

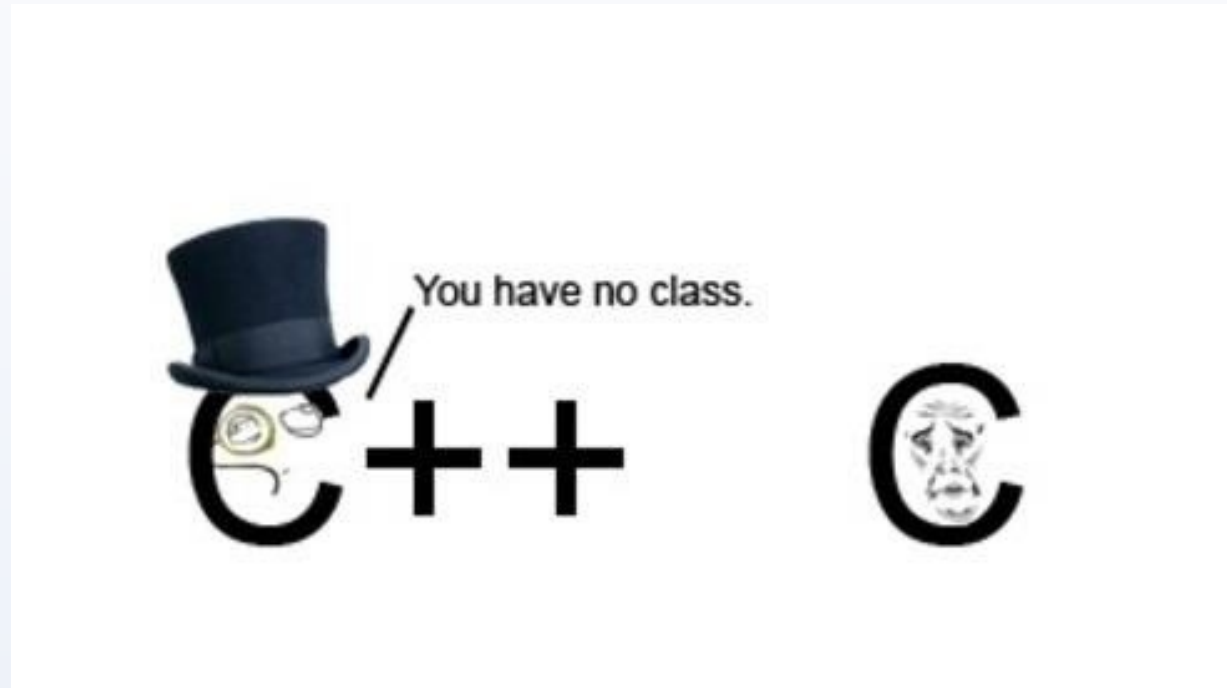
Programming with C and C++

First Steps to Code

- Part 2 -

@carlosjaimebh

C or C++ ? (Or any language)



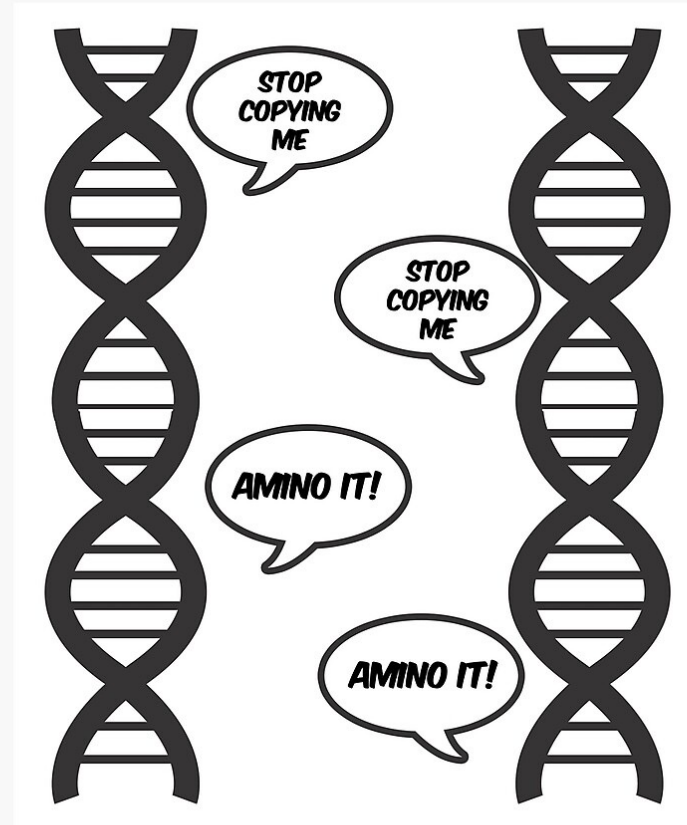
Depending of the oriented programming

Programming paradigm and oriented development is a process (scientific, technical and engineering).

