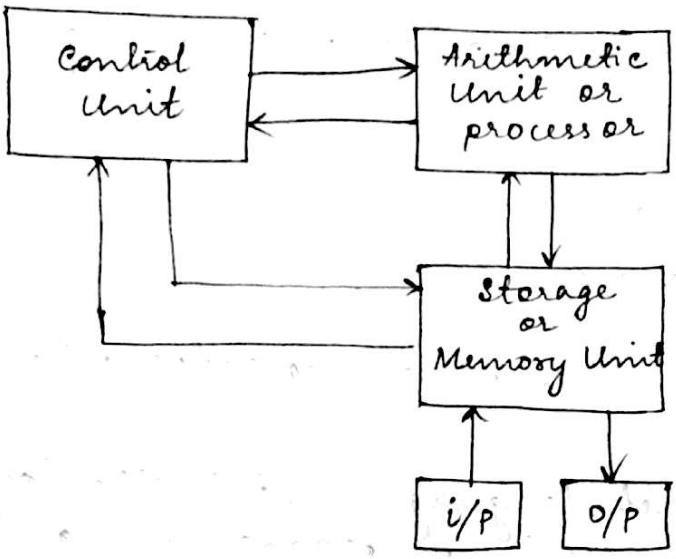
Fundamentals of Digital Computer :

11/1/16

✓ **Computer** : It is an electronic or programmable device that takes input from the user, process it and gives output in desirable form.

Bits : Basic unit of information (0 & 1).

Byte : Combination of bits (1 byte = 8 bits).
: represents 1 alphabet.
: Smallest unit of data.



① **Control Unit** — It is an essential part of CPU that controls the data flow and information. It maintains the sequence of operations, performed by the CPU.

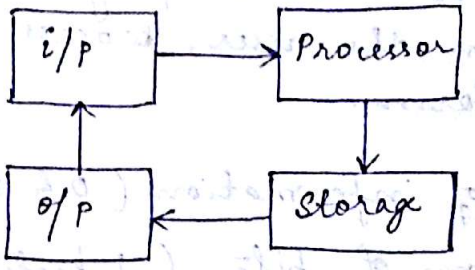
②. **Arithmetic Unit** — It performs the arithmetic operations on the data

③ **Storage / Memory Unit** — It stores the information or data processed by the CPU.

4. i/p — input to the computer eg. keyboard, mouse

5. o/p — output from the computer eg. monitor.

✓ Processing / Life cycle of processing any data



Language : Language can be used to create programs to control the behaviour of a machine or to express an algorithm.

Algorithm → Flowchart → Coding.

Language :—

① Low level language

② High level language

✓ ① Low level languages are machines codes which are close to the machine.

(i) Machine level language — It is the lowest & most elementary level of programming language and was the first type of language to be developed.

✓ It is fast & efficient. (advantage)

✓ It is platform dependent. (disadvantage).

(ii) Assembly language — In assembly language, we use alpha-numeric codes (pneumonics)

✓ ② High level languages are languages where english words are used to write programs easily.

Advantages : — user friendly.

— similar to English with vocabulary ~~with~~ of words & symbols.

— less time to write, easier to maintain.

— platform independent.

Disadvantages :— requires a translator to convert the english to machine level language which leads to increase in competitional time.

eg. C, C++, Java, php

13/1/16.

Algorithm

In programming, algorithms are the set of well-defined instructions in sequence to solve a problem.

Qualities of a good algorithm →

1. Inputs & outputs should be precisely defined.
2. Each step in algorithm should be clear and unambiguous.
3. An algorithm should not have computer codes.
4. It should have a distinct starting and stopping point.

---

① Write an algorithm to add two numbers.

Step 1 : Start
Step 2 : Declare variables a, b and sum
Step 3 : Read a and b.
Step 4 : Add the variables a and b ; and store it in sum.
Step 5 : Display sum
Step 6 : End

---

② Write an algorithm to find the largest among three 3 numbers.

Step 1 : Start
Step 2 : Declare variables a, b and c
Step 3 : Read a, b and c
Step 4 : If a > b and a > c , display a is largest
If b > a and b > c , display b is largest
else, display c is largest.
Step 5 : End

---

④ Write an algorithm to find factorial of a number.

