11-09-19
Wednesday

UNIT-2

Algorithm : step by step process for particular task.

Properties (or) characteristic of an algorithm:

1, Finiteness
2, Definiteness
3, Effectiveness
4, Generality
5, Input / Output

An algorithm is a step by step process to perform a particular task.

(or)

An algorithm is a method of representing the step by step procedure for solving a problem.

Properties (or) characteristics of algorithm.

(1) Finitness: An algorithm should terminate in a finit no. of steps.

(2) Definitness: Each step of the algorithm must be defined clearly.

(3) Effectiveness: Each step must be effective, in the sense that it should be easily convertable into program statements and can be performed exactly in the finite amount of time.

(4) Generality: The algorithm should be complete in itself, so that it can be used to solve all the problems of a given type of any input data.

(5) Input/Output: Each algorithm must take zero, one or more quantities as input data and yield one or more output values.

* Write an algorithm for adding two numbers:

step-1 : Start.
step-2 : Declare a, b, sum
step-3 : Define a, b;
step-4 : Sum = a+b
step-5 : Print sum
step-6 : Stop

* Write an algorithm for adding two number read the numbers using keyboard (or) read data at run time

step-1 : Start
step-2 : Declare a, b, sum
step-3 : Read Print enter a, b values.
step-4 : Read scanf a, b values ——
step-5 : Print sum.
step-6 : Stop

* Write an algorithm multiplication of two numbers:

step-1 : Start
step-2 : Declare p, q, M
step-3 : Define p, q
step-4 : M = p * q
step-5 : Print M
step-6 : Stop.

* Write an algorithm for swapping two numbers:

step-1 : Start
step-2 : Delare a, b, temp (temporary variable)
step-3 : Define a, b values (ex: a = 10, b = 20).
step-4 : Print a, b values before swapping.
step-5 : temp = a; a = b; b = temp

Step – 6: Print a, b values after swaping.
step – 7 : Stop.

* Swapping Write an algorithm for calculating area of the circle.

step – 1 : Start
step – 2 : Declare ~~x, y~~, π, r, A
step – 3 : ~~Print enter~~ Define π, r
step – 4 : Print area (A) Sum-Area = πr^2 = π * r * r
step – 5 : Print area (A)
step – 6 : Stop.

12-09-19 Thursday

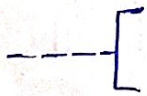
Flow Chart:

The diagramatic or pictorial representation of an algorithm is known as flow chart. Flow chart is build by using different types of boxes and symbols. The operation to be performed and should be written in the box or symbol. All the symbols are interconnected by arrows to indicate the flow of information and processing.

Flow chart is used for following purposes:

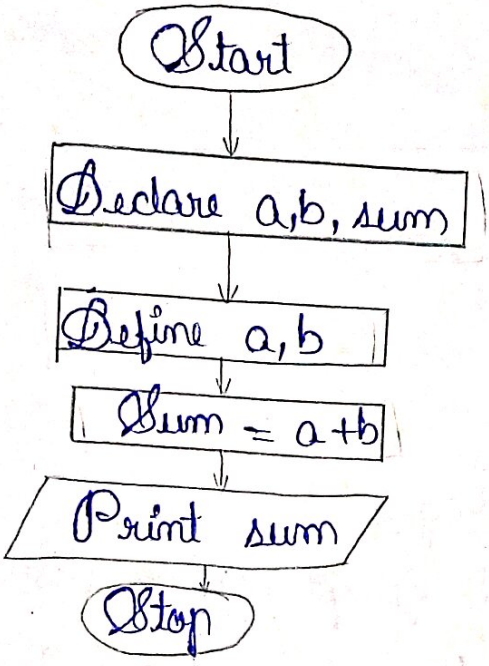
(1) To avoid the programmer in developing the program logic and to be used as documentation for a complete program.
(2) To analyse or define a process.
(3) To explain in detail about the action and decisions defined in a process.
(4) To assist the programmer while searching potential problem points in a process.
(5) To help the programmer, while examining the performance of a process.

Following are the standard symbols used in drawing flowcharts.

Oval		terminal	start/stop/begin/end
Parallelogram		input/output	making data available for processing (input) or recording of the processed information (output)
Rectangle		Process	any processing to be performed. An assignment operation normally represented by this symbol.
Diamond		decision	decision or switching type of operation that determines which of the alternative paths is to be followed.
Circle		connector	used for connecting different parts of the flow chart.
Arrow		flow	joins to symbols and represents execution flow.
Bracket with broken lines		annotation	descriptive comments or explanations.
Double sided rectangle		predefined process or user defined process	modules or subroutines can be represented by using this symbol.

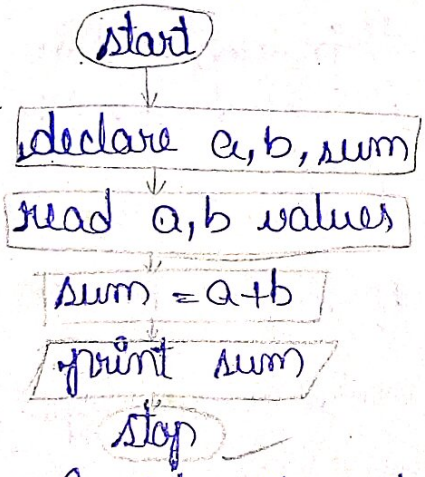
Ex: 1

Write an flowchart for adding two numbers.



Ex-2:

Draw a flow chart for adding two numbers read the numbers using keyboard.



14-09-19 Saturday

Pseudo code: English like statements that follow a loosely defined syntax and are used to convey the design of an algorithm.

(or)

An algorithm written in english like language is called pseudo code.

Note: Every pseudo code is an algorithm, but every algorithm is not a pseudo code.