- You need to see the requirements (or expected outcomes)
- **Programability (Remember, the idea is that you're a scientists, so your time is to make science)**
- Skills
- Platforms (not only computer machines), runtime.
- Likes and comfort!
 - **Not always, the popular is good**

Special Recommendations about Copying

- Programming in real life, copying is strongly encouraged. (The idea is not to reinvent the wheel (and not waste time getting it wrong)
 - Copying saves time;
 - Copying avoids typing mistakes;
 - Copying allows you to focus on your new programming challenges.



Introducing C++

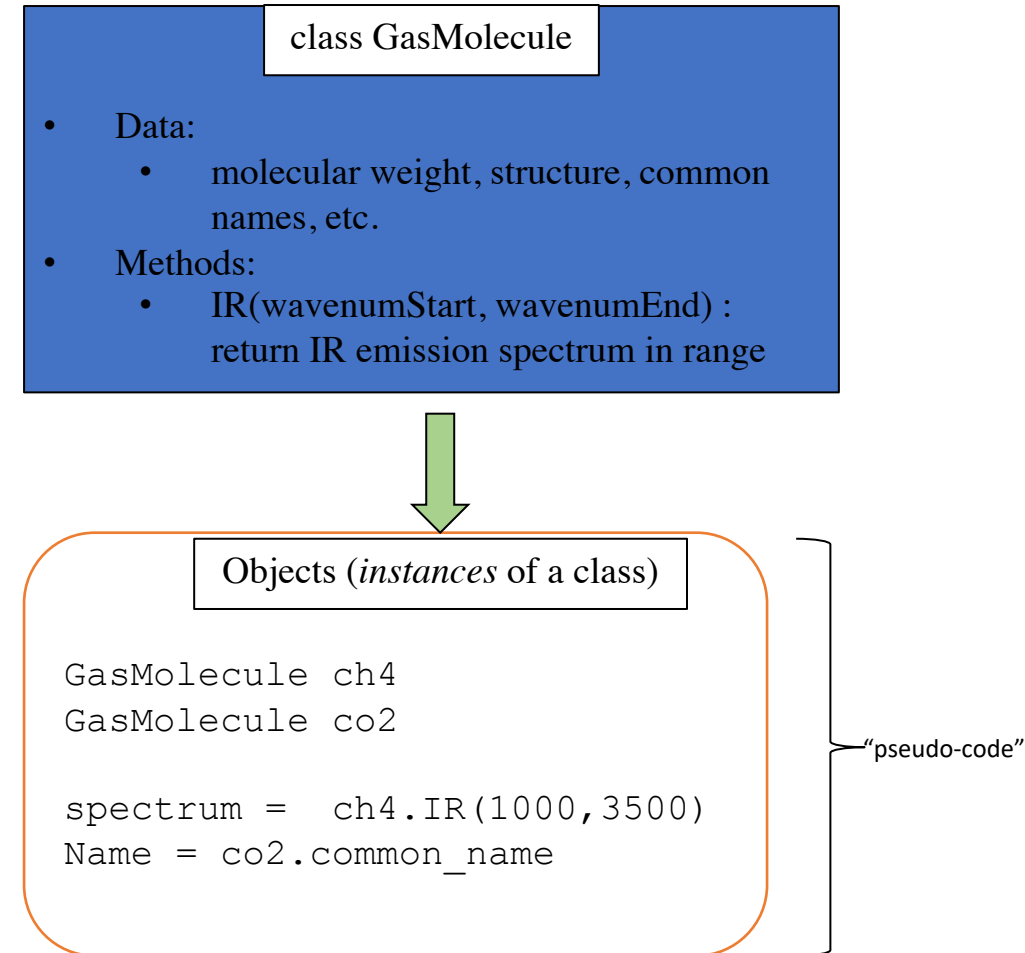
- The C++ programming language (Stroustrup, 1988) evolved from C (Ritchie 1972) and is emerging as the standard in software development.
- C++ facilitates a structured and disciplined approach to computer programming called object-oriented programming.
- C++ is a general-purpose programming language with a bias towards systems programming that:
 - supports data abstraction
 - supports object-oriented programming
 - supports generic programming.
- The C++ Foundation (<https://isocpp.org/>) supports the C++ community development.



Bjarne Stroustrup (<https://www.stroustrup.com/>)

Object-oriented Programming

- Object-oriented programming (OOP) seeks to define a program in terms of the *things* in the problem (files, molecules, buildings, cars, people, etc.), what they need, and what they can do.