Step 1 : Start
step 2: Declare variables n, i and fact
step 3 : Initialize fact = 1 , i = 1
Step 4: Read the value n
Step 5 : Repeat the step until i=n
fact ← fact * i
i ← i+1
Step 6 : Display fact

⑤ Write an algorithm to find Fibonacci series.

step 1 : Start
Step 2 : Declare variables i, first, second, third and n
Step 3 : Initialize i = 1, first = 0, second = 1
step 4 : Read the value of n
Step 5 : Repeat the step until i <= n
third ← first + second
first ← second
second ← third
i ← i+1
step 6 : Display ~~first~~ third.

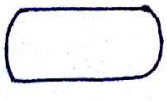
✓ **Flow chart :**

A flowchart is a diagrammatic or pictorial representation of any program. It depicts the nature and flow of the steps in a process. It generally uses graphic ~~syst~~ symbols.

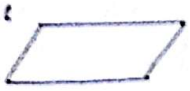
18/1/16

Flowchart symbols :

① '⟶' Flow line : It is used to indicate the flow of logic from one stage to the another, by connecting ~~It is used~~ symbols.

⑪  Terminal (start / End) : It is used to represent the start and end of flowchart

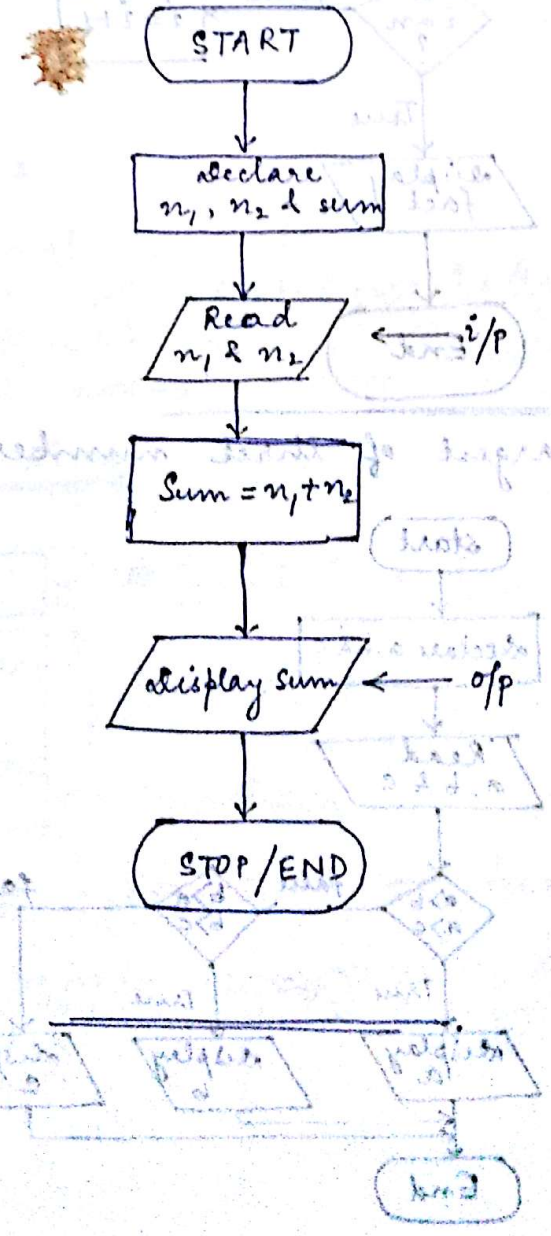
(iii)  Process / Declaration : It is used for declaration of variables and processing of any operation.

(iv)  i/p & o/p : It is used to indicate or display i/p & o/p of any data.

