Concept of Algorithm>>>

C.P.PATEL & F.H.SHAH COMMERCE COLLEGE
(MANAGED BY SARDAR PATEL EDUCATION TRUST)
BCA, BBA(ITM) & PGDCA PROGRAMME
BCA SEM : 01 (Programming Fundamental Using "C")
UNIT 01 : Concept of Algorithm, Flowchart and Languages

UNIT 01 : Concept of Algorithm, Flowchart and Languages	
Sr No.	Topics
1	• Concept of an algorithm and a flow chart
2	• Need and definition and characteristics of algorithm
3	• Flow chart: Symbols, Rules, Advantages, Disadvantages
4	• Typical examples of flow charts and algorithms(*Refer to Journal list marked ****)
5	• Generalization of Computer language
6	• High level and low level languages and Assembly language: Concept, Advantages, Disadvantages
7	• Translators: Interpreter, Compiler, Assembler
8	• Introduction to Editors : Definition, Examples(only names)

Reference Books: Books:
1. Balagurusami: Programming in ANSI C Tata McGraw Hill Publication 3rd Edition
2. Kernighan B., Ritchie D:THe C programming Language,Prentice Hall
3. Computer Fundamentals 4th Edition P.K. Sinha, Priti Sinha

Concept of Algorithm and Flow Chart Development

Need of Algorithm and Flow Chart OR Purpose of Problem Planning

Suppose you are asked by your teacher to solve an arithmetic problem and you are not familiar with the steps involved in solving the problem. In such a situation, you will not be able to solve the problem. The same principle applies in writing a 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 write each and every instruction in the proper sequence.

Definition 1: **Algorithm can be defined as a finite sequence of steps written in simple English language on paper which, if followed enable a particular task to be accomplished.**

Definition 2: An algorithm is composed of a finite set of steps, each of which may require one or more operations.

Definition 3: An 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.

An Algorithm produces one or more outputs (O/P) and may have zero or more inputs (I/P), which are externally supplied.

The instructions in the algorithm should be unambiguous and the result should be obtained after a finite number of executable steps i.e. an algorithm must terminate and should not repeat one or more instructions infinitely. In short, the algorithm represents the logic of the processing to be performed.

There are various ways in which an algorithm can be expressed. When an algorithm is expressed in a programming language, it becomes a program. Algorithms are often expressed in the form of what is known as a Flow Chart.

The following points must satisfy along with all the algorithms:

1. **Input:** The user must externally supply one or more quantities.
2. **Output:** At least one quantity must be produced as output.
3. **Definite:** Each of the instructions mentioned must be clear and unambiguous.
4. **Finite:** On going through all the instructions of any algorithm one by one, the algorithm must end after the execution of a finite number of steps.
5. **Effective:** All the instructions must be basic so that they can easily be performed on paper.
6. **Feasible:** Each operation must also be feasible apart from being definite.

Rules for writing Algorithms:

1. **START – STOP:** First step of any algorithm must be START and last step must be STOP, which indicates starting or beginning of an algorithm and End point or stop algorithm process.

2. **INPUT – OUTPUT:** Input (I/P) to the algorithm and output (O/P) from the algorithm should be indicated by READ and WRITE words. E.g. suppose we want to input two numbers to the algorithm and final answer we want to print after processing then by
   READ A, B
   statements (steps) two inputted numbers stored in two variables A and B and suppose we want to store the result in another variable C then we can print the result by
   WRITE C
   statement (step). In both the statements more then one variables (e.g. A and B in READ statement) should be separated by comma symbol only.

3. **Condition (Decision):** In some cases there is need for making condition which results in either **YES (TRUE) or NO (FALSE)** only. And depending on the answer remaining steps follows.

4. **Looping:** In some cases there is need for executing or repeating one or more steps for the number of times, so in that case you can use
   Repeat – Until Loop

FLOW CHART:

Definition:

"A Flow Chart is a diagrammatic representation of the algorithm, using a standard set of symbols".

OR

"A Flow Chart is a pictorial representation of an algorithm in which boxes of different shapes are used to denote different types of operations".

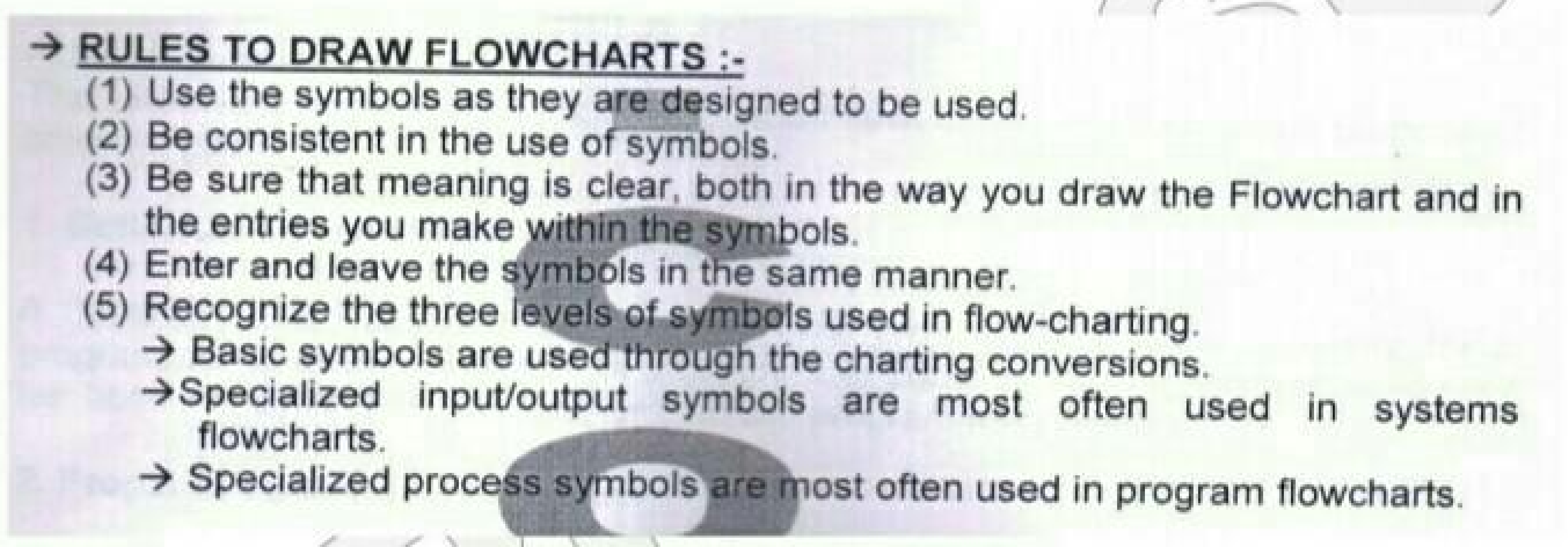
The actual instructions are written within these boxes using clear and concise statements. These boxes are connected with solid lines having arrow marks to indicate the flow of operation that is the exact sequence in which the instructions are to be executed.

