CHARUTAR VIDYA MANDAL'S
SEMCOM
Vallabh Vidyanagar

Faculty Name: Ami D. Trivedi
Class: FYBCA
Subject: US01CBCA01 (Fundamentals of Computer Programming Using C)

*UNIT – 1 Concept of Algorithm, Flowchart and Languages

*NEED OF ALGORITHM AND FLOWCHART OR PURPOSE OF PROGRAM PLANNING

Suppose you are asked by your teacher to solve an arithmetic problem and you are not familiar with the steps involved in solving that problem. In such a situation, you will not be able to solve the problem. The same principle applies in writing program also.

A programmer cannot write the instructions to be followed by a computer unless the programmer knows how to solve the problem manually.

Suppose you know the steps to be followed for solving the given problem but while solving the problem, you forget to apply some of the steps or you apply the calculation steps in the wrong sequence.

Obviously, you will get a wrong answer. Similarly while writing a computer program, if programmer leaves out some of the instructions for the computer or write the instructions in the wrong sequence, then the computer will calculate a wrong answer.

Thus, to produce an effective computer program, it is necessary that the programmer writes each and every instruction in the correct sequence. However, the logic (instruction sequence to solve a problem) of a computer program may be very complex. So we must define logic to plan a program.

Algorithm refers to logic of a program. There are various ways in which an algorithm can be presented.

When an algorithm is expressed in a programming language, it becomes a program.

Algorithm can also be represented in the form of flowcharts. Algorithm and Flowcharts are helpful to programmers to develop logic for their programs.

ALGORITHM

The word algorithm comes from the name of the 9th century Persian mathematician Abu Abdullah Muhammad bin Musa al-Khwarizmi.

What is an algorithm?

The term "algorithm" is often used to refer to logic (correct sequence of instructions needed to solve the problem on hand) of a program. It is a step-by-step description of how to arrive at the solution of given problem.

The term algorithm may be formally defined as:

"A sequence of instructions designed in such a way that if the instructions are executed in the specified sequence, the desired results will be obtained".

OR

An algorithm tells computer how to solve problem systematically to get desired output.

Characteristics of an algorithm

In order to qualify as an algorithm, a sequence of instructions must process the following characteristics:

1. Each and every instruction should be precise and unambiguous.
2. Each instruction should be executed in a finite time.
3. One or more instructions should not be repeated infinitely. This ensures that the algorithm will ultimately terminate.
4. After performing the instructions (when algorithm terminates), the desired results are obtained.

FLOWCHARTS

Definition: A flowchart is a pictorial representation of an algorithm.

Programmers use flowchart as a program planning tool for visually organizing a sequence of steps to solve a problem.

It uses different shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and brief statements. These boxes are connected with solid lines having arrow marks to indicate the flow of operation, i.e. the exact sequence in which the instructions are to be executed.

Why use Flowcharts?

Normally, an algorithm is first represented in the form of a flowchart and the flowchart is then expressed in some programming language to prepare a computer program.

The main advantage of this two step approach in program writing is that while drawing flowchart one is not concerned with the details of the programming language. Hence he can fully concentrate on the logic of the procedure.

Moreover, since a flowchart shows the flow of operations in pictorial form, any error in the logic of the procedure can be detected more easily than in the case of a program.

Once the flowchart is ready, the programmer can forget about the logic and can concentrate only on coding the operations in each box of the flowchart in terms of the statements of the programming language. This will normally ensure an error – free program.

A flowchart is a picture of the logic to be included in the computer program.

It is simply a method of assisting the programmer to lay out, in a visual, two – dimensional format, ideas on how to organize steps necessary to solve a problem by a computer.

It is basically the plan to be followed when the program is written. It acts like a road map for a programmer and guides him how to go from the starting point to the final point while writing a computer program.

Experienced programmers sometimes write programs without drawing the flowchart. However, for a beginner it is recommended that a flowchart be drawn first in order to reduce the number of errors and omissions in the program.

Moreover, it is a good practice to have a flowchart along with a computer program because a flowchart along with a computer program because a flowchart is very helpful during the testing of the program as well as while incorporating further modifications in the program.