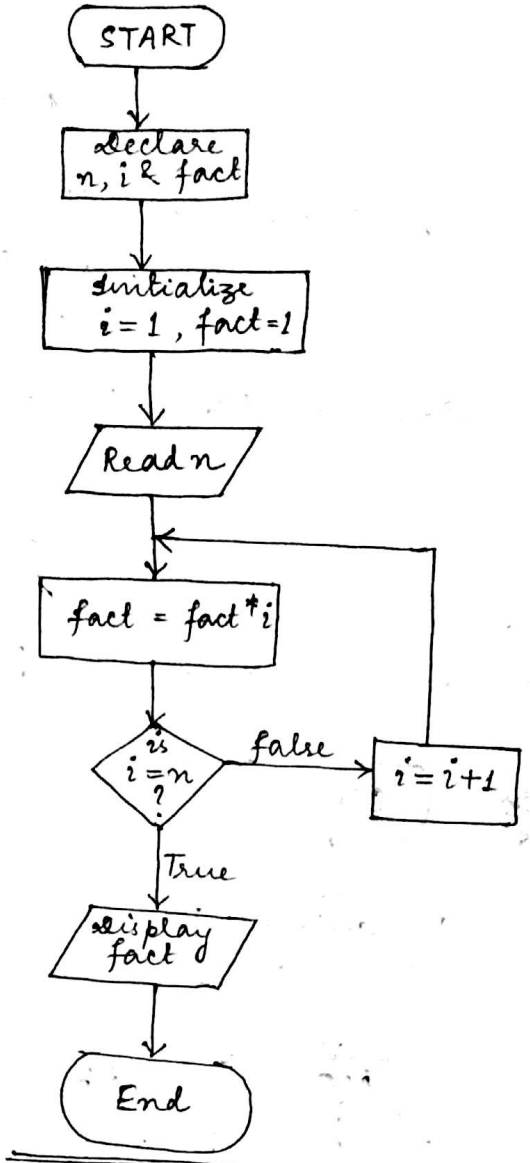
(v)  Decision/Conditional : It is used to represent an operation in which there are two alternatives

---

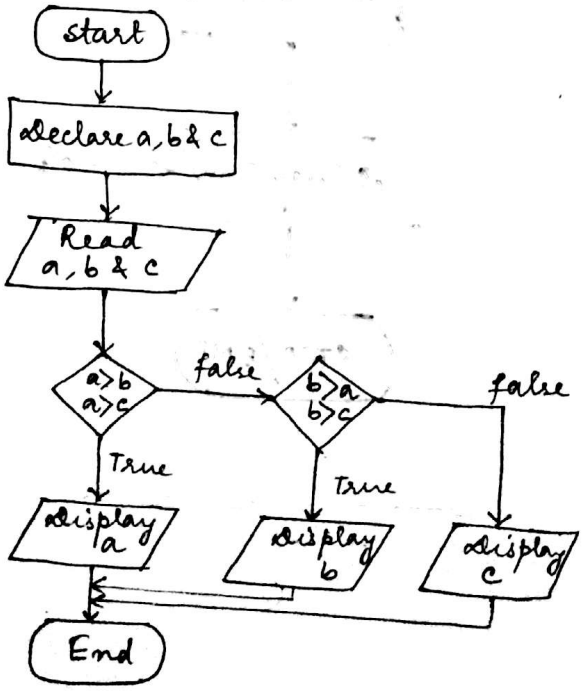
eg: flowchart for sum of two numbers :



eg flowchart for factorial of a number :



eg flowchart for largest of three numbers.



Number System :

- Binary : 0, 1 (base 2) Decimal : 0 – 9 (base 10).

$(10101)_2 \Rightarrow (21)_{10}$

$1\times2^0 + 0\times2^1 + 1\times2^2 + 0\times2^3 + 1\times2^4 = 21.$

2 | 21
2 | 10 — 1
2 | 5 — 0
2 | 2 — 1
1 — 0

$\Rightarrow 10101_2$

Binary :
00 — 0
01 — 1
10 — 2
11 — 3

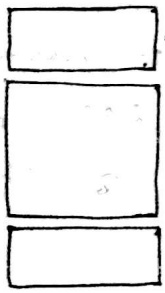
Octal :
000 — 0
001 — 1
010 — 2
011 — 3
100 — 4
101 — 5
110 — 6
111 — 7

$(010\ 101)_2 = (25)_8$

Hexadecimal : 0 – 15
0, 1, 2, ... 9, A, B, C, D, E, F

Basic C Structure :