Ex: Write a pseudo code for addition of two numbers.

step-1 : Start

step-2 : Declare a, b, sum

step-3 : Define a, b.

step-4 : Add a, b store the resultant value in sum.

step-5 : Print sum

step-6 : Stop

16-09-19 Monday

Structure of "C" program:

C" is a structured programming language

(1) Documentation section	/*-------*/ single line comands /*------- -------*/ multiple line comands
(2) Link / Hedrafile section	#include <stdio.h> #include <math.h>
(3) Definition section	#define pi 3.14 #define M 30
(4) Global variable section	syntax : data type variable 1, variable 2... ex: int a=10;

(5) Main function section	main ( )
	{
	declaration statement;
	:
	executable statement;
	}

(6) User Defined function section (or) sub program section	void add ( )
	{
	}
	
	syntax:
	return written type function name
	{
	}

(1) Documentation section: Documentation section consists of set of coment lines giving the name of the program, author and other details like date and time... The coment lines are represented by delimitors (/* ... */).

(2) Hedrafile section: In hedrafile section we include pre defined hedrafiles using the preprocessor directive (#include) #include is called as pound.

(3) Definition section: Definition section is used to define symbolic constants.

Ex: # define M 100, here M is symbol whenever we declare M in the program, the value 100 is alloted everytime.

(4) Global variable section: Some variables that are used in more than one function such variables are called gobal variables.

Ex: int a = 100;

(5) Main function section: Every 'C' program must have one main function the program excuation always starts from main function only. The main function consists of collection of declaration statements and excutable statement. Must be written between open curly braces and closed curly braces. Every 'C' program statement must terminated with semicolumn (;).

(6) User defined function section or sub program section: 'C' language provides facility for the users to define their own functions. This section is not mandatory in a 'C' program.

17-09-19
Tuesday

Identifiers: Identifiers are names of entities in a 'C' program such as variable names, array names, function names and so on. For declaring identifiers we use character set that is characters like a to z, A to Z, one underscore character ("_") then and ten digits 0 to 9.

Rules of forming and constructing identifiers:

An identifier begins with alphabitic character or underscore.

Ex: age, _, _age. ⟶ valid identifiers

An identifier must not begin with a digit (0-9)

Ex: 1age, 6x, 1y ⟶ invalid identifiers.

An identifier can not include a special character like #, $, @, ^, ? and so on. and except underscore character.

Ex: #x, $y, @abc ⟶ invalid indentifiers.

Rules:

*"C" reserved keywords can not used as identifiers.

*Upper case and lower case letters are distinct i.e., identifiers are case sensitive.

Ex: FIRST, First, first, ...

*Commas, blank spaces are not allowed in identifiers.

Ex: x, y, roll no, emp id ⟶ invalid identifiers

x_y, roll_no, emp_id
xY, rollNo, empId
} ⟶ valid identifiers.

*Identifiers must be meaningful, small, quickly accessed, easily types and easily read

Ex: add, cp_marks, cpMarks, roll_no ⟶ valid identifiers

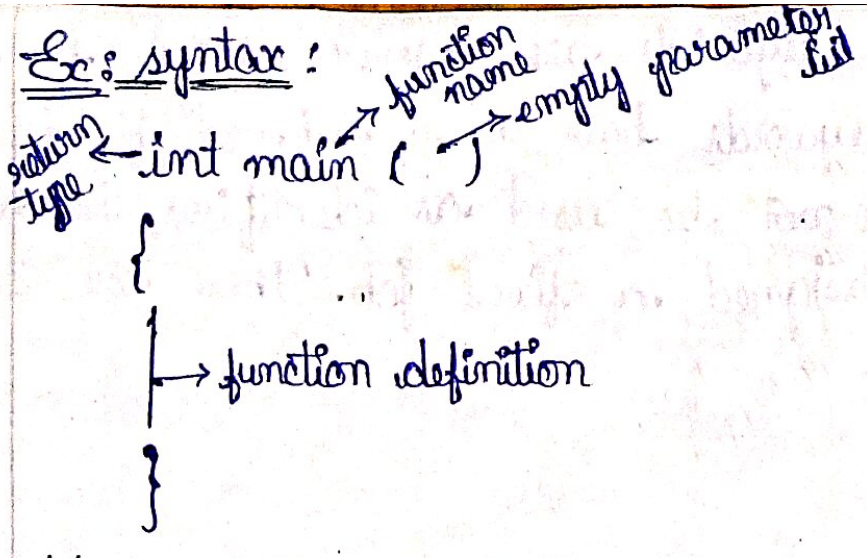
Keywords: The 'c' keywords are reserved words by the compiler all 'c' keywords have been assigned a fixed meaning. They can not be used as identifiers because they have been assigned a fixed job. There are 32 reserved keywords.

auto	int
break	long
case	register
char	returns
const	short
continue	signed
default	sizeof
do	static
double	struct
else	switch
enum	typedef
extern	union
float	unsigned
for	valid
goto	volatile
if	while.

data => collection of char, numbers & digits
information => processed data
software => set of instruction

Main function: Main function is a user defined function. The program execution starts from main function in every 'c' program consisting only one main function. By default the main function has integer written type. The program executed successfully it returns '0' to the operating system are else if there any errors it returns a number to the operating system: Main fuction consists of declaration statements and executable statements enclosed by open curly braces and closed curly braces. In main function, every statement ends with ";"

Ex: syntax:



Main function with parameter list: Many function consisting of two parameters. The parameters are used for reading command line, arguements.

(1) argc: The the argc stands for argument count. The argc parameter stores the no. of arguments to be passed at command line.

(2) argv: The argv stands for argument vector. The argv stores the values of arguments passed at command line.

Syntax: int main (int argc, char * argv[])

```
{
  declaration statement
  :
  executable statement
  :
}
```

19-09-19 Thursday.

