2 || numDivides3 || numDivides5 || numDivides7 ){
            cout << "Your number (" << number << ") is not prime."
                 << endl;
        } else {
            cout << "Your number (" << number << ") is prime."
                 << endl;
        }

    } else {
        cout << "Your number (" << number << ") is not in the range "
             << "10-100!" << endl;
    }

    return 0;
}
```