**C++ Reference Material**

*Created by – Hossain Muhammad Mamun*

*Assistant Professor, Dept. of CSE,*

*Bangladesh Army University of Science and Technology (BAUST)*

*Saidpur Cantonment, Saidpur, Nilphamary.*

**C++: An Object Oriented Programming Language**

**C++ at a glance:**

* C++ is an expanded version or **superset** of C.
* The C++ extension was first invented by **Bjarne Strousstrup** in **1979** at Bell laboratories in Murry Hill, New Jersey.
* Initially called the new language “**C with class**” however in **1983** the name was changed to C++.
* Since the invention, C++ has undergone three major revisions.

First revision was in **1997**, second in **19990** and the third revision, an ANSI/ISO (in which Herbert Scheldt was a member) standard which is referred as ***standard C++*** was come to light in **1998**.

* The event that **greatly expanded C++** is the creation of **STL** (Standard Template Library) by **Alexander Stepanov**.
* STL is a set of generic routines that we can use to manipulate data. It is both **powerful and elegant** but also quite large.

**Main topics:**

Objects and classes, inheritance, constructor, destructor, this, static, polymorphism, abstraction, abstract class, interface, namespace, encapsulation, arrays, strings, exception handling, File IO, STL etc.

**What is C++**

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming

C++ is a middle-level language, as it encapsulates both high and low level language features.

**Features of C++:**

* C++ supports the object-oriented programming, the four major pillar of object oriented programming used in C++ are:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

* Block structured (Class Based) Programing Language

**Details Features of C++:**

C++ is object oriented programming language. It provides a lot of **features** that are given below.

1. C:\Users\User\Desktop\cpp-features1.pngSimple
2. Machine Independent or Portable
3. Mid-level programming language
4. Structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible
11. Object Oriented
12. Compiler based

## Usage of C++:

By the help of C++ programming language, we can develop different types of secured and robust applications:

* Window application
* Client-Server application
* Device drivers
* Embedded firmware etc

## Standard Libraries:

Standard C++ programming is divided into three important parts:

* The core library includes the data types, variables and literals, etc.
* The standard library includes the set of functions manipulating strings, files, etc.
* The Standard Template Library (STL) includes the set of methods manipulating a data structure.

## History:

|  |  |  |
| --- | --- | --- |
| **Language** | **Year** | **Developed By** |
| Algol | 1960 | International Group |
| BCPL | 1967 | Martin Richard |
| B | 1970 | Ken Thompson |
| Traditional C | 1972 | Dennis Ritchie |
| K & R C | 1978 | Kernighan & Dennis Ritchie |
| C++ | 1980 | Bjarne Stroustrup |

**Compare with C: At a Galance**

|  |  |
| --- | --- |
| Parent of C++ | Super set of C |
|  |  |
| C is a low to mid level language | C++ is a mid to high level language |
|  |  |
| Less keywords: only 32+5(C99) | More Keywords: 63 |
|  |  |
| Five Fundamental Data type | Six Fundamental Data type (bool) |
|  |  |
| Structured Programing Language: Use function as main structural component | Block structured Programing Language: Use class as main Block structural component |
|  |  |
| No OOP features | Includes OOP features |
|  |  |
| Difficult to handle large complex code over 25,000 to 100000 line. | Allow programmer to comprehend and manage large complex code/programs. |
|  |  |
| Usages: OS, Interpreter, Compiler, Editor, Device Drivers, develop other programming language like C++ | Usages: OS, Device Drivers, Web Server, Search Engine, Game Engine, develop other programming language like C#, Java etc. |

# C Vs. C++ in Details:

|  |  |  |
| --- | --- | --- |
| **No.** | **C** | **C++** |
| 1) | C follows the **procedural style programming.** | C++ is multi-paradigm. It supports both **procedural and object oriented.** |
| 2) | Data is less secured in C. | In C++, you can use modifiers for class members to make it inaccessible for outside users. |
| 3) | C follows the **top-down approach.** | C++ follows the **bottom-up approach.** |
| 4) | C does not support function overloading. | C++ supports function overloading. |
| 5) | In C, you can't use functions in structure. | In C++, you can use functions in structure. |
| 6) | C does not support reference variables. | C++ supports reference variables. |
| 7) | In C, **scanf() and printf()** are mainly used for input/output. | C++ mainly uses stream **cin and cout** to perform input and output operations. |
| 8) | Operator overloading is not possible in C. | Operator overloading is possible in C++. |
| 9) | C programs are divided into **procedures and modules** | C++ programs are divided into **functions and classes.** |
| 10) | C does not provide the feature of namespace. | C++ supports the feature of namespace. |
| 11) | Exception handling is not easy in C. It has to perform using other functions. | C++ provides exception handling using Try and Catch block. |

**Keywords:**

This page contains a list of all the reserved words in Standard C++, and a few predefined identifiers for the sake of comparison.