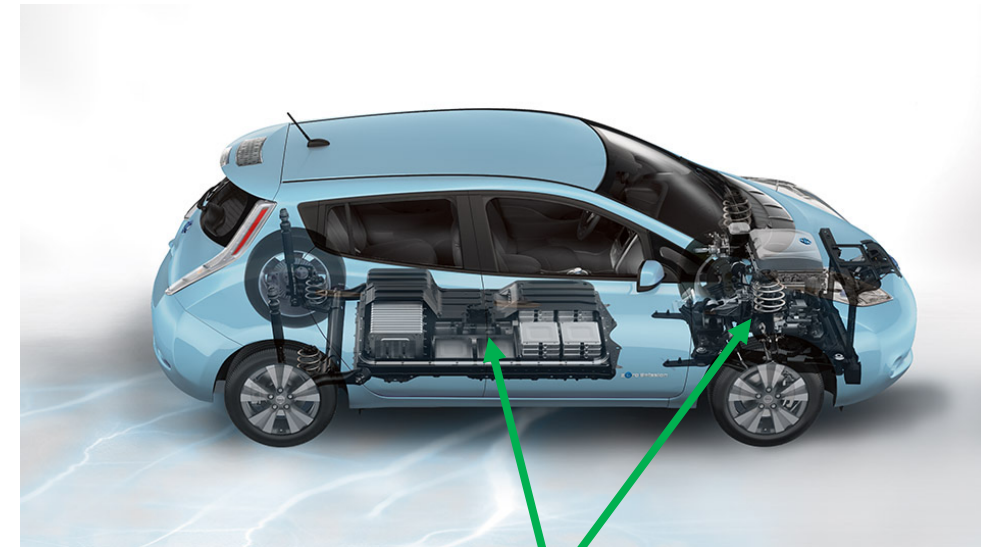


Object-oriented Programming

- OOP defines *classes* to represent these things.
- Classes can contain data and methods (internal functions).
- Classes control access to internal data and methods. A public interface is used by external code when using the class.
- This is a highly effective way of modeling real world problems inside of a computer program.

“Class Car”

public interface



More information in: <https://www.bu.edu>

private data and methods

Editing, Compiling and Executing a Simple Program

```
//Program to add two integers typed by user at keyboard
#include <iostream>
using namespace std;
int main()
{
    int a, b, total;
    cout << "Enter integers to be added: " << endl;
    cin >> a >> b;
    total = a + b;
    cout << "The sum is " << total << endl;
    return 0;
}
```

```
// Comment statements, which are ignored by
computer but inform reader
#include <header file name>
int main()
{
    declaration of variables;
    statements;

    return 0;
}
```

To compile in GNU CPP:

```
g++ SimpleAdd.cpp -o SimpleAdd
```

To execute:

```
./SimpleAdd
```

(Also, you can use the extension .cc)

A Hello World Example

(To see Compiler Error Messages)

```
// Hello world example in C++
#include <iostream>
using namespace std;
int main()
{
    for (int i=1; i<=10; i++) {
        cout << i << " Hello World,
Folks" << endl ;
        if ( i == 7) {
            cout << " that was lucky!\n" <<
endl ;

            } else {
                cout << endl ;
            }
        }
    }
}
```

Compiling using the command:

```
g++ -Wall Hello_1.cc -o Hello_1
```

What happens if:

- Omitting a quote character (")
- Inserting an extra quota character
- Omitting semicolon (;) and the end of the line.
- Adding a left-brace ({) or right-brace (}

How many types of error can you get the compiler to produce?
Make a list.

- Replace (i==7) by (i=7)

The flag -Wall means “Warnings (all)”

Suggested lesson: always compile with Warnings switched on.

Another (simple) Helloworld

(To try make)

```
// Hello World Version 2
#include <iostream>
int main() {
    std::cout << "Hello World" <<
std::endl;
    return 0;
}
```