	Data Type	Size	Range	Format Specifier.
1.	Int (signed int)	2 bytes (16 bits)	-32768 to 32767 (-2^{n-1} to $+2^{n-1}-1$)	%i (or) %d
2.	Unsigned int	2 bytes (16 bits)	0 to 65535 (0 to 2^n-1).	%u
3.	Short int (signed short int)	1 byte (8 bits)	-128 to 127	%d
4.	Unsigned short int	1 byte (8 bits)	0 to 255	%u.
5.	Long int (signed long int)	4 bytes (32 bits)	-2147483648 to 2147483647	%ld. l or L
6.	Unsigned long int	4 bytes (32 bits)	0 to 4294967295	%lu u or UL
7.	Char (signed char)	1 byte (8 bits)	-128 to 127	%c
8.	Unsigned char	1 byte (8 bits)	0 to 225	%c
9.	Float	4 bytes (32 bits)	3.4e-38 to 3.4e+38	%f or %g
10.	Double	8 bytes (64 bits)	1.7e-308 to 1.7e+308	%lf
11.	Long double	10 bytes (80 bits)	3.4e-4932 to 3.4e+4932	%Lf

Flag	Meaning
-	Left justify the display
+	Display the positive or negative sign of value
0	Pad with leading zeroes
space	Display space if there is no sign

30-09-19
Monday.

```
/* Program for arithematic operators */
#include <stdio.h>
main ( ) {
int a = 10, b = 5;
printf ("Arithematic operators");
printf ("%d + %d = %d", a, b, a+b);
printf ("%d - %d = %d", a, b, a-b);
printf ("%d * %d = %d", a, b, a*b);
printf ("%d / %d = %d", a, b, a/b);
printf ("%d %% %d = %d", a, b, a%b);
}
```

a+b = 15
printf ("a+b = %d", a+b);

=> 10+5 = 15
=> 10-5 = 5
=> 10*5 = 50
=> 10/5 = 2
=> 10%5 = 0.5

Operators, which performs operations on operants.

Relation Operators (<, <=, >, >=, ==, !=):

The relation operator compare two operators, it is also called comparision operator. The ouput of relation operator gives an ~~Bullian~~ boolean value i.e., either true or false. i.e., 1 or 0.

2 or more operations is called compound statement
statement or command instruction
performs some action

```
/* Program for relation operator */
#include <stdio.h>
main ( ) {
printf ("
int a = 10, b = 5;
printf ("%d > %d = %d \n", a, b, a > b);
printf ("%d < %d = %d \n", a, b, a < b);
printf ("%d >= %d = %d \n", a, b, a >= b);
printf ("%d <= %d = %d \n", a, b, a <= b);
printf ("%d == %d = %d \n", a, b, a == b);
}
```

Logical operators (&&, ||, !):

Logical operators performs relate operations on the results of relation operators.

a	b	a&&b
0	0	0
0	1	0
1	1	1
1	0	0

a	b	a||b
0	0	0
0	1	1
1	0	1
1	1	1

a	!a (inversion)
0	1
1	0

/*Program of logical operator*/

```
#include<stdio.h>
main( ) {
int a=10, b=5;
printf("%d a>b && a==b  %d \n", a>b && a==b);
printf("a<b && a==b %d \n", a<b && a==b);
printf("a>=b && a==b %d \n", a>=b && a==b);
printf("a<=b && a==b %d \n", a<=b && a==b);
printf("a>b && a!=b %d \n", a>b && a!=b);
printf("a<b && a!=b %d \n", a<b && a!=b);
printf("a>=b && a!=b %d \n", a>=b && a!=b);
printf("a<=b && a!=b %d \n", a<=b && a!=b);
}
```

```
printf("a>b || a==b %d \n", a>b || a==b);
printf("a>b || a!=b %d \n", a>b || a!=b);
printf("a<b || a==b %d \n", a<b || a==b);
printf("a<=b || a!=b %d \n", a<=b || a!=b);
printf("a>=b || a==b %d \n", a>=b || a==b);
printf("a<=b || a==b %d \n", a<=b || a==b);
printf("a>=b || a!=b %d \n", a>=b || a!=b);
printf("a<=b || a!=b %d \n", a<=b || a!=b);
```

Conditional operators: Conditional operators are also evaluated from left to right. If exp_1 ? is true then exp_2 : is evaluated, it becomes result and exp_3 ; is not evaluated. If exp_1 ? is false then exp_2 : is not evaluated then exp_3 ; is evaluated, it becomes result.

/* Program for conditional operator */

```
#include <stdio.h>
void main ( )
{
int a = 20, b = 15, big;
big = (a>b ? a : b;);
printf ("The biggest value is %d \n", big);
}
```

program = set of statements

03-10-19 Thursday

```
#include <stdio.h>
void main ( )
{
int a=10, b=20, c=30; large;
{
large = (a>b)&&(a>c) ? a : (b>c) ? b : c;
printf ("the large value is %d\n", large);
}
```

/* Write a program on conditional operator */.

```
#include <stdio.h>
int main ( ) {
int a, b, c, largest;
printf ("Enter three different values for a,b,c \n");
scanf ("%d %d %d", &a, &b, &c);
largest = (a>b) && (a>c) ? a : (b>c) ? b : c;
printf ("the largest value is %d\n", largest );
return 10;
}
```

10-10-19 Thursday

Assignment operator (=):

operator$_1$ = operator$_2$

a+=10 → a=a+10 a/=10 → a=a/10
a-=10 → a=a-10 a%=10 → a=a%10.
a*=10 → a=a*10

Increment and Decrement operators (++ / --);
+1 -1

++a → pre increment
--a → pre decrement
a++ → post increment
a-- → post decrement

```
#include <stdio.h>
void main ( )
{
int a=10, b=20;
printf ("a = %d", a);
printf ("a = %d", a++);
printf ("a = %d", ++a);
printf ("b = %d", b);
printf ("b = %d", b--);
printf ("b = %d", b);
printf ("b = %d", --b);
}
```

```
#include <stdio.h>
void main ( )
{
int x = 20, y;
  y = x++
printf ("x = %d, y = %d", x, y);
}
```

```
#include <stdio.h>
void main( )
{
int x = 10;
printf ("%d", ++x);
printf ("%d", x);
printf ("%d", x++);
printf ("%d", x);
}
```

