Chapter - 1

The Evolution of Programming Languages

Programming Languages Evolution

A Programming language is the language through which user can communicate with the computer by writing program instructions.

Every computer programming language contains a set of predefined words and a set of rules (syntax) that are used to create instructions of a program.

The Computer programming languages are classified as –

1) Low Level Language
2) Middle Level Language
3) High Level Language

1) Low Level Language

Low level language is also known as Machine Language. Machine language is also known as Machine code. Binary language is an example of low level language. The binary language contains only two symbols 1 and 0. All the instructions of binary language are written in the form of binary numbers 1's and 0's. A computer can directly understand the binary language. Low Level language is as the First generation language.

Advantages

- A computer can easily understand the low level language.
- Low level language instructions are executed directly without any translation.
- Low level language instructions require very less time for their execution.

Disadvantages

- Low level language instructions are very difficult to use and understand.
- Low level language instructions are machine dependent, that means a program written for a particular machine does not executes on other machine.
- In low level language, there is very difficult to find errors, debug and modify.

2) Middle Level Language

Middle level language is also known as Assembly language or Symbolic language. Assembly language is an example of Middle level language. In Assembly language, the instructions are created using symbols such as letters, digits and special characters. In assembly language, we use predefined words called mnemonics. A program written is an assembly language using mnemonics called assembly language program or symbolic program.

The process of translating an assembly language program into its equivalent machine language program with the use of an assembler. Assembler is used to translate middle level language to low level language.

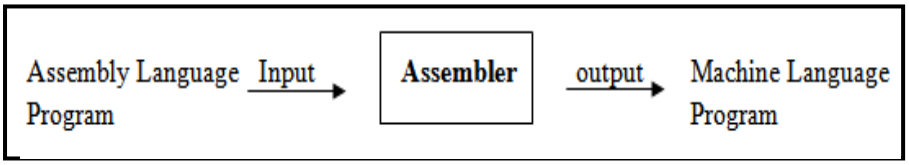


Figure -1: Translate assembly language program into machine language

Advantages

- In middle level language, writing instructions is easier.
- Middle level language is more reliable.
- Middle level language is easy to understand, find error and modify.

Disadvantages

- Middle level language is machine dependent.
- Middle level language needs to be translated into low level language.
- Middle level language executes slower compared to low level language.

3) High Level Language

High level can be easily understood by the users. It is very similar to the human language and has a set of grammar rules that are used to make instructions more easily. Every high level language has a set of predefined words known as keywords and a set of rules known as syntax. High level language is a programming language. Languages like COBOL, BASIC, FORTRAN, C,C++, JAVA etc. All these programming languages are to write program instructions. These instructions are converted to low level language by the complier or interpreter.

Advantages

- Writing instructions in high level language is easier.
- High level language is readable and understandable.
- High level language program can runs on different machines without any modification.
- It is easy to understand, create programs, find errors and modify.

Disadvantages

- High level language instructions need to be translated to low level language by using compiler or interpreter.
- Slower in execution as compared to low level and middle level language.
- Lack of flexibility.
- Lower efficiency.

Generation	Languages	Development Date	Example
First	Machine Language	1940s	10101111
Second	Assembly Language	1950s	MOV
Third	High Level Language	1960s	Read Sales
Fourth	Query and Database Languages	1970s	Select * from emp

Table -1: Evolution of Programming Languages

Programming Language Translator

A programming language translator is a software that translate computer program (instructions) written in some specific programming language into another programming language. A program written in high level language is called source code. To convert the source code into machine code, translators are needed.

There are three types of programming language translator –

1) Compliers
2) Interpreters

3) Assemblers
4) Linker and Loader

1) Compliers

Compiler is a translator which is used to convert programs in high level langue to low level language. It translates the entire program that is group of statements at a time and also reports the errors in source program encountered during the translation.

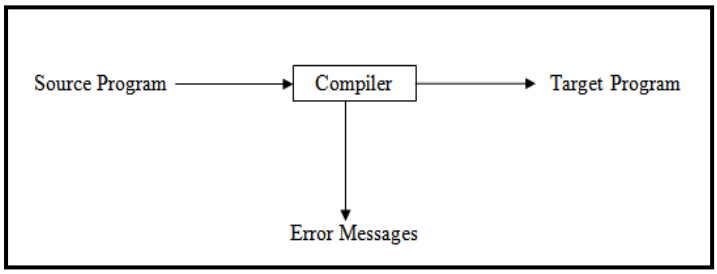


Figure - 2: Compiler translator

2) Interpreters

Interpreter is a translator which is used to convert programs in high level language to low level language. Interpreter translates line by line statements and reports the error once it encountered during the translation process. It gives better error diagnostics than a compiler.

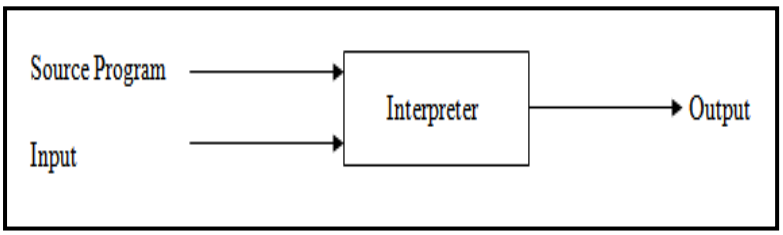


Figure – 3: Interpreter translator

3) Assemblers

Assembler is a translator which is used to translate the assembly language code into machine language code.

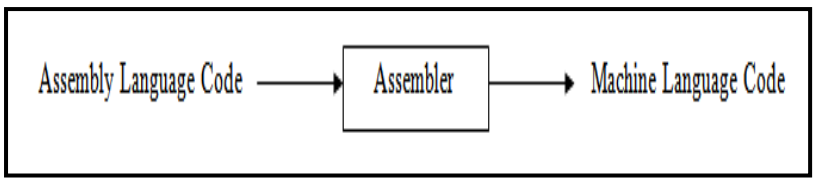


Figure – 4: Assembly translator

4) Linker and Loader

Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assembler.