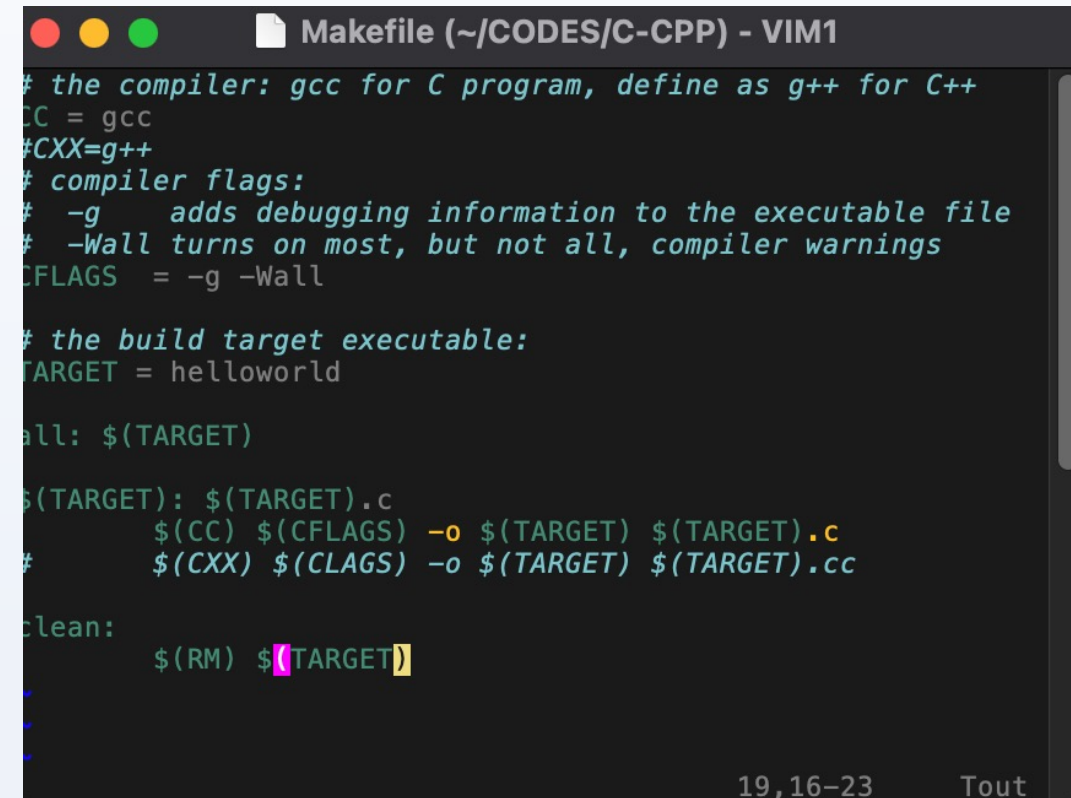
- Compiling using

make HelloWorld_2

What happens?

(Makefiles)

- The `make` allows to define personal rules to compiling your C / C++ code using the warning flags automatically. (Also, you can add other languages primitives)
- Make is a build automation tool that automatically builds executable programs and libraries from source code by reading files called Makefiles which specify how to derive the target program.



```
Makefile (~/CODES/C-CPP) - VIM1
# the compiler: gcc for C program, define as g++ for C++
CC = gcc
#CXX=g++
# compiler flags:
# -g    adds debugging information to the executable file
# -Wall turns on most, but not all, compiler warnings
CFLAGS = -g -Wall

# the build target executable:
TARGET = helloworld

all: $(TARGET)

$(TARGET): $(TARGET).c
    $(CC) $(CFLAGS) -o $(TARGET) $(TARGET).c
#    $(CXX) $(CLAGS) -o $(TARGET) $(TARGET).cc

clean:
    $(RM) $(TARGET)
```

19,16-23 Tout

Basic Syntax

- C++ syntax is very similar to C, Java, or C#. Here's a few things up front and we'll cover more as we go along.
- Curly braces are used to denote a code block (like the main() function):

```
{ ... some code ... }
```

- Statements end with a semicolon:

```
int a ;  
a = 1 + 3 ;
```

- Comments are marked for a single line with a `//` or for multilines with a pair of `/*` and `*/`:

```
// this is a comment.  
/* everything in here  
   is a comment */
```

- Variables can be declared at any time in a code block.

```
void my_function() {  
    int a ;  
    a=1 ;  
    int b ;  
}
```

Confusing syntax

```
int a, b;
```

```
int c = a * b;
```

```
int* d = &a;
```

```
int e = *d;
```

```
int& f = a;
```

*** means**

- multiplication, *or*
- pointer, *or*
- dereference pointer

& means

- get address of, *or*
- reference

Same symbol, different meanings!

Pass by X

pass by value

pass by pointer

pass by reference

```
void f(int a, int* b, int& c)
{
    // changes to a are NOT reflected outside the function
    // changes to b and c ARE reflected outside the function
}
```

DOES make a copy

does NOT make a copy

```
main()
{
    int a, b, c;
    f(a, &b, c);
}
```

PBP and PBR are *different* syntax for the *same* functionality

Variables and Constants

- Programs need a way to store the data they use. Variables and constants offer various ways to represent and manipulate data. Constants, as the name suggests, have fixed values. Variables, on the other hand, hold values that can be assigned and changed as the program executes.
- Every variable and constant has an associated type, which defines the set of values that can be legally stored in it. Variables can be conveniently divided into integer, floating point, character and boolean types for representing integer (whole) numbers, floating point numbers (real numbers with a decimal point), the ASCII character set (for example 'a', 'b', 'A') and the boolean set (true or false) respectively.

Example of variable declaration:

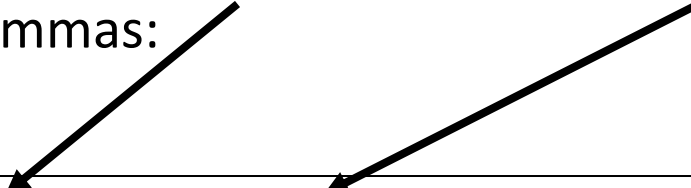
```
int count;  
float length;  
char firstInitial;  
bool switched_on;
```

int	to store a positive or negative integer (whole) number
float	to store a real (floating point) number
bool	to store the logical values true or false
char	to store one of 256 character (text) values

(To see in detail after)

Functions (First view)

- Functions are sections of code that are called from other code. Functions always have a return **argument type**, a **function name**, and then a list of arguments separated by commas:



```
int add(int x, int y) {  
    int z = x + y ;  
    return z ;  
}
```

```
// No arguments? Still need ():  
void my_function() {  
    /* do something...  
       but a void value means the  
       return statement can be skipped.*/  
}
```

- A *void* type means the function does not return a value.

- Variables are declared with a type and a name:

```
// Specify the type  
int x = 100 ;  
float y ;  
vector<string> vec ;  
// Sometimes types can be inferred  
auto z = x ;
```

Operators

- A sampling of arithmetic operators:
 - **Arithmetic:** + - * / % ++ --
 - **Logical:** && (AND) || (OR) !(NOT)
 - **Comparison:** == > < >= <= !=
- Sometimes these can have special meanings beyond arithmetic, for example the “+” is used to concatenate strings.
- What happens when a syntax error is made?
 - The compiler will complain and refuse to compile the file.
 - The error message *usually* directs you to the error but sometimes the error occurs before the compiler discovers syntax errors, so you hunt a little bit.