Bitwise operators: (&, |, ^, ~, <<, >>):

Bitwise operators perform operations only on characters (0,1).

x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

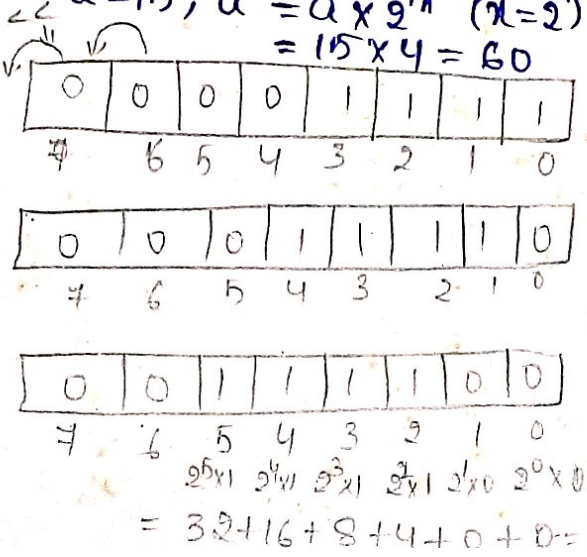
x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

x	~x
0	1
1	0

(<<) left shift: a<<2

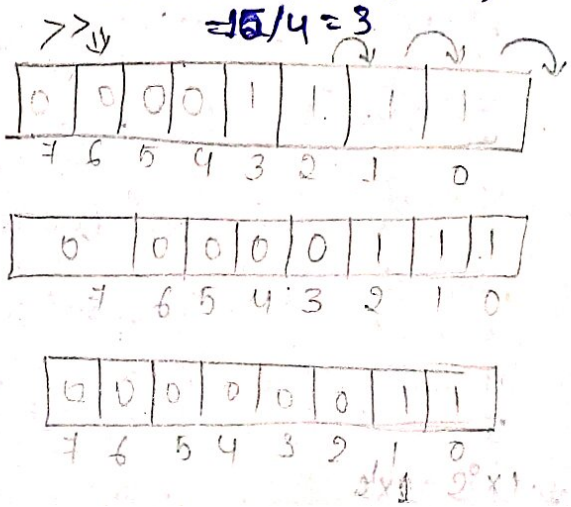
a=15, a = a x 2^x (x=2)
= 15 x 4 = 60

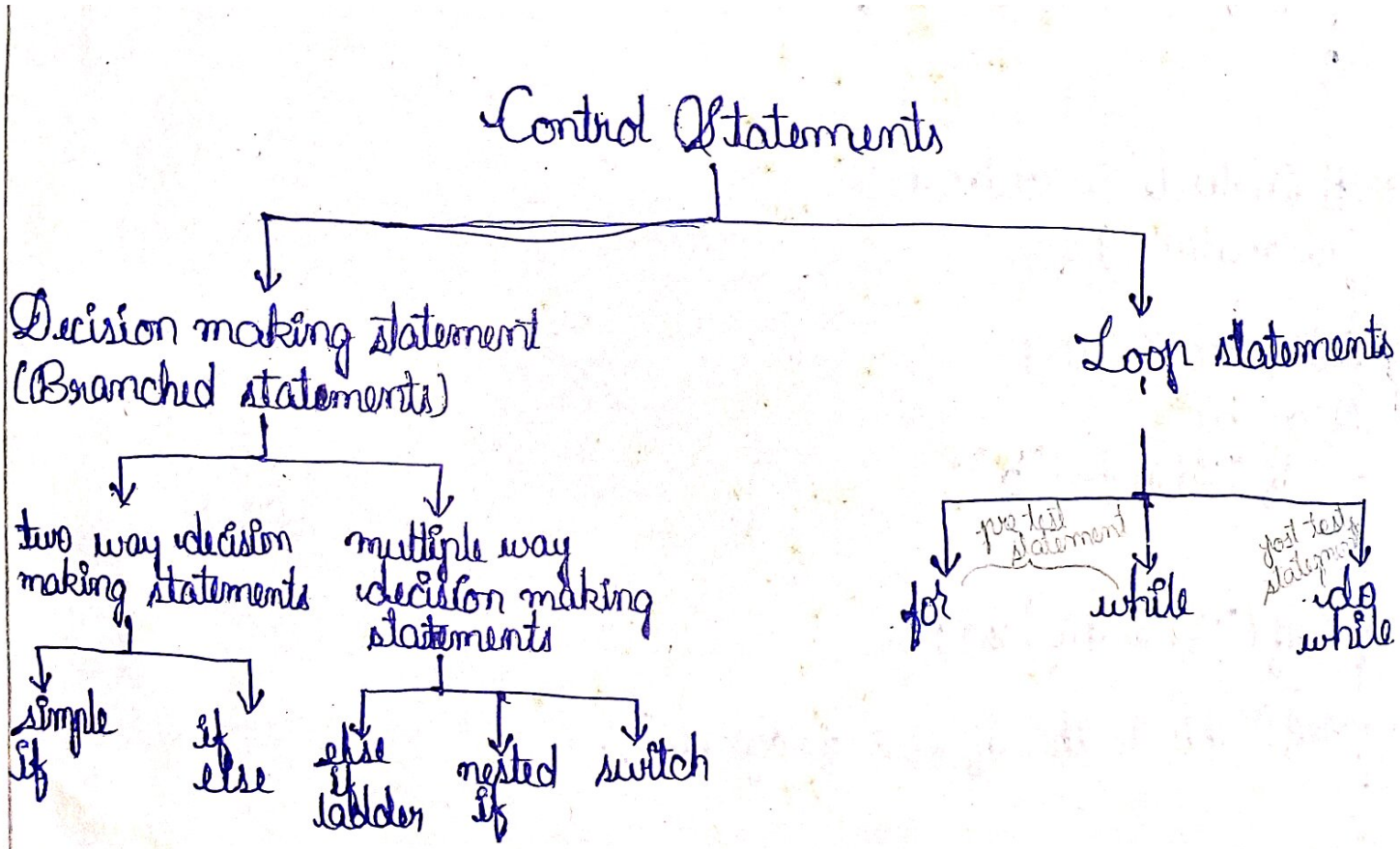


= 32+16+8+4+0+0 = 60

(>>) right shift: a>>2

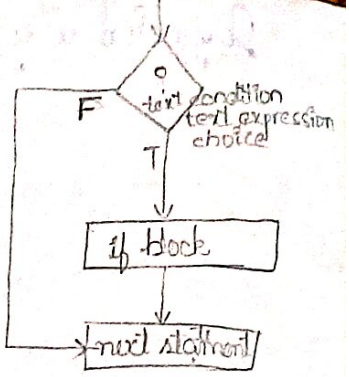
a=15, a = a/2^x (x=2)
=15/4 = 3




Control Statements
Decision making statement
(Branched statements)
Loop statements
two way decision making statements
multiple way decision making statements
pre-test statement
post-test statement
simple if
if else
else if ladder
nested if
switch
for
while
do while

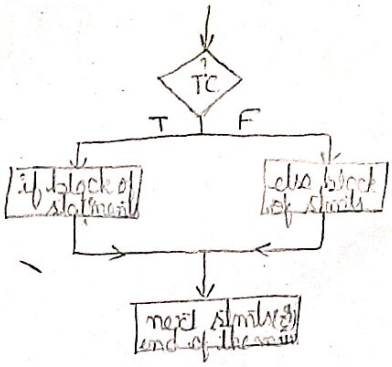
* If:

```
#include <stdio.h>
void main ( )
{
int a=40, b = 20;
if (a>b)
printf ("a is big ");
}
```



If else:

```
#include <stdio.h>
void main( )
{
int a=10, b = 20;
if (a<b)
{
printf ("%d is big", b);
}
else (a>b)
{
printf ("%d is big", a);
}
printf ("this is a If else condition ");
}
```




```
#include <stdio.h>
int main( )
{
int a=40, b=10;
If(a<b)
printf ("%d is big", b);
