UNIT - I

* Secondary storage devices :- (auxiliary storage)

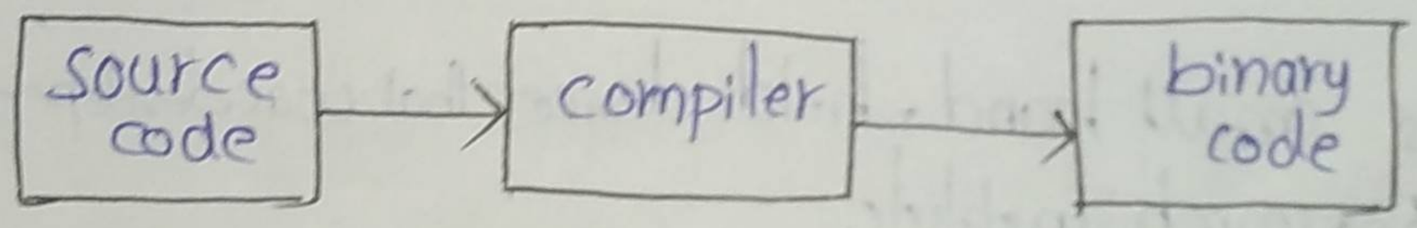
→ Permanent storage
→ Secondary memory is the slowest & cheapest form of memory.
→ Not possible to process such memory directly by the cpu.
→ necessary to copy the data present in the secondary memory into primary storage.
→ S.S. hold the information until it is deleted or over written.
→ Ex:- Magnetic tapes, Magnetic disks, optical disks.

* Magnetic disks :-

→ commonly used (secondary storage medium)
→ Advantages:-
→ They provide high storage capacity
→ They are much reliable
→ They have ability to directly access the stored data.

* Compiler :-

→ It refers to a software that transforms high level lang code into machine language code.
(or)
→ Transforms the source code into binary code

→ 

* Algorithm :-

→ a series of steps in a computer program which provides the solution for a particular problem.

Step for algorithm :-

1) understand the program
2) Identify the expected output
3) " " necessary input
4) Develop a logic that produces expected output
5) Finally test the algorithm with various sets of input

*~~Flow Chart~~ :-

Components of computer system :-

They are categorized into two major components, They are

1) Hardware components
2) software components

1. Hard ware components :-

Computer hardware is a physical device that includes the following components.

i) Input Devices :-

→ devices that are used to insert the data into the computer are called input devices.

→ Ex :- Keyboard, Mouse, Joystick, image scanners, voice systems.

a) Keyboard :-

→ The Keyboard is a primary input device that is used to input numeric and alphanumeric data & commands.

→ It is also used to select the menu options or graphic fun

→ There are 101 keys on a keyboard.

b) Mouse :-

→ A mouse is small hand-held device that was developed at stanford Research Institute.

→ It is used to position the screen cursor.

→ It " " for selecting elements such as tools, buttons and icons present on the screen by pointing and clicking them.

→ It is also called as pointing device

{ operations of mouse }.

c) Joystick :-

→ It consist of a stick (a small & vertical lever) that is attached to the base and is used to direct the cursor.

→ The screen positions can be selected ~~either~~ thus by stick movement or by applying pressure on the stick.

D) Image Scanners :-

→ Image scanners are is a machine, that scans the image and display's on the screen.

→ It is used for docs. scans, el.

E) Voice systems :- (voice recognition devices)

→ Voice recognition is a type of input method where a microphone is used to enter data in form of spoken words into the computers.

ii) Output Devices :-

→ It show the output to the users.

→ These devices take the result which is in machine language from the processor and translate them to form easily understood by the users.

* Hard copy :- Physical representation of o/p

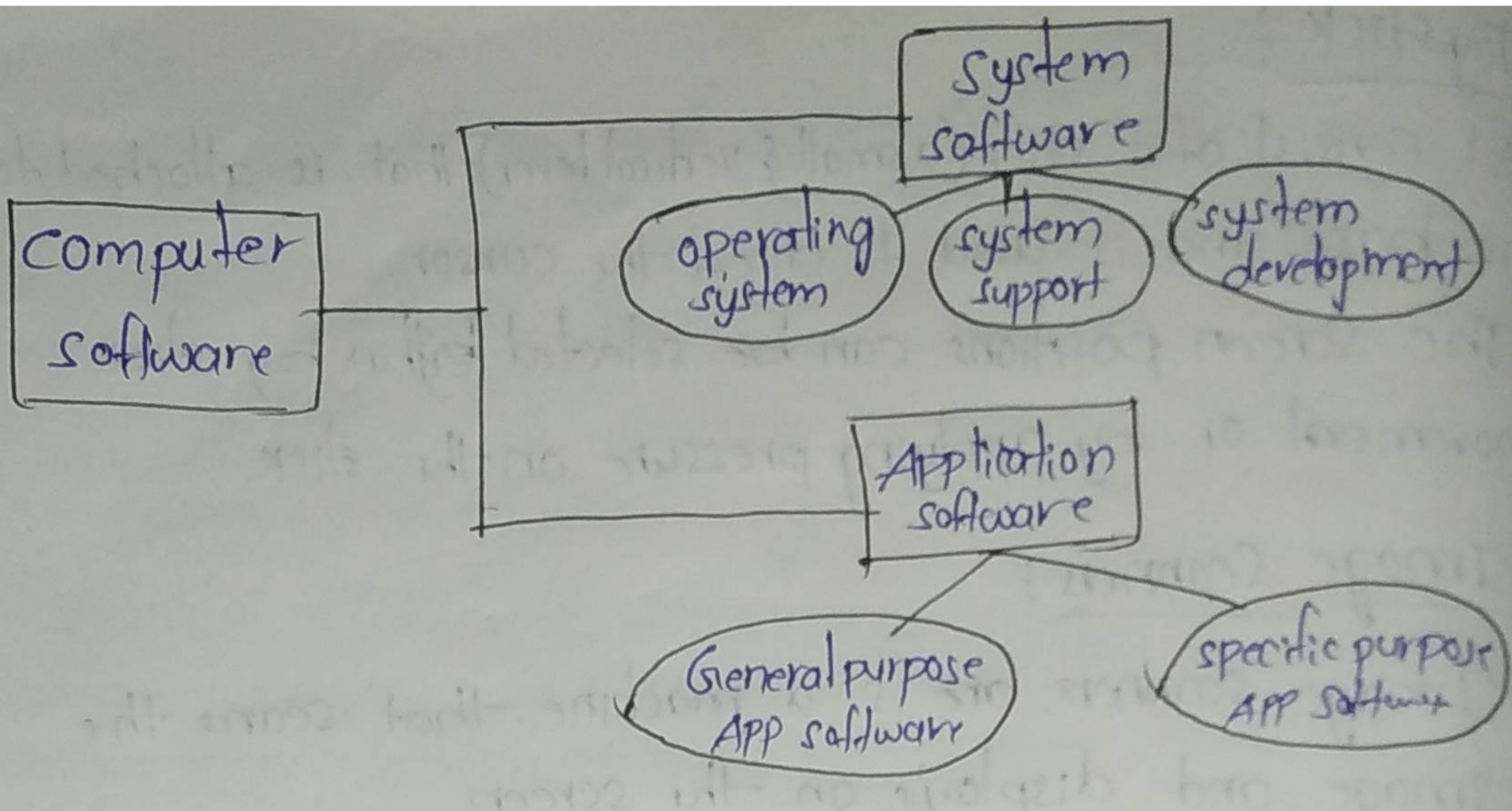
* Soft copy :- Electronic " " "

a) Display screens.

b) printers — [Impact printers (~~reports~~)
Non-Impact printers ( )

2. Software Components :-

Comp Software consist of a group of instructions that tells the computer hardware how to perform a task.



Solving Logical problems :-

Solving logical problems consists of below step

1) Comprehending the problem
2) Representing the problem in formal terms
3) Planning a solution
4) Executing the plan
5) Interpreting and evaluating the solution.

1) Comprehending the problem:

In this step, user need to visualize the situation

→ identify the problem and problem data.
→ Extract all the data related to the problem.

2) Representing the problem in Formal Terms

→ Represent the problem in terms of formal concepts and principles in specific area related to problem.
→ They are useful in simplifying the complexity of the problem.

Solve numerical Problems

steps for solving numerical problems are as follows

1. Analyze:-

→ determine starting point & ending point (inpd, outd).

→ Develop a plan to solve the problem

→ Table or graph.

2. Calculate:-

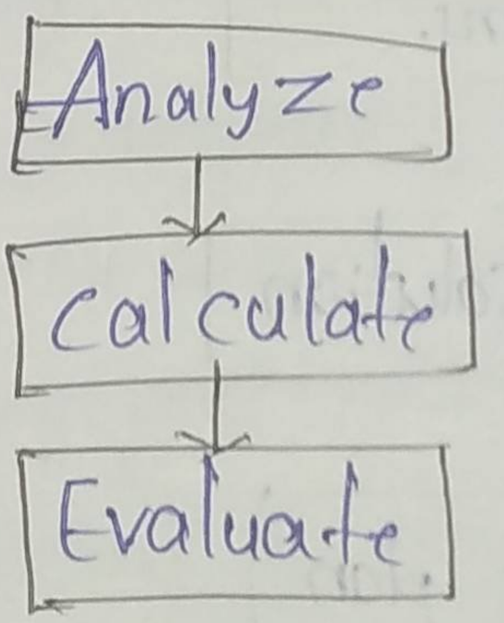
→ Calculations can be done easily if the planing is done effectively.

3. Evaluate:-

→ ~~It~~ involves verifying the problem solution, correctly or unit

→ Finally calculations are checked.

→ Scientific notations must be used in the answer.



Algorithm :-

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

→ An algorithm is very useful for finding the correct answer. to a problem by breaking the problem into simple steps.

Properties of Algorithm :-

An algorithm must process the following properties:

1) Finiteness
2) Definiteness
3) Effectiveness
4) Generality
5) Input/output

1) Finiteness :-
→ An algorithm should terminate in a finite number of steps.

2) Definiteness :-
→ Each step of the algorithm must be precisely stated.

