# C / C++

Alex S.[*]

## 1 C Language

C programming language was developed by Dennis Ritchie in the early 1970s (at AT&T Bell Labs) for the Unix operating system. It's an imperative programming language, meaning it describes the computation in terms of states (variables), and statements that change that state (statements that change variable values).

In 1983, ANSI formed a commitee to standardize C, and by 1989, the ANSI X3.159-1989 "Programming Language C" was born; also known as ANSI C. In 1990, the ANSI C was also turned into an ISO standard.

### 1.1 Efficiency

C (and presently C++; although this was not always the case) language is generally known to be the most efficient high level programming language. Some call it the 'high level assembly language'.

### 1.2 Types

In C language, one can convert from one type to another without much hassle. Pointers are often implicitly turned into "int" values and vice versa.

One can almost always 'typecast' any variable to any other variable—which leads to interesting issues related to aliasing.

### 1.3 Structures

The primary way of organizing data is to use Structures (more in class).

[*]alex@theparticle.com

# 2 C++ Language

C++ programming language was developed by Bjarne Stroustrup in 1983 (at AT&T Bell Labs) as a better version of C. It was originally named "C with Classes". It was designed to be multi-paradigm language—support many different programming styles and constructs: procedural programming, data abstraction, object oriented programming, generic programming.

C++ became an ISO standard in 1998. Many aspects of the language changed around that time—for example, namespaces were added, and made all the older C++ programs break, unless they used:

```
using namespace std;
```

## 2.1 Efficiency

C++ is currently considered to have the same efficiency as C. Most of the computational bits are "C language" anyway.

## 2.2 Types

C++ is statically typed (types are checked during compile type).

## 2.3 Classes

The primary way of organizing data is to use Classes. Classes are basically structures; that can also have functions operating on that structure—creating what are known as 'objects'.

## 2.4 Object Oriented Programming

Object Oriented Programming generally needs a few things:

### 2.4.1 Class

A class is a template (a structure) of the object. Basically a structure with functions (or rather, methods) that operate on that structure.

### 2.4.2 Object

An object is an *instance* of a class.

### 2.4.3 Encapsulation

Encapsulation is data hiding. A function's variables can only be accessed from that function, etc. In OOP sense, this means that an object's variables may be private. When someone is working with an object, they only have to concern themselves with the Object's public interface.

### 2.4.4 Inheritence

Inheritence is the ability to extend the functionality of a certain class—and re-use its code. For example, if our program needs to care about "Customer" and "Employee" objects—maintain lists of both, etc., then we may find it easier to create an "Entity" class, and inherit things from it.

Similarily, if we have an `ArrayStack` and a `LinkedStack`, we may find it convinient to create a `Stack` class and inheric things from that.

### 2.4.5 Abstraction

The ability to work with a more general form of an object. When we want to work with a `Stack`, we shouldn't have to care if it's an array based stack or a linked list based stack.

### 2.4.6 Polymorphism

Polymorphism is a complicated way of saying that different objects can behave/respond differently to same events. For example,

```
obj.speak();
```

Will certainly produce a different result if the `obj` is an instance of, say a `Cat`, `Dog`, `Person`.

## 2.5 Generic Programming

Generic programming is achieved through the use of templates. For example:

```
template<class In, class Out> void copy(In from, In too_far, Out to){
    while(from != too_far){
        *to = *from;
        ++to;
        ++from;
    }
}
```

The above basically copies a generic array, starting from variable `from` until we encounter `too_far`. This could be an array of anything—integers, doubles, some objects, etc.