The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded making the program instruction to have absolute reference.

Loader is a part of operating system and is responsible for loading executable files into memory and executes them.

It calculates the size of a program (instructions and data) and cerates memory space for it. It initializes various registers to initiate execution.

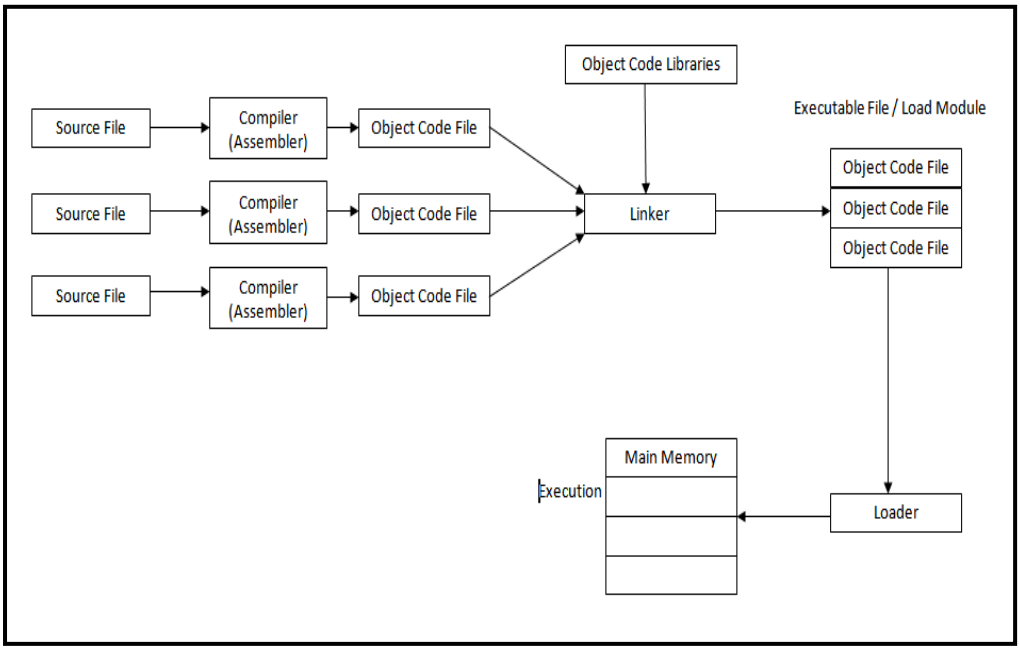


Figure -4 : Process of Linker and Loader

Differentiate between Compiler and Interpreter

Compiler	Interpreter
1) Compiler executes set of instructions at a time.	1) Interpreter executes only one instruction at a time.
2) Execution is faster.	2) Execution is slower.
3) It requires more memory for the generated intermediate object code.	3) It requires efficient memory as no Intermediate object code is generated.
4) It is difficult to debug.	4) It is easy to debug.
5) Locating an error is not instant.	5) Locating an error is instant.
6) C, C++ etc is an example of compiler	6) Python, BASIC, Ruby etc is an example of interpreter.

Compilation Process

The compilation process is a sequence of various phases such as Lexical analyzer, Syntax analyzer, Semantic analyzer, Intermediate code generator, Machine independent code optimizer, Code generator, Machine dependent code optimizer. Each phase takes input from its previous stage, has its own representation of source program, and feeds its output to the next phase of the compiler.

The different phases of compilation process are as

1) Lexical analysis
2) Syntax analysis
3) Semantic analysis
4) Intermediate code generation

5) Code optimization
6) Code generation

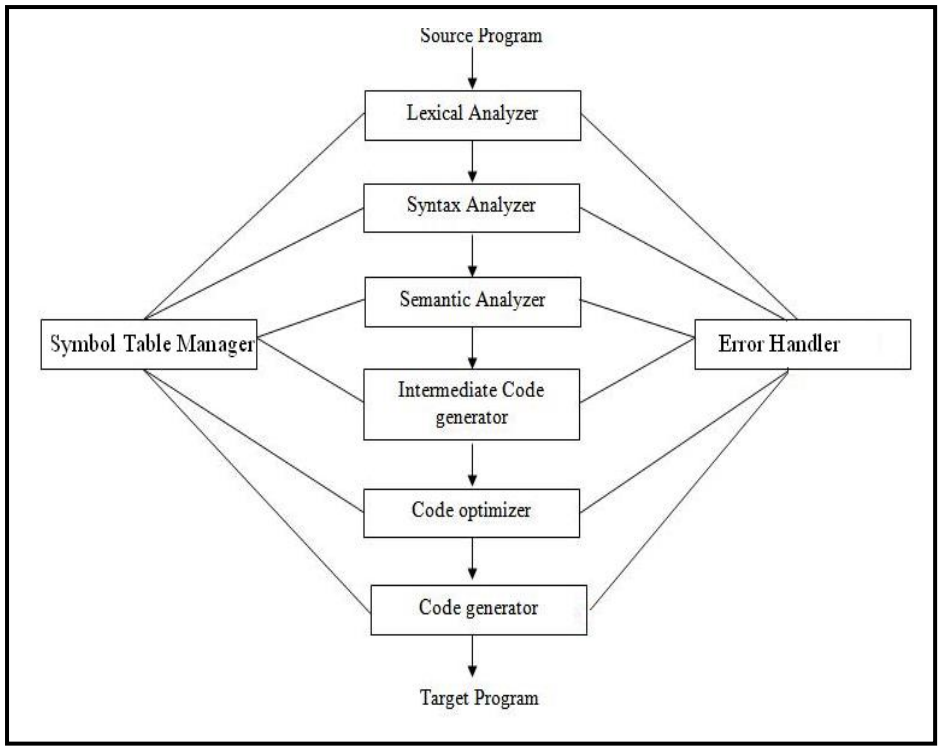


Figure – 5: Phases of Compilation process

1) Lexical analysis

This phase is the first phase of compiler process which scans the source program as a stream of characters and those characters are grouped to form a sequence called lexemes which produces token as output.