Precedence and nesting parentheses

- The use of parentheses (brackets) is advisable to ensure the correct evaluation of complex expressions. Here are some examples:

$4 + 2 * 3$ equals 10

$(4 + 2) * 3$ equals 18

$-3 * 4$ equals -12

$4 * -3$ equals -12 (but it's safer to use parentheses: $4 * (-3)$)

$0.5 (a+b)$ illegal (missing multiplication operator)

$(a+b) / 2$ equals the average value of a and b if they are of type float

Built-in (aka primitive or intrinsic) Types

- “primitive” or “intrinsic” means these types are not objects
- Here are the most commonly used types.
- Note: The exact bit ranges here are **platform and compiler dependent!**
 - Typical usage with PCs, Macs, Linux, etc. use these values
 - Variations from this table are found in specialized applications like embedded system processors.

Name	Name	Value
char	unsigned char	8-bit integer
short	unsigned short	16-bit integer
int	unsigned int	32-bit integer
long	unsigned long	64-bit integer
bool		true or false

Name	Value
float	32-bit floating point
double	64-bit floating point
long long	128-bit integer
long double	128-bit floating point

Need to be sure of integer sizes?

- In the same spirit as using *integer(kind=8)* type notation in Fortran, there are type definitions that exactly specify exactly the bits used. These were added in C++11.
- These can be useful if you are planning to port code across CPU architectures (ex. Intel 64-bit CPUs to a 32-bit ARM on an embedded board) or when doing particular types of integer math.
- For a full list and description see: <http://www.cplusplus.com/reference/cstdint/>

#include <cstdint>

Name	Name	Value
int8_t	uint8_t	8-bit integer
int16_t	uint16_t	16-bit integer
int32_t	uint32_t	32-bit integer
int64_t	uint64_t	64-bit integer

Reference and Pointer Variables

The object *hello* occupies some computer memory.

```
string hello = "Hello";
```

The asterisk indicates that *hello_ptr* is a pointer to a string. *hello_ptr* variable is assigned the memory address of object *hello* which is accessed with the "&" syntax.

```
string *hello_ptr = &hello;
```

The & here indicates that *hello_ref* is a reference to a string. The *hello_ref* variable is assigned the memory address of object *hello* automatically.

```
string &hello_ref = hello;
```

- Variable and object values are stored in particular locations in the computer's memory.
- Reference and pointer variables **store the memory location of other variables.**
- Pointers are found in C. References are a C++ variation that makes pointers easier and safer to use.
- More on this topic later in the tutorial.

Type Casting (1/2)

- C++ is strongly typed. It will auto-convert a variable of one type to another in a limited fashion: if it will not change the value.

```
short x = 1 ;  
int y = x ;    // OK  
short z = y ;  // NO!
```

- Conversions that don't change value: increasing precision (float → double) or integer → floating point of at least the same precision.
- C++ allows for C-style type casting with the syntax: (new type) expression

```
double x = 1.0 ;  
int y = (int) x ;  
float z = (float) (x / y) ;
```

- But since we're doing C++ we'll look at the **4** ways of doing this in C++ next...

Type Casting (2/3)

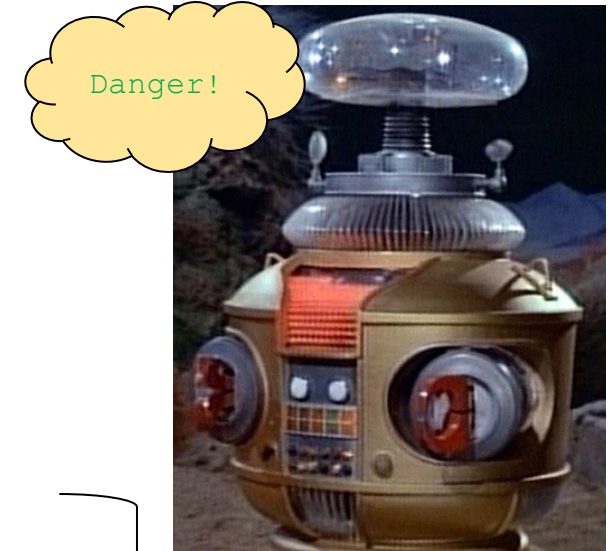
- `static_cast<new type>(expression)`
 - This is exactly equivalent to the C style cast.
 - This identifies a cast **at compile time**.
 - This will allow casts that reduce precision (ex. double → float)
 - ~99% of all your casts in C++ will be of this type.

```
double d = 1234.56 ;  
float f = static_cast<float>(d) ;  
// same as  
float g = (float) d ;
```

- `dynamic_cast<new type>(expression)`
 - Special version where type casting is performed at runtime, only works on reference or pointer type variables.
 - Usually handled automatically by the compiler where needed, rarely done by the programmer.

