19th October, 2023.

CPEN201: C++ PROGRAMMING

LAB 2

Control Structures

Lab 2a: Conditionals

If Statement

The if statement allows for your program to make a decision about what it should do. It asks a simple question: “Is this condition true?” If yes, then the computer will execute certain commands.

```
int x = 5;

if (x < 10) {
  cout << "Less than 10" << endl;
}
```

An if statement is comprised of the keyword if, followed by a boolean expression surrounded by parentheses (). Any code that should run if the boolean expression is true is surrounded by curly braces {}. It is best practice to indent this code, but it does not affect how the code runs.

If the boolean expression is false, the code in curly braces is skipped, and the program continues as normal.

```
int x = 20;

if (x < 10) {
  cout << "Less than 10" << endl;
}

cout << "And the program continues...";
```

if statements can be used to test multiple conditions. These conditions exist as boolean expressions inside the parentheses (). In addition, an if statement can exist inside another if statement. Both blocks of code below accomplish the same exact task.

```
int age = 15;

if (age < 20) {
  if (age > 10) {
    cout << "Teenager";
  }
```

```
int age = 15;

if ((age < 20) && (age > 10)) {
  cout << "Teenager";
}
```

If Else Statement

The if else statement gives your program the ability to ask a question, perform certain actions if the answer is true, and then perform another set of actions if the answer is false.

```
int x = 10;

if (x > 50) {
  cout << to_string(x) + " is greater than 50" << endl;
}
else {
  cout << to_string(x) + " is less than 50" << endl;
}
```

The if part of the if else statement is written as before. The else keyword is not indented; it should be aligned with the if keyword. else is followed by an open curly brace {. You do not use a boolean expression with else. All code that should run when the boolean expression is false should go before the closing curly brace }.

Be careful when expressing your boolean expression in terms of “less than or greater than”. This does not take into account when the values being compared are equal. Consider the code from above, but with x having the value of 50.

```
int x = 50;

if (x > 50) {
  cout << to_string(x) + " is greater than 50" << endl;
}
else {
  cout << to_string(x) + " is less than 50" << endl;
}
```

The output of the program does not make sense. 50 is not less than 50. Sometimes using <= and >= need to be used. Another solution is to update the output to be more inclusive such as replacing is less than 50 with is less than or equal to 50. In either case, be sure to think through all of the possible outcomes, and make sure your code can function properly in all of those scenarios.

Switch Statement

The switch case statement gives your program the ability to perform different actions based on the value of a given variable.

```
int size = 3;

switch (size) {
  case 1: cout << "Short"; break;
  case 2: cout << "Tall"; break;
  case 3: cout << "Grande"; break;
  case 4: cout << "Venti"; break;
  case 5: cout << "Trenta"; break;
  default: cout << "Grande";
}
```

The variable used to make the decision is in parentheses following the switch keyword. Inside curly braces, the cases listing the different values to check are followed by a : and then the code that should run if the variable is equal to that case's value comes next. The last case, default, runs if none of the other cases are true. Each code segment except the last one ends with break; to signal the program to jump to the closing curly brace.

Remember to include break; statements at the end of each case. Check out what happens when you remove them.

```
int size = 3;

switch (size) {
  case 1: cout << "Short";
  case 2: cout << "Tall";
  case 3: cout << "Grande";
  case 4: cout << "Venti";
  case 5: cout << "Trenta";
  default: cout << "Grande";
}
```

The output of the program does not make sense because the program continues through all of the cases after the initial case is matched to a value. In the example above, the program prints the command for case 3 as well as all of the commands that follow.

Task: Month of the Year

Write a program that determines the month of the year based on the value of a variable called month. The variable will be a number from 1 to 12 (1 is January, 2 is February, etc.). Use a cout << statement to write the month to the screen.

Create two C++ files. In one of the files, solve the task using ***if….else if*** statement and then in the other, solve the task using ***switch***.

Test your code with a few different values.

Lab 2b: Loops

For Loop

Copy the code below into your IDE (CodeBlocks). Then click on the ***Build and Run*** button to see the resulting output.

```
for (int x = 0; x < 11; x++) {
  if (x % 2 == 0) {
    cout << "Even" << endl;
  }
  else {
    cout << "Odd" << endl;
  }
}
```

Program Summary

1. The for loop runs through all the values of the variable x from 0 to 10 as specified in the loop header.
2. For each value of x, an expression is evaluated using a conditional if statement.