Token is a sequence of characters that represent lexical unit, which matches with the pattern, such as keywords, operators, and identifiers.

Lexeme is instance of token that is group of characters forming a token.

Form of tokens as

<Token-name, attribute-value>

Example c = a + b * 5;

Here, c, =, a, +, b, *, 5 are lexemes

a, b, c are identifiers, = is assignment symbol, + is addition symbol, * is multiplication symbol and 5 is number. These are called tokens.

Hence, <id, 1> < = > <id, 2> < +><id, 3> <* > < 5 >

2) Syntax analysis

This phase is the second phase of compiler process which is also called as parsing. Parser converts the tokens produced by lexical analyzer into a tree like representation called parse tree. A parse tree describes the syntactic structure of the input.

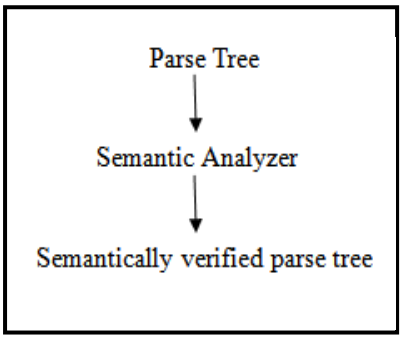


Figure – 6: Parse tree

Syntax tree is a compressed representation of the parse tree in which the operators appear as interior nodes and the operands of the operator are the children of the node for that operator.

Consider an expression c = a + b * 5;

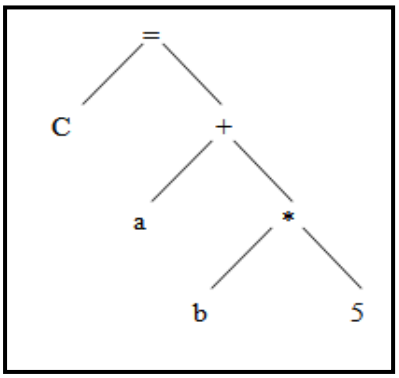


Figure – 7: Parse tree

3) Semantic analysis

Semantic analysis is the third phase of compiler. This phase checks for the semantic consistency. Type information is gathered and stored in symbol table or in syntax tree.

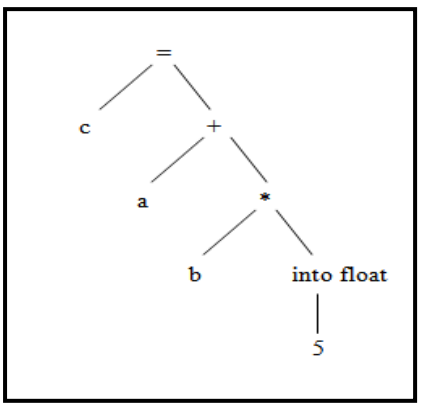


Figure – 8: Syntax tree

4) Intermediate code generator

Immediate code generation produces intermediate representations for the source program which are of the following forms-

- Postfix notation

- Three address code
- Syntax tree

Most commonly used form is the three address code.

t1 = int to float (5)

t2 = id3 * tl

t3 = id2 + t2

id1 = t3

Advantages of Intermediate code

a) It should easy to produce.

b) It should be easy to translate into target program.

5) Code optimization

Code optimization phase gets the intermediate code as input and produces optimized intermediate code as output. It results in faster running machine code. It can be done by reducing the number of lines of code for a program. This phase reduces the redundant code and attempts to improve the intermediate code so that faster-running machine code will result. During the code optimization, the result of program is not affected. To improve the code generation, the optimization involves-

- Deduction and removal of dead code (unreachable code).
- Calculation of constants in expressions and terms.
- Collapsing of repeated expression into temporary string.
- Loop unrolling.
- Moving code outside the loop.
- Removal of unwanted temporary variables.

t1 = id3 * 5.0

id1 = id2 + t1

6) Code generation

This phase is the final phase of compilation process. This phase gets input form code optimization phase and produces the target code or object code as result.

Intermediate instructions are translated into a sequence of machine instructions that performs the same task. The code generation involves-

- Allocation of register and memory.
- Generation of correct references.
- Generation of correct data types.
- Generation of missing code.

LDF R2, id3

MULF R2, #5.0

LDF R1, id2

ADDF R1, R2

STF id1, R1

Symbol Table Management and Error Handling

Symbol table is used to store all the information about identifiers used in the program. It is a data structure containing a record for each identifier, with fields for the attributes of the identifier. It allows finding the record for each identifier quickly and to store or retrieve data from that record. Whenever an identifier is detected in any of the phases, it is stored in the symbol table.

Example

int a, b; float c; char z;

Symbol Name	Type	Address
a	int	1000
b	int	1002
c	float	1004
z	char	1008

In Error handling, each phase can encounter errors. After detecting an error, a phase must handle the error so that compilation can proceed. In lexical analysis, errors occur in separation of tokens. In syntax analysis, errors occur during construction of syntax tree. In semantic analysis, errors occurs during type conversion, In code optimization, errors occur when the result is affected by the optimization. In code optimization, errors occur when the code is missing.

Study of High Level Language (HLL)

The first high level language was introduced in the 1950's.

A high level language is a programming language that enables development of a program in much user friendly programming and independent of the computer's hardware architecture.

High level languages are designed to be used by the human operator or the programmer. In other words, their programming style and context and context is easier to learn and implement than low level languages , and the entire code focuses on the specific program to be created.

A high level language does not require addressing hardware constraints when developing a program. Every single program written in a high level language must be interpreted into machine language before being executed by the computer.

Ada, Algol, BASIC, COBOL, FORTRAN, C/C++, JAVA, Pascal, Prolog etc are examples of high level languages.

Advantages

- It is easier to write
- It is easier to read
- It is easier to maintain

- It is portable.