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 these two step approach in program writing is that while drawing flowchart one in 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 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 therefore, 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 a step 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.



→ **RULES TO DRAW FLOWCHARTS :-**

(1) Use the symbols as they are designed to be used.
(2) Be consistent in the use of symbols.
(3) Be sure that meaning is clear, both in the way you draw the Flowchart and in the entries you make within the symbols.
(4) Enter and leave the symbols in the same manner.
(5) Recognize the three levels of symbols used in flow-charting.
→ Basic symbols are used through the charting conversions.
→ Specialized input/output symbols are most often used in systems flowcharts.
→ Specialized process symbols are most often used in program flowcharts.

→ **CONSTRUCTING FLOWCHARTS:-**

Eight steps that will insure complete preparation and execution of a flowchart are:

1. Research the problem
2. Decide on the method of solution
3. Plan the solution
4. Outline the system by drawing the system flowchart
5. Check the system flowchart
6. Draw a detailed program flowchart for each program
7. Check the program flowchart.
8. Test the program flowchart and system flowchart by using data that has been prepared.

As preparation and planning processed, more information may be needed. These steps may not be treated as an absolute rule.

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. Hence, 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. Owing 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.

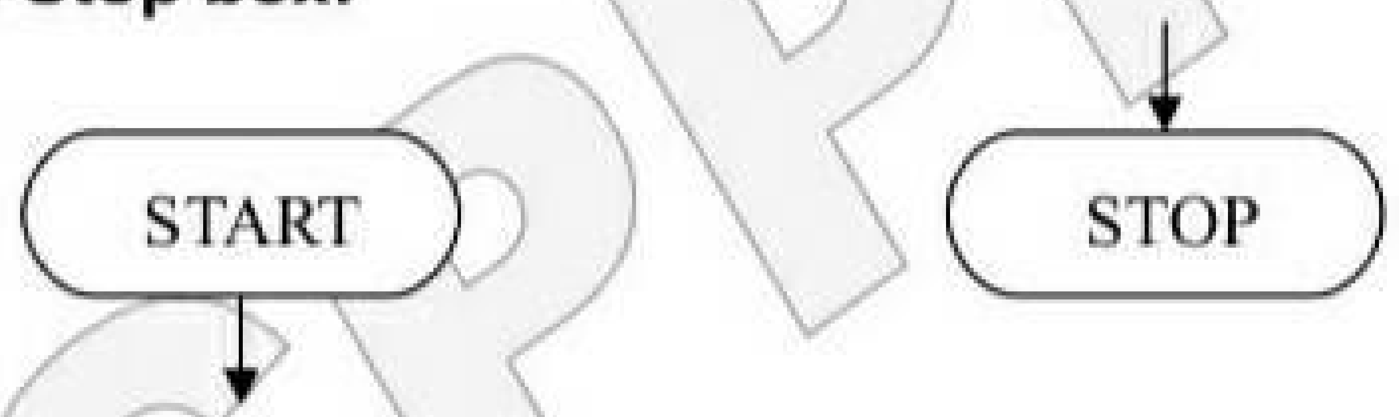
These software tools read the program's instructions and draw a flowchart of its logic. That is, this is a backward approach in which flowcharts are drawn from program codes mainly for documentation purpose.

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

Symbols used to draw a Flow Chart:

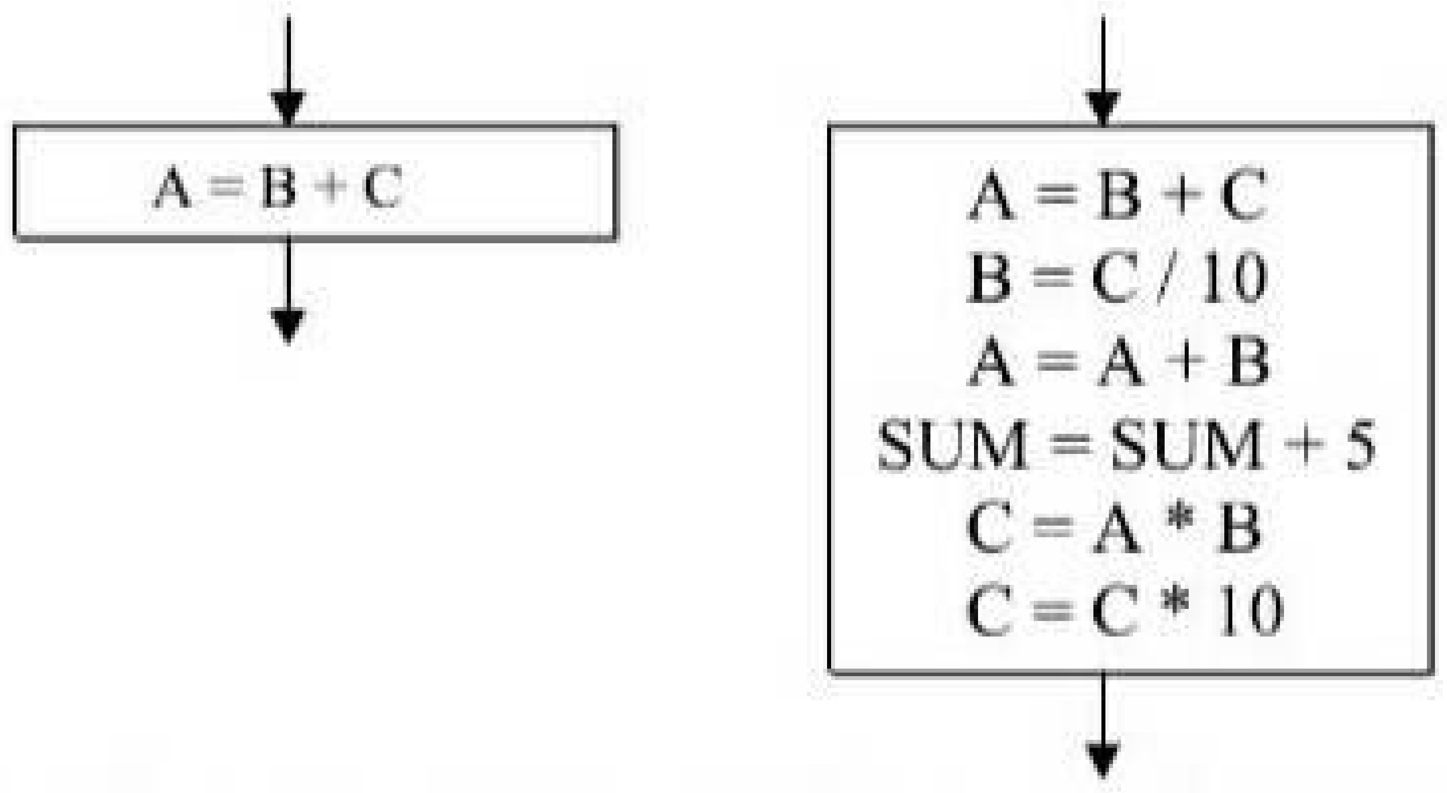
Shapes of the boxes used in a flowchart are relatively standard, many variations are also in existence. We shall use the following boxes:

1. Start-Stop box:



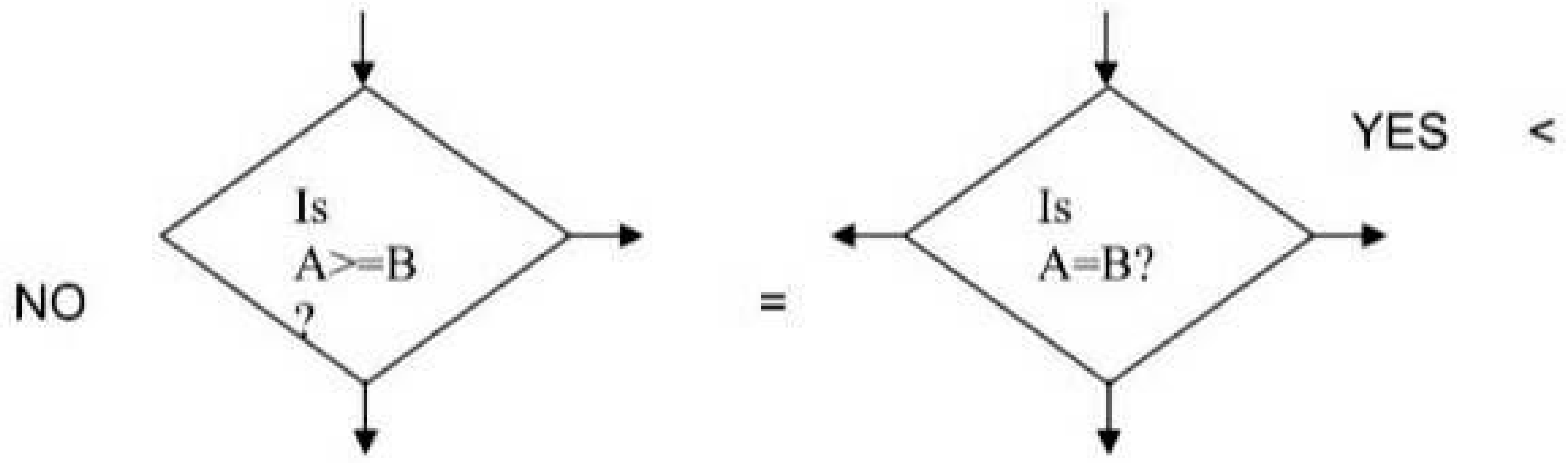
Used to indicate the point at which an algorithm begins and the point where it terminates or stops.

2. Process (Assignment) box:



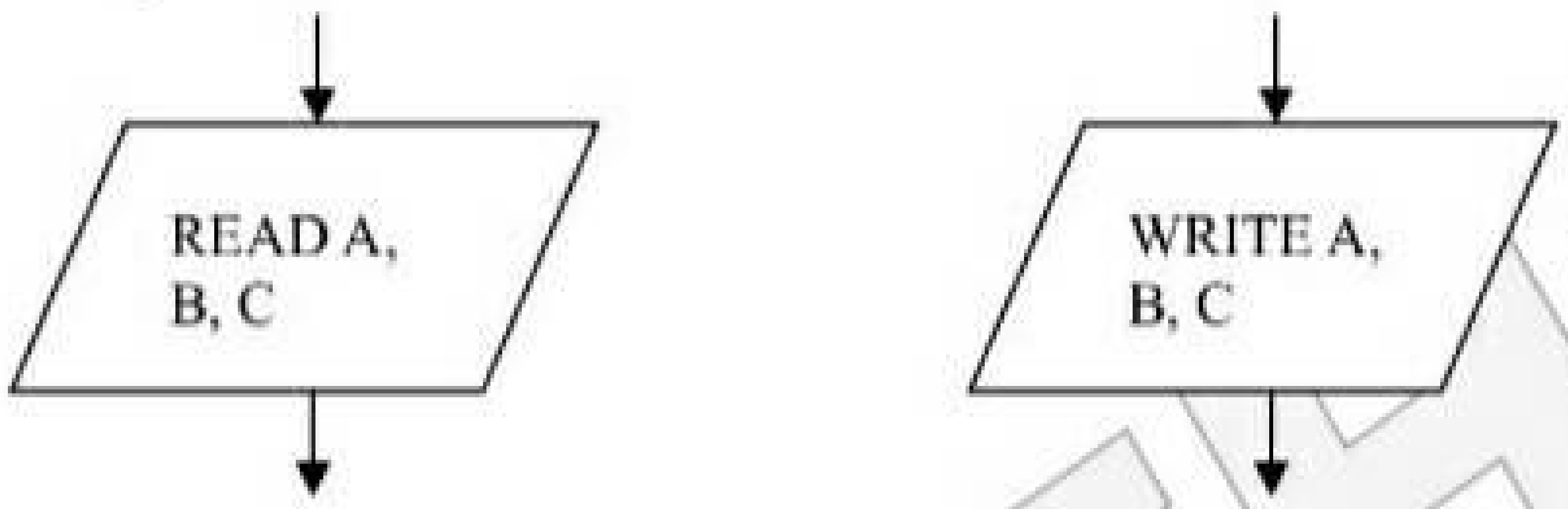
Used to indicate straight forward computation of certain quantity. A rectangle is used to represent process box.

3. Decision box:



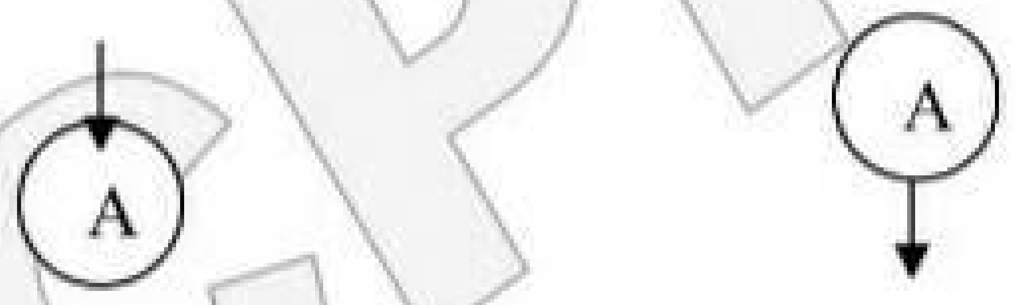
The point at which a decision has to be made and the algorithm has to select between two or three branches leading to the other parts of a decision box indicates the flow chart. A diamond shape is used to represent such a box.