3) Effectiveness :-
→ Each step must be effective, it should be easily convertable into program statement and can be performed exactly in a finite amount of time.

4) Generality :-
→ The algorithm should be complete, so, used to solve all problems of a given data for any input.

5) Input/output :-
→ Each algorithm must take 0, 1 or more quantities as input data and yield one or more output values.

★ Average of three numbers

Step1: start
Step2: Read three numbers a, b and c
Step3: compute the sum of a, b and c
Step4: Divide the sum by 3
Step5: Store the result in variable d
step6: Print the value of d
step7: Stop.

★ Greatest among three numbers.

step 1: start
step 2: Read three numbers a, b and c
step 3: check if (a > b) and (a > c) then
step 4: print a
else if (b > c) then
print b
else
print c
step 4: stop

(or)

Step 1: start
" 2: Declare the variables num1, num2 and num3
" 3: Read the variables num1, num2 and num3.
" 4: If num1 > num2
If num1 > num3
Display num1 is the largest number
else
Display num3 is the largest number
else
if (num2 > num3)
Display num2 is the largest number
else
display num3 is the largest number.
" 6: stop.

✡ Factorial of a number

5

5*4*3*2*1.

step 1: start

" 2: Declare the variables num, factorial, i.

" 3: Initialize factorial to 1

" 4: Initialize i to 1

" 5: Read 'num' for which factorial has to be found.

" 6: Repeat the following two steps until i = num

a) calculate factorial = factorial * i

b) Increment i by 1

" 7: Display factorial

" 8: stop.

✡ Perform linear search on an array.

step 1: Initialize index (i.e position) i=0 and number of elements in list = n.

" 2: Check, if (i<n) then

Goto step3

else

Goto step5

" 3: compare, if (key matches with list element at i) then.

printf ("Element found! successful search")

Return the index (i) of the found element

Goto step 6

else

" 4: Increment i by 1

Goto step 2

" 5: printf ("End of list; Element not found! unsuccessful search")

" 6: stop

☆ Binary Search on an array

Step 1: start

" 2: Initialize low_val = 0 and high_val = n and key

" 3: Read key value.

" 4: While (high_val >= low_val) do

mid_val = (low_val + high_val)/2

Goto step 5

otherwise when condition fails

printf ("Unsuccessful search, element not found")

Goto step 8

" 5: check, if (L[mid_val] = key) then

printf ("Element found at mid_val")

Goto step 8

else

" 6:

check, if (L[mid_val] > key) then

high_val = mid_val - 1

goto step 4

else

" 7:

check, if (L[mid_val] < key) then

low_val = mid_val + 1

Goto step 4

" 8: stop

✡ Bubble Sort

step 1: Initialize index element i=1

step 2: Repeat steps 3 to step 11 until all elements are stored

step 3: set j to n-1

" 4: If sort List[j] ≤ sort List[j+1] then goto step 8

" 5: set temp to sortList[j+1]

" 6: Set sortList[j+1] to sort List[j]

" 7: Set sortList[j] to temp.

" 8: Increment i and store the value in itself

" 9: If i ≤ j then goto step 4

" 10: set j to j-1

" 11: If j ≥ 1 then go to step 1

" 12: print the elements after sorting

" 13: End.

✡ Selection sort

Step 1: start

" 2: Read all the elements from the list

" 3: store each element in an array

" 4: set the index value i=1

" 5: Repeat step 6 to step 11 until all the elements in the list are sorted.

" 6: set first to i

" 7: set smallest to i+1

" 8: Repeat step 9 and step 10 while smallest < size_of_the_list.

" 9: If smallest < min, swap min and smallest.

" 10: Increment smallest

" 11: Increment i

" 12: End.

☆ remainder of a divisionoperation.

step 1: start

" 2: Read two integers as dividend and divisor

" 3: if divisor is eqvual is eqvual to zero(0) then
print "Divided by zero arithemetic exception"

" 4: else
remainder = dividend / divisor

" 5: print remainder

" 6: stop.

☆ Given 2 Numbers, determine whether(or) not their sum is greater than 100.

step 1: start

" 2: Read any two Numbers
declared

" 3: Add the two numbers and assign the result to the third variable say, num.

" 4: If the sum is greater than 100, then print "The sum of two given number is greaterthan 100".

" 5: If the sum of desser than 100, then print "The sum of two given number is not greater than 100".

" 6: Stop.

☆ Check whether given year leapyear or not.

Step 1: start

" 2: Declare & Read year.

" 3: rem = year%4

" 4: if (rem==0) then
printf 'leap year'
else
Printf 'not leap year'

" 5: stop

☆ Scope of variables

* Scope of variable is defined as a space with in a program, where value of a given variable can be accessed.
* The effect of a variable can be felt within the block where the variable has been declared.
* similarly in scope.

Local scope :-

* The variables which are declared within any function are called local variables.
* These variable can be accessed only by the function in which they are declared.
* Default value is garbage value.

Example :-

```
#include<stdio.h>
Void main()
{
void display() // Local function
{
 int c; // Local variable
 c=7;
 printf("value of c: %d\n", c);
}
}
```

output :-

Value of c : 7

Global scope :-

* The variable which are declared outside any function are called global variables.
* These variables can be accessed by all the functions in the program.
* Default value is 0 (zero).

Example :-

```
#include<stdio.h>
 int d; // Global variable
void display(); // Global function declaration
Void main()
{
d= 8; //Assignment to global variable `d`
void display() // Global function called by main
```

```
{
printf("value of d is : %d\n", d);
}
void display()
{
printf("%d", d); // Global variable is accessible herealso.
}
}
```

Output :-

value of d is : 8

✡ Operator precedence

* Each operator in 'C' has a precedence associated with it.
* This precedence is used to determine, how to evalute the expression involving more than one operator.
* There are distinct levels of precedence and operator may belong to one of the level
* The operator at higher level of precedence are evaluted first and the operators of the same precedence are evaluated from left to rigright or right to left.

Precedence of Arithmetic operators

* An arithemetic expression involving paranthesis will be evaluted from left to right using the rules of precedence of operators.
* There are two levels.
  1) High priority arithmetic operators are *, /, % and
  2) Low priority arithemetic operators are +, -
* The basic evalution procedure includes two left to right passes through the exp.
* During the first pass high priority operators are evalut and In seconds pass the low priority operators are evaluated.

Examples :-

x = a-b/3+c*2-1;

if a=9, b=12, c=3

Evaluation...

First pass

step1 : x=9-12/3+3*2-1

Step 2 : x= 9-4+6-1

Second pass

step 3: x = 576-1

" 4: x = 11-1

" 5: x = 10

Program:-

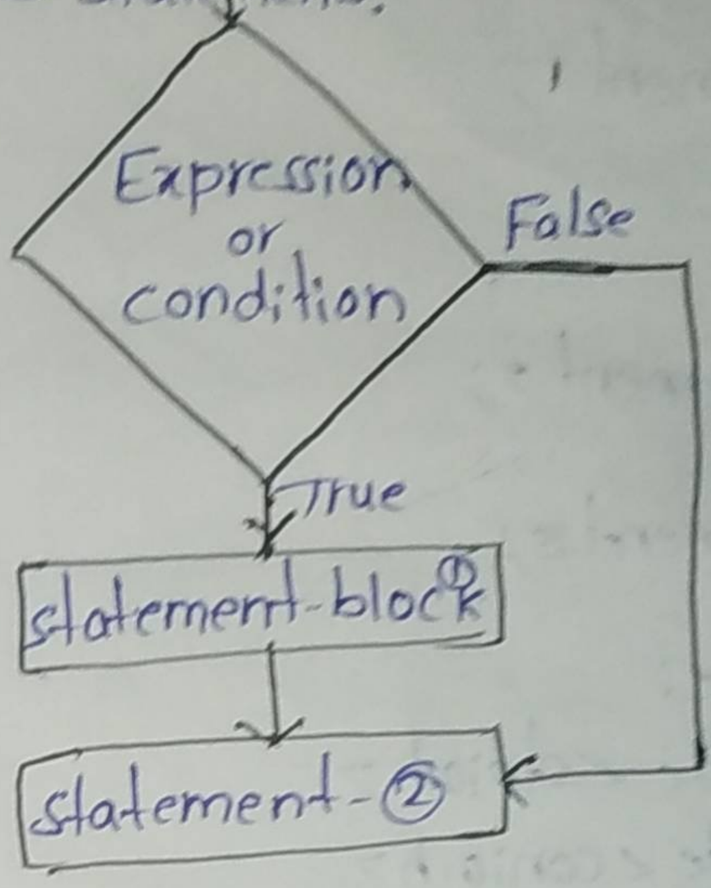
```
# include <stdio.h>
# include <conio.h>
Void main()
{
  int l, m, n;
  float P, q, r;
  clrscr();
  printf("Enter the values of l, m, n-\n");
  scanf("%d %d %d", &l, &m, &n);
  P= 2+m*5-n/3+1;
 q = 2+m*(5-n)/(3+1);
 r = (2+m*5) - (n/3+1);
 printf("%f \n %f \n %f \n", P, q, r);
  getch();
}
```

Output :-

```
Enter the values of l, m, n
10
20
30
102.000000
-36.000000
 80.000000
```

☆ If statement :-

* If statement is used for decision making.
* It allows the computer to first evaluate the condition and depending on the condition control to a particular statements in the program.
* This statement performs an action if the condition is true, otherwise it skips that action and executes other statements.



Syntax:

```
if(condition)
{
statement 1;
}
statement 2;
```

Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
int n;
clrscr();
printf("Enter the value of n:");
scanf("%d",&n);
if(n<=5)
{
printf("Hello\n");
}
printf("Welcome");
getch();
return 0;
}
```

Output:

```
Enter the value of n:8
Welcome.