Characteristics of Good Programming Language

Every computer requires appropriate instruction set (programs) to perform the required task. The quality of the processing depends upon the given instructions. If the instructions are improper or incorrect then it is the result will be superfluous.

A good programming language must be simple and easy to learn and use. It should provide a programmer with a clear, simple and unified set of concepts that can be grasped easily.

A good computer program should have following characteristics

1) **Portability**

   Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes.

2) **Readability**

   The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the programmers to understand their own program in a better way. Even if some computational efficiency needs to be sacrificed for better readability, it is advisable to use a more user-friendly approach.

3) **Efficiency**

   Every program requires certain processing time and memory to process the instructions and data. Code efficiency is directly linked with algorithmic efficiency and the speed of runtime execution for software. It is the key element in ensuring high performance.

4) **Structural**

   To develop a program, the task must be broken down into a number of subtasks. These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask. If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.

5) **Flexibility**

   A program should be flexible enough to handle most of the changes without having to rewrite the entire program. Most of the programs are developed for a certain period and they require modifications from time to time. For example, in case of payroll management, as the time progresses, some employees may leave the company while some others may join. Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.

6) **Generality**

Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain. For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.

7) Documentation

Documentation is one of the most important components of an application development. A well-documented application is also useful for other programmers because even in the absence of the author, they can understand it.

QUESTIONS

1. What is programming language?
2. What are the types of programming languages.
3. What is low level language?
4. Write advantages and disadvantages of low level language.
5. What is middle level language?
6. Write advantages and disadvantages of middle level language.
7. What is programming language translator?
8. What are the types of language translator? List them
9. What is compiler?
10. What is assembler?
11. What is Interpreter?
12. Define: Linker and Loader
13.

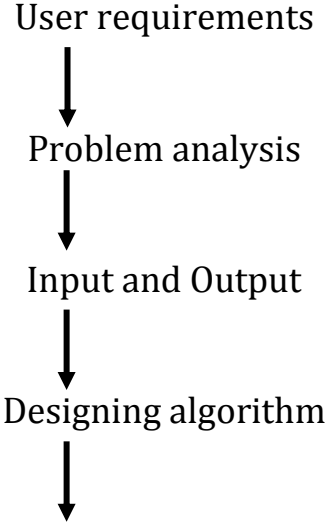
Programming Logic

Introduction Programming Logic and Techniques

1.1 Programming Logic

Software is a collection of programs and a program is a collection of instructions given to the computer. Development of software is a stepwise process. The first step is to understand the user requirements. Problem analysis arises during the requirements phase of software development. Problem analysis is done for obtaining the user requirements and to determine the input and output of the program.

For solving the problem , an " algorithm" is implemented. Algorithm is a sequence of steps that gives method of solving a problem. This "algorithm" creates the logic of program. On the basis of this "algorithm" , program code is written. The steps before writing program code are as –



Program coding

Fig shows 1.1 Process of program development

1.2 Design Methods

Designing is the first step for obtaining solution of a given problem.

The purpose of designing is to represent the solution for the system.

There are three types of Design Methods

1.2.1 Top-down design

1.2.2 Bottom-up design

1.2.3 Modular approach

1.2.1 Top-down Design

Top-Down method starts from top-level component to lowest level (bottom) component. In this design method , the system is divided into major components. Then each major component is divided into lower level components.

1.2.2 Bottom-Up Design

Bottom-Up design method is the reverse of Top-Down approach. It starts from the lowest level component to the highest-level component. It first designs the basic components and from these basic components the higher-level component are designed.

1.1.3 Modular Approach

In a programming , module is logically a well-defined part of program. Each module is a separate part of the program. It is easy to modify a program written with modular approach because changes in one module don't affect other modules of program. It is also easy to check bugs in the program in module level programming.

1.3 Program development Life Cycle

A program consists of a sequence of operations. It contains only those operations , which the computer can perform. Problem solving can be defined as " The task of expressing the solution of complex problems in terms of simple operations understood by the computer".

In order to solve a problem using a computer following stages as given below –

1. Problem definition
2. Problem analysis
3. Design a solution using design tools such as flowcharts and algorithms
4. Programming the computer (Coding)
5. Checking and Correcting errors (Testing and Debugging)
6. The documentation of the program
7. Program enhancement or maintenance

1. Problem definition

Problem definition requires to develop exact specification of the problem. In other words to extract from the problem statement a set of precisely defined tasks. The problem definition should be in the users language such as English or some other natural language. It may include charts , tables and equations of different kinds. The exact notation used to depends on the problem area.

2. Problem analysis

In this part of problem solving to understand the requirements of the problem to be solved. This process is the first step towards the solution domain. Explicit requirements about the input – output , time constraints , processing requirements, accuracy , memory limitations , error handling and interfaces are understood at this stage. The end result of this analysis is the selection of a method , which is to be used on the computer or a decision that a computer should not be used because of constraints as it may be seen that manual methods are better.

3. Design of Problem Solutions and Use of Design tools

The process of problem definition and problem analysis is complete to define the solution to the problem. The solution should include a sequence of well-defined steps that will input and manipulate the data and produce the desired output. The process of good designing can be done efficiently with the choice of certain design tools. Algorithm and Flowcharts are two design tools ,which help in the representation of a solution to a problem.

4. Coding

The problem has been analyzed it must be coded in a language which the computer can understand. This code is called a Program. Coding is the translation of algorithm into a suitable computer language such as C , Pascal , COBOL, FORTRAN etc.

5. Testing and Debugging

The program is created it must be complied and executed. During the complication process , errors are detected by the compiler. These errors are similar to grammatical errors in English and are known as syntax errors. When to informed by the compiler that such an error has occurred. They must determine what correction should be made to the source program, make the correction and then recompile. If the program compile correctly , then can processed with the execution of the resulting object program , which is in machine-readable form.

