DISCUSSION SECTION

WEEK 2

EXPRESSIONS, CONDITIONALS, LOOPS, & C++ PROGRAM STRUCTURE

C vs. C++

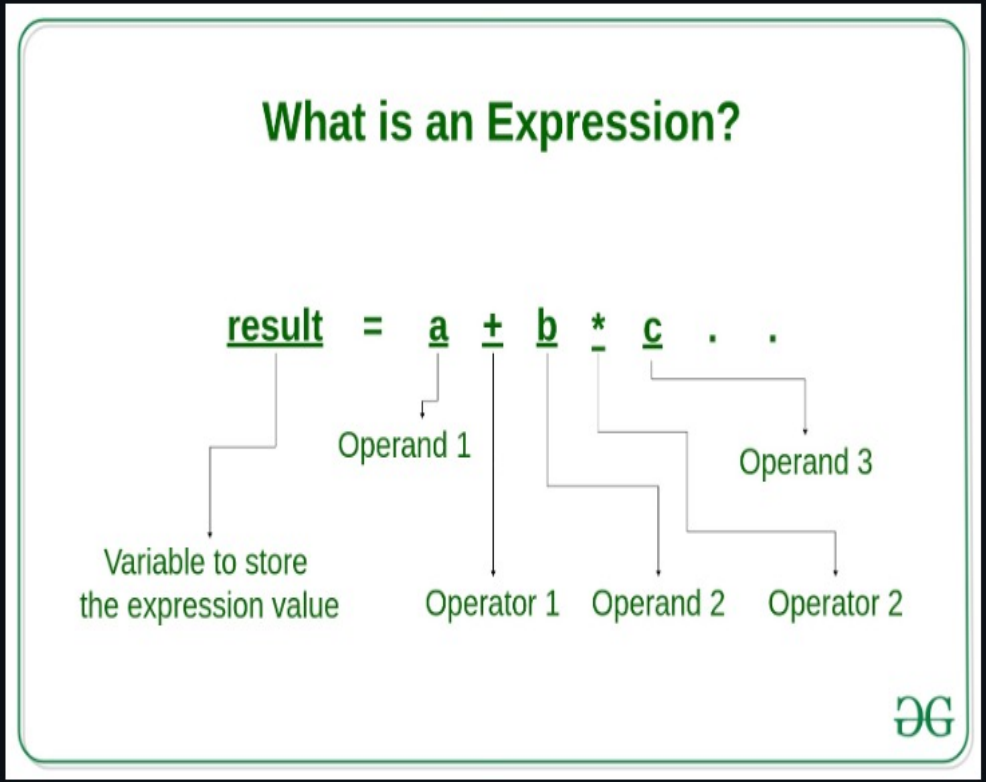
- ✓ C++, also known as “C with Classes,” is an extension of C with object-oriented programming support
- ✓ You can write C code in your C++ program; C++ cannot be written in C programs
- ✓ C has approx. 32 keywords, C++ has 95 keywords
  - You don’t need to memorize them; you will get familiar with them as you write more C++ code
- ✓ You will be working in C++ in future classes; probably C as well

Refresher…

- ✓ C++ is a strictly typed language
  - You must explicitly declare the type of a variable (or function) when creating it
- ✓ All C++ programs must have a main function.

  ALWAYS!
  - By C++ Standards, the main function cannot be called within a program.
- ✓ All statements in C++ must end with a semicolon
- ✓ C++ is not whitespace sensitive, but is case sensitive

Remember your algebra and start thinking logically - let's talk about expressions as well as their relation to conditionals and program structure!



> An expression is a combination of operators, constants and variables. An expression may consist of one or more operands, and zero or more operators to produce a value.

via GeeksForGeeks.org

Using Boolean Expressions for Conditionals

Knowing that expressions can come in a variety of different forms, let's think about how we can utilize them for conditional statements. So, let's take a look at Boolean expressions.

First, what is a Boolean expression? A **Boolean expression** is a specific kind of expression whose value when evaluated results in **true** or **false** (sometimes **1** or **0**, respectively).

Using Boolean Expressions for Conditionals

Knowing that expressions can come in a variety of different forms, let's think about how we can utilize them for conditional statements. So, let's take a look at Boolean expressions.

First, what is a Boolean expression? A **Boolean expression** is a specific kind of expression whose value when evaluated results in **true** or **false** (sometimes **1** or **0**, respectively).

For example, the code snippet below assigns a Boolean variable in C++ to true or false depending on whether or not the value of **x** is greater than or equal to 25.

```
bool myBooleanExpression = (x >= 25);
```

Another example: the code snippet below assigns a Boolean variable in C++ to true or false depending on whether or not the value of **x** is equal to the square root of **y**.

```
bool myBooleanExpression = (x == sqrt(y));
```

Note: **==** is an equality operator that is used to compare right side with left side. Returns true/false.

= (single equal sign) is an assignment operator used to assign values to variables.

Understand when to use either!

What would be the value of our Boolean variables?

int x = 25;

int y = 62;

1. *bool variable1 = (x > 0);*
2. *bool variable2 = (y % 2 == 0);*

   Remember that % gets the remainder after an integer division
3. *bool variable3 = (x == y);*
4. *bool variable4 = (x % 10 != 0);*

Compound Boolean Expressions

We can certainly combine two or more individual Boolean expressions into a single compound expression, using *logical operators*!

But first, we must understand the Truth table.