Enter the value of n:2
Hello
Welcome.
```

☆ ☆

If condition is satisfied, then statement-1 is executed then statement 2.

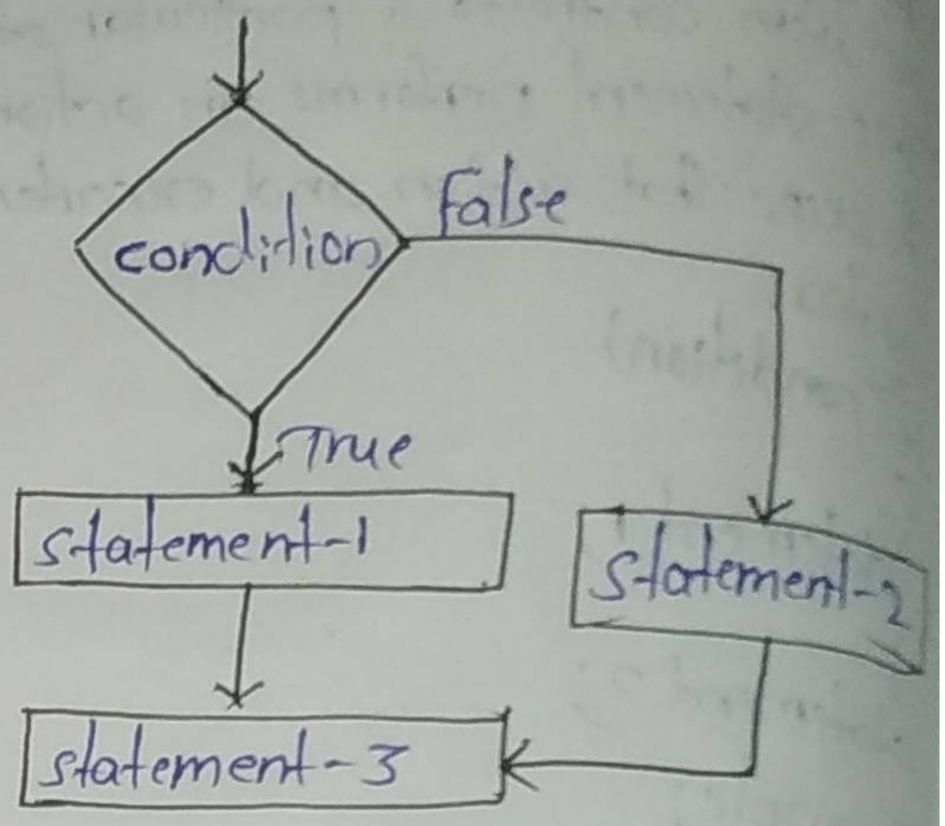
otherwise statement-1 is skipped and statement 2 is executed

✱ If-else statement:-

✱ This statement is an extension of simple 'if' statement.

Syntax:-

```
if(condition)
 {
  statement 1;
 }
 else
 {
  statement 2;
 }
  statement 3;
```



✱ Here when the condition is true statement 1 will be executed followed by statement-3. Otherwise, statement-2 will be executed followed by statement-3.

Example:

```
#include <stdio.h>
#include <conio.h>
main()
{
 int n;
 clrscr();
 printf("Enter the value of n:");
 scanf("%d", &n)
 if(n%2==0)
 {
  printf("The number entered is even\n");
 }
 else
 {
  printf("The number entered is odd\n");
 }
 printf("End of the program\n");
 getch();
 return 0;
 }
```

Output:-

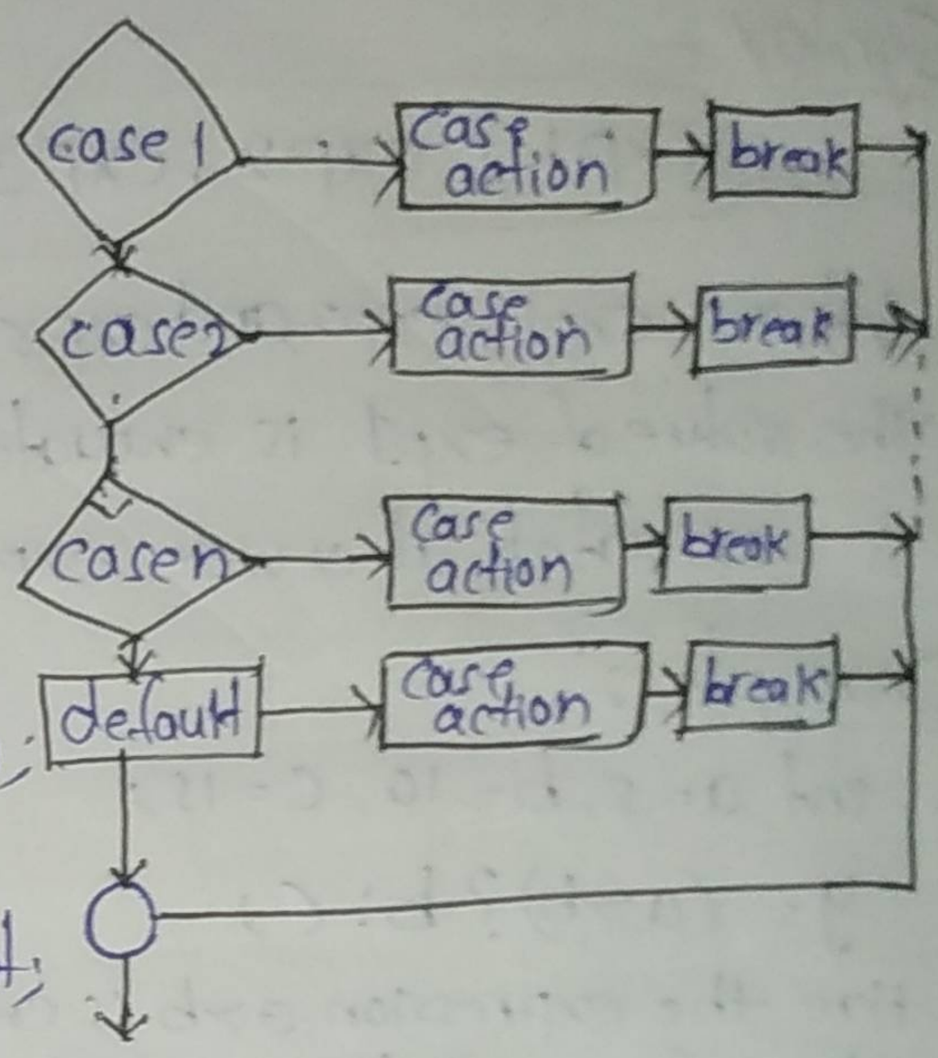
```
Enter the value of n:5
The number entered is odd
End of the program.
```

✱ switch Statement:

* A switch statement is a multi-way decision making statement

Syntax:

```
switch (expression)
{
    case constant1 : statement-1;
                     break;
    case constant2 : statement-2;
                     break;
    .
    .
    .
    case constantn : statement-n;
                     break;
    default        : default-statement;
                     break;
}.
statement-next;
```



Example

```
#include <stdio.h>
#include <conio.h>
main()
{
  char c;
  clrscr();
  printf("Enter the character: ");
  scanf("%c", &c);
  switch(c)
  {
    case 'a': printf("a is vowel");
              break;
    case 'e': printf("e is vowel");
              break;
    case 'i': printf("i is vowel");
              break;
    case 'o': printf("o is vowel");
              break;
    case 'u': printf("u is vowel");
              break;
    default : printf("The alphabet entered is a consonant");
  }
  getch();
  return 0;
}.
```

output

```
Enter the character: i
i is a vowel.
```

If statement 1 will be executed

☆ Ternary operator

* Conditional operator is also known as ternary operator.
* since it takes three arguments.

Syntax :-

```
exp1 ? exp2 : exp3;
```

Where exp1, exp2 and exp3 are expressions.

* The value of exp1 is evaluated first. If it is true, value of exp2 is evaluated otherwise exp3 is evaluated.

Examples:-

1. int a=5, b=10, c=15;

   y = (a>b) ? b : c;

   Here the expression a>b is evaluated first and since it is false the value of c will be assigned to y. so y=15.

   ⇒ y = (a<b) ? b : c;

   Here, a<b is true, so y=b. therefore y = 10

2. Greatest of three numbers using ternary or conditional operator.

   x = (a>b) ? (a>c) ? (b>c) ? a : b : c;

   greatest of three num is stored in x.

Program :-