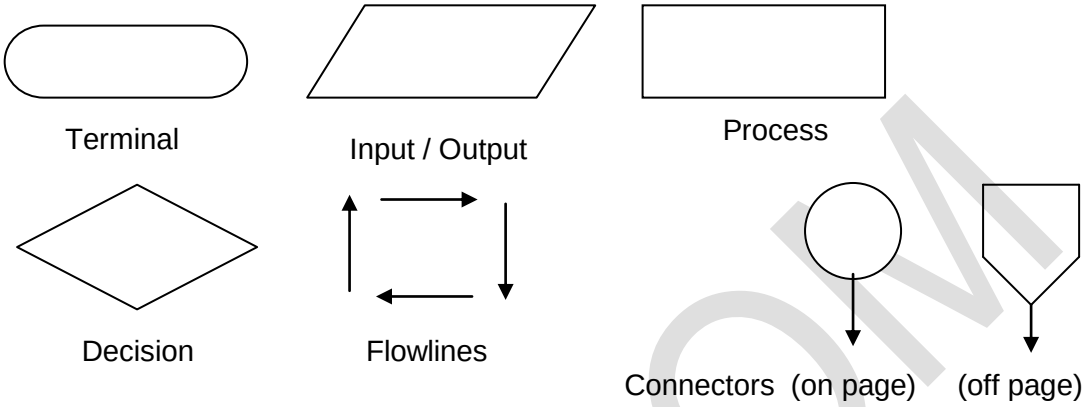
Flowchart Symbols

A flowchart uses boxes of different shapes to denote different types of instructions.

The communication of program logic through flowchart is made easier through the use of symbols that have standardized meanings. For example, a diamond always means a decision. Only a few symbols are needed to indicate necessary operations in a flowchart.

These symbols have been standardized by the American National Standards Institute (ANSI).

These symbols are drawn below:



Functions of all symbols are explained below:

1. Terminal:

The terminal symbol is used to indicate the beginning (START), ending (STOP) in the program logic flow. It is the first symbol and the last symbol in the program logic.

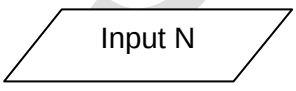
e.g. 

2. Input / Output:

The input /output symbol is used to denote any function of an input / output device in the program.

If there is a program instruction to input data from a disk, tape, terminal or any other type of input device, that step will be indicated in the flowchart with an input / output symbol.

Similarly all output instructions, whether it is output on a printer, magnetic tape, magnetic disk, terminal screen, or any output device, are indicated in the flowchart with an input / output symbol.

e.g. 

3. Processing:

A processing symbol is used in a flowchart to represent arithmetic and data movement instructions.

Thus, all arithmetic processes of adding, subtracting, multiplying and dividing are shown by the processing symbol. The logical process of moving data form one memory location to another (assignment statements) are also denoted by this symbol.

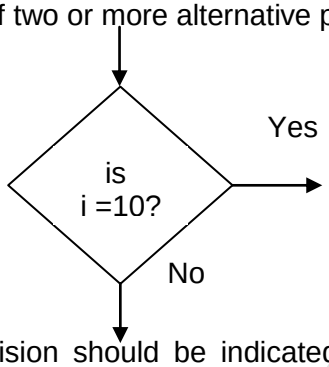
When more than one arithmetic and data movement instructions are to be executed consecutively, they are normally placed in the same processing box and they are assumed to be executed in the order of their appearance.

e.g.

```
sum =sum+r
a=b
```

4. Decisions :

The decision symbol is used in a flowchart to indicate the point at which a decision has to be made and a branch to one of two or more alternative points is possible.



Criteria for making the decision should be indicated clearly within the decision box. During execution, the appropriate path should be followed depending upon the result of the decision.

5. Flowlines:

Flowlines with arrowheads are used to indicate the flow of operations, that is, the exact sequence in which the instructions are to be executed.

The normal flow of flowchart is from top to bottom and left to right.

6. Connector :

If the flowchart becomes very long then number of flowlines increases and direction of Flowlines becomes confusing. Sometimes flowchart spread over more than one page. Also flow lines start crossing at many places that causes confusion. It reduces understandability of the flowchart.