Type Casting (3/3)

- `const_cast<new type>(expression)`
 - Variables labeled as *const* can't have their value changed.
 - `const_cast` lets the programmer remove or add *const* to reference or pointer type variables.
 - If you need to do this, you probably want to re-think your code.
- `reinterpret_cast<new type>(expression)`
 - Takes the bits in the expression and re-uses them **unconverted** as a new type. Also only works on reference or pointer type variables.
 - Sometimes useful when reading in binary files and extracting parameters.



“unsafe”: the compiler will not protect you here!

The programmer must make sure everything is correct!

A class

Classes are an expanded concept of *data structures*: like data structures, they can contain data members, but they can also contain functions as members.

```
class Date {  
public:  
    enum Month {Jan, Feb, Mar, ...}  
    Date(int year, Month month, int day);  
    int GetDay() const;  
    void SetDay(int day);  
    Date& operator+=(int days);  
private:  
    Month m_month;  
    int m_year, m_day;  
};
```

member
functions
(methods)

member
variables

Classes are defined using either keyword `class` or keyword `struct`

Struct vs. class

- In C++, no difference b/w struct and class (except default public vs. private)
- In C++, struct can have
 - member variables
 - methods
 - public, private, and protected
 - virtual functions
 - etc.
- Rule of thumb:
 - Use struct when member variables are public (just a container)
 - Use class otherwise

Class Example

```
class class_name {  
    access_specifier_1:  
        member1;  
    access_specifier_2:  
        member2;  
    ...  
} object_names;
```

```
class Rectangle {  
    int width, height;  
public:  
    void set_values  
(int,int);  
    int area (void);  
} rect;
```

- Where `class_name` is a valid identifier for the class, `object_names` is an optional list of names for objects of this class. The body of the declaration can contain *members*, which can either be data or function declarations, and optionally *access specifiers*.
- Classes have the same format as plain *data structures*, except that they can also include functions and have these new things called *access specifiers*. An *access specifier* is one of the following three keywords: `private`, `public` or `protected`.
- By default, all members of a class declared with the `class` keyword have private access for all its members. Therefore, any member that is declared before any other *access specifier* has private access automatically
- The [example](#), Declares a class (i.e., a type) called `Rectangle` and an object (i.e., a variable) of this class, called `rect`. This class contains four members: two data members of type `int` (member `width` and member `height`) with *private access* (because `private` is the default access level) and two member functions with *public access*: the functions `set_values` and `area`, of which for now we have only included their declaration, but not their definition.

More information in: <https://www.cplusplus.com/doc/tutorial/classes/>

Basic Input/Output

- C++ uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen, the keyboard or a file.
- A *stream* is an entity where a program can either insert or extract characters to/from. There is no need to know details about the media associated to the stream or any of its internal specifications.

stream	description
<code>cin</code>	standard input stream
<code>cout</code>	standard output stream
<code>cerr</code>	standard error (output) stream
<code>clog</code>	standard logging (output) stream

Standard input (cin)

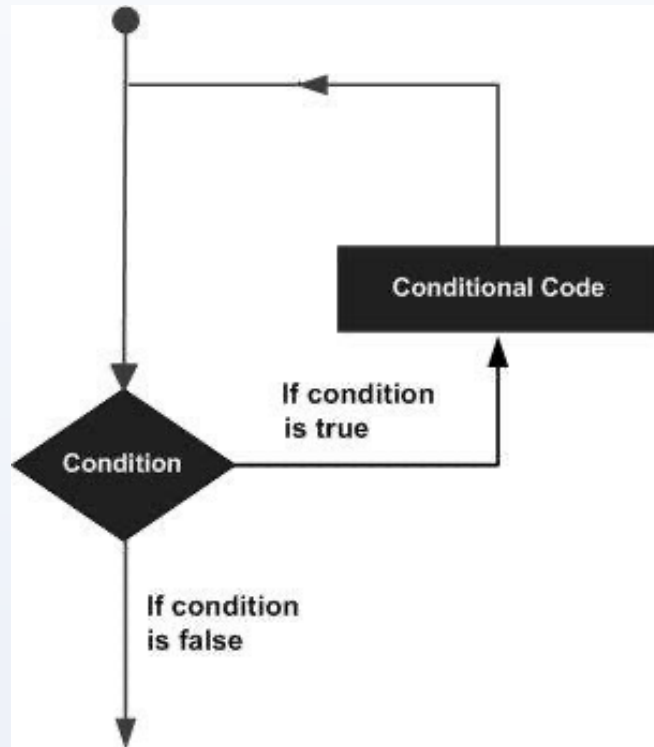
- In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is `cin`.
- For formatted input operations, `cin` is used together with the extraction operator, which is written as `>>` (i.e., two "greater than" signs). This operator is then followed by the variable where the extracted data is stored.

Example:

```
int age;  
cin >> age;
```

The first statement declares a variable of type `int` called `age`, and the second extracts from `cin` a value to be stored in it. This operation makes the program wait for input from `cin`; generally, this means that the program will wait for the user to enter some sequence with the keyboard.

Loops



Loop Type & Description

[while loop](#)

Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.

[for loop](#)

Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.