4. Input-Output box:



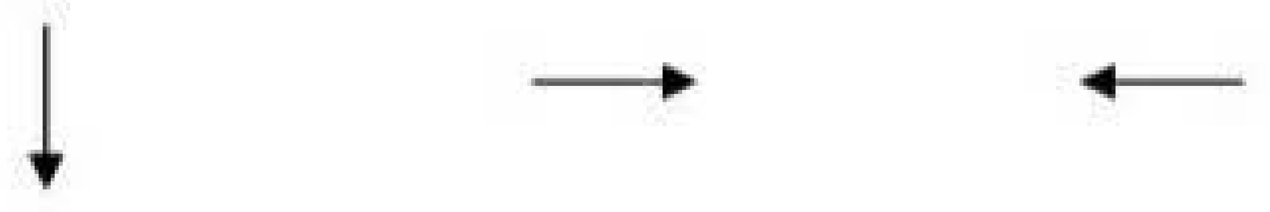
The point at which values of some data items have to be read or some results written are indicated by input / output boxes. Both the boxes are represented by parallelogram. The box on the left-hand side is a read box, which indicates the read operation. The box on the right-hand side is an output box.

5. Connectors:



Frequently a flow chart becomes too long to fit in a single page Thus when a flowchart spreads over more than one page, connector boxes are used to serve as links among sections in different pages.
A Circle is used to represent a connector and a letter or digit is placed within the circle to indicate the link.

6. Flowlines:



Flow lines 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.

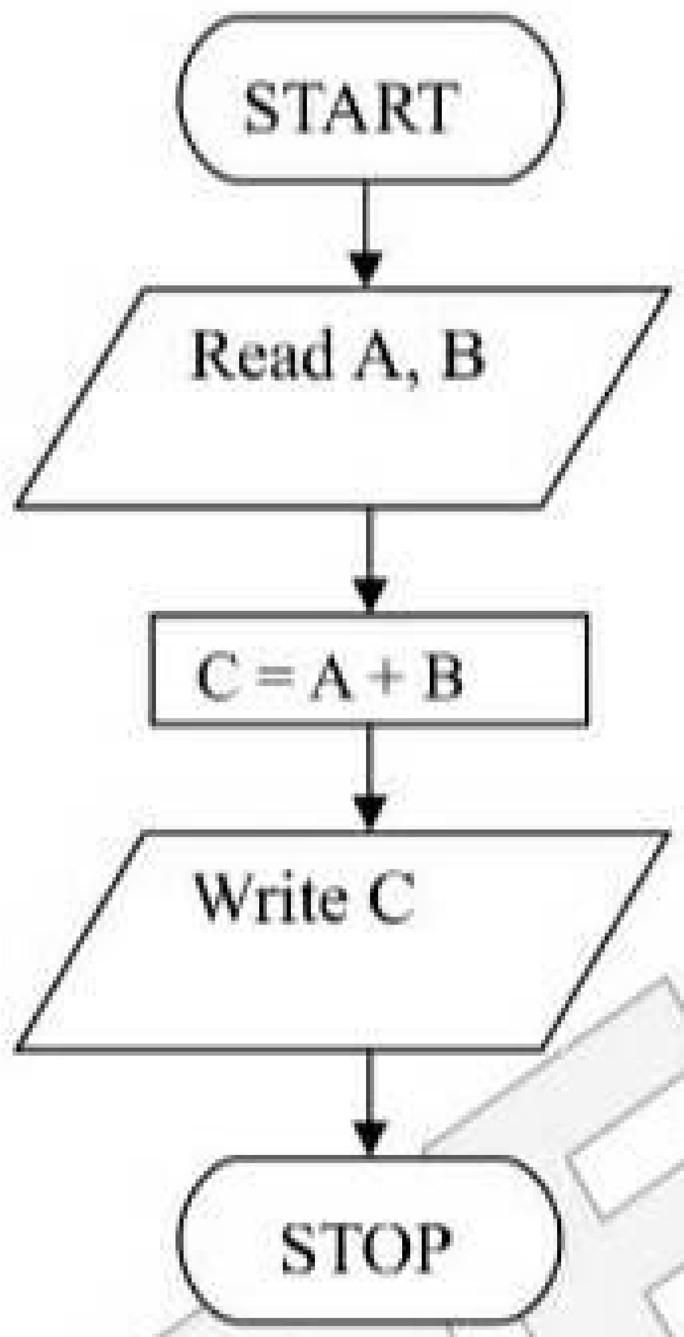
EXAMPLES:

1. Read two numbers and add them and print the result.

Algorithm: [Add two numbers]

```
Step 1: START
Step 2: Read A, B
Step 3: C = A + B
Step 4: Write C
Step 5: STOP
```

Flowchart: [Add two numbers]