Then the connector symbol is used to connect the flow chart at another page (off page) or on same page (on page). It is represented by a circle and a letter or digit is placed within the circle to indicate the link.

On page connector is used to indicate link on same page while off page connector is used to indicate link on another page.

Rules to draw flowcharts:

1. First chart the main line of logic and then incorporate detail.
2. Maintain a consistent level of detail for a given flowchart.
3. Do not chart every detail otherwise the flowchart will only be a step-by-step graphic representation of the program.
4. Words in the flowchart symbols should be common statements which are easy to understand.
5. Be consistent in using names, variables and symbols in the flowcharts.
6. Go from left to right and top to bottom in constructing flowcharts.
7. Keep the flowchart as simple as possible. The crossing of flow lines should be avoided as far as possible.
8. If a new flowcharting page is needed, it is recommended that the flowchart be broken at an input or output point. Moreover, properly labeled connecters should be used to link the portions of the flowchart on different pages.

Advantages and Limitations of Flowcharts

Advantages

The following benefits may be obtained when flowcharts are used for the purpose of program planning:

1. Better Communication

A flowchart is a pictorial representation of a program. So, it is easier for a programmer to explain the logic of a program to some other programmer, or to his / her boss through a flowchart, rather than the program itself.

2. Proper Program Documentation

Program documentation involves collecting, organizing, storing and otherwise maintaining a complete historical record of programs, and the other documents associated with system.

Good documentation is needed for the following reasons:

- Documented knowledge belongs to an organization, and does not disappear with the departure (resignation / retirement) of a programmer.
- If projects are postponed, documented work will not have to be duplicated.
- If programs are modified in future, the programmer will have a more understandable record of what was originally done.

Flowcharts often provide valuable documentation support.

3. Efficient Coding

Once a flowchart is ready, programmers find it very easy to write the corresponding program, because the flowchart acts as a road map for them.

It guides them to go from the starting point of the program to the final point, ensuring that no steps are omitted. The ultimate result is an error-free program, developed at a faster rate.

4. Systematic Debugging

A flowchart is very helpful in detecting, locating, and removing mistakes (bugs) in a program in a systematic manner, because programmers find it easier to follow the logic of the program in flowchart form.

The process of removing errors (bugs) in a program is known as debugging.

5. Systematic Testing

Testing is the process of confirming whether a program will successfully do all the jobs for which it has been designed under the specified constraints.

For testing a program, different sets of data are fed as input to that program to test the different paths in the program logic.

A flowchart proves to be very helpful in designing the test data for systematic testing of programs.

Limitations

In spite of their many obvious advantages, flowcharts have some limitations, which are as follows:

1. Flowcharts are very time consuming, and laborious to draw with proper symbols and spacing, especially for large complex programs.

It would be difficult to develop a detailed program flowchart for a program containing over thousands of statements.

2. Due to the symbol-string nature of flowcharting, any changes or modifications in the program logic will usually require a completely new flowchart.

Redrawing a flowchart being a tedious task, many programmers do not redraw or modify the corresponding flowchart when they modify their programs.

This leaves the program and its flowchart in an inconsistent state. That is, the logic used in the program, and that shown in its flowchart, do not match. This defeats the purpose of use of flowcharts as documentation support for programs.

To take care of this problem, many companies use software tools, which automatically generate flowcharts directly from the program code.

These software tools read the program's instructions and draw a flowchart of its logic. That is, this is a backward approach in which flowcharts are drawn from program codes mainly for documentation purpose.

3. There are no standards determining the amount of detail that should be included in a flowchart.

*GENERATION OF COMPUTER LANGUAGES

A language acceptable to a computer system is called computer language or programming language. And process of writing instructions in such a language is called programming or coding.

Language is a means of communication. We use a natural language such as English to communicate our ideas and emotions to others. Similarly, a programmer uses a computer language to instruct a computer what he/she wants it to do.

All natural languages use a standard set of words and symbols for communication. The set of words allowed in a language is called its vocabulary. Similarly, every computer language has a vocabulary of its own.

Every natural language has a systematic method of using the words and symbols in that language defined by the grammar rules of the language. Similarly, the words and symbols of a computer language must also be used as per the set rules known as syntax rules of the language.