6. Documentation and Maintenance

Documentation is the process of writing explanation about the program in the form of comments and remarks. Documentation may be divided into two categories , namely technical documentation and user-level documentation. The technical documentation is meant for the programmer who may try to modify the program and it includes information about the formula used , data and programs imported from other programs, developed by different programmers. The user-level documentation helps the users, who may not be programmers , to understand the program and to make use of the program.

Program maintenance means periodic review of the programs to ensure that it meets the revised needs of the organization. Many a times to enhance the clarity of the programs fresh documentation may be needed. Changes in program may also be needed due to availability of new and more sophisticated equipment. A lot of time and money can be saved if attention is paid to maintenance of the programs which is a continuing task.

1.4 Development tools

There are two basic development tools , which help in the representation of a solution to a problem.

1.4.1 Algorithm

1.4.2 Flowcharts

Psedocode

1.4.1 Algorithm

1.4.1.1 Definition

Algorithm refers to the logic of a program. It is a step-by-step procedure to solve a given problem.

1.4.1.2 Characteristics of Algorithm

1. Each instruction should be precise and unambiguous (Confusion)
2. Each instruction should be executed in a finite time.
3. One or more instructions should not be repeated infinitely.
4. Simple statement and Structures should be used in the development of the algorithm
5. After executing the instructions , the desired results are obtained.

1.4.1.3 Advantages of algorithm

1. It is plain English language
2. It is step-by-step representation of a solution to a given problem
3. It is easy to understand the logic
4. It is first develop an algorithm, and then convert it into a flowchart and then into a computer program
5. It is independent of any programming language
6. It is every easy to create program
7. It is easy to debug as every step has got its own logical sequence

1.4.1.4 Disadvantages of algorithm

1. It is very time consuming
2. If not make appropriate changes in the process and repeat the process.

Example : Write an algorithm To find the area of a rectangle

1. Start
2. Read length
3. Read breadth

4. Area = length * breadth

5. Display area

6. Stop

1.4.2 Flowcharts

1.4.2.1 Definition

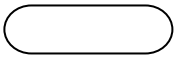
A flow chart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instruction.

This boxes are connected of solid lines having arrow marks to indicate the flow of operation. Thus the diagrammatic representation indicates the exact sequence in which the instruction are to be execute.

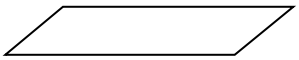
1.4.2.2 Flowchart symbols

There are some symbols which indicates the necessary operation in a flow charts, these symbols have been studies by **A**merican **N**ational **S**tandard **I**nstitute (**ANSI**). These symbols are as follows,

1. **Terminator:** This symbol is used to indicate the beginning (start) and ending (stop), pause (halt) in the program logic flow.

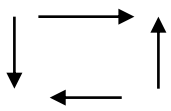


2. **Input / Output:** This symbol is used to denote any function of an Input / Output device in the program.

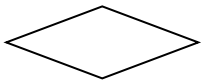


3. **Processing:** This symbol is used to represented data movement instruction. This symbol also used for logical process of moving data from one location of the main memory to another.

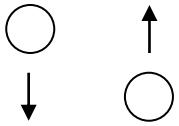
4. **Flow Line:** The flow lines with arrow heads are used to indicate the flow of operation, where the exact sequences of instruction are to be executed. The normal flow of flow chart from top to bottom and left to right.



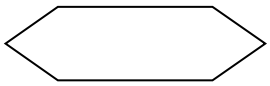
5. **Decision Box:** This symbol is used in a flow chart to indicate a point at which a decision has to be made.



6. **Connectors:** This symbol are do not represent any operation and their used only the sake of convenience and clarity.



7. Preparation [Looping] : This symbol is used an instruction or group of instructions which changes the program.



8. Predefined process : This symbol is a group of operations not detailed in the particular set of flowcharts.

1.4.2.3 Level of Flowchart

Flow chart having two types of level-

1. Micro Flow chart
2. Macro flow chart

A flow chart which shows less details is a macro flow chart and a flow chart with more details is called micro flow charts.

1.4.2.4 Rule of Flowchart

These are some rules for drawing a flow chart -

1. Draw the main line of logic and then incorporate the details.
2. Maintain a consistent level of details for a given flow chart.
3. The floe chart should always be in graphics representation. They should not be in every details.
4. The statements, which are used in a flow chart, should be easy to understand.

5. The flow chart should be consistence by using names and variable.
6. The flow of flow chart should always from left to right and top to bottom.
7. The flow chart should always in simple from i.e the crossing a flow lines should be avoided.

1.4.2.5 Limitations of Flowchart

There are limitation for drawing a flow chat -

1. Drawing flow chart with proper symbol and spacing for large complex program are very time.
2. Any change or modification in a program redrawing a flow chart is tedious job.
3. These are no standard to determine the amount of details that should be included in a flow charts.

1.4.2.6 Advantage of Flowchart

1. **Better Communication** : A flow chart is a pictorial representation of a program, it is easier for a programmer to explain the logic of a program to some other programmer.
2. **Effective Analysis** : A macro flow chart that charts the main line of logic of a software system becomes a system model that can be broken down into detailed parts for study and further analysis of the system.
3. **Effective Synthesis** : A group of programmers are normally associated with the design of big software system. Each programmer is responsible for designing only a part of the entire system. So initially, if each programmer draws a flow chart for his part of design, the flow chart of all the programmer can be placed together to visualize the overall system design.
4. **Proper Program Documentation** : Program documentation involves collecting organizing, storing and otherwise maintaining a complete historical record of programs and other documents associated with a system.
5. **Efficient Coding**: Once flow chart is ready, programmers, find it very easy to write the concerned program because the flow chart acts as a read 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.

6. **Systematic Debugging** : Even after taking full care in program design, some errors may remain in the program because the designer might have never thought about a particular case.
7. **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 set of data is fed as input to that program to test the different paths in the program paths.