Standard i/o Library function



← Global declaration : # include <stdio.h>
preprocessor command

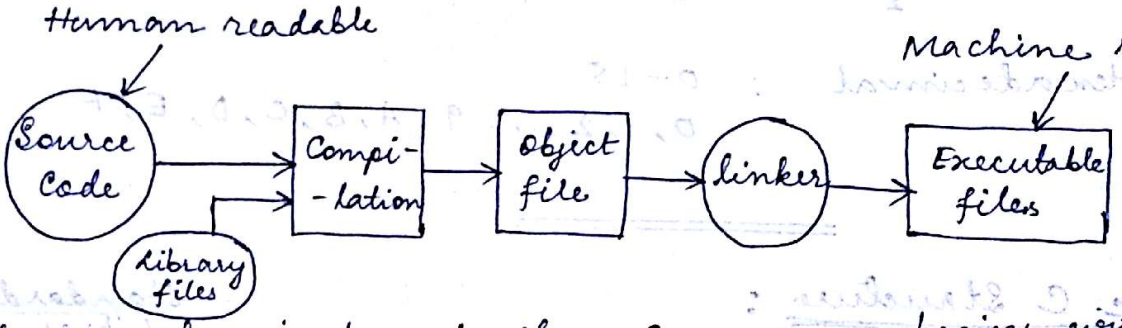
Main body

← Functions

```
# include <stdio.h>        ← header file.
{  int main()  /* My first program in C */
   printf (" Welcome to C programming ");
   return 0;
}
```

* The first line of the program #include <stdio.h> is a preprocessor command which tells C compiler to include standard i/o files before going for actual compilation.
* Main function is the function, from where actual execution of the program begins.
* /*.... */ is a comment, written only for the user, it is not executed by the compiler.
* 'printf' is another function available in C which causes the message "Welcome to C programming" to be displayed in screen
* 'return 0' will terminate the main function and returns value 0.

Overview of compilation & execution process: 22/01/16.



The mechanical part of a C program begins with one or more program source file, and ends with an executable file, which can be run on computer. The programming process starts with creating the source file, consist of statements of the program which is human readable.

The source file is then processed by a special program called as compiler. The compiler translates the source code into an object code.

The object code consists of the machine instruction for the CPU and calls to the operating system.

The object code is processed with another special program called linker.
While there is a different compiler for every individual language, the same linker is used for object files.
(Linker translates the object file into a low level language).

Features of C language :

① Low level features —
C programming provides low level features that are generally provided by the low level language.

② Portability —
C programs are portable, i.e. they can be run on any compiler with little or no modification.

③ It is powerful —
It provides wide variety of data types, functions and useful control and loop statement.

④ High level features —
It is more user-friendly.

⑤ Efficient use of pointers —
Pointers hold the address of another variable.

Keywords :

C has a set of reserved words often known as, keywords, that are basically a sequence of characters that have a fixed meaning. All the keywords are written in lowercase.
eg: for, auto, break, int, float, double, char, do, if, while .....

Identifiers :

Identifiers, as the name suggests, identifies data and other objects in the program. Identifiers are basically the name given to program elements such as variables, arrays and functions.

Rules for forming identifier names —

- It cannot include any special character or punctuation marks like #, $, ? ... etc, except '_' underscore.
- There cannot be two successive underscores.

eg. roll_number, marks, name, emp_id ....

Basic data types in C :

Datatype	Keyword	size in bytes
Character	char	1
Integer	int	2
floating point	float	4
Double	double	8
Valueless	void	0

25/01/16

(Q1) Write a program to add two integers

(Q2) Write a program to interchange two integer values, i.e. $x \leftrightarrow y$, with using temporary variables.

Ans (1)

```
# include <stdio.h>
main()
{  int a, b, sum;
   printf ("Enter the values of a and b");
   scanf (" %d %d", &a, &b);
   sum = a+b;
   printf (" sum = %d", sum);
}
```

int → %d
float → %f
character → %c
string → %s

(2) amperand → to hold the address of variables.

→ The 'printf' function is used to print the character, string, float, integer onto the output screen.

→ The 'scanf' function is used to read char, string and numeric data from keyboard.

→ '%d' is used in 'scanf' statement, so that the value entered is received as an integer.

→ 'Ampersand' holds the address of a variable.