One can use poor or incorrect vocabulary and grammar and can still communicate his/her thoughts. But it case of a programming language, a computer will not understand his/her instructions unless a programmer obeys exact syntax rules of the language.

The term 'Generation' of computer is used to categorize the generic enhancements in the various computer languages that have evolved over the last 50 yrs. Each generation indicates significant progress towards making computers easier to use.

Computer languages **by generation** are classified as follows.

First generation (late 1940s)	-	Machine languages
Second generation (early 1950s)	-	Assembly languages
Third generation (late 1950s to 1970s)	-	High level languages
Fourth generation (late 1970 onwards)	-	Whole range of query languages & & oth tools.

**COMPUTER LANGUAGES

Computer languages can be classified into 3 categories:

1. **Machine languages**
2. **Assembly languages**
3. **High – level languages**

(1) MACHINE LANGUAGES

Computers can be programmed to understand many different computer languages. But every computer understands only one language without using a translation program. This language is called machine language of the computer.

A computer understands information composed of only zeros and ones. The computer's instructions are therefore coded and stored in the memory in the form of 0's and 1's. A program written in the form of 0's and 1's is called a machine language program. There is a specific binary code for each instruction.

The circuitry of a computer is wired in such a way that it immediately recognizes the machine languages and converts it into electrical signals needed to run the computer.

Two part machine code:

An instruction prepared in any machine language has a two-part format, as shown below.

OPCODE	OPERAND
(Operation code)	(Address / Location)

Operation code: The first part is the command or operation and it tells the computer what function to perform. Every computer has an operation code or opcode for each of its functions.

Address: The second part of the instruction is the operand and it tells the computer where to find and store the data or other instructions that are to be manipulated.

Thus each instruction tells the control unit of the CPU what to do and what is the length and location of the data fields that are involved in the operation. Typical operation involves reading, adding, subtracting, writing and so on.

Some computers use single address instruction while many computers use multiple address instructions which include two or more operand addresses.

Advantages of machine language:

Programs written in the machine language can be executed very fast by the computer. This is mainly because machine instructions are directly understood by the CPU and no translation of the program is required.

Disadvantages / Limitations of machine language:

1. Machine dependent

Because the internal design of computers is different from one another and needs different electrical signals to operate, the machine language is also different from one type of computer to another.

It is determined by the actual design or construction of the Arithmetic Logic Unit, the control unit and the word length of the memory unit.

Hence, it is important to note that after becoming proficient in the machine code of a particular computer, if a company decides to change to another type, the programmer will be required to learn a new machine code and would have to write all the existing programs again.

2. Difficult to program

Although machine language is easily used by the computer, it is very difficult to write a program in this language.

It is necessary for the programmer either to memorize dozens of operation code numbers for the commands in the machine's instruction set or to constantly refer to a reference card.

A programmer is also forced to keep track of the storage locations of data and instructions. A machine language programmer must also be knowledgeable about the hardware structure of the computer.

3. Error prone

For writing a program in machine language, the programmer has to remember the opcodes and must keep a track of the storage location of data and instructions.

It therefore becomes very difficult for him to concentrate fully on the logic of the problem. This results in errors in programming.

4. Difficult to modify

It is very difficult to correct or modify machine language programs. Checking machine instructions to locate errors is is very difficult and time consuming.

Similarly, modifying a machine language program at a later date is so difficult that many programmers would prefer to code the new logic afresh instead of incorporating the necessary modifications in the old program.

In short, writing a program in machine language is so difficult and time consuming that it is rarely used today.

(2) ASSEMBLY LANGUAGE

Programming in machine language is difficult and error-prone because a programmer needs to:

1. Write numeric codes for the instructions.
2. Write numeric storage locations of data and instructions.
3. Keep track of storage location of data and instruction while writing a program.

Assembly language helped in overcoming these limitations of machine language programming.

The instructions, words which direct the computer are stored in the machine in numerical form. The programmer rarely writes his instructions in numerical form.

Instead each instruction to the computer is written using a letter code to designate the operation to be performed, plus the address in the memory of the number to be used in this step of the calculation.