else
printf ("%d is big", a);
printf("This is the if else program");
}
```

Nested If:

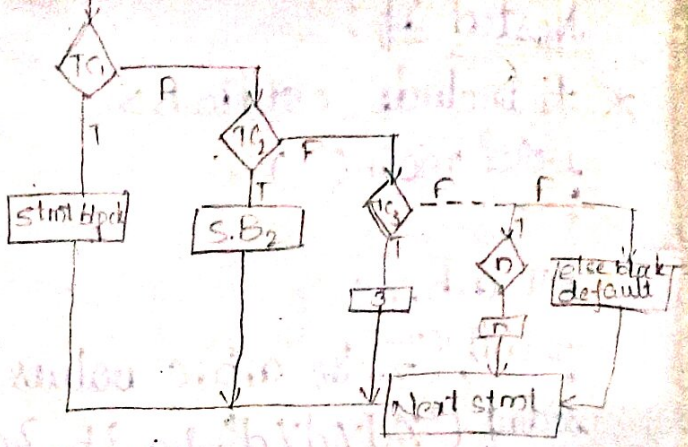
```
#include <stdio.h>
void main( )
{
int a,b, c;
printf ("Enter a,b,c values ");
scanf ("%d%d%d", &a, &b, &c );
if (a>b)
    {
    if(a>c)
    printf (%d is big",a);
   else
   printf ("%d is big", c);
    }
else
     {
     if (b>c)
     printf ("%d is big", b);
     else
     printf ("%d is big", c );
      }
}
```

```
#include<stdio.h>
int main( )
{
int a=10, b=30, c=20;
if (a>b)
   {
    if(a>c)
   printf("a is big");
   else
   printf("c is big");
}
else
   {
    if(b>c)
   printf("b is big");
   else
   printf("c is big");
    }
}
```

else if ladder :

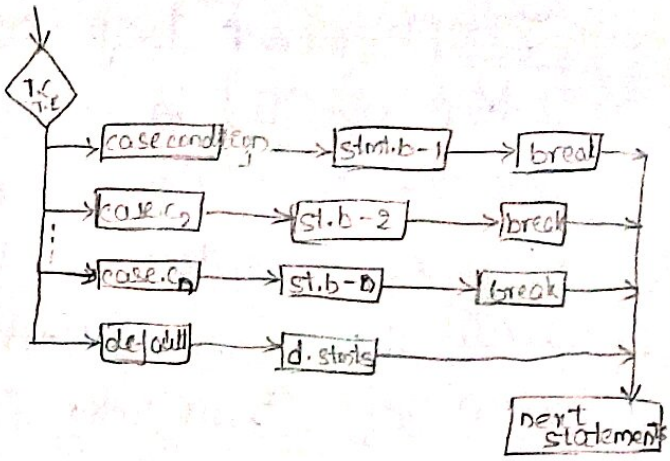
15-10-19 Tuesday

```
#include <stdio.h>
void main( )
{
int s1, s2, s3, s4, s5, s6, total;
printf{
float avg;
printf("Enter 6 subjects marks");
scanf("%d%d%d%d%d%d", &s1, &s2, &s3, &s4, &s5, &s6);
total = s1+s2+s3+s4+s5+s6;
avg = (float) total/6;
if (avg >= 80)
   {
   printf ("excellent");
   }
else if (avg>=70)
   {
   printf ("super");
   }
else if (avg>=60)
   {
   printf ("good");
   }
else if (avg>=50);
   {
   printf ("average");
   }
else
   {
   printf ("fail");
   }
}
```