Recall the distinction between ***reserved words* and *predefined identifiers*, which are collectively referred to (by us, at least) as *keywords***. But be aware that this terminology is not standard. For example, some authors will use ***keyword*** in the same sense that we have used ***reserved word***.

**C++ Reserved Words:**

The reserved words of C++ may be conveniently placed into several groups. In the first group we put those that were also present in the C programming language and have been carried over into C++. A keyword is a reserved word. You cannot use it as a variable name, constant name etc.

**A list of 32 Keywords in C++ Language which are also available in C language are given below.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| auto | break | case | char | const | continue | default | do |
| double | else | enum | extern | float | for | goto | if |
| int | long | register | return | short | signed | sizeof | static |
| struct | switch | typedef | union | unsigned | void | volatile | while |

**A list of 30 Keywords in C++ Language which are not available in C language are given below.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| asm | dynamic\_cast | namespace | reinterpret\_cast | bool |
| explicit | new | static\_cast | false | catch |
| operator | template | friend | private | class |
| this | inline | public | throw | const\_cast |
| delete | mutable | protected | true | try |
| typeid | typename | using | virtual | wchar\_t |

The following 11 C++ reserved words are not essential when the standard ASCII character set is being used, but they have been added to provide more readable alternatives for some of the C++ operators, and also to facilitate programming with character sets that lack characters needed by C++.

***and bitand compl not\_eq or\_eq xor\_eq***

***and\_eq bitor not or xor***

**Some Predefined Identifiers:**

Beginning C++ programmers are sometimes confused by the difference between the two terms ***reserved word*** and ***predefined identifier***, and certainly there is some potential for confusion.

One of the difficulties is that some keywords that one might "expect" to be reserved words just are not. The keyword (word) ***main*** is a prime example, and others include things like the ***endl*** manipulator and other keywords from the vast collection of C++ libraries.

For example, you could declare a variable called main inside your main function, initialize it, and then print out its value (but you probably shouldn't, except as an experiment to verify that you can!). On the other hand, you could *not* do this with a variable named ***else*** or ***for***. The difference is that **else is a *reserved word***, while **main is "only" a *predefined identifier*.**

Here is a very short list of some of the predefined identifiers you may have encountered:

***cin endl INT\_MIN iomanip main npos std***

***cout include INT\_MAX iostream MAX\_RAND NULL string***

The following tokens are recognized by the [preprocessor](http://en.cppreference.com/w/c/preprocessor) when in context of a preprocessor directive:

[***if***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***elif***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***else***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***endif***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***defined***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***ifdef***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***ifndef***](http://en.cppreference.com/w/cpp/preprocessor/conditional)[***define***](http://en.cppreference.com/w/cpp/preprocessor/replace)[***undef***](http://en.cppreference.com/w/cpp/preprocessor/replace)[***include***](http://en.cppreference.com/w/cpp/preprocessor/include)[***line***](http://en.cppreference.com/w/cpp/preprocessor/line)[***error***](http://en.cppreference.com/w/cpp/preprocessor/error)[***pragma***](http://en.cppreference.com/w/cpp/preprocessor/impl)

The following tokens are recognized by the preprocessor *outside* the context of a preprocessor directive:

***\_Pragma(since C++11)***

**C++ Variable**

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

type variable\_list;

**int** x;    **float** y;    **char** z;

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

**int** x=5,b=10;  //declaring 2 variable of integer type

**float** f=30.8;    **char** c='A';

Rules for defining variables

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword e.g. char, float etc.

Valid variable names: **int** a;    **int** \_ab;    **int** a30;

Invalid variable names: **int** 4;    **int** x y;    **int** **double**;

**Data Type in C++: At a Galance**

Character char : char ch; ch =’A’;

Integer int : int m, n; m=10,n=20;

Floating point float : float cgpa; cgpa=3.75;

Double precision double : double price; price =96969693.50

Void void : valueless

Boolean bool : bool flag; flag= true;

**Data Type in C++: In Details**

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

Cpp Data typews 1

There are 4 types of data types in C++ language.

|  |  |
| --- | --- |
| **Types** | **Data Types** |
| Basic Data Type | int, char, float, double, etc |
| Derived Data Type | array, pointer, etc |
| Enumeration Data Type | enum |
| User Defined Data Type | structure |

## Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

* **Character types:** They can represent a single character, such as 'A' or '$'. The most basic type is char, which is a one-byte character. Other types are also provided for wider characters.
* **Numerical integer types:** They can store a whole number value, such as 7 or 1024. They exist in a variety of sizes, and can either be *signed* or *unsigned*, depending on whether they support negative values or not.
* **Floating-point types:** They can represent real values, such as 3.14 or 0.01, with different levels of precision, depending on which of the three floating-point types is used.
* **Boolean type:** The boolean type, known in C++ as bool, can only represent one of two states, true or false.