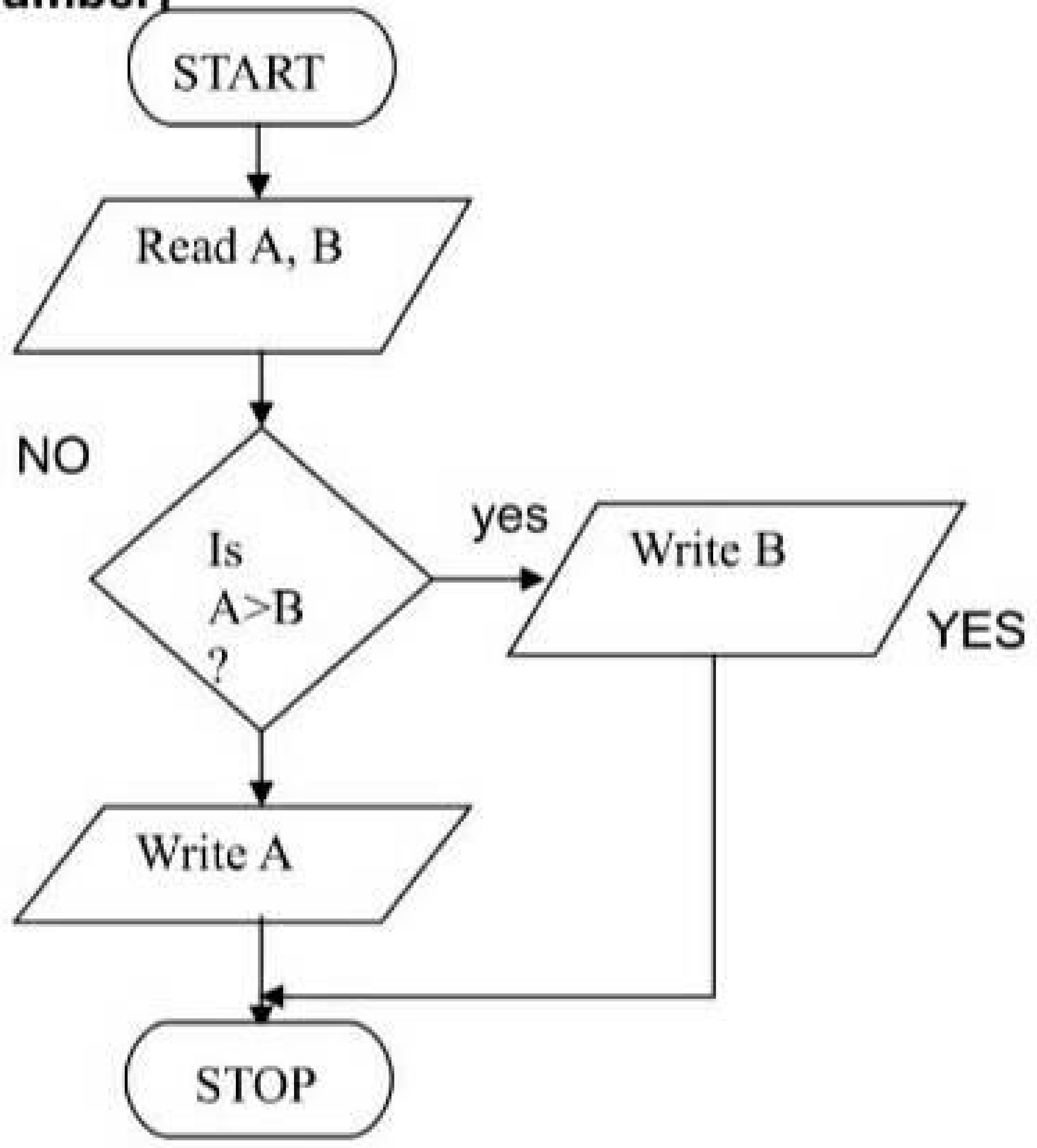


2. Read two numbers and finds larger number and print larger number.

Algorithm: [Find larger number]

```
Step 1: START
Step 2: Read A, B
Step 3: Is A > B No, Go to step 6.
Step 4: Write ('A is larger')
Step 5: Go to step 8
Step 6: Write ('B is larger')
Step 7: STOP
```

Flowchart: [Find larger number]

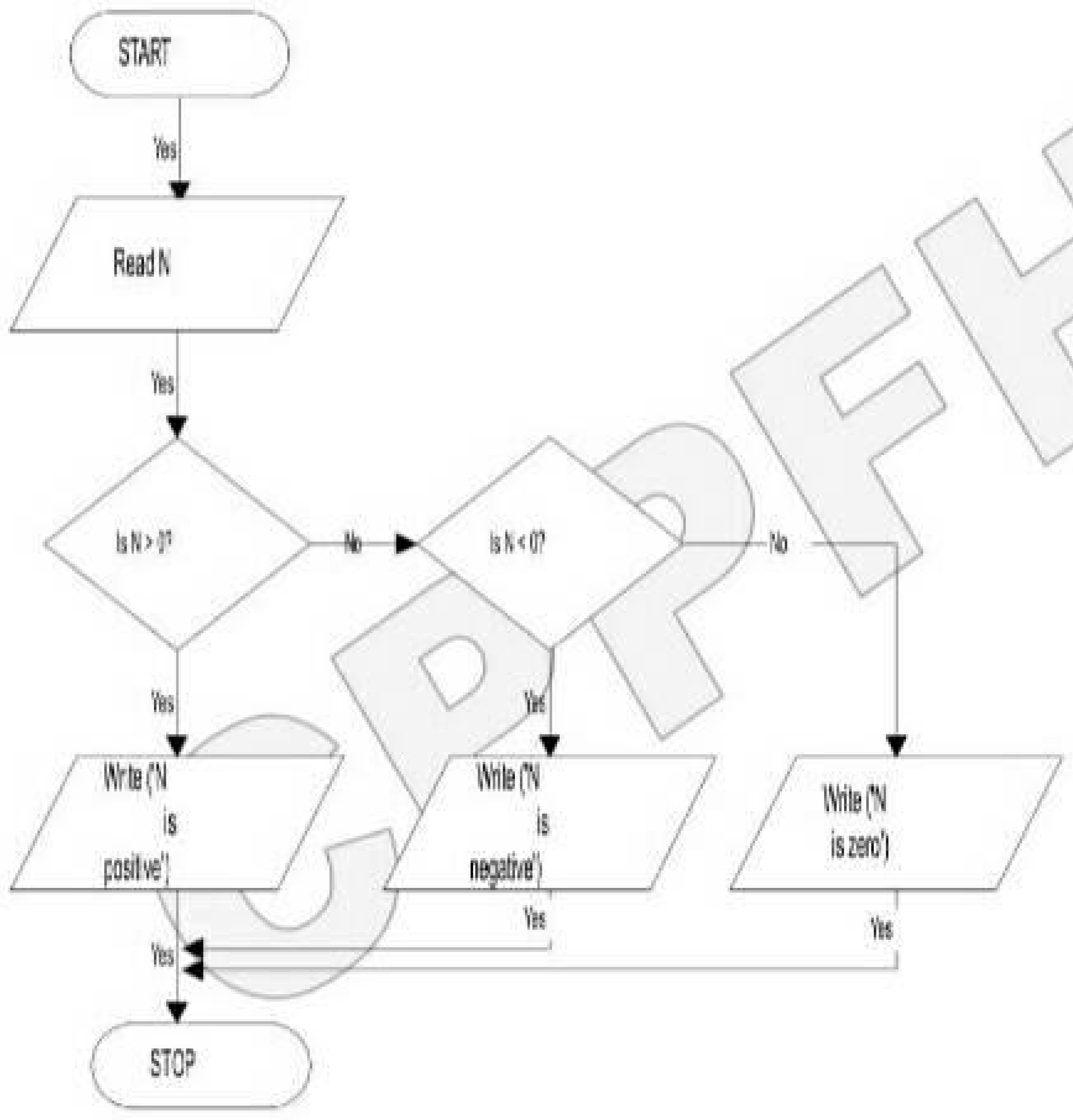


3. To determine whether the given number is positive, negative or zero.

Algorithm: [Positive, Negative or Zero]

```
Step 1: START
Step 2: Read N
Step 3: Is N > 0 No. Go to step 5.
Step 4: Write ('N is positive'). Go to step 8
Step 5: Is N < 0 No. Go to step 7
Step 6: Write ('N is negative'). Go to step 8
Step 7: Write ("N is zero')
Step 8: STOP
```

Flowchart: [Positive, Negative or Zero]