15-10-[illegible] Wednesday

switch:




```
#include <stdio.h>
void main( )
{
int a,b,choice;
printf ("Enter any two values");
scanf ("%d %d",&a,&b);
printf ("1.add \n 2.sub \n 3.multiplication \n 4.division \n 5.modula \n");
printf ("Enter your choice : ");
scanf ("%d",&choice);
switch (choice) {
case 1 : printf ("Addition of two numbers :- %d",a+b);
         break;
case 2 : printf ("Subtraction of two numbers : %d",a-b);
         break;
case 3 : printf ("Multiplication of two numbers : %d",a*b);
         break;
case 4 : printf ("Division of two numbers : %d",a/b);
         break;
case 5 : printf ("Modulo Division of two numbers : %d",
                                                a%b);
         break;
default : printf ("Wrong Choice");
 }
}
```

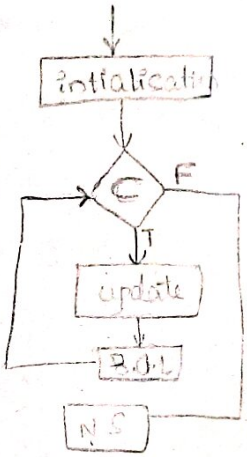
Looping: A loop contains a sequence of statements which specified once and executed several times. The code inside the loop is called as "body of the loop". It can be executes a several specified number of times or until some condition is met or once for each collection of items (or) infinitely. The type of loops are given below.

1, Counter controlled loops: In counter controlled loops the body of the loop executes in a specified number of times.

for(initialisation; condition; update/increment/decrement)
evaluated 1 time (optional) define variable

syntax-1: for i=1 to N
xxxx (body of the loop).
next i.

syntax-2: for i=1 to N
do begins
xxxx (body of the loop).
end



syntax-3: for (i=1; i<=N; i++)
{
xxxx (body of the loop).
}

2, Condition controlled loop: In condition controlled loops the body of the loop will be executes until some condition is met some variations test the condition at the starting of the loop, others test it at the end. Initially the condition is failed if the test is at the start, then the body of loop is skipped completely, if it is at the end the body of the loop is always executed atleast once.

syntax: while (test)
{
xxxx; (body of loop)
}

while(test)
{
xxxx;
(body of loop)
}

do
xxxx
while (test);

3. Collection controlled loop: Several programming languages (ex: Java, PHP, C++, python etc) have special constructs which allow implicit looping through all elements of an array (or) all members of a collection.

Syntax:
```
for each (item : my collection)
{
 xxxx ;
}
for each (element : my array).
{
 xxxx; (body of the loop).
}
```

Infinite loop: An infinite loop (or endless loop) is a endlessely executed loop due to the loop having no terminating condition, having condition can never be met.

An example in 'C':
```
while (1)
{
printf ("Infite loop");
}
```

An example in 'python':
```
while (true)
  printf ("Infinite loop");
```

An example in 'java':
```
while (true)
{
  system out printf ("Infinite loop");
}
```

Programming features (or) characteristics: Easy computer requires appropriate instruction set (program) to perform the required task. The quality of the process depends

upon the given instructions.

If the instructions are improper or incorrect then it generates the inconsistent results. Therefore, the proper and correct instructions should generate the accurate results.

Hence, a program should be developed. In such a way, that it ensures proper functionability of the computer. In addition, a program should be written in such a manner that it is easier to understand the underlying logic.

A few important features (or) characteristics are:

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

2, Readability: The program should be written in such a way that it makes either programmers or users follow the logic of the program without much effort.

If a program ~~without~~ written structurally it helps the programmers to follow their ~~can~~ own program in a better way.

3, 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 require modifications from the time to time.

4, Generality: The program should be general. If a program is developed for a particular task then it should also be used for all similar tasks of the same domain.

5) Structural: To develop a program, the task must be broken into a number of sub-tasks.

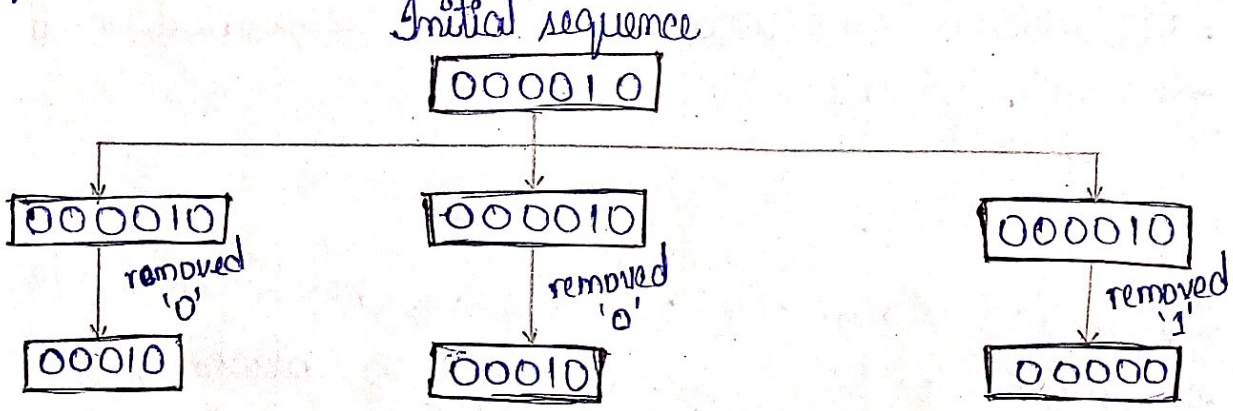
These sub-tasks are developed independently and sub-task is able to perform the assigned job without the help of any other sub-tasks.

If the program is developed structurally, the program becomes readable, the ~~tasking~~ testing and documentation also gets easier..

The Zero-one game: The Zero-one game starts with a sequence of Zero's and one's. The players required for this game has minimum two plays and maximum any number of players. The game allows the alternative turns between players. During each turn, a player removes one element from the seq sequence that satisfies the following rules.

