Logic Handling in any programming language, If/Else

# CS 10A – PROGRAMMING LOGIC PART 1

Allen Zhao - SRJC CS 10A

# Making Decisions

- Computers have to be able to make decisions in order for them to be useful.
- Computers do logic at a binary level, but in higher level languages like C/C++, logic can be more explicitly stated by users for the ease of convenience.
- The if/else statement is the basic building block for all decision making.
- Else if statements can be added to increase the number of possible decisions within a single logic block.

# If/Else If/Else

```
int main()
{
        if(/* logic statement goes here*/)
                // This line executes if true. Otherwise, this statement is skipped.
        else if(/* logic goes here */)          // A group of commands following a logic statement can be referred to as a block.
        {
                // If the else if is true, then all items within the brackets are executed. Without brackets, only the first line
                // following the logic statement is executed (or else completely breaks your program).
        }
        else
                // Executes if none of the above statements are true

        // Else if and Else statement blocks are optional. Statement blocks that turn out false are skipped.
        // You can use just an if statement or as many else if statements as needed
        // If a statement is true in multiple places, then only the first true statement block is executed. The rest are skipped.
        return 0;
}
```

# Designing Logic Blocks

```
int main()
{
        if(0)           // Every if statement starts a new logic block
                        cout << “No output” << endl;

        if(0)           // Choose carefully on whether or not if statements should be connected to each other via else statements
                        cout << “No output” << endl;
        else if(1)      // Else if statements within one block render all logic statements in that block mutually exclusive
                        cout << “Output” << endl;
        else if(0)      // One logic block executes on one set of statements or nothing at all
                        cout << “No output” << endl;

        if(1)           // Multiple if statements mean multiple sets of logic can execute simultaneously in one run
                        cout << “Output” << endl;
        else            // Else statements guarantee that logic block to always execute on one set of commands
                        cout << “No output” << endl;

        return 0;
}
```

# Logic Statements and Analysis

- In C/C++, any non-zero value is considered True. Only 0 is considered False.
- You can use any variable type to stand as a Boolean (as it is in C). In C++, you can use the type bool. Boolean variables are always defined as true or false.
- Logic can be evaluated using comparative operators:

| Symbol | Definition | Symbol | Definition |
|---|---|---|---|
| < | Less than | <= | Less than or equal to |
| > | Greater than | >= | Greater than or equal to |
| == | Equal to | != | Not equal to |

# Logic Statements in C/C++

```
int i0 = 5, i1 = 9, x;
string str0 = “Moon”, str1 = “moon”;   // strings are case-sensitive
int main()
{
        x = (i0 > i1);          // After this, x = 0
        i0 < i1;                // Statement returns 1, does nothing
        str0 == str1;           // Statement returns 0, does nothing
        x = str0 != str1;       // Parentheses not necessary, x = 1

        return 0;
}
```

# Logical Operators vs. Bitwise Operators

- ~, &, |, and ^ are all known as bitwise operators as they are intended to be used for Boolean Algebra.
- Their logic equivalents are !, &&, and ||, respectively. There is no equivalent for XOR. They're specifically for handling Boolean LOGIC rather than Algebra.
- Make sure you use the right symbol for a given context! Since any non-zero value is considered true, using the bitwise operators will often return the wrong result.

# Combining Logic Statements

```
int i0 = 3, i1 = 4;
bool ex_bool0 = true, ex_bool1 = false, ex_bool2;
int main()
{
        if(i0 <  i1 && ex_bool0)
                // This line should execute
        if(i0 >= i1 || !ex_bool1)
                // This line should also execute. If this was an else if statement,
                // then it would be linked to the above if statement and skipped
        ex_bool2 = i0 && i1;            // true with &&, false with &
        return 0;
}
```

# Full Program Using Logic

## Program

```
int i0 = 7, i1 = 8;
bool ex_bool0 = true, ex_bool1 = false;
int main()
{
        if(i0 < i1)
                cout << “i0 is less than i1\n”;
        else
                cout << “Skipped\n”;

        if(ex_bool1)
                cout << “Skipped\n”;
        else if(ex_bool0 && i0 < i1)
                cout << “Statement is true\n”;
        return 0;
}
```