```
# include <stdio.h>
# include <conio.h>
main()
{
int num1, num2, num3, largest;
clrscr();
printf("Enter any three numbers:");
scanf("%d %d %d", &num1, &num2, &num3);
largest = (num1>num2 && num1>num3 ? num1 : num2>num3 ? num2 : num3);
printf("the largest among three numbers is : %d", largest);
getch();
return();
}
```

output :-

Enter any three number :

8
9
9

the largest among three numbers is : 9.

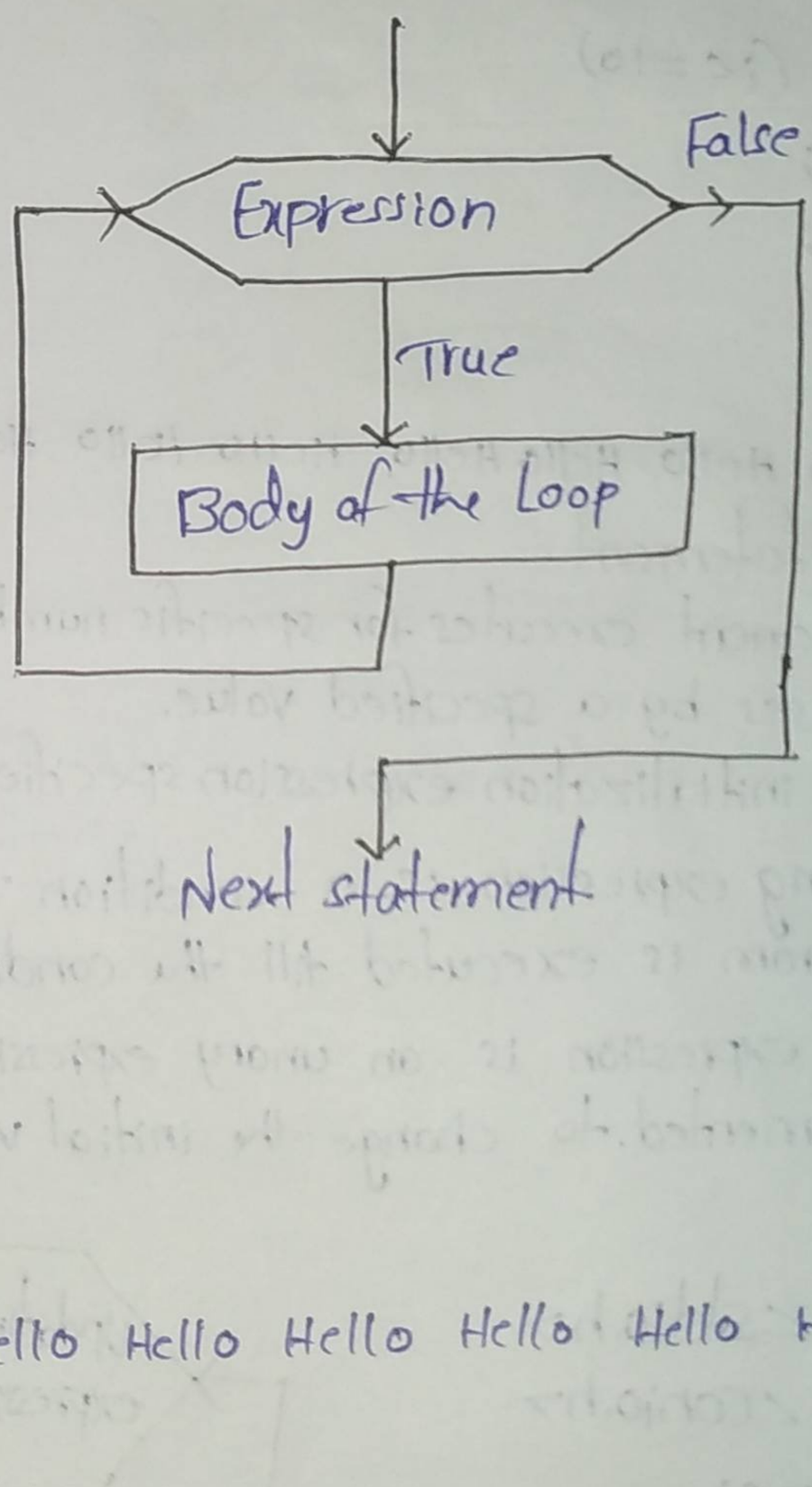
☆ While Statement

* The while statement will be executed repeatedly as long as the expression remains true.

Syntax:

```
While (expression)
{
body of the loop;
}
```

* When the while is reached, if the expression or condition is false, body of the loop will not be executed & loop terminates otherwise, the body of loop will be executed. till the expression becomes false



Example:

```
#include <stdio.h>
#include <conio.h>
main()
{
  int i=1;
  clrscr();
  while (i<=10)
  {
    printf ("Hello \n");
    i++;
  }
getch();
return 0;
}
```

Output:-

Hello Hello Hello Hello Hello Hello Hello Hello Hello Hello.

Output:-

FAWAZ
FAWAZ
FAWAZ
FAWAZ
FAWAZ

* Program to evaluate the power series,

$e^x = 1 + x + x^2 + x^3 + \ldots\ldots + x^n$, $0 < x < 1$

```
# include <stdio.h>
# include <conio.h>
# include <math.h>
void main ( )
{
long float res = 0.0;
float x;
int n, i;
clrscr();
printf("The given series is 1+x+x^2+x^3+...x^n \n");
printf("As n -> infinity; x^n -> 0 \n");
printf("Enter the value of x between 0 and 1: ");
scanf("%f", &x);
if((0<x)&&(x<1))
{
printf("Enter the value of n: \n");
scanf("%d", &n);
   for (i=0; i<=n; i++)
   res = res + pow(x,i);
printf("\n The sum of the series upto power %d of x is: ", n);
printf("\n E^x = %lf", res);
}
else
printf("\n x should lie between 0 and 1");
getch();
}
```

Output:

```
The given series is 1+x+x^2+x^3+...x^n
As n -> infinity; x^n -> 0
Enter the value of x between 0 and 1:.5
Enter the value of n: 4
The sum of the series upto power
 4 of x is:
E^x = 1.937500
```

☆ To find the squares of N numbers using Do-while.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   int i, n, res;
   int a[10];
   clrscr();
   printf("Enter the value of n: \n");
  scanf("%d", &n);
   printf("Enter numbers");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
   i=0;
   do
   {
     res = a[i] * a[i];
     printf("The square of number %d is %d\n", a[i], res);
     i++;
   }
   while(i<n);
   getch();
}
```

Output:

```
Enter the value of n: 5
Enter numbers 25 15 10 8 12
The Square of number 25 is 625
The Square of number 15 is 225
The Square of number 10 is 100
The Square of number 8 is 64
The square of number 12 is 144.
```

☆ Array :-

* An array is a collective name given to a group of similar elements.
* An array is a variable that is capable of holding many values whereas, an ordinary variable can only hold a single value at a time.
* It can stored values of same type.

One-dimensional Array :-

* It is the simplest form of array in which list of elements are stored in contiguous memory locations and are accessed using only one subscript.

Declaration of one-dimensional array :

The general format for declaring one dimensional array is

syntax :-

```
datatype array-name[size];
```

Here datatype - specifies data type of array elements.
array name - indicates the name of the array variable
size - specifies the number of elements in the array.
It is fixed, it cannot be changed

Example :

```
int a[10]
```

Here array 'a' is of type int, that can store 10 elements.

Initialization of one-dimensional Array:

* User can initialize array elements at the place of their declaration itself.

Format :-

```
type array-name[size] = {list of elements separated by commas};
```

Example!

* `int a[4]={5,10,1,55};`
* `int a[3]={8,5,10};`
* `int xyz[]={5,6,7,9};`

Printing One-dimensional Array:

Array elements can be printed by using "for" loop and a printf statement.

Example:

```
int a[5] = {10,20,30,40,50};
for(int i=0; i<=5; i++)
  printf("The elements of array are: %d", a[i]);
```

✱ Functions

A function consist of three elements

i) Funtion Declaration - Declares a function, consist of function type, fun name, parameter list, semicolon

Syntax :- Function-type function-name (parameter-list);

Example :- float sum (float x float y);

ii) Function Defination - ~~Defines~~ complete description of fuction.

Syntax :-

return_type function_name (parameter_list)

```
{
  local variable declaration;
  ------
  statements;
  -------
  return (expression);
}
```

Example

```
float sum (flotd x, floaty)
 {
   floatz;
   Z= x+y;
   return (z);
 }
```

iii) Function call :- A fuction can be called with fuction name and alist of actual parameters enclosed in parameters.

Syntax :- F(...) F- operand or fuction name

Example :- A fuction can be called in various ways → 'Faz ( )'

→ Faz (8, 6)

→ Faz (z, 6)

→ Faz (8, z)

→ Faz (Expression1, Expression2)

Program :-

```
#include <stdio.h>
#include <conio.h>
int mul(intx, inty);
Void main()
{
```

```
int x,y,z;
clrscr();
printf("Enter the first number \n");
scanf("%d", &x);
printf("Enter the second number \n");
scanf("%d", &y);
z= mul(x,y);
printf("%d * %d = %d", x,y,z);
getche();
}
int mul(int x, int y)
{
   int p;
   p = x*y;
   return(p);
}
```

output:-

```
Enter first number
5
Enter the second number
9
5*9 = 45
```

✱ Library Functions:-

✱ sqrt(x):-

* It return the non-negative square root of given x value.
* However x value should be greater than zero.

Example:-

```
# include <stdio.h>
# include <conio.h>
# include <Math.h>
int main()
{
 float n=144, res;
 clrscr();
 res = sqrt(n);
 printf("The square root of %f is %f", n, res);
 getche();
 return 0;
}
```

output:-

```
The square root of 144.000000 is
12.000000
```

✡ toupper(x):-

* It transforms lower case letters into uppercase letters.
* The declaration of toupper(x) function is,

```
int toupper(int ch);
```

