In early 1980s, Bjarne Stroustrup invented C++, wrote its early definitions, and produced its first implementation. He formulated the design criteria for C++, designed all its major facilities, and was responsible for the processing of extension proposals in the C++ standards committee. During the 80s the C++ language was being refined. From 1990, ANSI committee began the development of an own standard for C++.

C++ is a general-purpose language and supports a variety of programming styles. All are based on a strong static type checking, and most aim at achieving a high level of abstraction and a direct representation of the programmer’s ideas. Not every peace of code can be well structured, hardware-independent, or easy to read. C++ has features that are intended for manipulating hardware facilities in direct and efficient way. It also has facilities for hiding such code behind elegant and safe interfaces. In addition, the aim was to have an average line of C++ code express much more than the average line of C or Pascal code.

The language core application domain is systems programming in the broadest sense. In addition, C++ is successfully used in many application areas that are not covered by this label. Implementations of C++ exist from some of the most modest microcomputers to the largest supercomputers and for almost all operating systems. You can practically compile the same C++ code in almost any type of computer and operating system without making changes. Code written in C++ is very short in comparison with other languages, since the use of special characters is preferred before key words.

The resulting code from a C++ compilation is very efficient, due indeed to its duality as high-level and low-level language and to the reduced size of the language itself.

The language owes much to C. Except for closing a few serious loopholes in the type system C is retained as a subset. A programming language serves two related purposes; it provides a vehicle for the programmer to specify actions to be executed, and it provides a set of concepts for the programmer to use when thinking about what can be done. The first purpose requires a language that is “close to the machine” so that all important aspects of a machine are handled simply and efficiently in a way that is obvious to the programmer. The C language was primarily designed with this in mind.
The second purpose requires a language that is “close to the problem to be solved” so that the concepts of a solution can be expressed directly and concisely. The facilities added to C to create C++ were primarily designed with this in mind. This is where the name came from. The ++ operator means increment in C. So C++ is kind of a result of incremental operation on the C language. The main extensions to the traditional C programming techniques are data abstraction, object-oriented programming, generic programming, exception handling, standard library, and providing powerful input/output system thru standard library.

Object-oriented programming is a technique for programming. C++ is an object-oriented programming language. This means that it provides the facilities that make it reasonably easy, safe and efficient to use this style. These facilities include data encapsulation, inheritance, and polymorphism.

Data abstraction is provided using class facility. C++ has no built-in high-level data types and no high-level primitive operations. For example, the language does not provide a matrix type with an inversion operator or a string type with concatenation operator. If a programmer wants such a type, it can be defined using the facilities provided by the language. Defining a new general-purpose type or application-specific type is the most fundamental programming activity in C++. A well-designed user-defined type differs from a built-in type only in the way it is defined, not in the way it is used. The C++ standard library provides many examples of such types and their use.

Generic programming. There are some general concepts such as stack, map/multi maps, set/multi set, list, and vector. The idea of these data structures is to act as a container for build-in and user-defined types. Therefore, they need to be type-independent. Generic algorithms are the algorithms that operate on these containers. The language provides the template facility to support generic programming.

Examples:

Using C++ standard library.

```
string s1( “ABCDE” );
```
string s2 = “FGHIJ”;
string s3 = s1 + s2;
s1 < s2, s1 == s2, s1 > s2, s1 != s2

string::size_type i = s1.find( “CD” ); // i = 2

// Get C-representation string
char *cstring = s1.c_str( );

// Construct a string from different types
int i = 10;

string s = “int i has value”;
char c = ‘:’;

ostringstream os;
os << s << c << ‘ ‘ << i;

// print it out
cout << os.str(); // os.str() return a string – “int I has value: 10”

Input/Output

int i;

double d;

float f;

char c;

cout << i << ‘ ‘ << d << ‘ ‘ << f << ‘ ‘ << c << ‘ ‘ << s << endl;
cin >> i >> d >> f >> c;

*******************************************************
The program is included to demonstrate some of language’s features.