**OBJECT:**

- Introduction of C++ programming Language
- Introduction of Dev C++ Programming Environment
- Basic Program writing and error correction.
- Output using cout, Preprocessor Directives

**INTRODUCTION TO PROGRAMMING LANGUAGES**

A computer is a computational device which is used to process the data under the control of a computer program. Program is a sequence of instruction along with data. While executing the program, raw data is processed into a desired output format. These computer programs are written in a programming language which are high level languages. High level languages are nearly human languages which are more complex than the computer understandable language which are called machine language, or low level language.

Between high-level language and machine language there are assembly language also called symbolic machine code. Assembly language are particularly computer architecture specific. Utility program (Assembler) is used to convert assembly code into executable machine code. High Level Programming Language are portable but require Interpretation or compiling to convert it into a machine language which is computer understood.

**HIERARCHY OF COMPUTER LANGUAGE**

Programmers write instructions in various programming languages, some directly understandable by computers and others requiring intermediate translation steps. Hundreds of computer languages are in use today. These may be divided into three general types

1. Machine languages
2. Assembly languages
3. High-level languages

Any computer can directly understand only its own machine language. Machine language is the "natural language" of a computer and as such is defined by its hardware design. Machine languages generally consist of strings of numbers (ultimately reduced to 1s and 0s) that instruct computers to perform their most elementary operations one at a time. Machine languages are machine dependent (i.e., a particular machine language can be used on only one type of computer).

Machine-language programming was simply too slow, tedious and error-prone for most programmers. Instead of using the strings of numbers that computers could directly understand, programmers began using English-like abbreviations to represent elementary operations. These abbreviations formed the basis of assembly languages. Translator programs called assemblers were developed to convert early assembly-language programs to machine language at computer speeds.

Computer usage increased rapidly with the advent of assembly languages, but programmers still had to use many instructions to accomplish even the simplest tasks. To speed the programming process, high-level languages were developed in which single statements could be written to accomplish substantial tasks. Translator programs called compilers convert high-level language programs into machine language. High-level languages allow programmers to write instructions that look almost like everyday English and contain commonly used mathematical notations.

From the programmer's standpoint, obviously, high-level languages are preferable to machine and assembly language. C, C++, Microsoft's .NET languages (e.g., Visual Basic .NET, Visual C++ .NET and C#) and Java are among the most widely used high-level programming languages.
The process of compiling a high-level language program into machine language can take a considerable amount of computer time. Interpreter programs were developed to execute high-level language programs

directly, although much more slowly. Interpreters are popular in program development environments in which new features are being added and errors corrected. Once a program is fully developed, a compiled version can be produced to run most efficiently.

**INTRODUCTION TO C++:**

C++ programming language is an enhanced version of C language that provides object-oriented programming (OOP) capabilities. It is a superset of C, which means that you can use a C++ compiler to compile C programs. Object oriented programming techniques differ significantly from the sequential programming used in C programming language.

**HISTORY OF C AND C++**

C++ evolved from C, which evolved from two previous programming languages, BCPL and B. BCPL was developed in 1967 by Martin Richards as a language for writing operating-systems software and compilers for operating systems. Ken Thompson modeled many features in his language B after their counterparts in BCPL and used B to create early versions of the UNIX operating system at Bell Laboratories in 1970.

The C language was evolved from B by Dennis Ritchie at Bell Laboratories. C uses many important concepts of BCPL and B. C initially became widely known as the development language of the UNIX operating system. Today, most operating systems are written in C and/or C++. C is now available for most computers and is hardware independent. With careful design, it is possible to write C programs that are portable to most computers.

The widespread use of C with various kinds of computers (sometimes called hardware platforms) unfortunately led to many variations. This was a serious problem for program developers, who needed to write portable programs that would run on several platforms. A standard version of C was needed. The American National Standards Institute (ANSI) cooperated with the International Organization for Standardization (ISO) to standardize C worldwide; the joint standard document was published in 1990 and is referred to as ANSI/ISO 9899: 1990.

C++, an extension of C, was developed by Bjarne Stroustrup in the early 1980s at Bell Laboratories. C++ provides a number of features that "spruce up" the C language, but more importantly, it provides capabilities for object-oriented programming.