Example:-

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
int main()
{
  int ch='x';
  clrscr();
  printf("Uppercase of %c is %c\n",ch,toupper(ch));
  getch();
  return 0;
}
```

output:-

Uppercase of x is X.

adecimal

Remainder

4 ↑

5

nal

maindel

12 ↑

5

ructions

processor

compiler

* Call-by-value:

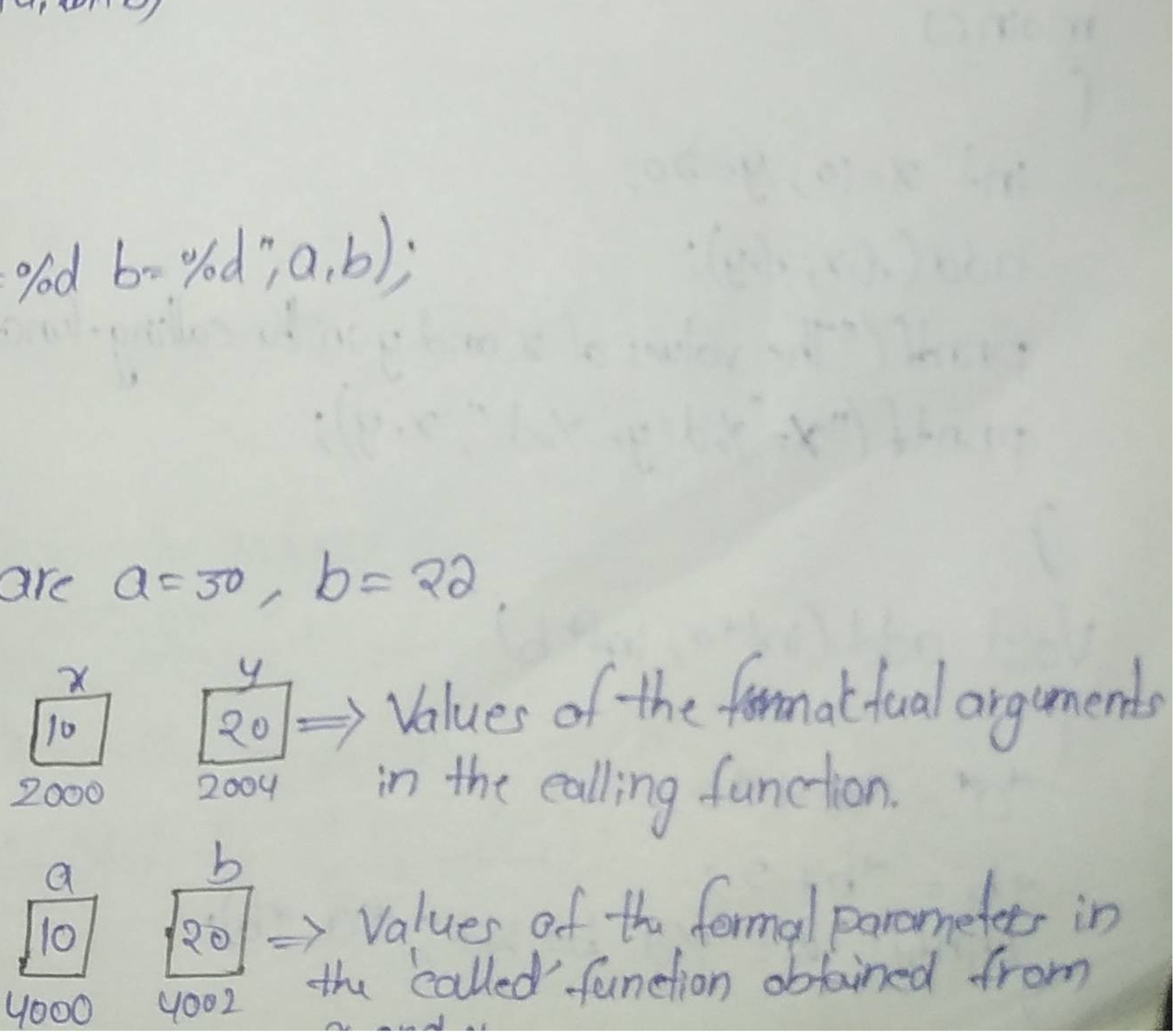
In this method, values of the actual arguments in the 'calling function' are copied into the corresponding parameters in the 'called function'. Hence, the modification done to the arguments in the called function will not be reflected (or) changed in the actual arguments.

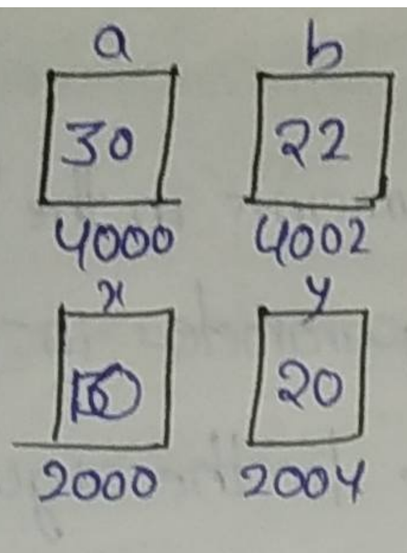
Example:

```
#include<stdio.h>
#include<conio.h>
Void add(int a, int);
main()
{
 int x=10, y=20;
 add(x,y);
 printf("The values of x and y in the calling function are");
 printf("%d %d", x,y);
}
Void add(int a, int b)
{
 a=a+b;
 b=b+2;
 printf("a=%d b=%d", a,b);
}
```

output:-

The values are a=30, b=22.





=> Values of 'a' and 'b' in the called function after executing the 'add' function

=> The values of x and y in the calling function remains unaffected as the 'call by value' mechanism is used.

Fig: Call by Value Mechanism

* Call - by - Reference Mechanism:

In this method, the addresses of the actual arguments are copied to the corresponding parameters in the 'called' function Hence, any modifications done to the formal parameters in the 'called function' causes the actual parameters to change.

Example:-

```
#include <stdio.h>
#include <conio.h>
Void add(int*, int*);
main()
{
   int x=10; y=20;
   add(&x, &y);
   printf("The values of x and y in the calling-function\n");
   printf("x= %d y= %d", x, y);
}
Void add(int*a, int*b)
{
  *a*= *a + *b;
  *b = *b + 2;
   printf("a = %d  b= %d", *a, *b);
}
```

output:

The values are a=30, b=22.

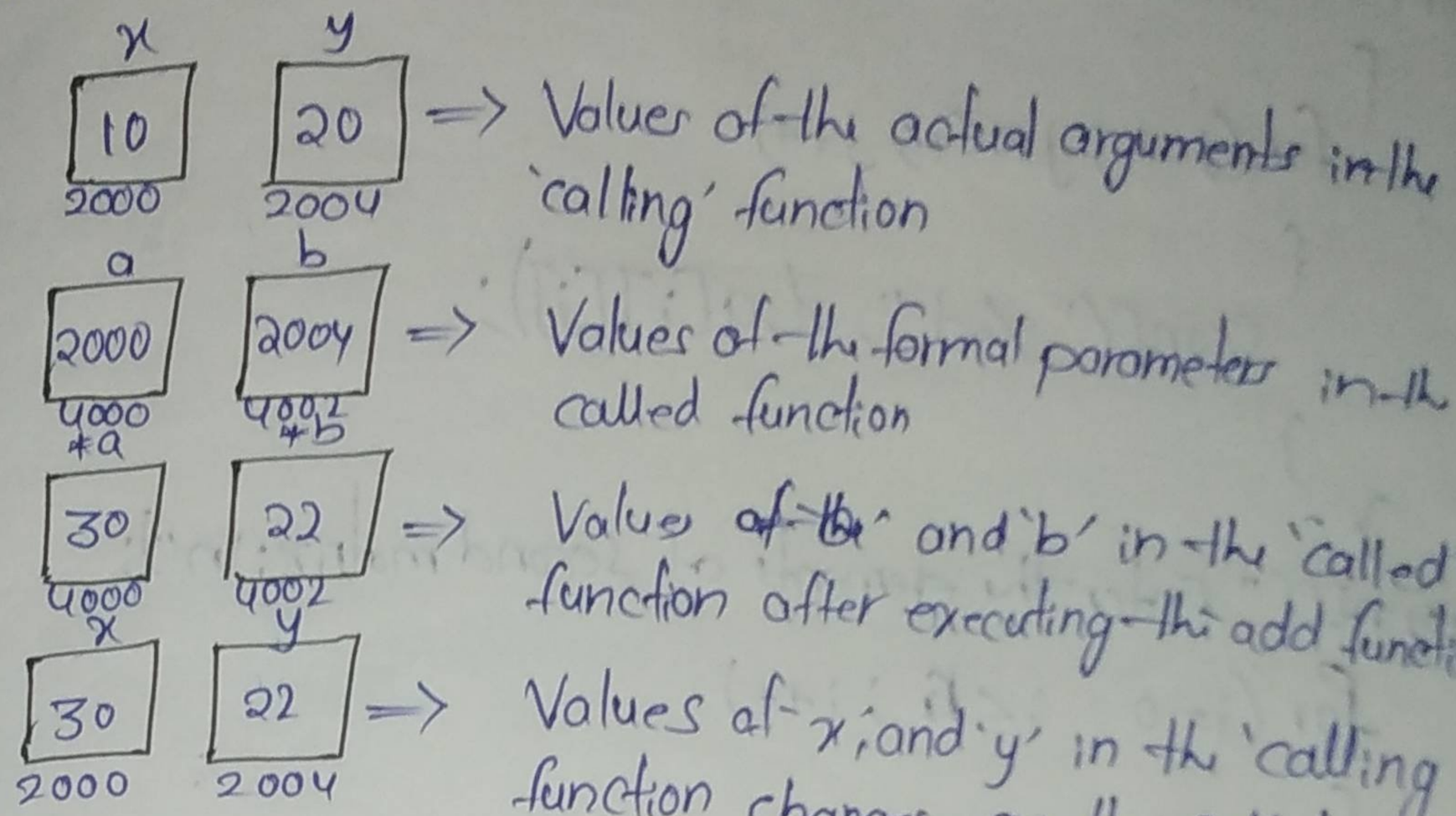


Fig: Call-by-Reference Mechanism

Program for Multiplication of Two Matrices:

```
// Program to multiply two matrices
#include <stdio.h>
#include <conio.h>
Void main()
{
  int r1, c1, r2, c2, i, j, k;
  int a[10][10], b[10][10], c[10][10];
  clrscr();
  printf("Enter the order of first matrix:\n");
  Scanf("%d %d", &r1, &c1);
  Printf("Enter the order of second matrix:\n");
  Scanf("%d %d", &r2, &c2);
```

```
Printf("Enter the elements of first matrix:\n");
 for(i=0; i<r1; i++)
    {
      for(j=0; j<c1; j++)
       {
        scanf("%d", &a[i][j]);
       }
    }
printf("Enter the elements of second matrix:\n");
   for(i=0; i<r2; i++)
   {
     for(j=0; j<c2; j++)
      {
       scanf("%d", &b[i][j]);
      }
// Matrix Multiplication
if(c1==r2)
{
    for(i=0; i<r2; i++)
      {
       for(j=0; j<c2; j++)
       {
        c[i][j] = 0;
        for(k=0; k<r2; k++)
        c[i][j] = c[i][j]+a[i][k]*b[k][j];
       }
}
```