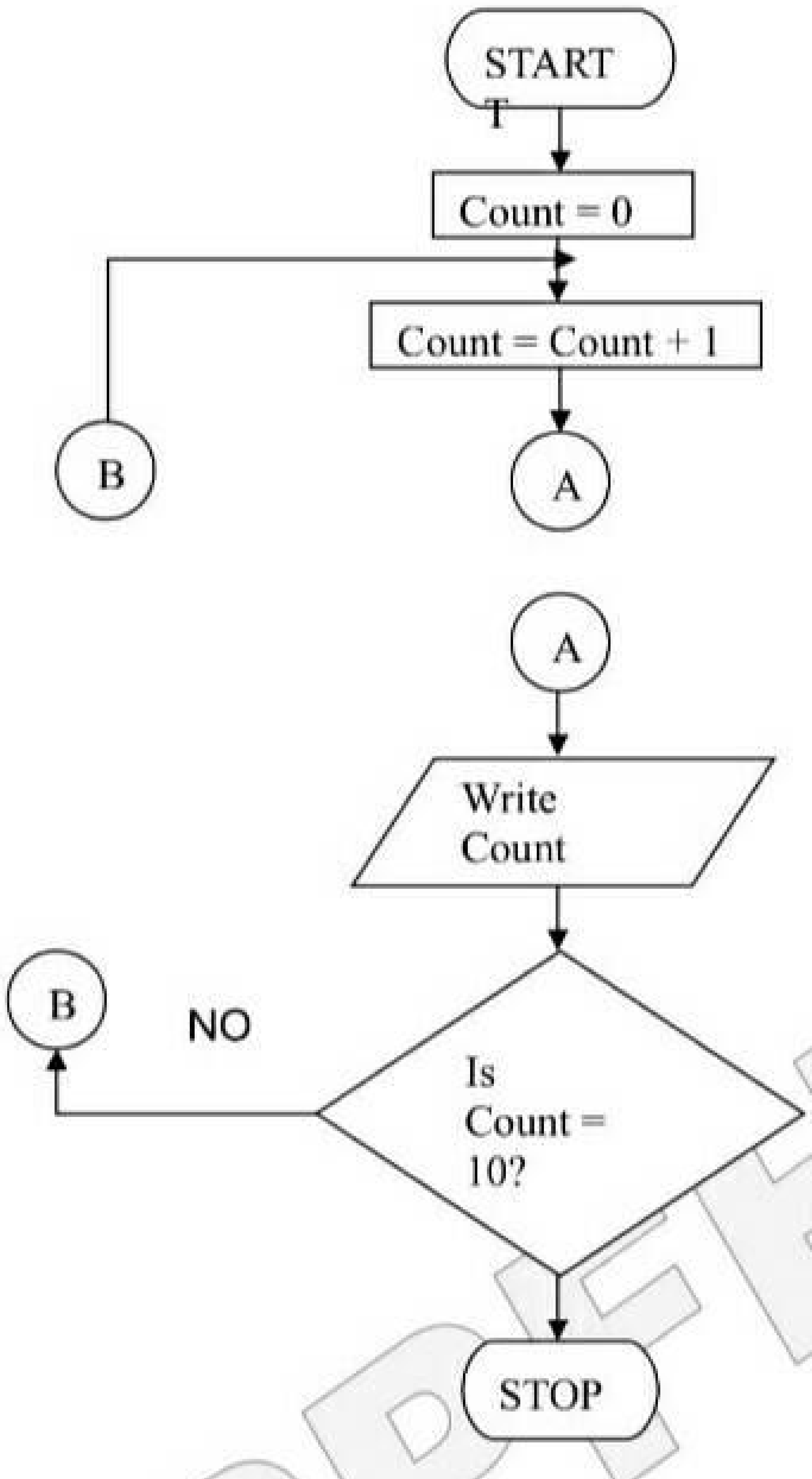


4. Write algorithm and draw flowchart for printing first 10 natural numbers.

Algorithm: [Print first 10 natural nos.]

```
Step 1: START
Step 2: Count = 0
Step 3: Count = Count + 1
Step 4: Write Count
Step 5: Is Count = 10? No. Go to step 3
Step 6: STOP
```

Flowchart: [Print first 10 natural nos.]



- **Generation of Computer Languages>>>**

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. In the early days of computing, it was assumed that only a few elite technical specialists would learn to use computers, but now their use by a larger proportion of population is taken for granted.
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) – including a whole range of query languages & other tools.**

Computer languages can be classified into 3 categories:

1. Machine language
2. Assembly language
3. High – level language

- Machine languages:

The set of instructions codes, whether in binary or in decimal notation, which can be directly understood by the computer without the help of a translating program is called a machine code or machine language program.
A computer understands information composed of only zeros and ones. This means that a computer uses binary digits for its operation. 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 binary code (machine code or object code) for a certain operation differs from one computer to another.
Two part machine code:
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. 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 operations involve reading, adding, subtracting, writing and so on.

- **Advantages of machine language:**

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

- **Disadvantages of machine language:**

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, the programmer will be required to learn a new machine code and would have to write all the existing programs again, in case the computer system.

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 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 expert who knows about the hardware structure of the computer.

Error prone:

For writing a program in machine language, the programmer not only has to remember the opcodes, but also has to 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 frequently causes errors in programming.

Difficult to modify:

It is very difficult to correct or modify machine language programs. Checking machine instructions to locate errors is about as tedious as writing them initially. 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.

❖ **Assembly language:**

The instructions, words which direct the computer are stored in the machine in numerical form. The programmer, however, 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 as written by the programmer therefore consist of two parts

a) The **operation code** part that designates the operation (addition, subtraction, multiplication etc.) to be performed.
b) The **address** of the number 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.

- **Advantages of assembly language:**

a) 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.

b) **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.
c) **Easy to locate and correct errors**: It is easier to find and correct errors because of the use of mnemonics and symbolic field names.
d) **Easier to modify:** Assembly language program are easier for people to modify than machine language program. This is mainly because they are easier to understand and it is easier to locate, correct and modify instructions as and when desired.

• **Disadvantages of Assembly language:**

a) Programming is **difficult and time consuming.**
b) Assembly languages are **machine dependent.** The programmer must have detailed knowledge of the structure of computer he is using. He must have the knowledge of registers and instruction sets of the computer, connection of ports to the peripherals etc.
c) 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 its own assembly language.
d) 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 an assembly language program.
e) In case of an assembly language program, instructions are still written at the machine – code level – that is one assembler instruction is substituted for one machine – code instruction.

❖ **High – level languages (HLL)**

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, etc.

• **Advantages of high – level language:**

a) **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.

b) **Portability:** High – level languages are independent of computer architecture. The same program will run on any other computer that has a compiler for that language. The compiler is machine dependent and not language dependent.

c) **Easy 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. The programmer need not learn any thing about the computer he is going to use. He need not worry about how to store his numbers in the computer, where to store them, what to do with them, etc. That is the programmer need not know the machine instructions, the data formats, and so on.

d) **Fewer errors:** In case of high – level languages, since the programmer need not write all the small steps carried out by the computer, he is much less likely to make errors made by the programmer. So errors can be easily located and corrected by programmer.

e) **Lower program preparation cost:** Writing programs in high – level languages requires less time and effort which ultimately leads to lower program preparation cost

- **Disadvantages of high – level languages**
  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**.

