# C++ guide and key points:

C++ code is medium level language which can be directly converted to machine level using compilers. Therefore, it is faster than other high-level languages such as java, python and C#.

Code written in Java -> Compiler (Javac) converts to byte code -> Run in Virtual Machine

Code written in C++ -> Compiler (g++ or clang) convert to direct machine code (Platform specific) -> Run on machine

# Variables:

The main thing is we make programs to manipulate data. So, it is important how we store it and access it.

|  |
| --- |
| 1 byte = 8 bits  Char = 1 byte  Short = 2 bytes  Int = 4 bytes (1 bit for sign and 31 bits for storing data) (range: +2^31 to – 2^31)  Unsigned int = 4 bytes (total 32 bits are used for storing data)  Long = 8 bytes  Float = 4 bytes  Double = 8 bytes  Bool = 1 byte |

Char is treated as character, so even a number is assigned to it, it treats as character.

Char a = ‘A’ same as char a = 64

**Sizeof(datatype**) 🡪 provides the size of data type (size of datatype is typically defined by compiler and can be different sometimes for different compilers)

# Functions:

* Reusable code blocks, specifically doing a defined task, can have inputs (parameters) or outputs.
* We can define functions with out having any classes and can still call them just by having declaration to it.
* If function is declared with a return value, the definition must return a value or else you get error. However, there is exception for int main() function as it by default return 0 which means, the main function returned success.

# Preprocessor commands

* Preprocessor directives start with a hash sign (#).
* These lines are not program statements but directives for the preprocessor. The preprocessor examines the code before actual compilation of code begins and resolves all these directives before any code is generated by regular statements.
* #include “file.h” 🡪 #include is preprocessor command which includes file from local program
* #include <file.h> 🡪 #include is preprocessor command which includes files from standard library
* #pragma once 🡪 Includes header file once in a cpp file (cpp 🡪 single translation unit which compiled to .obj)

# Header Files

* To store variable, function, class declarations
* #include “file.h” 🡪 #include is preprocessor command which includes file from local program
* #include <file.h> 🡪 #include is preprocessor command which includes files from standard library

# Conditions and branches:

If , else if

# Loops:

**For loop:**

|  |
| --- |
| For(int i = 0; i<5; i++)  Print(“hello”)  Same as  Int i = 0;  For( ; i<5; )  Print(“hello”)  I++;  Or  Int i = 0;  Bool condition = true  For( ; condition ; )  {  Print(“hello”)  I++;  If(! (i<5))  condition = false;  } |

**While loops:**

For loop require initial i variable to be initialized. If you dont want to initialize any variable then, while loop is better.

|  |
| --- |
| While(x<5)  Execute code. |

**Do while loops:**

Do something at least one time and check the condition to loop it again.

|  |
| --- |
| Do  {  Executed once always.  } while(x<5) |

# Control Flows:

**Continue:**

Skips the next lines in loop and go to next iteration of loop if there is one.

**Break:**

Break the current execution in the loop and jump out of loop

**Return:**

Return from the current function which has return type

# Pointers:

In computers, the memories are simply arranged as a single linear array. Assume a city that has a single straight road and houses are just aligned with this road. The addresses to this houses are stored in pointers.

Pointers are integers that store memory address of a variable. You can write the whole c++ program with out using a single pointer. But, if you want to improve performance and use efficiently pointers are great.

Void\* ptr = 0;

Here, we have declared a pointer with out any type and assigned memory address 0 (0 is not a valid memory address ofcourse!)

|  |
| --- |
| Int i = 5  Void\* ptr = &i;  Print(ptr) // prints address  Print(\*ptr) //prints value stored at address |

* Here, ptr stores the address of i.
* “&” provides the address to variable data.
* \*ptr provides the actual value stored at address that is stored in pointer.
* Ptr 🡪 provides address (same as &i)
* \* ptr 🡪 value that is fetched from the memory that is stored at pointer

Instead of void pointer you can use int, char, short or long pointers. They all are same integer type but differed in size of memory. Long pointer stores big memory addresses.

Char\* buffer = new char[8];

This means we have created a memory of 8 bytes and buffer stores the address to the initial char.

# References:

|  |
| --- |
| Int i = 5;  Int& ref = i;  ref = 10 // which means i = 10 |

References are just an alias to variable. They are not a specific variable that is declared and uses memory. They are just used for internal code purpose and has no impact on memory as they are not stored. You can not initialize with out assigning it. You should always assign it to an existing variable. Once a reference is assigned to a variable, it can not be assigned to another variable.

|  |
| --- |
| Int i = 5;  Int j = 10;  Int& ref; // not possible and gives compilation error  Int& ref = i;  ref = 10 // which means i = 10  ref = j; // this means j value is assigned to i. The reference is not changed to j. |

References are useful especially when you want to modify a variable inside a function that is declared outside of function.

|  |
| --- |
| Int increment(int& i)  { i++; }  Int main()  {  Int x = 5;  Increment(x); // now we are sending a reference to that x so that x will be incremented.  } |

# Classes:

Reusable code that is organized to a particular object and contain various datatypes and functions.

Classes are user defined types.

By default, all the variables in class are private.

**Class vs Struct:**

Basically, there is no major differences. But, there is one small difference.

Class: All default variables are private

Struct: All default variables are public.

Structs are still there in c++ just because of backward compatibility to C. As we all know, C uses structs instead of classes. Most of the times, structs are used when we have a different type of data.

# Static keyword:

* Static outside a class
  + Accessible only in that translation unit (cpp file)
  + Linker specifically looks for the variable defined in the same cpp file.
  + To link or access a static variable defined in different file, use “extern” keyword.
  + For best code practices always use static keyword if you don’t want to access the variable outside the cpp file.
* Static inside a class
  + All the instances of the class will have only one instance of this variable. (All class instances share this variable)
  + Any class instance changes the instance of that variable, it effects in all other classes.

# Constructor and Destructor:

* Every class normally has a default constructor. Constructer method is called when class is initialized.
* Constructor method has same name as class.
* Constructor is used to initialize class variables so that they are not assigned with random values.
* Destructors are called automatically when the object runs out of scope. Destroying heap allocated memory is done normally in destructors.

# Inheritance:

* To extend the class functionality and reuse class functions or give different definition to class functions
* To override function or variable in child class from base class
  + Use virtual keyword in base class
  + Use override keyword in child class
  + Otherwise, the child class object calls base class function

# Interfaces – Pure virtual:

Interfaces are just normal classes that contain abstract methods. Other languages specifically use interface keyword to create interface but in c++ it is just class with pure virtual functions.

* Pure virtual function is abstract function or known as interface.
* To create a pure virtual function
  + The base class function uses virtual function () = 0
  + This implies that the child class will override automatically.

# Visibility: #

* Private: accessible with in class only
* Public: accessible outside class also
* Protected: Accessible inside the class and all child classes