```
for(i=0; i<r1; i++)
{
  printf("\n");
  for(j=0; j<c1; j++)
  {
   printf("%d\t", c[i][j]);
  }
 }
}
else
printf("Multiplication not possible");
getch();
```

Enter the order of first matrix:

2 2

Enter the order of second matrix:

2 2

Enter the elements of first matrix:

3 4

5 6

Enter the elements of second matrix:

4 5

6 8

$$\begin{bmatrix} 36 & 39 \\ 56 & 57 \end{bmatrix}$$

What is array ? Explain two dimensional arrays.

⇒ An array is a collective name given to a group of similar elements whereas a variable is any entity that may change during program execution.

⇒ An array is a variable that is capable of holding many values.

2-D array

When the data must be stored in the form of a matrix two-dimensional arrays are used.

Int a[5][3];

Above declaration represents a two-dimensional array consisting of 5 rows & 3 columns. so. the total no. of elements which can be stored in this array are 5X3 = 15.

Declaration of Two-dimensional Array.

type array-name [row size][column-size];

- type → datatype
- array-name → name of array
- row size → no. of rows in array
- column-size → no. of column in array.

Initialization of 2-D array

type array name [row size][column-size] = {{list of elements in row1},
{list of elements in row2},
⋮
{list of elements in rown}};

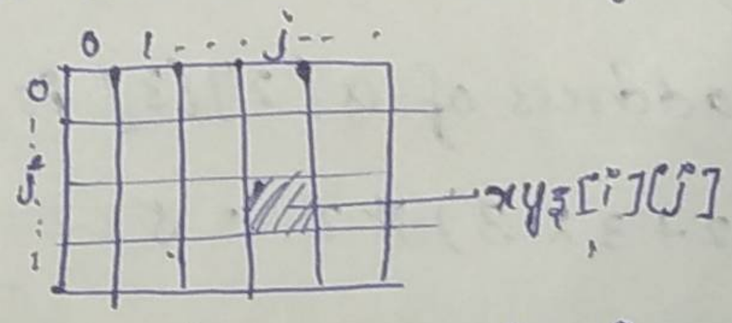
Ex:- int xyz[2][3] = {{2,3,4},{5,6,7}};

Referencing Elements of 2-D array.

The elements of an array can be referenced using two indices, one for obtaining row & another for obtaining column number.

Ex:-

xyz[i][j] is used to access i^{th} row & j^{th} column element



The no. of elements in a 2-D array is given as,

No. of rows x No. of columns.

Storage Representation of 2-D array :-

⇒ 2-D array consists of rows & columns. but when it is stored in memory, no facility for 2-D storage is available. The memory is linear. Hence, the actual storage is differ from matrix presentation.

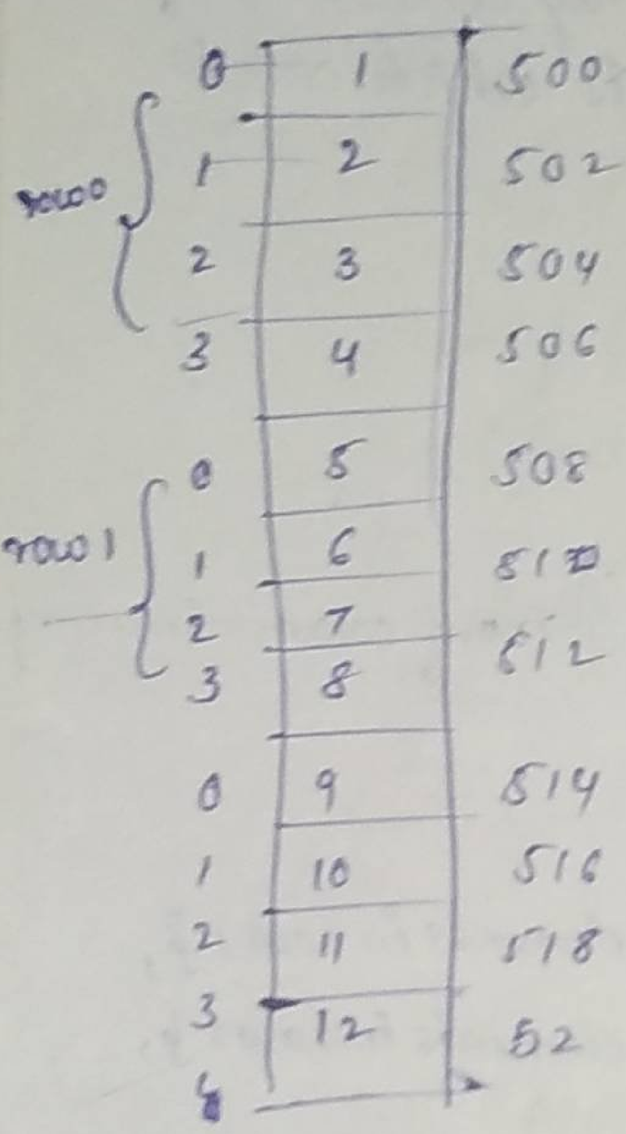
a) Row major representation

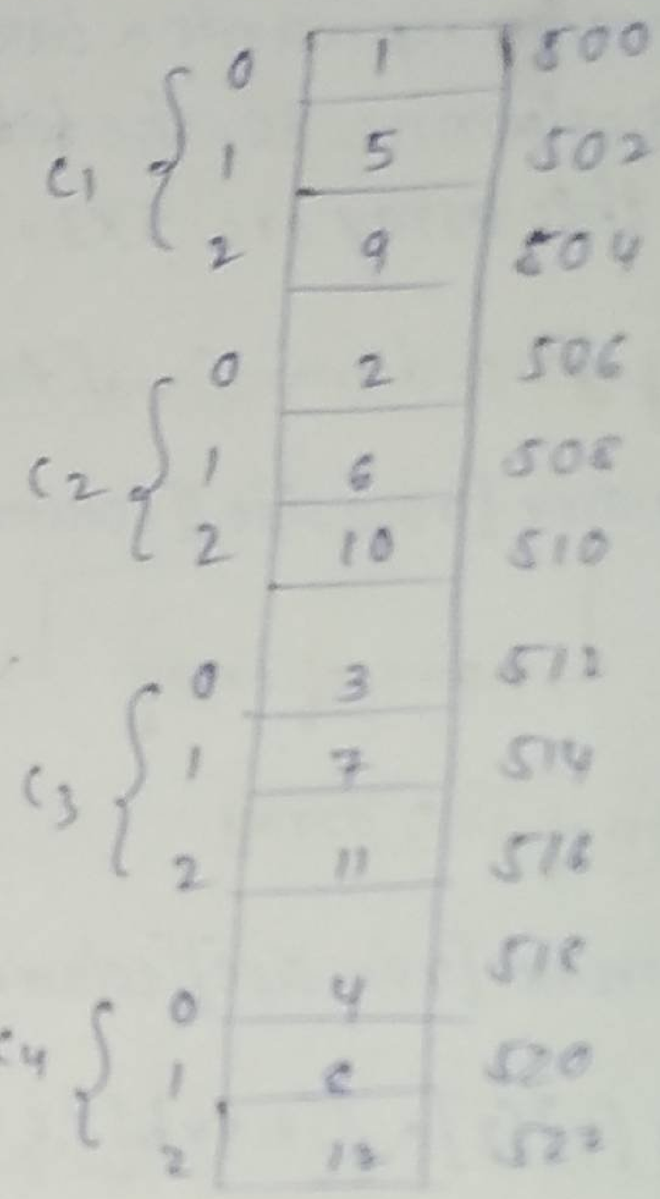
b) Column major repre

Ex:-

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

int a[3][4];





Accessing Address of an array Element

In an array a[m][n] stored using column major order, the address of a[i][j] is given as,

Base address + (i+j*m) x size of array element

Ex:-

a[3][4], address of a[2][3] is,

500 + (2+3x3) x 2 = 522

Explain the process of accessing a variable through its pointer. Give an example

Accessing a variable through pointer

When a variable (eg: int n=5) is declared in c, the complier will perform the following functions

i) Allocates space in the memory for the variable.

ii) Name is given to the memory location

iii) Stores the value at the memory location.

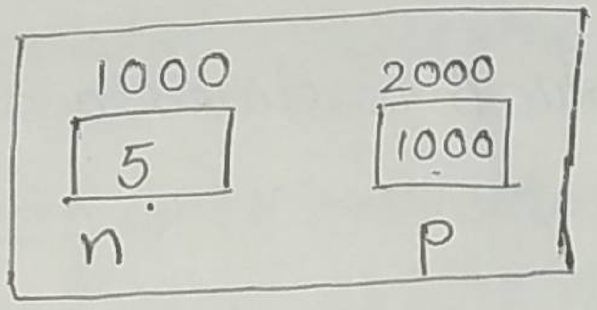
iv) C provides '&' (address of operator) which returns the address of associated variable.

⇒ &n → Returns the address of the variable 'n'

Exi- int *p

p = &n

p → It is a pointer variable.



Memory.

⇒ '*' is the "value of address" operator.

⇒ The variable 'n' can be accessed through its pointer p.

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int n=5;
   int *p;
   p=&n;
printf("\n Address of n = %ou", &n);
printf("\n Address of n = %u", p);
printf("\n Address of p = %ou", &p);
printf("\n Value of p = %d", p);
printf("\n Value of n = %d", n);
printf("\n Value of n = %d", *(&n));
```

Explain the process of declaring & initializing pointers. Give an example.

Declaration of pointers

⇒ Pointers are declared as follows,

datatype * ptr_Variable ;

Here '*' indicates that ptr_variable is a variable representing value stored at a particular address.

Ex:- int *p.

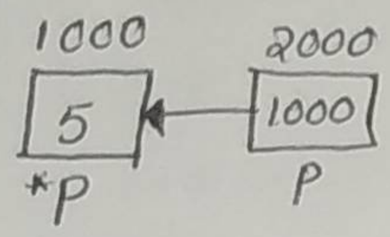
p = pointing to address location where an integer data type is stored

Initialization of pointer variable.

⇒ pointer variable can be initialized after declaration as given below.

int *p

*p = 5;



int n = 5

int *p

*p = 5

p = &n.

Here p = 1000 and *p gives value 5.

⇒ on compilation of above statements the value 5 is stored in an address pointed by p.

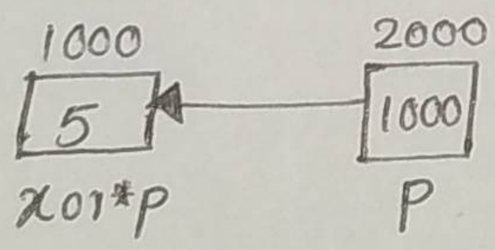
Assigning of Address of variable to pointer variable.

⇒ A pointer variable is used to point address of other variables.

int x = 5;

int *p;

p = &x;



Here address of variable x is assigned to pointer p.

*p gives the values at location 1000 i.e, 5. which is same as x.