A revolution is brewing in the software community. Building software quickly, correctly and economically remains an elusive goal, and this at a time when the demand for new and more powerful software is soaring. Objects are essentially reusable software components that model items in the real world. Software developers are discovering that using a modular, object-oriented design and implementation approach can make them much more productive than they can be with previous popular programming techniques. Object-oriented programs are easier to understand, correct and modify.

**THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE):**

Turbo C++ compiler has its own built-in text editor. The files you create with text editor are called source files, and for C++ they typically are named with the extension .CPP, .CP, or .C.

The C++ Developing Environment, also called as Programmer's Platform, is a screen display with windows and pull-down menus. The program listing, error messages and other information are displayed in separate windows. The menus may be used to invoke all the operations necessary to develop the program, including editing, compiling, linking, and debugging and program execution.

**USING DEV C++ IDE**

Dev-C++ is a full-featured integrated development environment (IDE), which is able to create Windows or DOS-based C/C++ programs. You have to install the Dev C++ IDE then to invoke the IDE double click on the icon available on desktop.

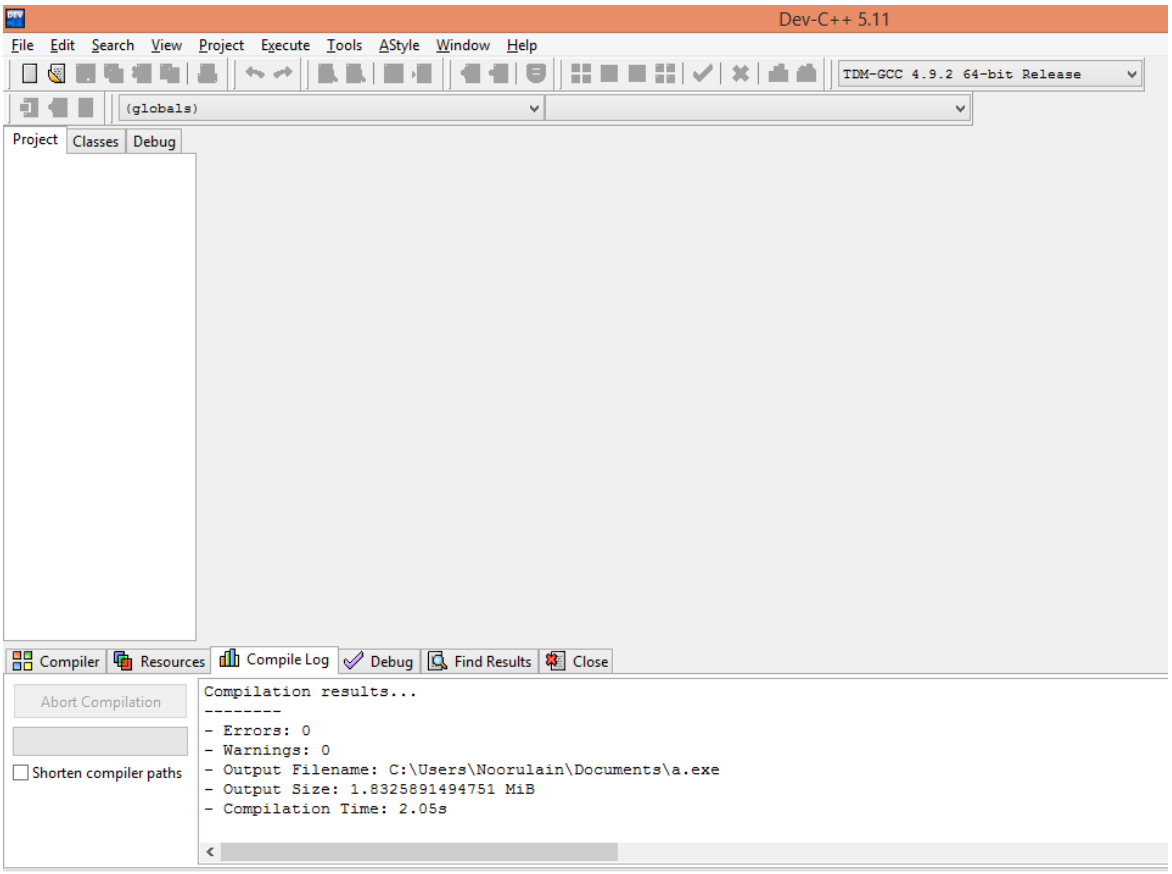



**Figure 1: Dev C++ IDE**

**CREATING NEW SOURCE FILE**

Click on **"new"** icon and select **"Source File"** see figure 2. You can also press the **ctrl + N** shortcut.