1.4.2.7 Disadvantages of Flowcharts

1. Complex logic
2. Alternation and Modification
3. Reproduction
4. Link between conditions and actions

1.5 Pseudo code

1.5.1 Definition

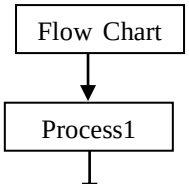
Pseudo code is another programming analysis tool that is used for planning program logic "Pseudo" means imitation or false and "code" refers to the instruction written in a programming language. Pseudo code is also called program design language(PDL). This Pseudo code is an imitation of actual computer instruction.

1.5.2 Classification of Pseudo code

Pseudo code is made up of the following basic logic,

1. Sequence
2. Selection (If----Then-----Else)
3. Iteration (Do----While Or Repeat----Until)

1. Sequence Logic : Sequence logic is also known as sequence process logic. This logic is used for performing instruction one after another on a sequence. The flow of pseudo code is from top to bottom.



Pseudo code

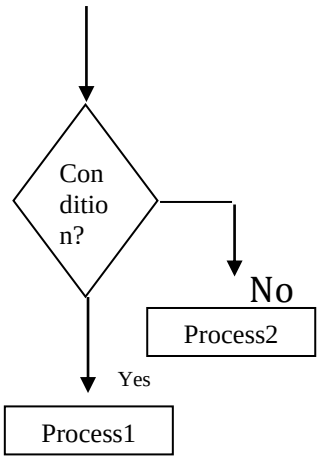
Process1

Process2

2. Selection Logic: Selection logic is also known as decision logic, it is used for making decision, selection logic, selecting proper path out of two or more alternative path in a program logic selection logic used either IF---THEN---ELSE condition structure.

The IF---THEN---ELSE structure says that if the condition is true then do process1 and if it is not then skip over from process. The E?NDIF is used to indicate the end of the decision structure.

Flow Chart



Pseudo code

```
If condition=true then
        Process1
Else
        Process2
End if
```

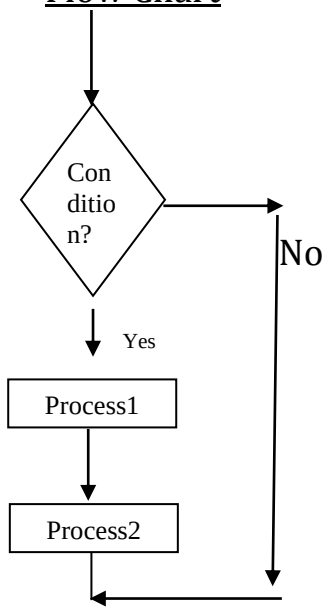
3. Iteration Logic: Iteration logic is used to produce loop when one or more instruction may be executed several times depending on same condition. This logic uses two structure

a. Do-----While
b. Repeat------Until

The Do---While is used for looping. The looping continuous as long as the condition is not true. In Do----While structure that will change the condition that

controls the loop. The condition is tested of the top of the loops. The ENDDO marks the end of the Do----While loop.

Flow Chart



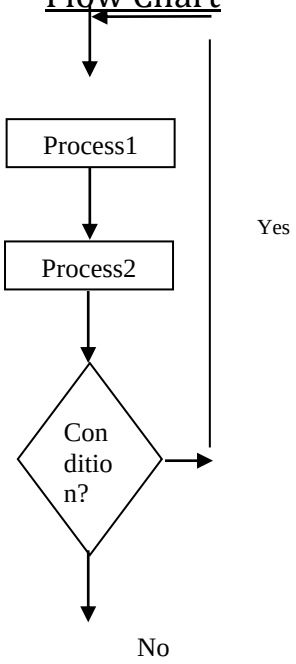
Pseudo code

```
.....
Do while condition is true
        Process1
        Process2
        .....
ENDDO
```

Repeat……Until:

The Repeat….Until is used for looping. The looping until the condition becomes true. That is, the execution of the statement with in the loop is repeated as long as the condition is not true. The condition is tested of the bottom of the loop. Until followed by some condition marks the end of the Repeat….Until structure.

Flow Chart



Pseudo code

```
.....
Repeat
Process1
Process2
.....
Until condition
```

1.5.3 Advantages of Pseudo code

1. Converting a pseudo code to a programming language is much more easier as compared to converting a flowchart.
2. As compared to a flow chart, it is easier to modify the Pseudo code of a program logic when program modification are necessary/.
3. Writing of Pseudo code involves much less times and effect than drawing an equivalent flow chart.

1.5.4 Limitations of pseudo code

1. In case of Pseudo code, a graphics representation of program logic is not available.
2. There are no standard rules to follows in using Pseudo code.
3. For a beginner it is more difficult to follow the logic of or write Pseudo code, as compared to flow chart.

1.7 Language Translator

Language translators perform the translation of high level languages or assembly language into machine language.

There are two types of translator. They are

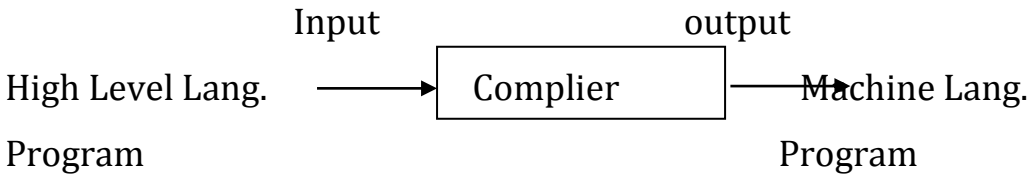
1.7.1 Compiler

1.7.2 Interpreter

1.7.3 Assembler

1.7.1 Complier

A complier is a translator program, which translate a high level language program into its equivalent machine language program. A compiler a set of machine language instruction for every program instruction of a high level language . As shown in fig



In the above fig the input to the complier is the high level language program (called source program) and its output is the machine language program (called object program). High level instruction are macro instruction , the

compiler translate each high level language instruction into a set of machine language instruction. During the process of translation of a source program into its equivalent object program by the complier , the source program is not being executed. It is only converted into a form which can be executed by the computer processor. Complier are large program which is permanently on secondary storage. When source program is to be translated , the complier and source program are copied from secondary storage into the machine language of the computer. The complier is executed with the source program as its input data. It generates the equivalent object program as its output , which is saved in a file on secondary storage. There is no need to execute the program , the object program is copied from secondary storage into the main memory of the computer and executed. There is no need to repeat the compilation process every time to execute the program, reason is that the object program stored on secondary storage is already in machine language . To load the object program from the secondary storage into the main memory of the computer and execute it directly. The compilation is necessary to modify the program. If changes in the program , it must load the original source program from secondary storage into the main memory of the computer , recompile the modified source program and create and store an updated object program for execution.