⇒ '&' is used as address of operator.

Ex:- int *p = &x;

Pointers in self Referential structures

⇒ A structure defination in which includes atleast one member as a pointer to the same structure is known as self-referential structures.

⇒ Pointer can be used to point the member of self-refertial structure

Ex:-

```
struct student
{
 char name[50];
 int rollno;
 struct student *ptr;
};
```

In the above example, the pointer will point the structure element student.

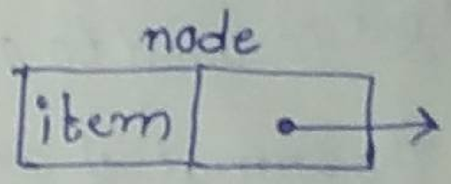
Self Referential structures In Linked list.

⇒ Self Referential structures are used in implementation of linked data structures like list of trees etc.

⇒ A linked list is a collection of nodes, where each node points to the next node of the list.

⇒ data can be added or deleted easily by using linked list.

⇒ each node consists of two items
1) data 2) address of next element.



⇒ In a linked list, nodes represent self-referential structures & arrow represents pointers that makes a list (linked) of self refrential structures

```
struct node
{
 char item[50];
 struct node *ptrnode;
};
```

Write short notes on,

(a) Pointer constants (b) Pointer values (c) Pointer variables.

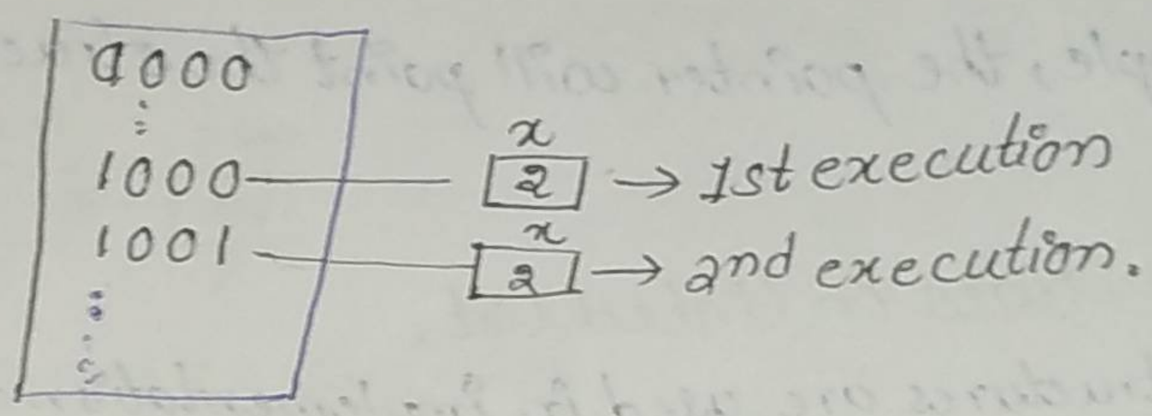
(a) Pointer constants.

⇛ Pointer constants are the collection of addresses or locations in the memory which are assigned to the variables.

⇛ Pointer constants are cannot be changed themselves, but they can change during runtime due to modern operating system for its convienence.

Ex:- Consider a Variable 'x' its value is '2'. Is stored at memory addrees 1000 in first execution.

⇛ It is not necessary during next execution it stores the variable at same address It may store at 1001.



(b) Pointer values

⇛ Pointer values are nothing but the values of pointer constants (addrees) that are assigned to the variables.

⇛ '&' is used to obtain the address of variable

⇛ The format of an address expression is.

| &Variable Name |.

⇛ Depending on data types the address is assigned to the variables.

⇛ If data type is Integer it takes 4 bytes.

⇛ If data type is character it takes 1 byte.

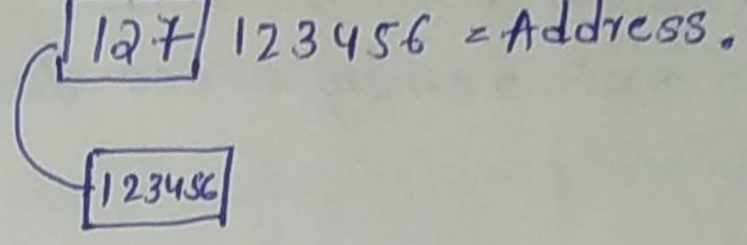
(c) Pointer Variables

⇒ A pointer Variable is a variable that stores the address of another variable.

Ex:- consider variables 'var' & 'ptr-var'.

⇒ The var stores a value '127' at address 123456

⇒ The 'ptr-var' stores the address of 'var' i.e 123456.



File

⇒ A file is a permanent named unit that holds large volumes of related d
⇒ It stores data in secondary memory such as CDs, tapes, DVDs etc.
⇒ It is used to support volatile nature of main memory.
⇒ main memory looses data when the system is shut down & depend on seconda
⇒ main memory gets loaded with data from the files when system is on
⇒ main memory cannot hold all the data at a time.
⇒ I/ access the data from files as when needed.
⇒ The operating speed between main memory & secondary memory differs largely.
⇒ A buffer is a temporary storage used for data transfer between main memory & Secondary memory.
⇒ The entire reading and writing process is by buffer.
⇒ The entire process of buffer is managed by operating system.

File Name

⇒ The file name must be specified for a particular file.
⇒ The file name is typically a string of characters.
⇒ The name of file's can be Data.txt, Add.c PRI.C & soon.
⇒ The extensions can be given depending upon the usage of file.
⇒ If files is used for storing text use .txt extension.
⇒ If files is used for storing program use .c extension
⇒ If it is a header file use .h ff extension.

File Information Table

⇒ To read or write a file in program, certain information is require like file name, the location of the current character of the file.
⇒ Such Information is stored in a predefined structure called file Information table.
⇒ This table is defined by 'stdio.h' header file with 'FILE' as its identifier.

rewind()

This function takes file pointer to the starting of the file and resets it.

```
rewind(fp);
```

ftell()

⇒ This function gives the current position of file pointer in terms of bytes from the begining.

```
n = ftell(fp);
```

⇒ 'n' gives the current position of file pointer from the beginning.
⇒ It gives the information regarding number of bytes read or written already

fseek()

⇒ This function is used to move the file pointer to desired location in file.
⇒ The main purpose of file fseek() is to position of file pointer to any location on the stream.

```
fseek(fileptr, offset, position);
```

offset specifies the number or variable of type reposition the file pointer

fileptr → is a file pointer.

offset tells the number of bytes to be moved

offeset can be positive or negative integers.

positive → reposition the pointer forward
negative → reposition the pointer backward.

1) fputc()

⇒ The fpute functions are commonly used to write a character to a predefined file stream

⇒ The 1st parameter in this function represents character to be written

⇒ The 2nd parameter in this function represents file itself.

⇒ It returns Eof if an error occured.

int fputc(intonechar, file *spout);

2) fputs()

⇒ The fputs() function is used to write file.

⇒ It removes the null charater from the string.

⇒ This function returns non-negative integers if no error occur otherwise it returns EoF.

int fputs(const char *pstring, file *p);

3) fgetc()

⇒ The fgetc() function is used to read the next character from the predefined file stream and converts it to integer.

⇒ The If an error is encountered then this function retuns Eof

int fgetc(File * spMyfile)

4) fgets()

⇒ The fgets() function accepts data from a file.

⇒ The parameters of this functions. string pointer, array size

char* fgets(chair* strptr, intsize, file *sp);

fprintf()

⇒ fprinf() function is used for writing records into a file.

⇒ This function outputs the matter given in quotations in fprintf()
to the display screen

```
fprinf(file_descriptor, "format string", list);
```

fScanf()

⇒ This function is used for reading records from the file

```
fscanf(file-descriptor "formatstring", address of list);
```

fopen()

⇒ This fopen() function is used to open a file. It has two parameter
i) file name ii) file open mode.

⇒ The fopen() is common for both the text files & binary files

```
fopen(file_name, file_open_mode);
```

fclose()

⇒ This function fclose() is used to close the file after required
operations

```
fclose(file_pointer);
```

⇒ If multiple files are to be closed, the multiple fclose() function
must be written.

Function	Recursive function.
A function is a block of code which performs specific task.	A recursive function is a special type of function which calls itself by performing some task repeatedly
It has less time complexity	It has high complexity compared to functions complexity.
It is easy to understand	It is difficult to understand.
It cannot be converted into algorithm easily	It can be converted into algorithm easily.

Object code	Executable code
The program that is written in the form of binary digit	The program consists of set of code encoded form to be performed by the computer.
It is difficult to modify program	Processor can easily read and modify the program.
It is also known as object program.	It is also known as executable program.

Compiler	Interpreter.
The compiler translates the program at single instance	The Interpreter translate the program line by line. (sequentially)
It consumes less execution time	It consumes more execution time
It converts the program after removal of syntax error after completion of execution.	It check every line for syntax error for executing the program & executed and then converts it
Its debugs the program at very slow pace.	It debugs the program very fastly.

Structure	Array.
Structure is a user-defined data type.	Array is a derived data type.
It contains heterogenous data type	It contains homogeneous data type.
It uses dot operator for accessing elements	It uses subscript for accessing array elements.
It is known as dynamic memory allocation since the size of the structure can be changed dynamically.	It is known as static memory allocation since the size of an array cannot be change

break

The keyword 'break' is used for terminating loops.

⇒ It makes control to jump the next statement after the loop or block.

⇒ This statement is used in for, while, do-while and switch statements.

Syntax

```
{
statement-1;
statement-2;
:
statement-n;
break;
}
```

Continue

The keyword 'continue' is used for continuing the next iteration of the loop.

It makes control to continue the next statement after loop or block.

This statement is used with for, while & do-while loops.

The loop does not gets terminated rather it skips the statements after continue.

Syntax

```
{
statement-1;
continue;
statement-2;
}.
```

Algorithm

Alogrithm is a step-by-step process to solve a problem

It defines the Instruction in simple English language.

⇒ It is used for large and modular programs

⇒ It is easier to understand

flowchart

flowchart is the pictorial/ diagrammatic representation of an algorithm.

It defines the Instruction in various boxes of flowchart of an algorithm.

