Programming and Problem Solving through C Language
O Level / A Level

Chapter -1 : Introduction to Programming

Programming Language - Introduction

A programming language is a set of symbols, grammars and rules with the help of which one is able to translate algorithms to programs that will be executed by the computer. The programmer communicates with a machine using programming languages. Most of the programs have a highly structured set of rules.
The primary classifications of programming languages are: Machine Languages. Assembly Languages. High level Languages.

Machine Language

Machine language is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding. Machine level language is a language that supports the machine side of the programming or does not provide human side of the programming. It consists of (binary) zeros and ones. Each instruction in a program is represented by a numeric code, and numerical addresses are used throughout the program to refer to memory locations in the computer's memory. Microcode allows for the expression of some of the more powerful machine level instructions in terms of a set of basic machine instructions.

Assembly language

Assembly language is easier to use than machine language. An assembler is useful for detecting programming errors. Programmers do not have the absolute address of data items. Assembly language encourage modular programming

High level language

High level language is a language that supports the human and the application sides of the programming. A language is a machine independent way to specify the sequence of operations necessary to accomplish a task. A line in a high level language can execute powerful operations, and correspond to tens, or hundreds, of instructions at the machine level. Consequently more programming is now done in high level languages. Examples of high level languages are BASIC, FORTRAN etc

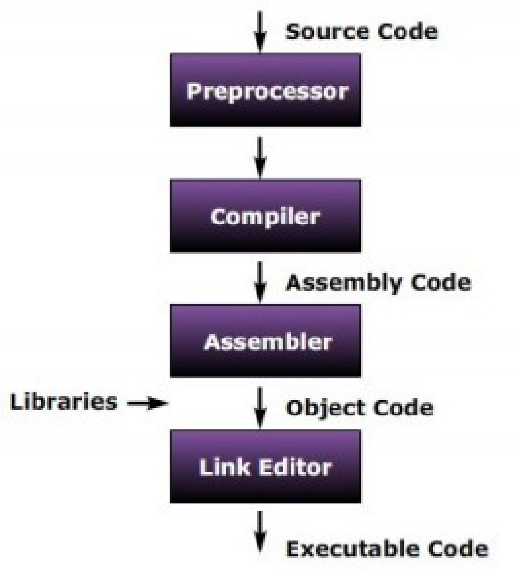
Compilation

The compiler program translates the instructions of a high level language to a machine level language. A separate compiler is required for every high level language. High level language is simply a programmer's convenience and cannot be executed in their source. The actual high - level program is called a source program. It is compiled (translated) to machine level language program called object program for that machine by the compiler. Such compilers are called self-resident compilers. Compiler compiles the full program and reports the errors at the end

Compilation Process

The compilation and execution process of C can be divided in to multiple steps:

- Preprocessing Using a Preprocessor program to convert C source code in expanded source code. "#include" and "#define" statements will be processed and replaced actually source codes in this step.
- Compilation Using a Compiler program to convert C expanded source to assembly source code.
- Assembly Using a Assembler program to convert assembly source code to object code.
- Linking Using a Linker program to convert object code to executable code. Multiple units of object codes are linked to together in this step.
- Loading Using a Loader program to load the executable code into CPU for execution. Compilation



Linking

- After all of the files are compiled, they must be "merged together" to produce a single executable file that the user use to run the program.
- In C, most compiled programs produce results only with the help of some standard programs, known as library files that reside in the computer. This process is called linking.
- The result obtained after linking is called the executable file.
- The linker′s primary function is to bind symbolic names to memory addresses.
- To do this, it first scans the files and concatenates the related file sections to form one large file. Then, it makes a second pass on the resulting file to bind symbol names to real memory addresses.
- Loading is loading the executable into memory prior to execution.

- There are two types of linking: Static linking. Dynamic linking.
- Static linking occurs at compilation time; hence it occurs prior to loading a program. With static linking the external symbols that are used by the program (e.g. function names) are resolved at compile time.
- Dynamic linking occurs at run time, so it occurs after or at the time of the loading of a program. With dynamic linking the symbols are resolved either at loading time, or at run time when the symbol is accessed (lazy binding).

Loading

- After the files are compiled and linked the executable file is loaded in the computer′s memory for executing by the loader. This process is called Loading.
- Program loading is basically copying a program from secondary storage into main memory so it ′s ready to run.
- In some cases, loading us just not copying the data from disk to memory, but also setting protection bits, or arranging for virtual memory map virtual addresses to disk pages.

Assignments

1. The linker ?
    a. is same as the loader
    b. is required to create a load module
    c. is always used before programs are executed
    d. None of above

2. A system program that combines the separately compiled modules of a program into a form suitable for execution ?
    a. Assembler
    b. Linking loader
    c. Cross compiler
    d. Load and Go

3. Loading process can be divided into two separate programs, to solve some problems. The first is binder the other is ?
    a. Linkage editor
    b. Module Loader
    c. Relocator
    d. None of these

4 A linker program
    a. places the program in the memory for the purpose of execution.

b. relocates the program to execute from the specific memory area allocated to it.
c. links the program with other programs needed for its execution.
d. interfaces the program with the entities generating its input data.

Q.5 Explain Linker.

Q.6 Explain Compiler and Interpreter.