* The removing element is not the first (or) last elements
* The removing element must be surrounded by Zero's and on both sides.
* The first player who won't take their turn loses the game. The second one won the game.
* Both players move optionally.

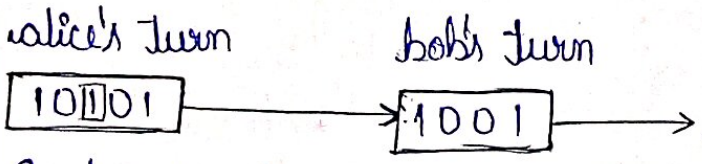
For example, the diagram below depicts the 3 possible first move for the sequence.



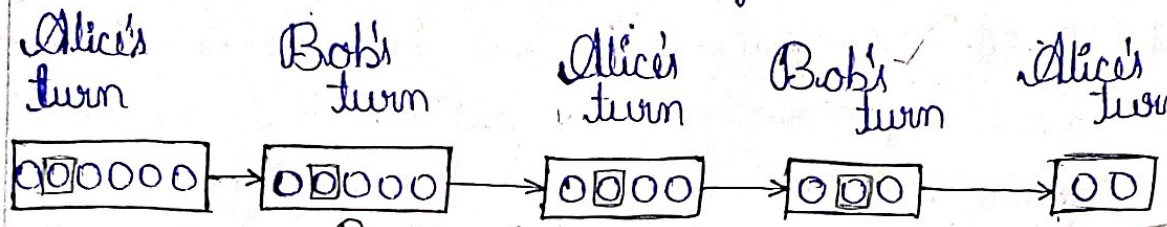
For example, Alice and Bob play games. Given the initial sequence of numbers for each game, print the

name of each game's winner to on a new line.

*Alice removes one element from and Bob is left with no moves.



*Each player removes some zero from until only two zeroes at either end remain at which point Alice is left with no moves. Thus we print Bob on a new line.



Structural Programming Concepts: Structural programming can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction (i.e., one after another). In this approach the instruction will executed in serial and structured manner. The language, that support structured programming approach are C, C++ etc. The structured programming mainly consists of following types of elements (concepts).

1) Sequence statements: The lines or block of the code are written and executed in a sequential order.

Ex: x = 5, y = 4;
z = x + y;
print (z);

2) Selection statements: Executes a block of code if a condition is true and it skips a block of code if a condition is false.

syntax: If condition then
action (or block of the code).
End if.

Ex: x = Read line ( )
If x mod 2 == 0
Write line ("The number is even");
End if.

(3) Repeatable statements: It repeats a block of the code (action) while a condition is true. There is no limit to the number of times that the block can be executed.

Syntax: while condition
action ("block of code");
end while.

Ex: x = 2
while x < 100
write line (x);
x = x * x;
end while.

The structured program consists of well structured and separated modules. But the entry and exit in a structured program is a single time event. It means that the program uses single-entry and single exit elements. And it is also well maintained neat clean program. This is the reason why the structured programming approach is well accepted in the programming world.

Advantages of stuctured approach:
* Easier to read and understand.
* User friendly.
* Easier to maintain.
* Development is easier as it requires less effort and time.
* Easier to debug.

Disadvantages of structured programming approach:
*Since it is machine-independent, so it is takes time to convert into machine code.
*The program depends upon changable factors like data type.
*Usually the development in this approach takes time as it is language dependent.

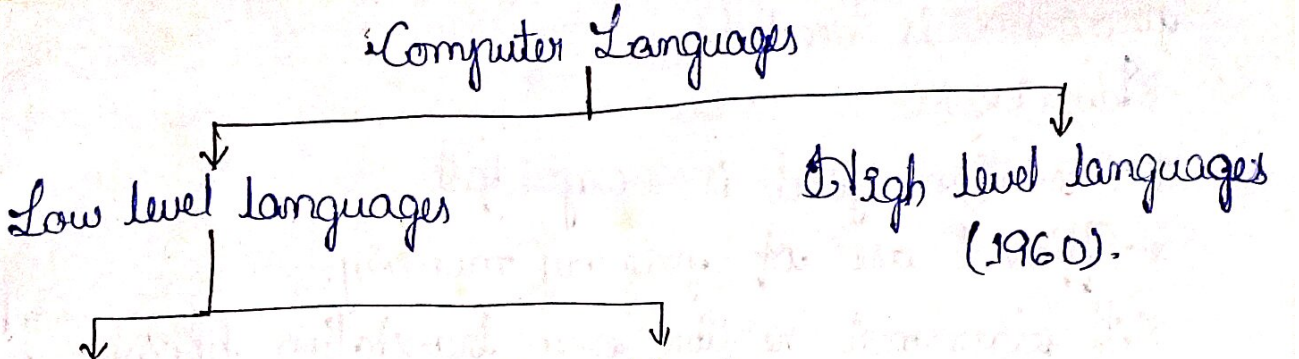
Documentation: Documentation is one of the most important compounds of an application development. It helps the end user to fully utilise the functionality of the application. It is also helps the programmer to understand it even in the absence of the another.

Programming languages: A language is meant for communication purpose. A programming language is used for communication between human beings (computer users) and computers.

(or)

A programming language consists of a set of vocabulary and grammatical rules, to express the computations and tasks that the computer has to perform.

Each language has a unique set of keywords and a special syntax for organising program instructions. The programming language should be understand by the programmer and computer. Basically languages can be divided into two categories according to how the computer understands them. They are (1) Low level languages.
(2) High level languages.



1, Low level languages: Low level languages are either machine codes (or) are very close them (i.e., A computer cannot understand instructions in high level language (or) in english. It can only understand and excute instructions given in the form of machine language i.e., binary language (0's and 1's). There are two types of low level languages.

(i), Machine language.

(ii), Assembly language.