❖ **Translators>>>**

❖ **Compiler:**

Since a computer hardware is capable of understanding only machine level instructions, so it is necessary to convert the instructions of a program written in high – level language to machine instructions before the program can be executed by the computer. In case of a high – level language, this job is carried out by a compiler. **Thus, a compiler is a translating program that translates the instructions of a high – level language into machine language.** 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.

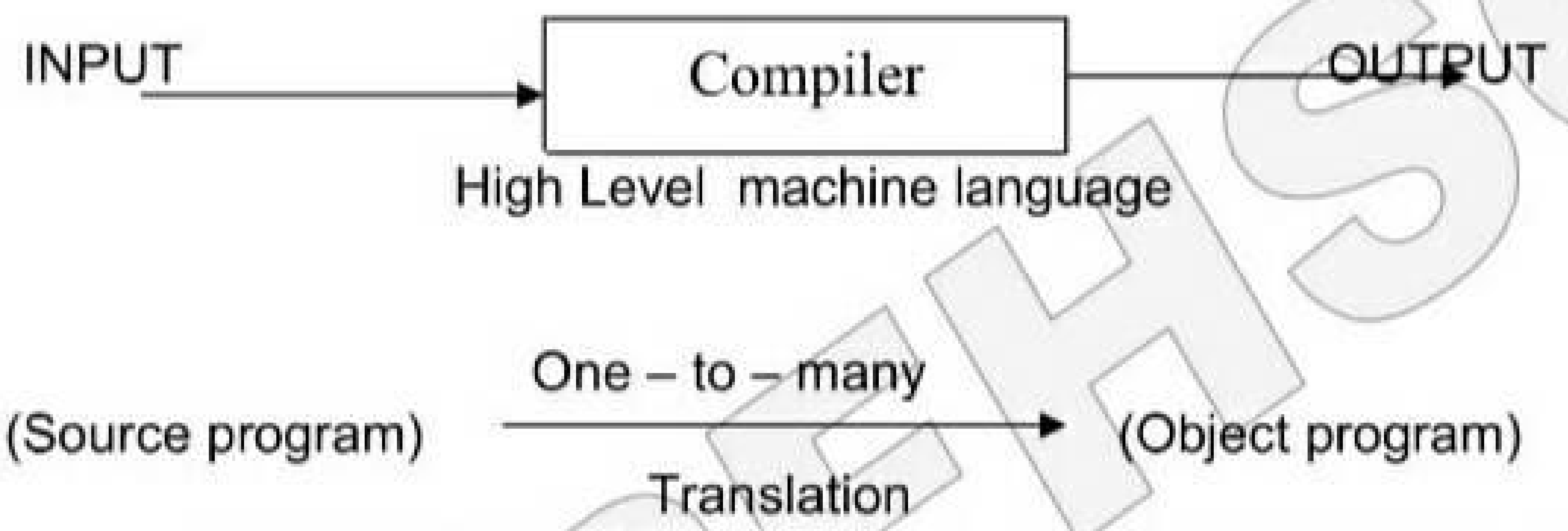


Figure: - Illustrating the translation process of a compiler

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

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

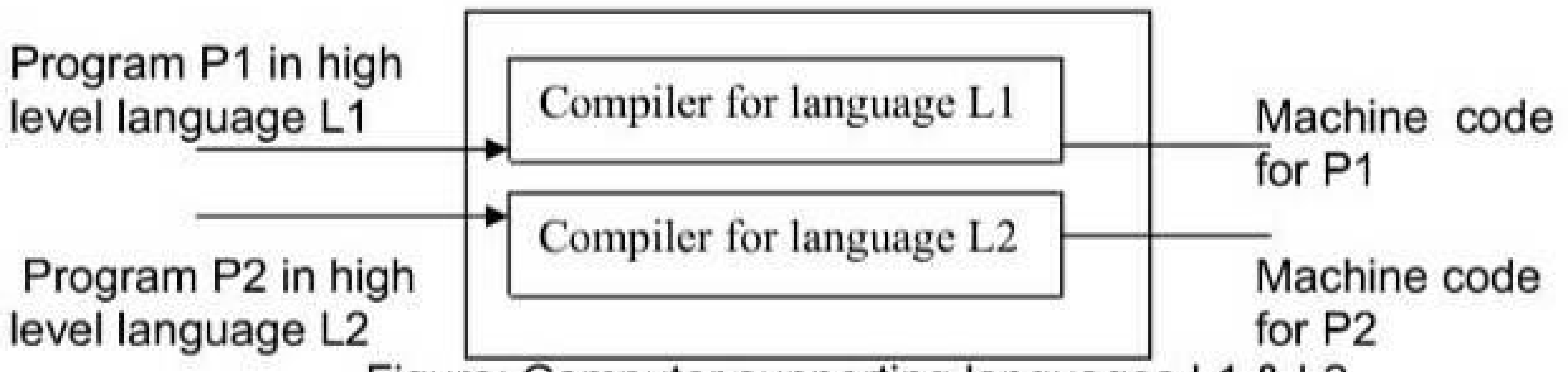


Figure: Computer supporting languages L1 & L2

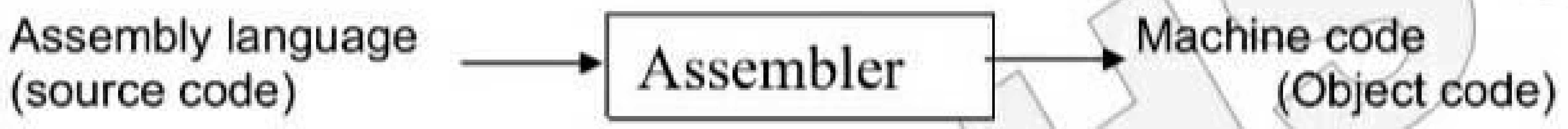
Compilers are large programs, which reside permanently on secondary storage. When the translation of a program is to be done, they are copied into the main memory of the computer. The compiler, being a program, is executed in the CPU. While translating a given program, the compiler analyses each statement in the source program and generates the sequence of machine instructions which, when executed, will precisely carry out the computation specified in the statement.

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

❖ <u>Interpreter:</u>

Interpreter is also a translator meant for translating high – level languages into machine code. Here translation and execution of the instructions goes on parallel. That is once the first instruction is translated it is executed immediately. So in case of interpreter no object code of the program is saved in the memory, and thus the program execution is slow as compared to the compiler. Each time translation is required whenever you execute the program while in case of compiler, once the object code is saved, no translation is required for execution.
Error solving process is less time consuming in case of interpreter. Interpreter itself is the small software as compared to the compiler, so it occupies less storage space.

❖ <u>Assembler:</u>
Assembler translates the assembly language into machine code



The translator program that translates an assembly code into the computer's machine code is called an assembler.

The assembler is a system program which is supplied by the computer manufacturer. It is written by system programmer by great care.