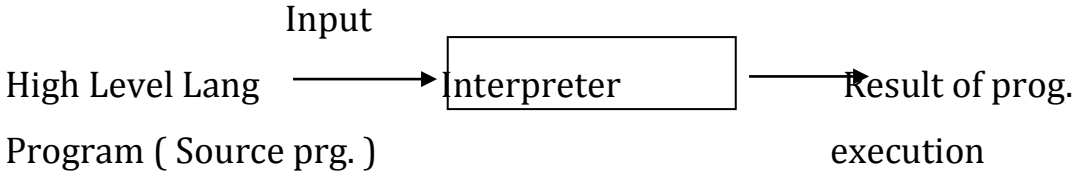
For translating high level instructions into machine language instructions , compiler also automatically detect and indicate types of errors in source program. These error are syntax error . They are-

1. Illegal character
2. Illegal combination of characters
3. improper sequencing of instruction in a program
4. Use of undefined variable name

A variable program containing one or more errors detected by the compiler will not be complied into the object program. In this case , the compiler will generate the list of coded error messages indicating type of errors committed. This error list is an invaluable to the programmer in correcting the program errors.

1.7.2 Interpreter

A interpreter is a type translator which is used for translating program written in high level language . It takes one statement of a high level language program translate it into machine language instructions , and then immediately executes the resulting machine language instructions . As shown in fig



In the above fig the input to an interpreter is the source program. No object program is saved for future use, repeated interpretation of a program is necessary for its repeated execution. An interpreter translate and executes a high level language program statement by statement a program , a program statement must be reinterpreted every time it is encountered during program execution . Interpreter are easier to write because they are less complex program then compliers. They also require less memory space for execution then compilers.

1.7.3 Assembler

Assembler is a software which translate a program written in assembly language in to its equivalent in machine language. The program written in assembly language deals with a low level language. An assembler translate the complete source program into an object program, identifying any errors. There are any errors the assembler will list or display these errors as well as the complete source and object programs.

1.8 Differentiate between Interpreter and Compiler

Interpreter	Compiler
1. It executes one instruction at a time.	1. It executes the complete instructions or set of instructions at a time.
2. It is a program that translates the English like statements f a HLL into the MLL of a computer.	2. It is a program that translates the English like statements of a HLL into the MLL of a computer.
3. It is easy to debug.	3. It is difficult to debug as compared with interpreter.
4. it is slower .	4. It is faster.
5. It used for languages such as BASIC.	5. It is use for languages such as FORTRAN , PASCAL , COBOL etc

6. Transaction & execution is simultaneous.	6. Transaction and execution are done at two different instances.
7. No object file is generated.	7. Object file is created.
8. Locating an error is instant.	8. Locating an error is not instant.
9. BASIC is an example of Interpreter.	9. TURBOC , PASCAL is an example of Compiler.

1.9 Solved Examples based on algorithm and flowchart

1.9.1 Design an algorithm to calculate the simple interest , given the principal [P] , rate [R] and time [T].

Step 1 - Start

Step 2 - Input P, T, and R

Step 3 - [Calculate]

SI ← P * T * R / 100

Step 4 - Output SI

Step 5 - Stop

1.9.2 Design an algorithm to find the average of four numbers

Step 1 - Start

Step 2 - Input A, B, and C

Step 3 - if((A > B) and (A > C)) then

Output A

else if B > C then

Output C

endif [End of if structure]

Step 4 - Stop

1.9.3 Design an algorithm to print all numbers from 1 to N.

Step 1 - Start

Step 2 - Input N

Step 3 I 1

Step 4 - Output I

Step 5 I I + 1

Step 6 - if(I <=N) then

Goto Step 4

endif [End of if structure]

Step 7 - Stop

1.9.4 Design a flowchart and an algorithm to find the sum of three numbers.

Algorithm

Step 1 Start

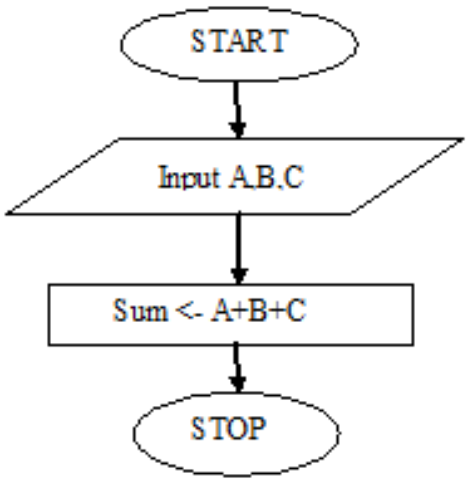
Step 2 Input A,B,C

Step 3 [Compute] Sum A+B+C

Step 4 Output Sum

Step 5 Stop

Flowchart



1.9.5 Design a flowchart and an algorithm to find the area of a square.

Algorithm

Step 1 Start

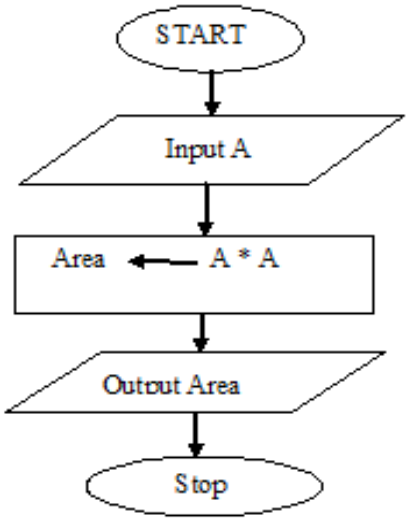
Step 2 Input the value for a [side]