Prob (2)

```
# include <stdio.h>
main()
{   int x, y, * temp ;
    printf (" Enter the values of x and y");
    scanf (" %d %d", &x, &y);
    temp = x;
    x = y ;
    y = temp;
    printf (" after interchange x = %d, y = %d ", x, y);
}
```

output: 10, 20

```
temp = 10 ;
x = 20 ;
y = 10 ;
```

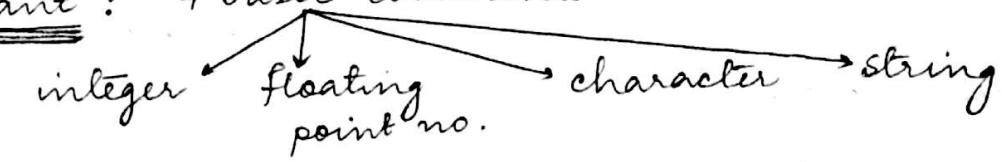
```
x = x+y ;
y = x-y ;
x = x-y ;
(without)
```

---

(3) Write a program to check whether a number is odd or even.

```
# include <stdio.h>
main()
{   int a ;
    printf (" Enter the value of a");
    scanf (" %d", &a);
    if ( a %2 == 0)
    printf (" ~~The number~~ %d is even",a);
    else
    printf (" ~~The number~~ %d is odd,",a);
}
```

Constant : 4 basic constants



→ An integer constant is an integer-valued number thus consists of a sequence of digits.

→ A decimal integer constant can consist any number from 0-9, if two or more digits exist the first digit must be other than 0.

→ In octal number system, any combination from 0-7. The first digit must be zero, in order to identify octal number system. eg. 0, 01, 0743...

→ In hexadecimal number system, any combination from 0-15. The number must begin with 0x or 0X. eg. 0x, 0x1, 0X7FFF.

→ Floating point constant is a base-10 number that contains either decimal point or exponent. It has a much greater range than integer constant. eg. 1.2×10^{-3} → 1.2E-3 or 1.2e-3

→ Character constant is a single character enclosed in apostrophes ' '. eg. 'A'

→ String constant consists of any no. of consecutive characters enclosed in double quotation mark. " " eg "Hello", "NITS".

→ For string, the compiler automatically places a null character (\0) at the end of every string constant as the last character within the string. (\0 indicates termination of the string.) eg "Hello\0"

Variables : A variable is an identifier that is used to represent some specified type of information, within designated portion of a program.

→ A given variable can be assigned different data items at various places within the program. Thus, information represented by variable can change during execution of program.

27/01/16

Expressions :

→ An expression represents a single data item such as a number or a character. It may consist of single entity such as constant, variable, array etc.

→ It may also consist of some combination of such entities (operands) interconnected by one or more operators.
eg. sum = a+b, x=y, x<=y, x==y.

Operators :

→ An operator is a symbol that tells the compiler to perform some specific mathematical or logical manipulation.

→ C language have built-in operators and provides the following type of operators :—

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Assignment operators
5. Bit wise operators.

1. Arithmetic operators (+, −, *, /, %)

- Increment operator (++) increases integer value by 1.
- Decrement operator (−−) decreases value by 1.

eg. A = 10, A++ = 11
A−− = 9.

② Relational operators

i) '==' → It checks if the values of two operands are equal or not, if yes, then condition becomes true.

ii) '!=' → It checks if the values of two operands are equal or not, if the values are not equal, then condition becomes true.

iii) > greater than

iv) < less than

v) >= greater than, equal to

vi) <= less than, equal to

③ Logical operators

i) '&&' → logical 'and' operator : if both the operands are nonzero, then conditions becomes true.

ii) '||' → logical 'OR' operator : if any of the two operands is nonzero, then condition becomes true.

iii) '!' → logical 'NOT' operator : it is used to reverse the logical state of its operand. If a condition is true, then logical 'NOT' will make false.

④ Assignment operators

i) '=' → simple assignment operator, it assigns value from right side operand to the left side operand.

ii) '+=' → add AND assignment operator, it adds right operand to the left operand and assigns result to the left. eg. C+=A
C=C+A

iii) '-=' → sub AND assignment operator, it substracts right operand from left operand and assigns result to the left. eg. C-=A
C=C-A

Operator precidence table ✓ (Unary operators holds highest precidency)

Unary operators :-

C includes a spical class of operators that act upon a single operand to produce a new value, is called an unary operator.

eg. int i ;
++i ;

i++ } both increments
++i } the value.

i = 5 ++i = 6
i++ = 6.