[do...while loop](#)

Like a 'while' statement, except that it tests the condition at the end of the loop body.

[nested loops](#)

You can use one or more loop inside any another 'while', 'for' or 'do..while' loop.

Control Statement & Description

[break statement](#)

Terminates the **loop** or **switch** statement and transfers execution to the statement immediately following the loop or switch.

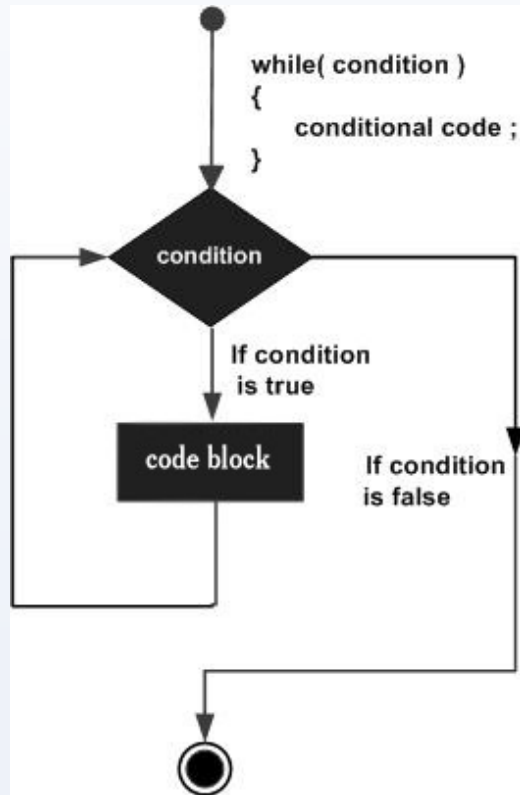
[continue statement](#)

Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

[goto statement](#)

Transfers control to the labeled statement. Though it is not advised to use goto statement in your program.

While

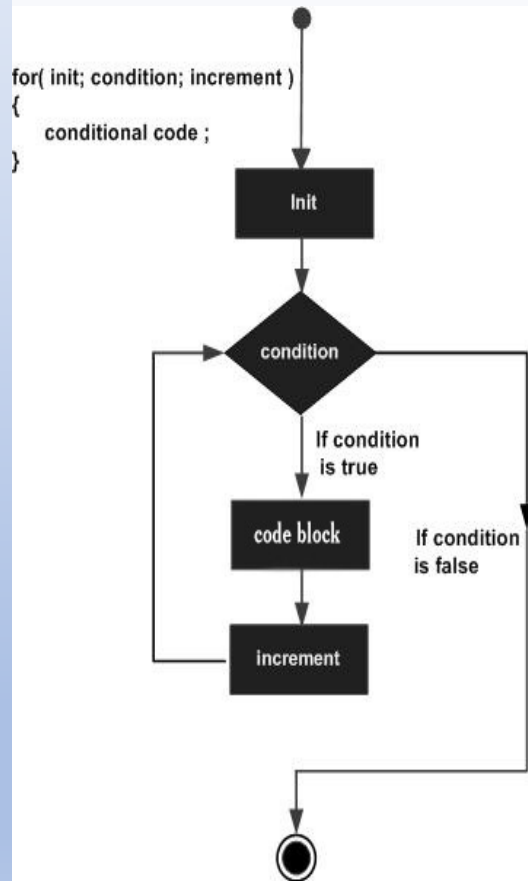


```
while( condition )  
{  
    conditional code ;  
    statement (s) ;  
}
```

Example:

```
#include <iostream>  
using namespace std;  
  
int main ()  
{  
    // Local variable declaration:  
    int a = 10;  
    // while loop execution  
    while( a < 20 ) {  
        cout << "value of a: " << a << endl; a++;  
    }  
    return 0;  
}
```

For



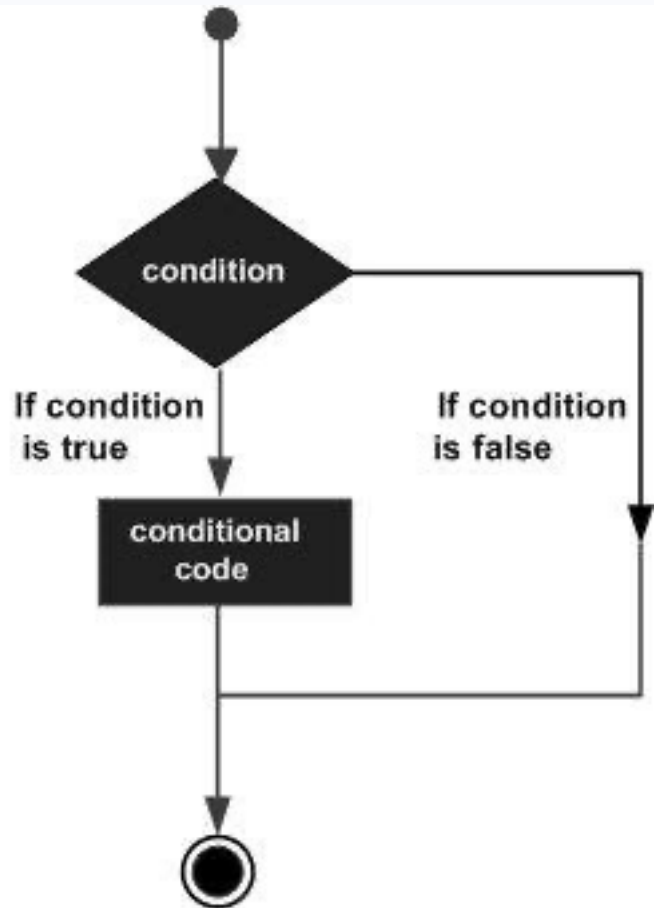
```
for ( init; condition; increment )  
{ statement(s) ;  
}
```