Name	Symbol	Description
Logical AND	&&	Returns true only if all the operands are true or non-zero
Logical OR	||	Returns true if either of the operands is true or non-zero

A	B	$A\&\&B$	$A\|\|B$
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

Examples:

```
bool myExpression = (x >= minVal && x <= maxVal && y >= minVal && y <= maxVal);
bool myExpression = (x > 0 && y < 10) || (z == 5 && w != 0);
```

Exercise 1 (5 minutes):

Consider a scenario where a company offers discounts based on the following criteria:

1. If the purchase amount is greater than $100 and the customer is at least 50 years old, they get a discount.
2. If the purchase amount is between $50 and $100 (inclusive) and the customer is less than 50 years old, they get a discount.

Write a Boolean expression that evaluates to **true** if a customer is eligible for discount based on the given criteria; and evaluates to **false** otherwise.

***You can use two variables of your choosing for age and purchase amount.

Answer

```
bool discountEligible = (amt > 100 && age >= 50) || (amt >= 50 && amt <= 100 && age < 50);
```

Now then, let's try applying what we've learned to a conditional statement. Conditional statements appear in most languages, often appearing and performing very similarly. In C++, the general syntax of a conditional statement is as follows:

```
if (boolean expression) {
    // Do something!
}
```

That said, there are a variety of different conditional statements. Here's a few more!

If Else

```
if (boolean expression) {
    // Do something if!
} else {
    // Do something else!
}
```

Short Hand If Else

```
bool variable = (boolean expression) ? <line to run if true> : <line to run if false>;
```

Else If

```
if (boolean expression) {
    // Do something if!
} else if (next boolean expression) {
    // Do something else if!
} else {
    // Do something else!
}
```

Switch (Similar to a chained else if but works directly with the value of any given expression rather than true/false in a boolean expression specifically.)

```
switch (expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

Example

```
int day = 4;
switch (day) {
  case 1:
    cout << "Monday";
    break;
  case 2:
    cout << "Tuesday";
    break;
  case 3:
    cout << "Wednesday";
    break;
  case 4:
    cout << "Thursday";
    break;
  case 5:
    cout << "Friday";
    break;
  case 6:
    cout << "Saturday";
    break;
  case 7:
    cout << "Sunday";
    break;
}
// Outputs "Thursday" (day 4)
```

This is how it works:

- The `switch` expression is evaluated once
- The value of the expression is compared with the values of each `case`
- If there is a match, the associated block of code is executed

The break Keyword

When C++ reaches a `break` keyword, it breaks out of the switch block.

This will stop the execution of more code and case testing inside the block.

When a match is found, and the job is done, it's time for a break. There is no need for more testing.

A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.

The default Keyword

The `default` keyword specifies some code to run if there is no case match:

Exercise 2 (10 minutes):

1. Convert the *switch* statement on the previous slide to conditional statements (using **if**, **else if**, & **else**).

2. Write a program that takes in an integer between 1 and 12 (inclusive), and prints out the month corresponding to that number. Use a **switch** statement.

    - Print “Invalid month” if the integer input is not between 1 and 12. *(Hint: this is the default case!)*

```cpp
#include <iostream>

int main() {

    int day = 4;

    if(day == 1) {
        std::cout << "Monday";
    } else if (day == 2) {
        std::cout << "Tuesday";
    } else if(day == 3) {
        std::cout << "Wednesday";
    } else if(day == 4) {
        std::cout << "Thursday";
    } else if(day == 5) {
        std::cout << "Friday";
    } else if(day == 6) {
        std::cout << "Saturday";
    } else if (day == 7) {
        std::cout << "Sunday";
    } else { // Case where "day" is not a number between 1 and 7
        std::cout << "Not a valid day";
    }
```

main.cpp

Users > yemifasina > Desktop > main.cpp > main()

```cpp
#include <iostream>

int main() {

    int month;
    std::cin >> month;

    switch (month) {
        case 1:
            std::cout << "Jan";
            break;
        case 2:
            std::cout << "Feb";
            break;
        case 3:
            std::cout << "Mar";
            break;
        case 4:
            std::cout << "Apr";
            break;
        // After making cases for 5 through 12..then
        // you have the default case:
        default:
            std::cout << "Invalid month";
            // A break is not required here,
            // but is good practice to have it
            // for consistency & future modification
    }


```

Loops..

```
for(init; boolean expression; update) {
    // some code to be executed
}
```

- init is usually the declaration of a variable to control the loop
- boolean exp. is the condition for executing the code block
- update modifies the variable in init

Range-based For Loop (or For-each loop)

```
for(type variableName : rangeExpression) {
        // some code
}
```

Example:

```
for(char letter : "Programming") {
        std::cout << letter;
}
```

- Note: You may get a warning message using the range-based loop if your version of C++ is prior to C++11. In that case, you can add a c++11 flag to your compilation as in:

g++ -std=c++11 main.cpp -o main

....or better still, don't use range-based loop at all

Nested Loop

✓ is a loop within a loop

Example Scenario:

Suppose you're a teacher in a classroom with 3 rows of desks, and each row has 7 students. You want to say "Hello" to every student in each row.

```
for(int numberOfRows = 1; numberOfRows <= 3; numberOfRows++ ) {
        for(int numStudentsPerRow = 1; numStudentsPerRow <= 7; numStudentsPerRow++) {
                std::cout << "Hello" << std::endl;
        }
}
```

• In this case, we are able to visit all students in a row (inner loop), before moving on to another row (outer loop).

Exercise 3 (15 minutes)

Write a program to print a 2x3 grid of numbers starting from 1.

Expected Output:

```
1 2 3
4 5 6
```

Answer

```
#include <iostream>

int main() {
    int number = 1;
    for (int rows = 1; rows <= 2; rows++) {   // 2 rows so outer loop runs twice
        for (int cols = 1; cols <= 3; cols++) {    // 3 numbers per row
            std::cout << number << " ";
            number++;
        }
        std::cout << std::endl;  // Move to the next line after each row has printed
    }
    return 0;
}
```

Output:

1 2 3
4 5 6