Later, the alphabetical section of the instruction word is converted to numerical form using an assembler.

An instruction word consists of two parts. The operation code part which designates the operation (addition, subtraction, multiplication etc.) to be performed. The address of the number is to be used.

A typical instruction word is: ADD 535

The operation code part consisting of the letter ADD, directs the computer to perform the arithmetic operation of addition and address part tells a computer the address in storage of the number to be used.

Mnemonics: A Mnemonic means a name or symbol used for some code or function. All computer languages are made up of Mnemonics except the machine language itself.

A language allows instructions and storage locations to be represented by letters and symbols instead of numbers are called assembly language or symbolic language. A program written in an assembly language is called assembly language program or symbolic program.

Advantages of assembly language:

1. Easier to understand and use

Assembly languages are easier to understand and use because mnemonics are used instead of numeric op – codes and suitable codes are used for data. The use of mnemonics means that comments are usually not needed, the program itself is understandable.

2. Easy to locate and correct errors

It is easier to find and correct errors because of the use of mnemonics and symbolic field names. Programmers need not keep track of storage locations of data and instructions, fewer errors are made while writing assembly programs.

3. Easier to modify

Assembly language program are easier to understand. So it is easier to locate, correct and modify instructions of assembly language program than a machine language program.

4. No worry about addresses

An important advantage of assembly language is that programmers need not keep track of storage locations of data and instructions while writing an assembly language program.

5. Easily relocatable

Assembly language programs can be moved easily from one section of memory to another. This is not possible easily with machine language programming.

6. Efficiency of machine language

An assembly language program also enjoys the efficiency of its corresponding machine language program. Because there is one-to-one correspondence between instructions of an assembly language program and its corresponding machine language program.

In other words, an assembly language program will be just as long as the resulting machine language program. So, only translation time required by assembler is additional. Otherwise, actual execution time for an assembly language and its equivalent machine language program would be same.

7. Faster execution than High level language

The advantage of assembly language over HLL is that the computation time of an assembly language program is less. An assembly language program runs faster to produce the desired result.

Disadvantages / Limitations of Assembly language:

1. Machine dependent

Each instruction of an assembly language program is translated into exactly one machine language instruction. So assembly language programs are machine dependent.

Assembly language differs from computer to computer. So a decision to change to another computer will require learning a new language of the new computer.

2. Knowledge of hardware required

The programmer must have good knowledge of characteristics and logical structure of his/her computer to write good assembly language programs.

3. Machine level coding

In assembly language program, instructions are still written at the machine code level. That is one assembler instruction is substituted for one machine language instruction. So, programming is difficult and time consuming.

4. Not portable

The program written in an assembly language for one computer cannot be used in any other computer, i.e. the assembly language program is not portable. Each processor has its own instruction sets and hence it has own assembly language.

5. More instructions than High level language program

An assembly language program contains more instructions as compared to a high level language program. Each statement of a program in a high – level language (such as FORTRAN, PASCAL etc) corresponds to many instructions in a assembly language program.

(3) HIGH – LEVEL LANGUAGES (HLL)

Both machine and assembly languages have following limitations:

1. They are machine dependent. A machine/assembly language program can not be executed on any computer other than the computer for which it is written.
2. They require programmers to have a good knowledge of the internal structure of the computer used.
3. It is difficult, error prone and time consuming to write programs in machine/assembly language. Because they deal with machine level coding.

Due to these limitations, machine and assembly languages are called low-level programming languages.

To overcome the difficulties associated with assembly languages, high – level or procedure – oriented languages were developed.

High – level languages permit programmers to describe tasks in a form which is problem oriented rather than computer oriented. A programmer can formulate problems more efficiently in a high – level language program. Besides he must not have a precise knowledge of the architecture of the computer he is using.

The instructions written in a high – level language are called statements. Examples of high – level languages are BASIC, PASCAL, FORTRAN, COBOL, ALGOL, PROLOG, LISP, ADA, SNOBOL, JAVA, C#etc.

Note:

C language was developed in 1972 at AT&T's Bell laboratories, USA by Dennis Ritchie and Brian Kernighan.