It is so called 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. 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. The input to an assembler is a source program written in assembly language and its output is an object program, which is in machine language.

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.

When we write a program in symbolic language, we first run the assembler program in symbolic language, we first run the assembler (program) to assemble the symbolic program into machine language, and then we run the machine language program to get the answer, but it also must first translate the original symbolic program into machine language. But symbolic programming saves so much time and effort of the programmer that the extra time spent by the computer is worth of.

❖ THE TURBO C ++ EDITOR

INTRODUCTION

The turbo C ++ offers everything you need to write, edit, compile, link, run, manage and debug your programs. You require TC. EXE file to active the Turbo C ++ i.e. TC editor. The menu bar at the top of the screen is the gateway of the menus.

❖ EDITORS EXAMPLE : TC editor, vi editor, KOMODO EDIT, Net Beans etc..

To go to the menu bar, there are three different ways.

1. Press F10 key or
2. Press Alt + ch, Where ch is the first character of the menu options.
3. Click any where on it. (if mouse facility is available)

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

- File	**- Edit**	**- Search**	**- Run**	**- Compile**
- Debug	**- Project**	**- Options**	**- Windows**	**- Help**

File Menu: The file menu provides commands for creating new files, opening Existing files, saving files, chaining directories, printing files, shelling to ODS & quitting Turbo C ++.

Edit Menu: The edit menu provides commands to cut, copy and paste text in Edit windows. You can also undo changes and reverses the changes you have just undone.

Search Menu: The Search menu provides command to search for text, function, declarations and error location in your files.

Run Menu: The run menu provides commands to run your program and to start and end debugging sessions.

Compile Menu: The compile menu provides commands to compile the program in the active edit window, or to make or build your project.

Debug Menu: The Debug menu provides commands to control all the features of the integrated debugger.

Project Menu: The project menu contains all the project management commands
To do the following:

- create or open a project
- add or delete files from your project
- set options for a file in the project
- view include files for a specific file in the project, etc.

Options Menu: The options menu contains for viewing and changing various default setting in Turbo C ++

Windows Menu: The window menu contains window management commands.

Help Menu: This Help menu provides access to the online help system.

EDITOR COMMANDS:

The Turbo C ++ offers variety of commands to do several tasks. Depending upon their functions, commands are classified into following categories:

1. Cursor Movement Commands
2. Insert & Delete Command
3. Block Commands
4. Miscellaneous Commands

[1]Cursor Movement Commands:
The following table shoes the cursor movement commands:

Commands	Function
←	To move cursor one character left
→	To move cursor one character right
↑	To move cursor one line up
↓	To move cursor one line down
PgUp	To move cursor one page up i.e. screen up
PgDn	To move cursor one page down i.e. screen down
Ctrl + W	To scroll up one line
Ctrl + Z	To scroll down one line
Ctrl + A or Ctrl +	To move cursor one word left
Ctrl + F or Ctrl +	To move cursor one word right
Home	To move cursor at beginning of line
End	To move cursor at end of line
Ctrl + Home	To move cursor at the top of window
Ctrl + End	To move cursor at the end of window
Ctrl + PgUp	To move cursor at the top of file
Ctrl + PgDn	To move cursor at the bottom of file

[2] Insert & Delete Commands :
The following table shows insert & delete commands.

Command	Function
Delete	To delete the character
Back Space or Shift + Tab	To delete the character to left
Ctrl + Y	To delete the line
Ctrl + T	To delete the Word
Ctrl +Q Y	To delete the from current cursor position to end
Ctrl + N	To insert the line
Insert	To make insert mode on / off

[3]Block Commands:

The following table shows block commands:

Command	Function
Ctrl + KB	To set beginning of the block
Ctrl + K K	To set end of the block
Ctrl +K C	To copy the block
Ctrl + KV	To move the block
Ctrl + KY	To delete the block
Ctrl + K H	To hide the marked block i.e.Unhide the block
Ctrl + K W	To write a block to the disk
Ctrl + K R	To read a block from the disk

[4]Miscellaneous Commands :

The following table shows miscellaneous commands:

Commands	Function
Ctrl + Q A	Search & Replace
Ctrl + L	Search
Ctrl + [or + Ctrl +]	Pair matching

HOT KEYS:

Turbo C ++ provides hot keys, or shortcut for your convenience. These hot keys can be classified into following category

a) General hot keys
b) Menu hot keys
c) Editing hot keys
d) Online Help hot keys
e) Window management hot keys
f) Debugging /Running hot keys

[a]General hot keys:

Commands	Function
F1	Displays a help screen
F2	**Saves the file that's in the active edit window**
F3	Brings up a dialog box so you can open file.
F4	Runs your program to the line where the cursor is positioned
F5	Zooms the active window
F6	Cycles through all open windows
F7	Runs your program in debug mode, tracing into functions
F8	Runs your 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

[b] Menu hot keys:

Commands	Function
Alt + Spacebar	Takes you to the (=) system menu
Alt + C	Takes you to compile menu
Alt + XD	Takes you to Debug menu
Alt + E	Takes you to Edit menu
Alt + F	**Takes you to File menu**
Alt + H	Takes you to Help menu
Alt + O	Takes you to Option menu
Alt + P	Takes you to Project menu
Alt + S	**Takes you to Run menu**
Alt + W	Takes you to Search menu
Alt + X	**Exits Turbo C ++**

[c] Editing hot keys:

Commands	Function
Alt + backspace	Undo
Shift + Alt + Backspace	Redo
Shift + Delete	**Places selected text in Clipboard, deletes selection**
Shift + Insert	**Pastes text from Clipboard into the active window**
Ctrl + Delete	**Remove selected text from window; doesn't put it in Clipboard**
Ctrl + Insert	**Copies selected text to Clipboard**

[d] Online help hot keys:

Commands	Function
F1	Opens a context-sensitive help screen
F1 F1 (Two time)	Brings up Help on Help
Shift + F1	Brings up help index
Alt + F1	Display previous Help Screen
Ctrl + F1	Calls up language –specific help in the active edit window

[e] Window management hot keys:

Commands	Function
Alt + #	Display a window, where # is the number of the window you want to view
Alt + O	Displays a list of open windows
Alt + F3	**Closes the window**
Alt + W T	Tiles all open windows
Alt + F5	**Displays user screen**
Ctrl + F5	Changes size or position of active window

[f] Debugging / Running hot Keys :

Commands	Function
Alt + F4	Opens an Inspector window
Alt + F7	Takes you to previous error
Alt + F8	Takes you to next error
Alt + F9	**Compiles to .OBJ**
Ctrl + F2	Resets running program
Ctrl + F3	Brings up call stack
Ctrl + F4	Evaluates an expression
Ctrl + F7	Adds a watch expression
Ctrl + f8	Sets or clears conditional breakpoints
Ctrl + F9	**Runs program**

$$$=====================$$$=======================$$$