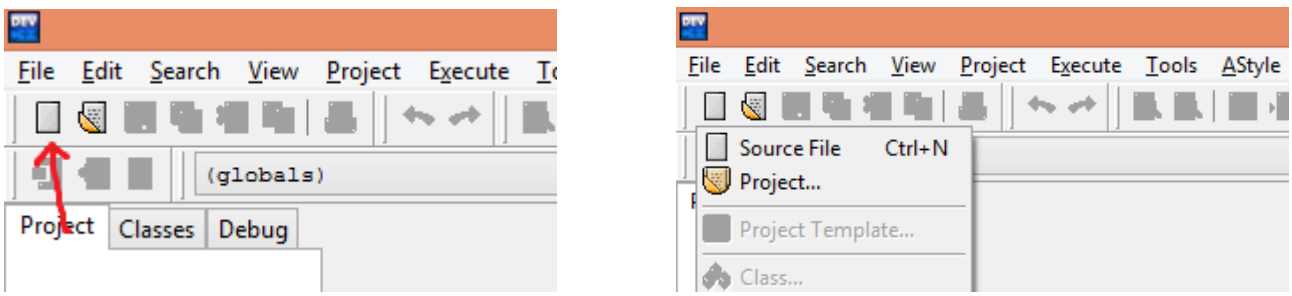



**Figure 2: Creating new Source File.**

**TYPING AND COMPILING YOUR FIRST PROGRAM**

Now type your program as shown in figure 3. Save you program by pressing **"ctrl + S"**, give any name to you file. Now click on "Compile and Run" icon to see the output of your program.

