CSCI 1300 CS1: Starting Computing
Ashraf, Cox, Spring 2020
Homework 3 Background
Homework Due: Saturday, February 8, by 6 pm
(5 % bonus on the total score if submitted by 11:59 pm February 7th)

# Conditional Statements

Conditional statements, also known as decision or branching statements, are used to make a decision using a condition. A condition is an expression that evaluates to a boolean value, either true or false. Conditional Execution in C++ is a good online resource for learning about conditionals in C++.

**IF Statements:** An if statement in C++ is composed of a condition and a body. The body is executed only if the condition is true. The condition appears inside a set of parentheses following the keyword "if" and the body appears within a set of curly brackets after the condition:

```
if (<CONDITION>)
{
 <BODY>
}
```

It is good practice to vertically align matching curly brackets and to indent the body.

The condition is interpreted as a boolean value, either true or false. Be careful, most expressions in C++ have such an interpretation. For instance, non-zero numeric values are true. Assignment operations are interpreted as true as well. A common mistake is to use a single equals sign inside a condition when a double equals sign is intended. For example:

```
int x = 5;
if (x = 1)
{
 cout << "The condition is true" << endl;
}
```

will print "The condition is true". On the other hand,

```
int x = 5;
if (x == 1)
{
 cout << “The condition is true” << endl;
}
```

will not print anything since the condition is false. Remember, “=” is for assignment and “==” is for checking equality.

**IF-ELSE Statements:** If statements may be paired with else statements in C++. If the condition associated with the if statement is false, the body associated with the else statement is executed. The else statement body is enclosed in a set of curly brackets:

```
if (<CONDITION>)
{
 <BODY>
}
else
{
 <BODY>
}
```

While an if statement does not need an else statement, there must be an if statement before every else statement.

**IF-ELSE IF-ELSE Statements:** Finally, an if statement may also be associated with any number of else-if statements. These statements each have an associated condition and an associated body. The body is executed if the condition is true and the conditions for all preceding if statements and else-if statements in the same group are false. An else statement may be included at the end of the group but is not required.

```
if (<CONDITION>)
{
 <BODY>
}
else if (<CONDITION>)
{
 <BODY>
}
```

```
else
{
 <BODY>
}
```

**Examples:**

(1) If the int num is negative, print “Changing sign” and make it positive.

```
if (num<0)
{
 cout << “Changing sign” << endl;
 num = -1*num;
}
```

(2) If the int num is 0, print “Can’t divide by 0!”. Otherwise, set num to num divided by 2.

```
if (num==0) //notice the double equals!
{
 cout << “Can’t divide by 0!” << endl;
}
else
{
 num = num/2; //integer arithmetic
}
```

(3) If the int num is greater than 0 and less than 10, set num to 5 times itself. Otherwise, if num is greater than 100, set num to itself divided by 10.

```
if (num>0 && num<10)
{
 num = 5*num;
}
else if (num > 100)
{
 num = num/10;
}
```

(4) Print “Positive” if the int num is positive, “Zero” if it is 0, and “Negative” if it is negative. Both of the approaches below work. Why? Can you think of other ways to achieve the same result?

```
if (num>0)
{
 cout << “Positive” << endl;
}
else if (num==0)
{
 cout << “Zero” << endl;
}
else if (num<0)
{
 cout << “Negative” << endl;
}
```

```
if (num>0)
{
 cout << “Positive” << endl;
}
else if (num==0)
{
 cout << “Zero” << endl;
}
else
{
 cout << “Negative” << endl;
}
```

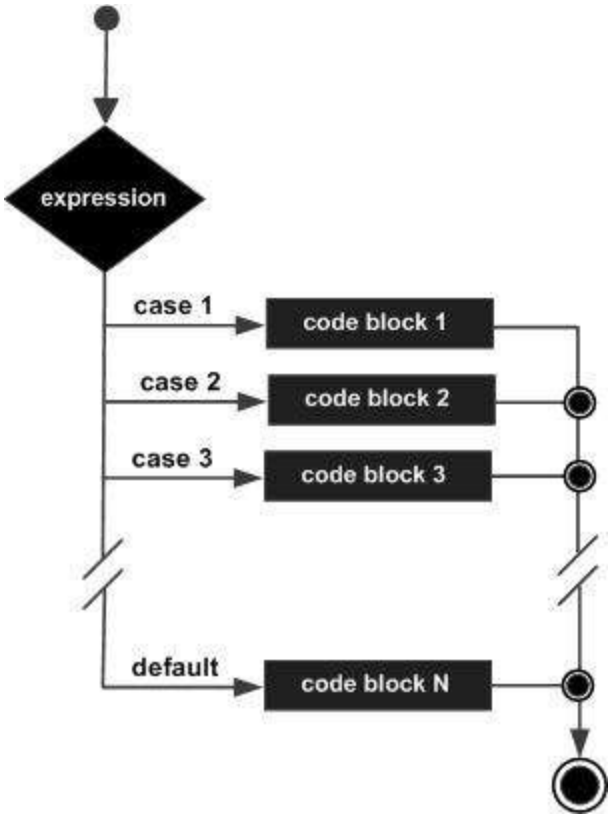
(5) Let score be an int between 0 and 100. Print the letter grade associated with score (A: 90-100, B: 80-89, C: 70-79, D: 60-69, F: 0-59)

```
if (score>= 90 && score<=100)
{
 cout << “A” << endl;
}
else if (score>=80 && score<=89)
{
 cout << “B” << endl;
}
else if (score>=70 && score<=79)
{
 cout << “C” << endl;
}
else if (score>=60 && score<=69)
{
 cout << “D” << endl;
}
else if (score>=0 && score<=59)
{
```

```
 cout << "F" << endl;
}
```

# Switch Statements

*Switch case statements* are a substitute for long if statements that compare a variable to several values.