3. If x modulo 2 evaluates to 0, then print Even followed by a newline character.

4. If x modulo 2 does not evaluate to 0, then print Odd instead followed by a newline character.

While Loop

Copy the code below into your IDE (CodeBlocks). Then click on the ***Build and Run*** button to see the resulting output.

```
int counter = 0;
while (counter < 10) {
  cout << counter << endl;
  counter = counter + 1;
}
cout << "while loop ended" << endl;
```

Program Summary

1. A counter variable is initialized to keep track of how many times the loop will be executed.

2. The loop will run as long as counter is less than 10.

3. Each time the loop runs, the integer value of counter is printed to the screen.

4. The value of counter is then incremented by 1.

5. When counter reaches 10, the boolean expression no longer evaluates to true and the program will exit the loop.

6. Before the program terminates, a statement is printed to the screen, indicating that the while loop has ended.

7. Recall that the while loop must have an exit condition. By incrementing the counter variable, we ensure that the loop will eventually end. If we do not increment counter in this loop, we will create an infinite loop because counter will never reach 10 or greater.

Break Statement

Breaking from the While Loop

Copy the code below into your IDE (CodeBlocks). Then click on the ***Build and Run*** button to see the resulting output.

What does cin >> input; do? The cin >> input; command records what a user enters on the screen and stores that information in the variable input. Note that input is of type double.

What do cin.good() and cin.fail() do? cin.good() checks to see if the input entered by the user was successful while cin.fail() checks to see if the input failed. Since input is of type double, only numerical values entered by the user will cause cin >> input to be successful, anything else will cause the input to fail.

```
double result = 0;
double input;

while (true) {
  cout << "Enter a number to add to sum. ";
  cout << "Or enter a non-number to quit and calculate sum." <<
        endl;
  cin >> input;
  if (cin.good()) {
    result += input;
  }
  if (cin.fail()) {
    cout << "Sum = " << result << endl;
    break;
  }
}
```

Program Summary

1. Declare the variable result and initialize it to 0. result will store the total of the summation.

2. Declare the variable input. input will store the information that the user enters.

3. Next we set up a while loop with true as the expression in the loop header. We do this because we want the loop to continue running and storing information from the user. Since we don't know how much information the user will enter, a while loop is best for the situation.

4. The user is prompted to enter some information and that information is stored in the variable input which was declared earlier.

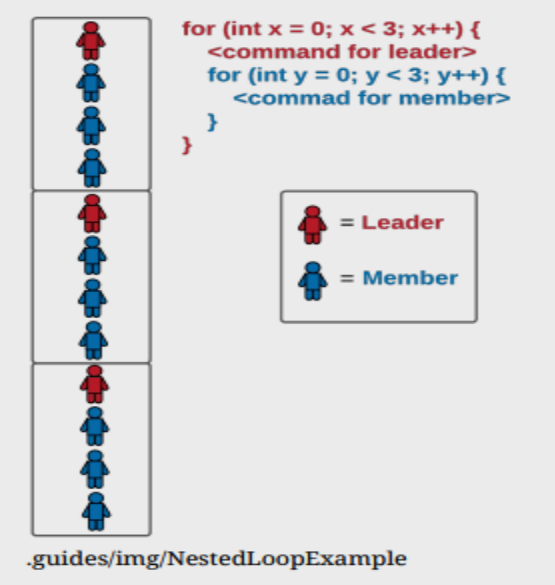
5. If the information was stored into input successfully, the value in input will be added to the value in result, our total summation.

6. If the information was not stored into input successfully, then the program will print out the total summation result and exit the while loop.

Task: Loop Patterns

Nested Loop Example

One of the benefits of nested loops is that they can be used to construct complex patterns. Imagine a classroom full of students and they are distributed evenly into smaller groups and asked to form a single line with their groups. The outer loop is like the group leader (represent in **red** and L) and the inner loop is like the rest of the group members (represented in **blue** and M.



.guides/img/NestedLoopExample

```
for (int x = 0; x < 3; x++) {
  cout << "L" << endl;
  for (int y = 0; y < 3; y++) {
    cout << "M" << endl;
  }
}
```

What is the pattern described by the above example? There are 3 leaders and each leader has 3 members. However, note that the example shows the students standing in a vertical line. What if you want to arrange the students in a horizontal line like this instead?

.guides/img/NestedLoopHorizontal

By removing the << endl commands from the code above, you can accomplish this task. Alternatively, you can also make use of an if and else statement instead of a nested loop. Both ways will produce the same result.

```
for (int c = 0; c < 3; c++){
    cout << "L";
    for (int y = 0; y < 3; y++){
        cout << "M";
    }
}
```

using nested loop

```
for (int x = 0; x < 12; x++){
    if ((x == 0) || (x == 4) || (x == 8)){
        cout << "L";
    }else{
        cout << "M";
    }
}
```

Using **if** and **else**

For this challenge, you will use your knowledge of patterns, conditionals, and nested for loops to produce the following output:

```
XOXOXOXOX
OXO
OXO
XOXOXOXOX
OXO
OXO
XOXOXOXOX
OXO
OXO
```

Requirement:

Your program must include at least two for loops, one nested within another, in order to receive credit. In addition, you are only allowed to use, at most, two cout statements.

Hint: You should start by determining a pattern that repeats itself. One noticeable pattern is:

```
XOXOXOXOX
OXO
OXO
```

Try creating that particular pattern first, then iterate that pattern by modifying the existing loop(s).

Happy coding