BASIC– Beginners All purpose Symbolic Instruction Code
COBOL – Common Business Oriented Language
SNOBOL – StriNg Oriented symbolic Language
FORTRAN– FORmula TRANslation
LISP – LISt Processing

Advantages of high – level language:

1. Machine Independence

High – level languages are machine independent. This is very valuable advantage because it means that a company changing computers – from one to different manufacture – will not be required to rewrite all the programs that it is currently using.

In other words, a program written in a high – level language can be run on many different types of computers with very little or practically no modification.

2. Easier to learn and use

These languages are very similar to the languages normally used by us in our day – to –day life. Hence they are easy to learn and use.

They are also easier to use because a programmer need not know the internal details of a computer for programming in a high level language.

3. Fewer errors

In high – level languages, the programmer need not worry about how and where to store the instructions and data of program.

He need not write machine level instructions carried out by the computer. This allows programmer to concentrate more on logic. And thus he/she is much less likely to make errors during program development.

Also compilers and interpreters detect and indicate syntax errors automatically. So syntax errors can be easily located and corrected by programmer.

4. Lower program preparation cost

Writing programs in high – level languages requires less time and effort which ultimately leads to lower program preparation cost (cost of coding, debugging, testing etc.).

5. Better documentation

Statements of a program written in a high level language are very similar to natural language statements used by us in our day-to-day life. So very few or no separate comment statements are required in programs written in high level languages. Due to this reason, high level languages are sometimes also referred to as self documenting languages.

6. Easier to maintain

Programs written in high level languages are easier to maintain than assembly/machine language programs. This is because they are easier to understand, and therefore it is easier to locate, correct and modify instructions whenever desired.

Disadvantages / Limitations of high – level languages:

1. Lower efficiency

Program written in assembly language or machine language is more efficient than one written in high – level language. That is, the program written in high – level language take more time to run and require more main storage.

2. Less flexibility

Generally, high level languages are less flexible than assembly language because they do not have instructions or mechanism to control a computer's CPU, memory and registers.

***Fourth Generation languages (4GL)**

Fourth generation programming languages allows users to create programs with much less effort than is required by high level languages.

A high level language is a procedural language. Programmers write a set of executable operation to be performed in a sequence to achieve the task.

The objectives of 4GL include:

1. Increases the speed of developing programs
2. Minimize users efforts to obtain information from computer
3. Decrease skill level required of users
4. Reduce errors or make programs that are easy to change

These languages are usually used in conjunction with a data base. 4GL include: data query languages, report generators, and application generators.

A database query language permits formulation of queries that may relate to several records from one or more files. The appropriate records can then be printed or displayed in a suitable format. E.g. IBM's SQL.

A report generator allows data from a database to be extracted and formatted in to reports. It also allows substantial arithmetic and logic operations to be performed on the data before they are displayed or printed.

****TRANSLATORS**

A computer can execute only machine language program directly. So assembly / high language program must be converted (translated) into its equivalent machine language program before it can be executed on a computer. This translation is done with the help of translator program.

A translator is a program which converts assembly / high language program into its equivalent machine language program.

***COMPILER**

A computer can execute only machine language program directly. So high level language program must be converted (translated) into its equivalent machine language program before it can be executed on a computer. This translation is done with the help of translator program called compiler.

So compiler is a translator program that translates a high level program into its equivalent machine language program. Following figure explains process of translating a high level language program into its equivalent machine language program.

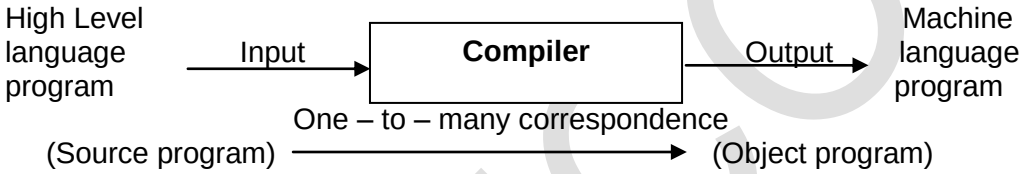


Figure: Illustrating the translation process of a compiler

As shown in the above figure the input to the compiler is a source program written in a high – level language and its output is an object program which consists of machine language instructions.