(i), Machine language: Machine language is the lowest level language. In machine level language the instructions are written as strings of binary number (0's and 1's). It is a language that can be directly understood and obeyed by a machine (computer) language that can be directly understood without conversion (translation). The machine language also know as "first generation programming language". The machine language is different for different computers. It is not easy to learn. The computer understand the programs written in machine language directly. No translation of the

program is needed.

Advantages:

*Execution speed is very fast.

*Efficient use of primary memory.

*It does not require any translation because machine code is directly understood by the computer.

Disadvantages:

*Machine dependent: The machine language is different for different types of computers.

*Difficult to write a program: A machine language programming must be knowledgable about the hardware structure of the computer.

*Difficult to modify: To correct (or) modify the instructions is very and time consuming.

ii) Assembly language: Assembly language is also known as "second generation of programming language". These are introduced in 1950's.

Assembly language consists of special codes called "Mnemonics" to present the language instructions instead of 0's and 1's. The code is also called as "operation code" or "op code".

Ex: ADD, SUB, MUL, JMP, LOAD etc.

Advantages:

*Easy to understand and use: Assembly language program is easy to use, understand and memorise it uses mnemonic codes inplace of binary codes.

*Easy to write input data: In assembly language programs, the input data can be written in decimal number system, later they are converted into binary.

*Easy to locate and modify: Assembly language are easier to understand it is easier to locate, correct and modify the instructions.

Disadvantages:

*Machine dependent: A program written for one computer might might not run in other computer with different hardware configuration.

*Knowledge of hardware required: Assembly languages are machine dependent, then the programmer must have a good knowledge of characteristics and logic structure of the computer.

*Translator required: To converting assembly languages to machine language instructions one translator is required (assembler).

2. High level languages: The instructions of high level languages are written by symbols and words. The high level languages are similar to english language. Each high level language has its own rules and grammar for writing program instructions. These rules are called "syntax of the language".

The program written in high level language must be translated to machine code before to it. Each high level language has its own translator programmer i.e., interpreter (or) compiler. The high level languages are further divided into

*procedural languages.

*non-procedural languages.

*object oriented programming languages.

Advantages:

*Easy to learn: The high level language are very easy to learn than low level languages. The statements written for the program are similar to english like statements.

*Easy to understand: The program written in high level language by one programmer can easily be understood by another because of the program instructions are similar to the english language.

*Easy to write program: In high level language, a new program can easily be written in a very short time. The larger and complicated software can be developed in few days (or) months.

*Easy to detect and remove errors: The errors in a program can be easily detected and removed. Mostly the errors are occured during the compilation of new program.

*Built-in library functions: Each high level language provides a large number of built-in functions (or) produces and saves time of the programmer.

*Machine independence: Program written in high level language is machine independent. It means that a program written in one type of computer can be excuted on another type of computer.

Disadvantages:

*A high level language has to translated into the machine language by a translator which takes time.

*The object code generated by a translator might

be inefficient compared to an equivalent assembly language program.

*Procedural languages: Procedural languages are also known as "third generation languages". In a procedural language, a program is designed using procedures (functions). A procedure is a sequence of instructions having a unique name the instructions of the procedure are executed with the reference its name to solve a specific problem some popular procedural languages are.

(i) Fortran (FORTRAN): "FORTRAN" stands for "Formula Traslation". It was developed in 1957 for "IBM computer"

(ii) COBOL: "COBOL" stands for "Common Business Oriented Language". It was developed in 1959 for "business and commercial applications". It is used for to process credit and debit cards.

(iii) PASCAL: This programming language is named in the honour of "Blaise Pascal", a mathematician and scientist, who invented the first mechanical calculator. It was developed in 1971 for scientific field.

(iv) ADA: It is developed in 1980 and is named in the honour of "Lady Augusta Ada". She was the first computer programmer. This language mainly used for "defence purpose" such as for controlling military weapons like missiles etc.

(v) C-language: "Dennis Ritchie" and "Barian Karnigh" developed it in 1972 at "Bell Laboratories" c-language is a middle level language. The c-language is a

structured programming language. The main feature of a C-language is that it uses a large number of built-in functions to perform various tasks.

*Non-procedural languages: Non-procedural languages are also known as "fourth generation languages". In a non-procedural language, the user/programmer writes english like instructions to retrive data from data base. These language provide the user friendly program development tools to write instructions. The most important non-procedural languages are

(i) SQL: It stands for "Structured Query Language". It is very popular database access language and is specially used to access and to manipulate the data of database. To perform various operations on data of database the queries (enquiries) are used.

(ii) RPG: It stands for "Report Program Generator". This language was introduced by IBM, to generate "business reports".

*Object Oriented Programming language: The object oriented programming was introduced in late 1960's, but now it becomes most popular approach to develop software. In object oriented programming the software is developed by using set of interfacing objects. The object is a component of the program that has a set of modules. The modules are known as methods. And these are used to access the data from the object. Once, an object for any program designed, it can be reused in any

other program. Most popular and commonly used objects oriented programming languages are C++, JAVA.

(i) Assembler
(ii) Compiler
(iii) Interpreter
(iv) Linker
(v) Loader.

(i) Assembler: Assembler is an program (or) softwa-are which converts assembly language instructions into machine language instructions (i.e., machine code).

(ii) Compiler: Compiler is a software that translates high level language instructions into machine language instructions. The compiler program is refered as object code. The compiler also reports syntax errors, if any in the source code.

(iii) Interpreter: Interpreter is a software which translates high level language instructions into machine langua-ge instructions.

The main difference between compiler and interpreter is compiler converts entire program at a time into machine code, the interpreter converts line by line into object code.

(iv) Linker: Linker is a program that links several object modules and libraries into a single execut-able program. Before execution of the program, the modules and required libraries are linked together using the linker software.

The compiled and linked program is called "executable code".

(v) Loader: The loader is a software is used to load and reload executable program into main memory. Software has to be loaded into the main memory during execution. Loader assigns storage space to the program in the main memory for execution.