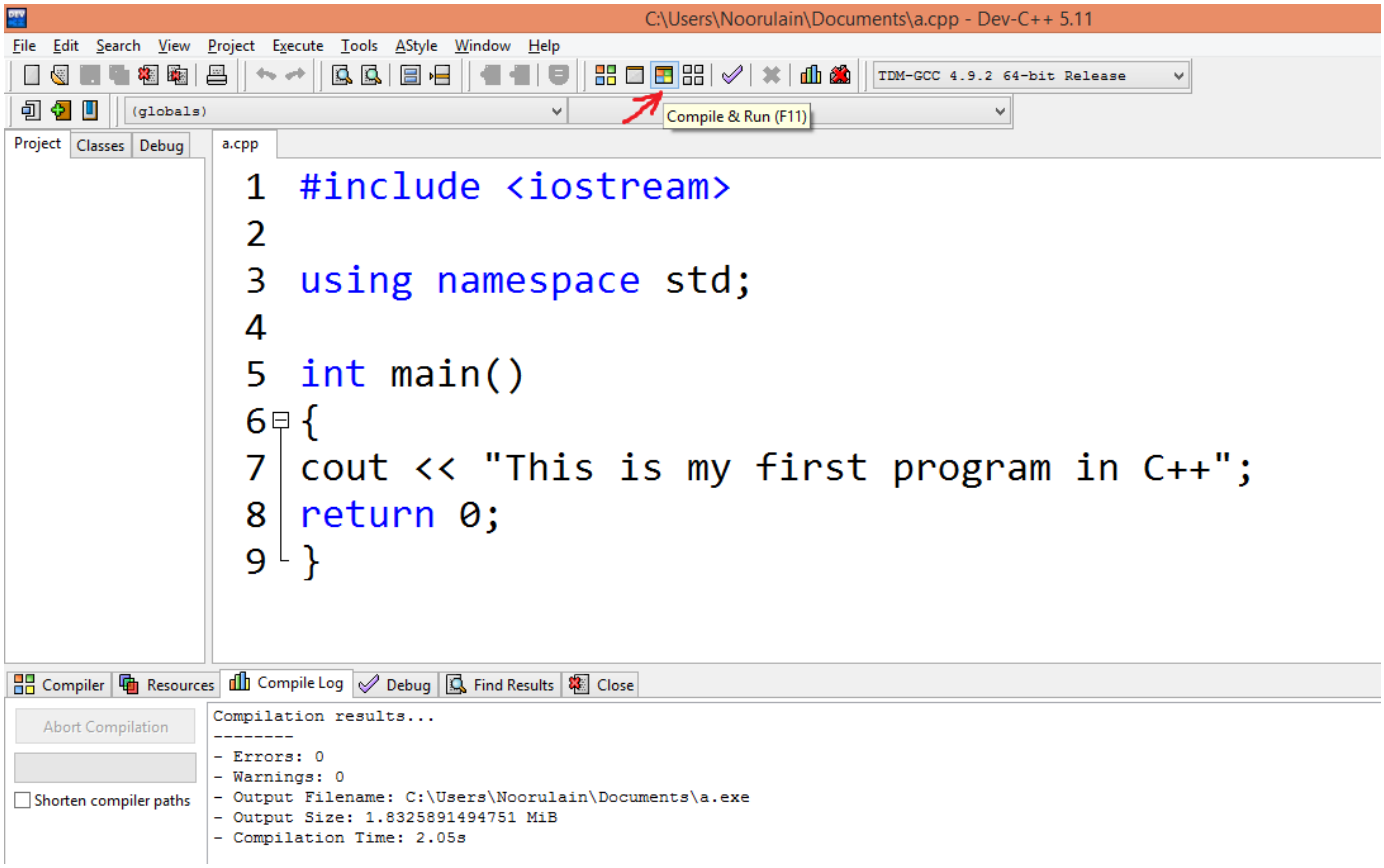
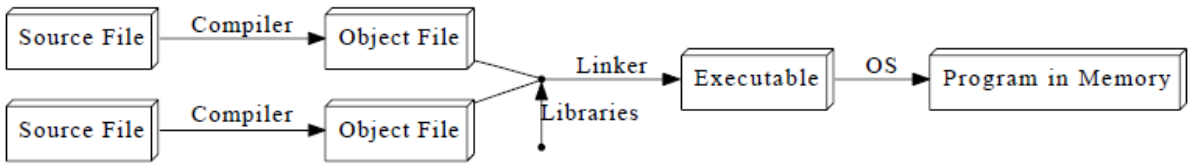



**Figure 3: Typing and running the program**

### THE COMPILATION PROCESS

A program goes from text files (or source files) to processor instructions as follows:




Object files are intermediate files that represent an incomplete copy of the program: each source file only expresses a piece of the program, so when it is compiled into an object file, the object file has some markers indicating which missing pieces it depends on. The linker takes those object files and the compiled libraries of predefined code that they rely on, fills in all the gaps, and spits out the final program, which can then be run by the operating system (OS).

The compiler and linker are just regular programs. The step in the compilation process in which the compiler reads the file is called parsing.
In C++, all these steps are performed ahead of time, before you start running a program. In some languages, they are done during the execution process, which takes time. This is one of the reasons C++ code runs far faster than code in many more recent languages.

**Now perform the following changes. Is the program running as before? Write Yes or No and explain why. Finally correct the program again after performing every error than perform the next change.**

1. Delete the line **using namespace std** from the program.

_____

_____

_____

**2.** Write **void main()** instead of **int main()**

_____

_____

_____

**3.** Write **#include<abc>** instead of **#include<iostream>**

_____

_____

_____

4. Remove any one semi colon ' **;** '.

_____

_____

_____

5. Remove any one of brace ' **{** ' or ' **}** '.

_____

_____

_____

6. Write **int main() ;** instead of **int main()**

_____

_____

_____

7. Write **cout >>** instead of **cout <<**

_____

_____

_____

8. Remove the line **return 0;**

_____

_____

_____

**9.** Write **Main()** instead of **main()**

_____

_____

_____

10. Write **COUT<<** instead of **cout<<**

_____

_____

_____

11. Remove inverted commas **" "** in line 7**;**

_____

_____

_____

**12.** Remove **#** from **#include<iostream>**

_____

_____

_____