In ++i, the compiler increments the value before reading i; first increments then stores in i.

eg: a = 10, b = 20 ;
X = ++a ;
Y = b++ ;
printf ("X = %d a = %d", &X, &a);
printf ("Y = %d b = %d", &Y, &b);

Output → X = 11 , a = 11
Y = 20, b = 21.

Y = (b)20++
Y will hold 20
b will get incremented.

---

(5) WAP to perform basic arithmetic operations in a single program.

```
# include <stdio.h>
main()
{   int x, y, sum, sub, mult, div, mod;
    printf ("Enter the values of x and y");
    scanf ("%d %d", &x, &y);
    printf ("y = %d",
    sum = x+y;
    sub = x-y;
    mult = x*y;
    div = x/y;
    mod = x%y;
    printf ("sum = %d sub = %d mult = %d div %d mod %d",
                &sum, &sub, &mul, &div, &mod);
}
```

⑨ Find the output of the following codes: 29/01/16

1) main()
{
int i = 5, j;
j = ++i + ++i + ++i
printf ("%d %d", i, j);
}

→ i = 5
j = 6 + 7 + 8
i = 8
j = 21.

2) main()
{
int i = 1;
i = 2 + 2 * i++;
printf ("%d", i);
}

→ i = 2 + 2×1 = 4
i = 4 + 1 = 5.

3) main()
{
int a = 2, b = 7, c = 10
c = a == b;
printf ("%d", c);
}

→ condition is false so output is zero.
c = false so 0.

4) main()
{
int x;
x = 10, 20, 30;
printf ("%d", x);
}

→ Output x = 10
(∵ The assignment operator gets precedence).

5)
```
main()
{
    int a=0, b=10;
    if (a=0)
    { printf ("true");
    }
    else
        printf ("false");
}
```

6)
```
main()
{
    int a;
    a = 015 + 0x71 + 5;
    printf ("%d", a);
}
```

$\longrightarrow$ $5\times8^0 + 1\times8^1 = 13$; $1\times16^1 + 7\times16^0 = 112+1 = 113$

$113+13+5 = 131$.

$\therefore$ Output: $a = 131_{10}$.

7)
```
main()
{
    int i=5
    int a = i++ + i++ + i++
    printf ("%d %d", i, a);
}
```

$\longrightarrow$ 18/21.

$i = 7$.

(7, 18).

Control statements :

① If - statement

```
if ( )
{ statement ;
}
```

② If - else - statement

```
if ()
{ statement ;
}
else
{ statement;
}
```

⑨ WAP to check the largest of three numbers.

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

Nested if-else

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

false → go for last else.

true → test condition 2

Switch statement

Switch statement test the value of a given variable against a list of case values and when block is found, a statement associated with that case is executed.

```
switch (expression)
{   case 1 : value-1 ;
             block-1 ;
             break ;
    case 2 : value-2 ;
             block-2 ;
             break ;
    ------
    ------
    default : default-block ;
              break ;
}
```

→ An expression usually includes an integer or char.

→ Value-1,2 are constant expressions known as case labels.

→ 'Break' causes exit from the switch statement.

---

⑧ WAP to perform basic arithmetic operations in switch case. (CALCULATOR)

```
# include <stdio.h>
main ()
{
    int a, b, sum, sub, mult, div, choice ;
    printf ("Enter the values of a and b");
    scanf ("%d %d", &a, &b);
    printf ("enter 1 for add \n 2 for sub \n 3 for
                mult \n 4 for div");
    printf ("enter your choice <1,2,3,4>");
    scanf ("%d", &choice);
```

```
switch (choice)
{
    case 1: sum = a+b;
            printf ("sum = %d", sum);
            break;
    case 2: sub = a-b;
            printf ("sub = %d", sub);
            break;
    case 3: mult = a*b;
            printf ("mult = %d", mult);
            break;
    case 4: div = a/b;
            printf ("div = %d", div);
            break;
    default: default;
             printf ("wrong choice");
             break;
}
printf ("sum = %d sub = %d mult = %d div = %d",
        sum, sub, mult, div);
}
```

Ternary operator / Conditional operator

(test condition) ? (statement-1 or expression1) : (statement-2 or expression2)

(These expressions) ; expression1 ↓ correct / true ; expression2 ↓ false.