Let's see the basic data types. It size is given according to 32 bit OS.

|  |  |  |
| --- | --- | --- |
| **Data Types** | **Memory Size** | **Range** |
| char | 1 byte | -128 to 127 |
| signed char | 1 byte | -128 to 127 |
| unsigned char | 1 byte | 0 to 127 |
| short | 2 byte | -32,768 to 32,767 |
| signed short | 2 byte | -32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 32,767 |
| int | 2 byte | -32,768 to 32,767 |
| signed int | 2 byte | -32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 32,767 |
| short int | 2 byte | -32,768 to 32,767 |
| signed short int | 2 byte | -32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 32,767 |
| long int | 4 byte |  |
| signed long int | 4 byte |  |
| unsigned long int | 4 byte |  |
| float | 4 byte |  |
| double | 8 byte |  |
| long double | 10 byte |  |

Here is the complete list of fundamental types in C++:

|  |  |  |
| --- | --- | --- |
| **Group** | **Type names\*** | **Notes on size / precision** |
| Character types | **Char** | Exactly one byte in size. At least 8 bits. |
| **char16\_t** | Not smaller than char. At least 16 bits. |
| **char32\_t** | Not smaller than char16\_t. At least 32 bits. |
| **wchar\_t** | Can represent the largest supported character set. |
| Integer types (signed) | **signed char** | Same size as char. At least 8 bits. |
| *signed* **short** *int* | Not smaller than char. At least 16 bits. |
| *signed* **int** | Not smaller than short. At least 16 bits. |
| *signed* **long** *int* | Not smaller than int. At least 32 bits. |
| *signed* **long long** *int* | Not smaller than long. At least 64 bits. |
| Integer types (unsigned) | **unsigned char** | (same size as their signed counterparts) |
| **unsigned short** *int* |
| **unsigned** *int* |
| **unsigned long** *int* |
| **unsigned long long** *int* |
| Floating-point types | **float** |  |
| **double** | Precision not less than float |
| **long double** | Precision not less than double |
| Boolean type | **Bool** |  |
| Void type | **Void** | no storage |
| Null pointer | **decltype(nullptr)** |  |

Size Unique representable values Notes

8-bit 256 = 28

16-bit 65 536 = 216

32-bit 4 294 967 296 = 232 (~4 billion)

64-bit 18 446 744 073 709 551 616 = 264 (~18 billion billion)

**C++ Key Terms:**

<iostream> 🡪 header file to perform I/O operation. ( like <stdio.h> in C)

namespace 🡪 declarative region to avoid name conflicts

std 🡪 all the library functions and features whatever are available in C++ are placed/defined under the std name space

int main(void) 🡪 ~ is invalid in C++, void is redundant here so we use

int main()🡪 is valid and should be use.

Left shift / output operator: << 🡪 Stream insertion operator

Right shift / input operator: >> 🡪 Extraction operator

endl 🡪 End line (like new line in C) defined under the namespace std;

:: 🡪 scope resolution operator;

* To indicates which class methods is being implemented.
* To define a function outside a class or
* To use a global variable but also has a local variable with same name.

: 🡪 colon used with access modifier or subclass

Don’t use signed and unsigned together.

**Object and Class:**

~ A Class is a logical abstraction, but an object has physical existence.

Class vs. Object: class is general type i.e. Human being whereas object is particular i.e. arif

**What is an object?**

**~** Any real world entity which has physical existence

**~** instance/occurrence of a class

~ has real existence

~ has some attributes or properties ~ exists inside the memory of a computer

~ is a variable of a user defined data type (class) or class type variable.

Example: Hillary, Salma, Dove etc.

**Instance: same as object**

~ is an occurrence of a class. Different instance can have their different set of values. If student is a class then student st1, st2; st1 and st2 are two different instance/object of student class.

**Instance variable:** - A variable which is declared inside the class but outside the method, is called instance variable

**What is a class?**

~ Blueprint to create an object

~ General form of an object

~ Class is used to define the nature of an object.