⇒ It is used for small pro

⇒ It is easier to understan when symbols are know

Define recursion

Recursion is a process or technique by which function calls itself. A recursive function contains a statement within its body. which calls the same function. Thus, it is also called as circular definination.

⇒ A recursion is classified as two types

1) direct recursion 2) Indirect recursion.

⇒ In direct recursion the function calls itself

⇒ In Indirect recursion the first function (f_1) calls another function (f_2) and the called function (f_2) calls the calling function (f_1).

Ex:- Implementation of factorials by using direct recursion.

Advantages

⇒ Recursion solves a problem in the most general way as possible.

⇒ Recursion function is small & simple compare to other coded versions of the program

⇒ For some problems it is easy to understand using recursion

⇒ Recursion is used to solve the more complex problems that have repetitive structure.

How function is declared

⇒ A function declaration declares a function that consists of a function type (or return type), function name, parameterlist (arguments) & and a terminating semicolon.

⇒ function declaration is also called function prototype.

⇒ function must be declared before it is invoked.

Syntax

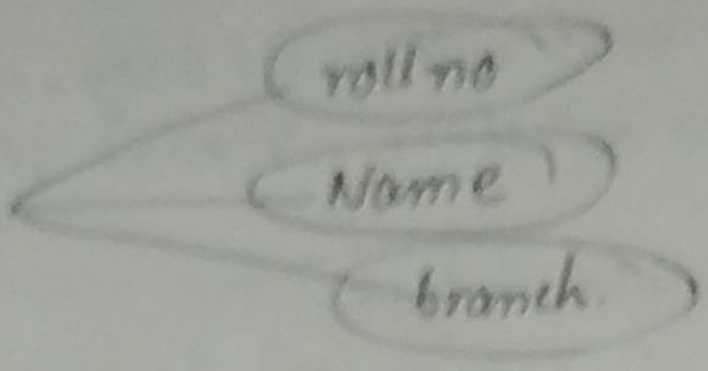
```
function-type function name (parameter-list);
```

where

parameter list is a list of arguments with datatype separated by comma.

Structure is a collection of heterogeneous data type. under a same or single name.

Ex:-



Advantages

- Structures can create a complex data type.
- They can easily represent complex numbers in mathematics that contain real & Imaginary parts
- They can hold the locations of points in multi-dimensional Space.
- Pointers to structure can be used to create linked lists etc.

Syntax:-

```
struct student
{
    int marks 1;
    int marks 2;
    int marks 3;
};
```

Array of structures

An Array is a collection of same data type. Grouping structure statements variables into one array is reffered as an array of structures.

To declare array of structures

```
struct structure student
{
    int m 1;
    int m 2;
    int m 3;
} st[3]
```

∴ st[3] is an array of 3 elements of students. with members that are marks 1, marks 2, marks 3,

Write the various steps involved in executing a 'C' program

⇒ The phases or steps required for successful execution of a 'C' program can be visualized as follows.

Creating a program
↓
Compiling the program
↓
Linking the program
↓
Executing the program

1) Creating a program :-

⇒ In this phase, the 'C' program is entered into a file (i.e source code) through a text editor.

⇒ A text editor allows the user to enter, modify and store the character data.

⇒ Corrections in the program at later stages are done through these editors

2) Compilation phase :-

⇒ This phase is carried out by a program called compiler. Compiler translates the source code into the object code (i.e machine understandable code)

⇒ The compilation phase cannot be proceeded successfully until & unless the source code is error free.

⇒ The error-free source code is translated into object code & stored in a separate file with an extension "obj".

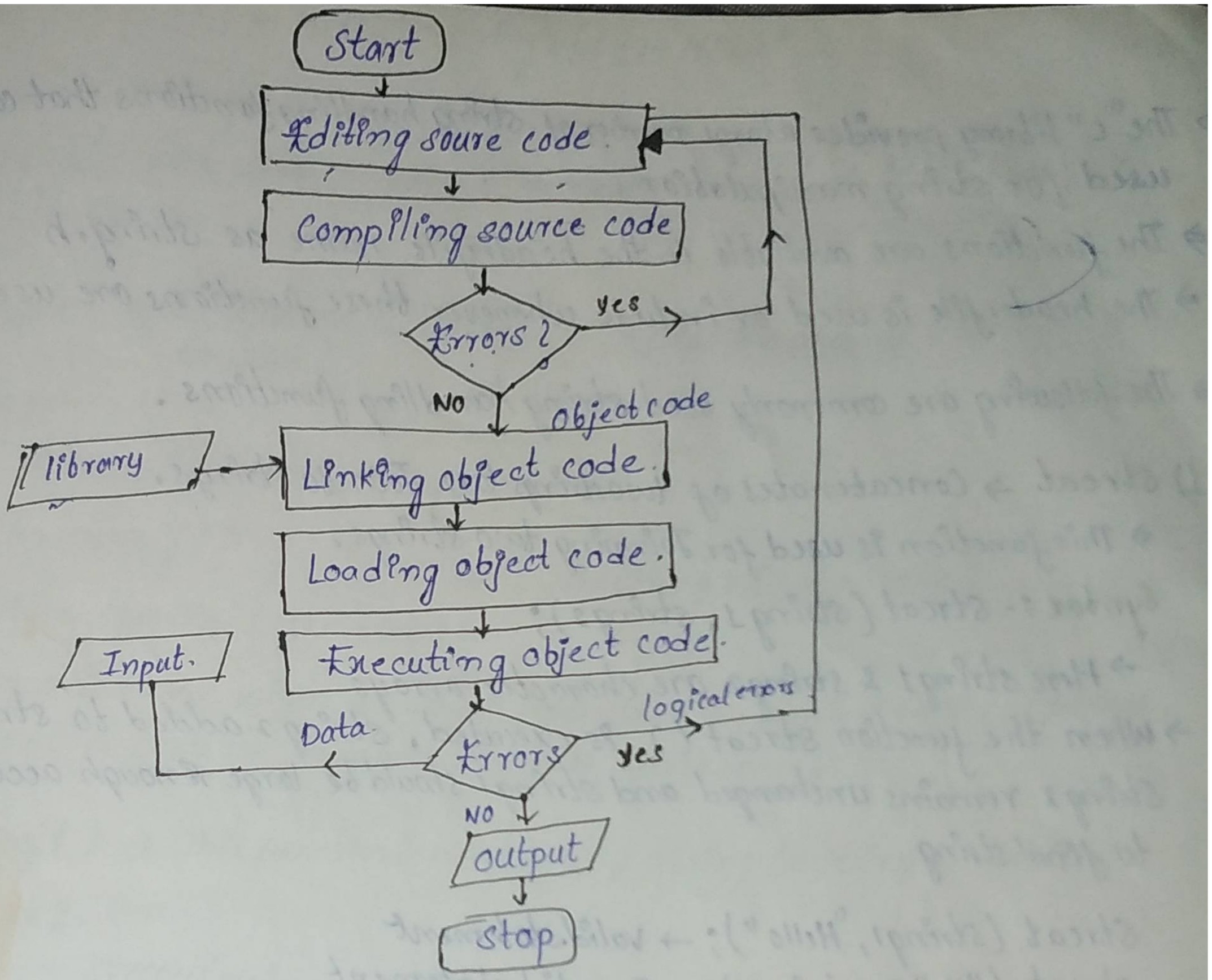
3) Linking phase :-

⇒ In this phase, the linker links all the files and functions with the object code under execution. Now the object code is ready for next phase. (i.e execution phase).

4) Execution phase

In this phase, the executable object code is loaded into the memory and the program execution begins

⇒ It executes even after it has errors. These errors can be corrected



⇒ The "C" library provides a large number of string handling functions that can be used for string manipulation.

⇒ The functions are available in the headerfile name as string.h

⇒ The header file is used or include whenever these functions are used.

⇒ The following are commonly used string handling functions.

1) Strcat → Concatenates of two strings (or) Join to strings.

⇒ This function is used for Joinning two strings.

Syntax :- strcat (string1, string2);

→ Here string1 & string2 are character arrays.

→ When the function strcat ( ) is executed, string2 added to string1. String2 remains unchanged and string1 should be large enough accomodate to final string.

strcat (string1, "Hello"); → valid statement

strcat ("Hello", string2); → Invalid statement.

char string1 [11] = "HELLO"

char string2 [6] = "WORLD"

strcat (string1, string2)

H	E	L	L	O	W	O	R	L	D	\0
0	1	2	3	4	5	6	7	8	9	10

2) strcmp ( ) → These functions is used to compare strings.

Syntax :- strcmp (string1, string2);

→ Here both string1 & string2 are character array.

→ One of the string, can be string constant.

⇒ string1 & string2 are compared as per "ASCII" sequences rules.

⇒ string1 = string2, then value '0' is return.

⇒ String1 > String2, positive value is returned

⇒ String1 < string2, Negative value is returned.

Ex:- string1[6] = "HELLO", string2[6] = "HELP"

Int i = strcmp(string1, "HELLO");

The value returned when the above statement is excuted is '0' because strings are equal.

3. Strcmpi() → It is same as strcmp(), except the case of characters not consider.

Syntax:-

Strcmpi(string1, string2);

Ex:- charstring1[ ] = "HELLO", charstring2[ ] = "hello"

int i = strcmpi(str1, str2);

The value returned when the above statement is excubed to '0'. Because those function does not consider the case characters.

4. Strcpy() → This function is used to copy string to string variable.

Syntax:-

strcpy(string1, string2);

→ Here str1, str2 are character arrays string2 may be a string variable or string constant.

→ Content of str2 are copied into str1, str1 should be large enough to accomodate to final string.

Ex:- strcpy(name, "HELLO WORLD");

when the above statement is executed the string "HELLO WORLD" will be assigned to string variable "name".

5. Strlen() → The function is used to determine the length of the string

Syntax:- Strlen(string1)

Ex:- string1 = "HELLO WORLD"

Int n = Strlen(string1)

It gives n = 10

6. Strrev() → The function is used to reverse the content of the given string.

Syntax:- strrev(string);

string1 = "hello"

Int s = strrev(string1);

It gives s = olleh