Step 3 [Compute] ~~Area~~a a * a

Step 4 Output Area

Step 5 Stop

Flowchart



1.9.6 Design a flowchart and an algorithm to find the largest of two numbers.

Algorithm

Step 1 Start

Step 2 Input the values for A and B

Step 3 if A > B then

Output A

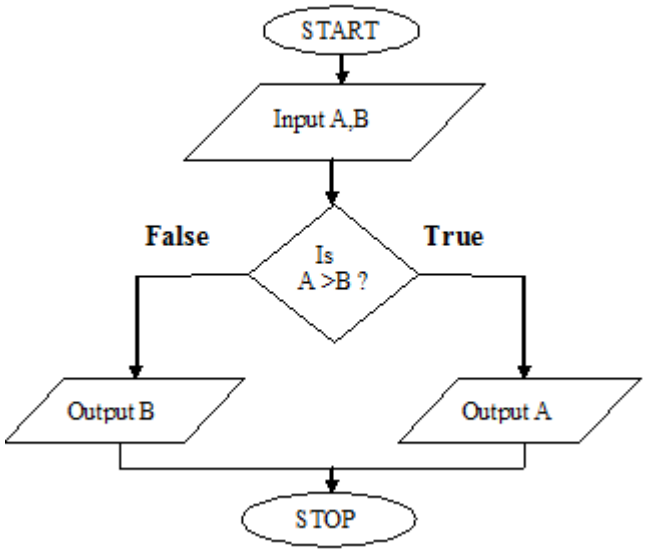
Else

Output B

end if

Step 4 Stop

Flowchart



1.9.7 Develop a flowchart and an algorithm to generate N natural numbers.

Algorithm

Step 1 Start

Step 2 Input N

Step 3 [Initialize]

Cont ← 1

Step 4 Output Count

Step 5 [Increment Counter]

Count ←Count + 1

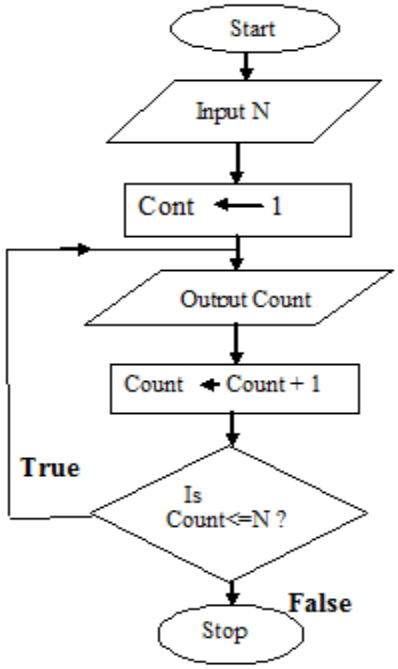
Step 6 if Count <= N

Goto Step 4

[End if]

Step 7 Stop

Flowchart



1.9.8 Develop a flowchart and an algorithm to reverse a given number [example N=12345 Reversed number=54321].

Algorithm

Step 1 Start

Step 2 Input NUM

Step 3 [Initialize] ~~REV~~ 0

Step 4 Repeat Step 5 While NUM <>0

Step 5 [Compute]

a. [Remainder] R← MOD[NUM,10]

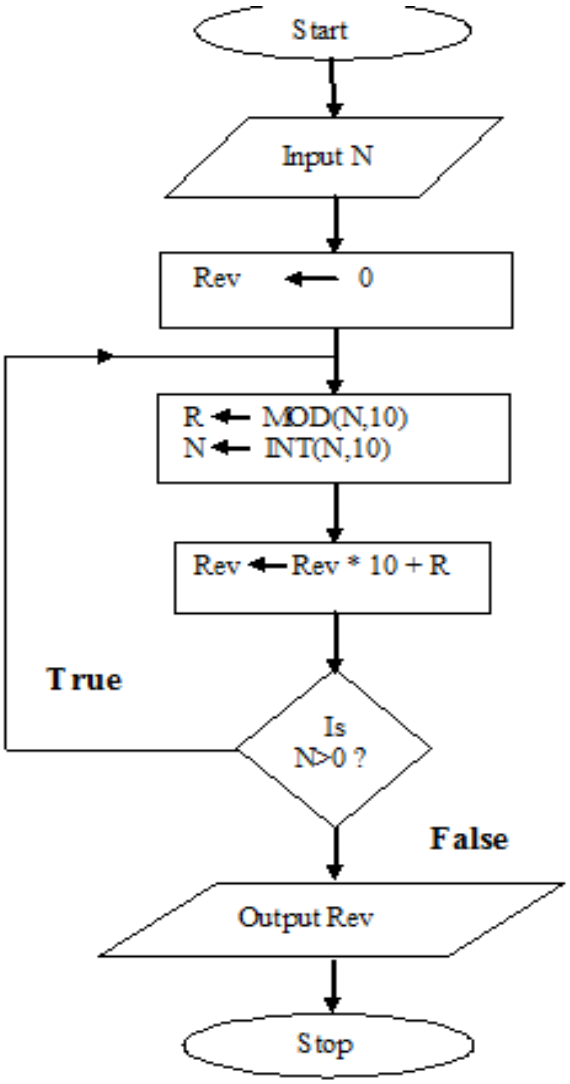
b. [Quotient] NUM ← INT [NUM / 10]

c. REV ← REV * 10 + R

Step 6 Output Rev

Step 7 Stop

Flowchart


Start
Input N
Rev ← 0
R ← MOD(N,10)
N ← INT(N,10)
Rev ← Rev * 10 + R
True
Is
N>0 ?
False
Output Rev
Stop