## Syntax:

With the switch statement, the variable name is used once in the opening line. A case keyword is used to provide the possible values of the variable, which is followed by a colon and a set of statements to run if the variable is equal to a corresponding value.

```
switch (n){
     case 1:
          // code to be executed if n = 1;
          break;
     case 2:
          // code to be executed if n = 2;
          break;
     default:
          // code to be executed if n doesn't match any cases
}
```

**Example:**
Write a switch case statement that prints the range corresponding to each letter grade value (A: 90-100, B: 80-89, C: 70-79, D: 60-69, F: 0-59).

```
char grade = 'A';
switch (grade){
     case 'A':
          cout << "90-100" << endl;
          break;
     case 'B':
          cout << "80-89" << endl;
          break;
     case 'C':
          cout << "70-79" << endl;
          break;
     case 'D':
          cout << "60-69" << endl;
          break;
     case 'F':
          cout << "0-59" << endl;
          break;
     default:
          cout << "The letter grade is not recognized" << endl;
}
```

## Important notes to keep in mind while using switch statements :

1. The expression provided in the switch should result in a constant value otherwise it would not be valid.
    a. `switch(num)` //allowed (num is an integer variable)
    b. `switch('a')`//allowed (takes the ASCII Value)
    c. `switch(a+b)` //allowed,where a and b are int variable, which are defined earlier
2. The **break** statement is used inside the switch to terminate a statement sequence. When a break statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
3. The break statement is optional. If omitted, execution will continue on into the next case. The flow of control will fall through to subsequent cases until a break is reached.
4. The **default** statement is optional. Even if the switch case statement does not have a default statement, it would run without any problem.

# Relational Operators

A *relational operator* is a feature of a programming language that tests or defines some kind of relation between two entities. These include numerical equality (e.g., 5 == 5) and inequalities (e.g., 4 ≥ 3). Relational operators will evaluate to either True or False based on whether the relation between the two operands holds or not. When two variables or values are compared using a relational operator, the resulting expression is an example of a *boolean condition* that can be used to create branches in the execution of the program. Below is a table with each relational operator's C++ symbol, definition, and an example of its execution.

| | | |
|---|---|---|
| **>** | greater than | 5 > 4 is TRUE |
| **<** | less than | 4 < 5 is TRUE |
| **>=** | greater than or equal | 4 >= 4 is TRUE |
| **<=** | less than or equal | 3 <= 4 is TRUE |
| **==** | equal to | 5 == 5 is TRUE |
| **!=** | not equal to | 5 != 4 is TRUE |

# Logical Operators

Logical operators are used to compare the results of two or more conditional statements, allowing you to combine relational operators to create more complex comparisons. Similar to relational operators, logical operators will evaluate to True or False based on whether the given rule holds for the operands. Below are some examples of logical operators and their definitions.

- **`&&`** (AND) returns true if and only if both operands are true
- **`||`** (OR) returns true if one or both operands are true
- **`!`** (NOT) returns true if an operand is false and false if the operand is true

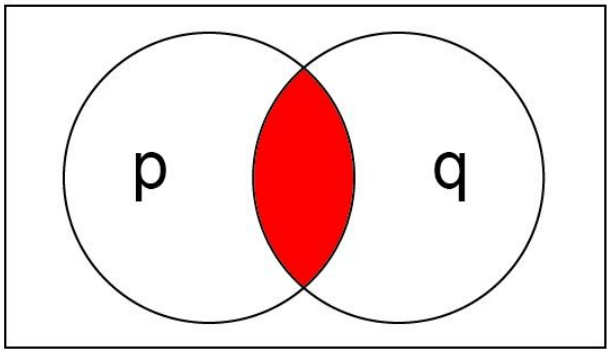
## Truth Tables

Every logical operator will have a corresponding *truth table*, which specifies the output that will be produced by that operator on any given set of valid inputs. Below are examples of truth tables for each of the logical operators specified above.

## AND:

These operators return true if and only if both operands are True. This can be visualized as a venn diagram where the circles are overlapping.

| p | q | p && q |
|---|---|---|
| True | True | True |
| True | False | False |
| False | True | False |
| False | False | False |




## OR:

These operators return True if one or both of the operands are True. This can be visualized as the region of a venn diagram encapsulated by both circles.

| p | q | p \|\| q |
|---|---|---|
| True | True | True |
| True | False | True |
| False | True | True |
| False | False | False |