## Console

- ./a.exe

```
i0 is less than i1
Statement is true
```

# Tips on Creating Logic Conditions

- You can use any sort of variables to create logic conditions, not just bool. You can even use numbers in place of an actual logic condition. Remember that only 0 is false.
- int, float, and double can all be directly compared using the comparative operators.
- char can be directly compared just like numbers. Remember to use single quotes to state characters.
  - if(ch0 > ‘a’)  // is ASCII of ch0 is greater than ASCII of a?
- string can only use ==. Remember that strings need double quotes to be used in C/C++.
  - if(rating == “PG-13”)  // is the string variable rating “PG-13”?

# Tips on Minimizing Logic

- Logic statements can be combined or nested in order to minimize the size of the code.
- Recommended practices are using OR to combine multiple logic statements that lead to the same actions.
  - if(bool0 || bool1 || bool2 && bool3 || x > y && bool4)
    - This example combines 4 unique sets of conditions that, if any one of them were true, the command(s) below would execute.
- Condition statements can be nested: logic statements used within logic statements. Similar to using && but has the advantage of allowing inclusion of intermediary steps.

# Minimizing Logic

```
int x = 0;      char ch0 = ‘a’;                 string str0 = “check”;
int main()
{
                if(x < 1)
                {
                                // You can insert additional commands here to take place if it’s not dependent on the second if statement
                                if(ch0 == ‘a’)
                                {
                                                // This second internal if statement is also known as a nested if statement
                                }
                                // Commands here take place regardless of the above condition, but only after the if statement has been checked
                }
                else if(str0 == “CHECK” || x > 1 || ch0 > ‘a’ && x == 1)
                {
                                // There are three conditions in this else if statement. Only one needs to be true to trigger the events here.
                }
                return 0;
}
```

# Logic Statements Are Not Math Statements!

```
int main()
{
	int x;
	cin >> x;		// You have to be very specific when writing logic statements for value ranges
	if(10 < x < 20)		// This compiles, but is functionally incorrect due to syntax rules
	{
		// When read left to right, the program sees 10 < x first, which becomes 1 or 0
		// Because both are less than 20, the above statement will ALWAYS be True
	}
	if(x > 10 && x < 20)		// This is the correct version of the above
	{
		// Each logic statement in assessed independently, then compared with &&
	}
	return 0;
}
```

# The Conditional Operator

- In the case of very simplistic if/else statements, use of a conditional operator (?) can condense four lines into one.
- Syntax is as follows:
  - Condition ? Execute this if true : execute this if false;
  - The line to execute can be a command or a value to return
    - x > y ? y++ : x++;  // If x > y, do y++. Otherwise, do x++.
    - int a = (x > y) ? 0 : 3;  // If x > y, assign 0 to a. Otherwise, a = 3.
  - Parentheses will make condition statements easier to read, but are usually unnecessary since ? has a low precedence.
  - Conditionals can be nested to be as complicated as need be.

# Using the Conditional Operator

## Program

```
int x = 1, y = 2, i;
int main()
{
        i = (x == 2) ? ++x : ++y;
        cout << i << endl;

        y == 2 ?
                (cout << “y is 2”) :
                (cout << “y is not 2”) ;
        // Semicolons are not allowed within the true
        // or false blocks of the conditional
        // y is 3 because ++y acts as y = y+1
        cout << endl;
        return 0;
}
```

## Console

- ./a.exe

```
3
y is not 2
```

# Nested Conditional Operators

```
int main()
{
        x <= y ? (x < y ? x++ : x--) : y++;
        // You can nest as many or as few statements as you want, in any position.
        // This can mimic the effects of using else if statements or other combinational logic.
        /* The above conditional is equivalent to the code block below
           if(x == y)
                   x--;
           else if(x < y)
                   x++;
           else
                   y++;
        */
        return 0;
}
```