~ User defined type which represents some real world entity.

~ Have two types of members

1. Data member : instance variable and class variables (static variable)
2. Member function: functions or method

Example: Human, Student, Bird, Box

## Difference between class and instance variables

|  |
| --- |
|  |

Now, it should be clear what the difference between instance and class variables is. Class variables only have one copy that is shared by all the different objects of a class, whereas every object has it’s own personal copy of an instance variable. So, instance variables across different objects can have different values whereas class variables across different objects can have only one value.

**What is an object Oriented Programming?**

~ Organized around data: Data controlling access to code

~ define data and routines/functions/methods that are permitted to act on data.

~ has three mainly characters – Encapsulation, Polymorphism and Inheritance

**Encapsulation:**

~ binds together code and the data it manipulates, and keeps both safe from outside interference and misuse.

Class is the best example through which C++ achieved its encapsulation feather.

**class Student {**

**public:**

**int id;//data member (also instance variable)**

**string name;//data member(also instance variable)**

**void say\_hello(){**

**cout << “Hi ” << name << endl;**

**}**

**};**

int main() {

Student s1; //creating an object of Student

s1.id = 201;

s1.name = "Sonoo Jaiswal";

s1.say\_hello(); // Hi Sonoo Jaiswal

return 0;

}

**Polymorphism:**

**~ one interface, multiple methods**

* **Overloading**: same method name with different type of parameters or number of parameter is different but return type may same or different

#include <iostream>

using namespace std;

class Cal {

public:

int add(int a,int b){

return a + b;

}

float add(float a,float b){

return a + b;

}

int add(int a, int b, int c)

{

return a + b + c;

}

};

int main() {

Cal C;

cout<<C.add(10, 20)<<endl;

cout<<C.add(12, 20, 23);

return 0;

}

* **Overriding** : number of parameter or parameter type and return type same but body different; overriding function resides in derived class have same return type, parameter like base class but different body.

#include <iostream>

using namespace std;

***class Animal {***

***public:***

***void eat(){***

***cout<<"Eating...";***

***}***

***};***

***class Dog: public Animal***

***{***

***public:***

***void eat()***

***{***

***cout<<"Eating bread...";***

***}***

***};***

int main() {

Animal a = Animal ();

a.eat(); // Eating

Dog d = Dog();

d.eat(); // Eating Bread

return 0;

}

**Inheritance:**

~ process by which one object can acquire the properties of another object.

~ The way/mechanism by which Derived class/Child class acquire the resources/properties of Base class/ Parent Class.

#include <iostream>

using namespace std;

class **Animal** {

public:

void eat() {

cout<<"Eating..."<<endl;

}

};

class **Dog**: public Animal

{

public:

void bark(){

cout<<"Barking...";

}

};

int main() {

Dog d = Dog();

d.eat(); // Eating …

return 0; }

**Access Modifier:**

Define from where your field class member (data or method) can be accessed

**public**: Public member can be accessed from anywhere

**Private**: Private member can be accesses inside the class and cannot be access from outside the class or from the code of any class derived from the Base/Parent class.

In C++ (class), private is default but in C++(struct) public by default.

**Protected**: Protected members of a class are not accessible outside of the class code, but is accessible from the code of any class derived from the class.

|  |  |  |  |
| --- | --- | --- | --- |
| **Access** | **public** | **protected** | **private** |
| Same class | Yes | yes | yes |
| Derived classes | Yes | yes | no |
| **Outside classes** | **Yes** | **no** | **no** |

**Constructor:**

* A special method/function with same name as Class and that is a member of the class.
* Has no return type
* Take as many parameters as required
* Must be declared as public
* Called automatically, immediately when any object of that class is created.
* Use: initialization, automatically initialized object when it is created

**Destructor: start with the tilled symbol (~)**

* Same name as constructor(class) preceded by an tiled(~) sign
* No return type
* Destructor is called automatically or never called
* Must be declared as public
* No more than one destructor is allowed in a class.
* Called when delete an object of that class.
* Use: free memory
* Can’t take parameter
* destructor is called through **delete** keyword when memory is dynamically allocated

class Human

{ string name;

int age;

public:

Human(){

name="No name"; age= 0;

cout << " \n \t Constructor is called when you create an object \n" << endl;

}

void display(){

cout << name << " age is " << age << endl;

}

~Human(){ cout << " \n Destructor is called when you delete an object \n" << endl;}

};

int main()

{ Human h;

h.display();

return 0;

}