A program written by a programmer in a high – level language is called is called a **source program**. After this source program has been converted into machine language by compiler, it is referred to as an **object program**.

Note that the source program and the object program are the same program, but at different stages of development.

Compiler translates each high level language instructions into a set of machine language instructions. Hence, there is one-to-many correspondence between high level language instructions of a source program and the machine language instructions of its equivalent object program.

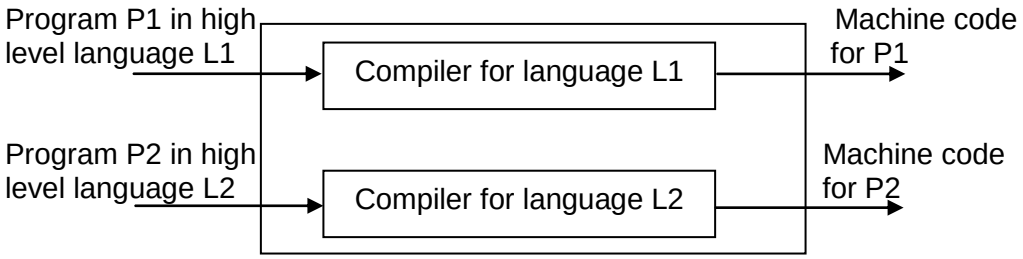


Figure: Computer supporting languages L1 & L2

The compiler can translate only those source programs which have been written in the language for which the compiler is meant. This is illustrated in the above figure.

For example, a FORTRAN compiler is only capable of translating source programs which have been written in FORTRAN and, therefore, each machine requires a separate compiler for each high – level language that it supports.

Compilers are large programs which reside permanently on secondary storage. When the translation of a program is to be done, they are copied into the main memory of the computer.

The compiler, being a program is executed with source program as its input data. While translating a given program, the compiler analyses each statement in the source program and generates the sequence of machine instructions. These machine equivalent instructions will be executed to carry out the computation specified in the statement.

There is no need to repeat the compilation process every time you wish to execute the program. Because object program stored in secondary memory is already in machine language. Compilation is needed whenever the program is modified.

A compiler cannot diagnose logical errors. It can only diagnose grammatical (syntax) errors in the program.

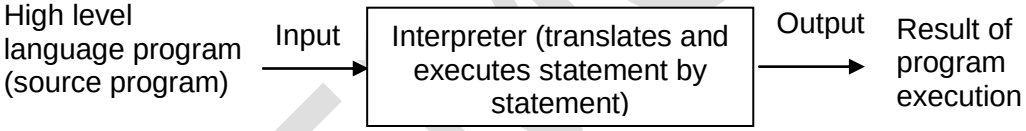
***INTERPRETER**

Interpreter is also a translator used to translate high – level language program into its equivalent machine language program.

It takes one statement of high level language program. Translates it into machine language instructions and then executes the machine language instruction immediately. i.e. translation and execution of the instructions goes on parallel.

This differs from a compiler. Compiler translates entire source program into an object program and source program is not involved in execution.

Input to an interpreter is a source program, but its output is result of program execution instead of object program.



In case of a compiler, after compilation of a source program, the resulting object program is saved permanently for future use. It can be used every time when the program is required to be executed.

In case of an interpreter no object code of the program is saved in the memory. So repeated interpretation of program (translation and execution) is necessary every time when program is required to be executed.

Advantages of interpreter:

1. As compared to compiler, interpreters are easier to write. Because they are less complex programs than compilers.
2. Interpreter itself is the small software as compared to the compiler, so it occupies less storage space.
3. They also require less memory space for execution than compliers require. Because no object code is saved.
4. Main advantage of interpreter is that a syntax error in a program statement is detected and shown to programmer as soon as the program statement is interpreted. So, interpreter makes it easier and faster to correct programs.

Disadvantage of interpreter:

Interpreters are slower than compilers when running a finished program. Because each statement is translated every time from source program when it is executed.

***ASSEMBLER**

A computer can directly execute only machine language programs that use numbers for representing instructions and storage locations.

So, an assembly language program must be converted (translated) into its equivalent machine program before it can be executed on the computer. This translation is done with the help of a translator program called assembler.