Example:

```
#include <iostream>  
using namespace std;  
int main () { // for loop execution  
for( int a = 10; a < 20; a = a + 1 ) {  
    cout << "value of a: " << a << endl;  
}  
return 0;  
}
```

More about in: https://www.tutorialspoint.com/cplusplus/cpp_loop_types.htm

Decision Making



Statement & Description

[if statement](#)

An 'if' statement consists of a boolean expression followed by one or more statements.

[if...else statement](#)

An 'if' statement can be followed by an optional 'else' statement, which executes when the boolean expression is false.

[switch statement](#)

A 'switch' statement allows a variable to be tested for equality against a list of values.

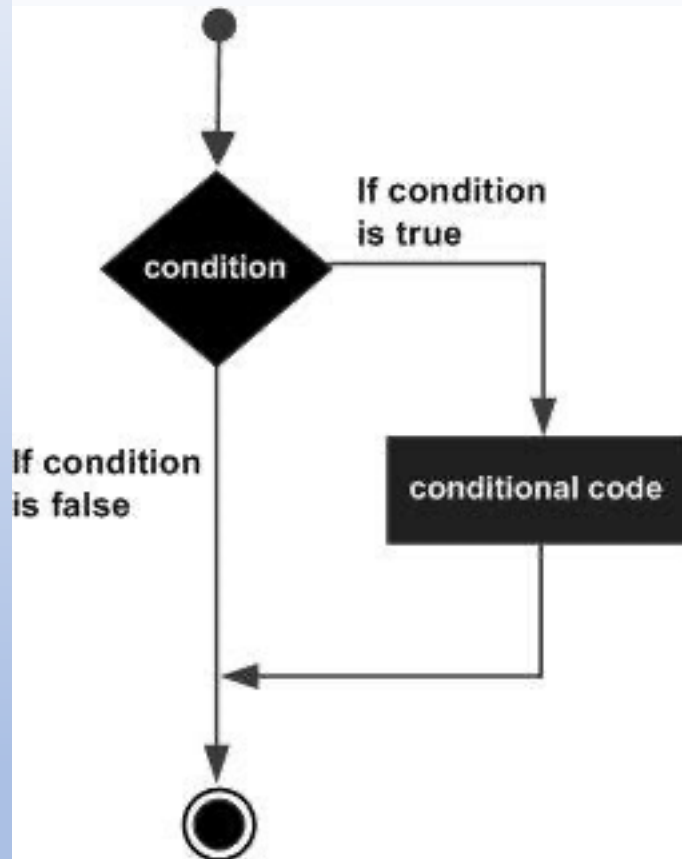
[nested if statements](#)

You can use one 'if' or 'else if' statement inside another 'if' or 'else if' statement(s).

[nested switch statements](#)

You can use one 'switch' statement inside another 'switch' statement(s).

If



```
if(boolean_expression) {  
    // statement(s) will execute if the boolean expression is true  
}
```

Example:

```
#include <iostream>  
using namespace std;  
int main () { // local variable declaration:  
    int a = 10; // check the boolean condition  
    if( a < 20 ) { // if condition is true then print the following  
        cout << "a is less than 20;" << endl;  
    }  
    cout << "value of a is : " << a << endl;  
    return 0;  
}
```

More information in: https://www.tutorialspoint.com/cplusplus/cpp_decision_making.htm

Now, Practice Time

1. Write a simple c code to show your name and your age.
2. Write a simple c code to show the sum of all digits of your student code.
3. Follow the <https://www.learn-cpp.org/> tutorial

Important References

- C++ Programming Tutorial and Instructions for Practical Sessions by Christopher Lester
Department of Physics (based on earlier versions by David MacKay, Roberto Cipolla and Tim Love) <https://www.hep.phy.cam.ac.uk> or a google search.
- Bjarne Stroustrup's site <https://www.stroustrup.com/>
- The C++ Foundation's site <https://isocpp.org/>
- Code Block's site: <http://www.codeblocks.org/>
- The cplusplus.com site: <https://www.cplusplus.com/>
- The C++ Point Tutorial's Site <https://www.tutorialspoint.com/cplusplus/index.htm>
- Learn C++ <https://www.learn-cpp.org/>

There are two types of people:

```
if (Condition) {  
    Statement  
    /* ....  
    */  
}
```

```
if (Condition)  
{  
    Statement  
    /* ....  
    */  
}
```

Questions?