Assembler is system software supplied by computer manufacturer. It is written by system programmer by great care.

Assembler translates an assembly language program into its equivalent machine language program. It is called assembler because in addition to translating the assembly code into machine code, it also , 'assembles' the machine code in the main memory of the computer and makes it ready for execution.

Following figure explains process of translating an assembly language program into its equivalent machine language program.

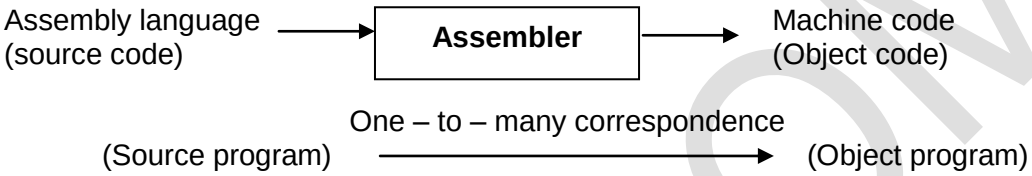


Figure: - Illustrating the translation process of an assembler

As shown in the above figure the input to the assembler is a source program written in an assembly level language and its output is an object program which consists of machine language instructions.

The symbolic program written by the programmer in assembly language is called a **source program**. After the source program has been converted into machine language by an assembler, it is referred to as an **object program.**

Since the assembler translates each assembly language instructions into its equivalent machine language instructions, there is one - to - one correspondence between the assembly instructions of source program and the machine instructions of object programs.

In case of assembly language program, the computer has to run the assembler program to translate the original assembly language program (source program) into its equivalent machine language program (object program). And then we run the machine language program to get the answer.

This means computer has to spend more time to get desired result from an assembly language program as compared to machine language program.

But symbolic programming saves so much time and effort of the programmer that the extra time spent by the computer is worth of. Because this conversion is one time. Once object code is obtained, it can be executed as many times as desired. Assembling is needed whenever the program is modified.

Note: Assemblers, compilers and interpreters are also known as **Language processors** because they are used for processing language instructions.

*EDITORS

Editor is a special program, which helps in creation and modification of the simple text files. It is also defined, as 'Editor is a program used to interactively review alters text material and other program instructions.'

Editor does not support any text or paragraph formatting. You can use it to type a simple memo, letter, etc. It can also be used to type programs for programming languages or applications such as BASIC, COBOL and SQL etc. Editor can create a document up to some limited characters.

TURBO C++ EDITOR

The Turbo C++ offers everything you need to write, edit, compile, link, run, manage, and debug your programs. You require TC.EXE file to activate the Turbo C++ i. e. TC editor.

Once you open this editor, it has following menu options.

- File -Edit - Search - Run - Compile
- Debug - Project - Options - Window - Help

Turbo C++ provides hot keys, or shortcut for your convenience. Popular are:

Command	Functions
Ctrl + K B	To set beginning of the block
Ctrl + K K	To set end of the block
Ctrl + K C	To copy the block
Ctrl + K V	To move the block
Ctrl + K Y	To delete the block
Ctrl + K H	To hide the marked block i.e. unhide the block
F1	Displays a help screen
F2	Saves the file that is in the active edit window
F5	Zooms the active window
F6	Cycles through all open windows
F7	Runs program in debug mode, tracing into functions
F8	Runs program in debug mode, stepping over functions calls
F9	Invokes the Project Manager to make an .EXE file
F10	Takes you to the menu bar
Alt + Backspace	Undo
Shift + Alt + Backspace	Redo
Shift + Delete	Places selected text in Clipboard, deletes selection
Shift + Insert	Pastes test from Clipboard into the active window
Ctrl + Delete	Remove selected text from window, does not put it in Clipboard
Alt + F9	Compiles to **.OBJ**
Ctrl + F7	Adds a watch expression
Ctrl + F9	Runs program

Other editor: NE (Norton editor), EDIT (dos editor), vi (UNIX editor), Notepad and wordpad (windows editor)

Disclaimer

The study material is compiled by Ami D. Trivedi. The basic objective of this material is to supplement teaching and discussion in the classroom in the subject. Students are required to go for extra reading in the subject through library work.