⑤ WAP to check whether a number is odd or even using ternary operator.

```
# include <stdio.h>
main()
{ int a;
  printf ("Enter the value of a");
  scanf ("%d", &a);
  (a%2==0)? printf("even"): printf ("odd");
}
```

03/04/16.

Looping statement

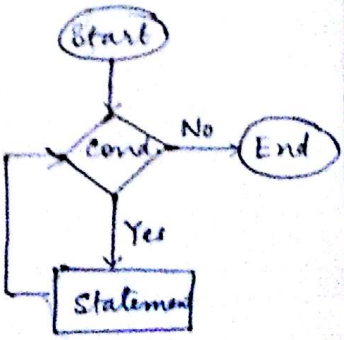
① While statement [Entry control loop].

```
while (condition)
{ statement;
}
```

The while statement is used to carry out looping operations in which group of statements is executed repeatedly until some condition has been satisfied. The statement is executed repeatedly as long as the expression is true.

⑥ WAP to display consecutive digits from 0 to 9 with one digit on each line.

```
# include <stdio.h>
main()
{   int i=0;
    while (i<=9)
    {
        printf ("%d\n", i);
        i++;
    }
}
```



② **for loop statement** [Entry control loop].

The for statement is the most commonly used looping statement in C. It is an entry control loop.

```
for ( expn1 ; expn2 ; expn3)
{                  ↓        ↓          └──→ increment
                initialization  test condition
}
```

expn 1 Initialization of control variable is done first using assignment statement such as $i=0$, count = 1 ....

i, count → loop control variables.

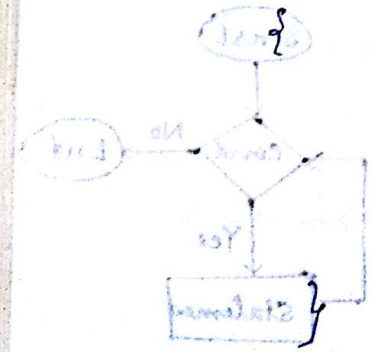
expn 2 The variable of control variable is tested using test condition. It use relational expression such as $i<10$, $i>0$, that determines when loop will exit.

expn 3 The control variable is incremented and alter the value of parameter initially assigned using an unary or assignment operator. And, the new value of control variable is again tested to see whether it satisfies the loop condition.

---

eg.

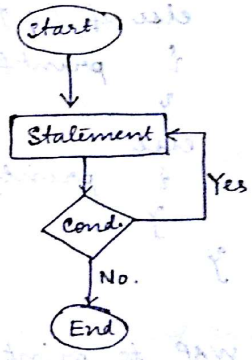
```
# include <stdio.h>
main ()
{
  int i ;
  for ( i=0; i≤9 ; i++)
  { printf ("%d\n", i)
  }
}
```



③ do-while statement [Exit control loop].

```
do
{
  statement ;
}
while (test condition);   ← semicolon reqd. only in do-while.
```

In case of 'while' loop, the test of continuation of loop is carried out at the beginning of each pass (or iteration). Sometimes it is desirable to have a loop ~~which~~ to with test for continuation at the end of each pass, it is done by using 'do-while' statement.



eg.
```
# include <stdio.h>
main ()
{
   int i = 0 ;
   do
   {  printf ("%d \n", i);
      i++ ;
   }
   while (i <= 9) ;
}
```

---

④ WAP to find the factorial of a number.

```
#include <stdio.h>
main ()
{
   ~~int i, n, fact ;~~
   ~~i=1 ; int n, i=1, fact=1 ;~~
        ~~for (~~
   int i, n, fact=1 ;
   printf ("Enter the value of n");
   scanf ("%d", &n);
   for ( i=1 ; i<=n ; i++)
    {    fact = fact * i ;  }
   printf ("%d", fact);
}
```

---

② WAP to find whether a given year is a leap year.

```
# include <stdio.h>
main()
{  int a;
   printf ("Enter the year:");
   scanf ("%d", &a);
   if (a%.100 ==0)
   {     if (a%.400 ==0)
         printf ("%.d is a leap year", a);
         else
         printf ("%.d is not a leap year", a);
   }
   else if (a%.4==0)
   {  printf ("%.d is a leap year", a);
   }
   else
   {  printf ("%.d is not a leap year", a);
   }
}
```

---

③ WAP to print the reverse of a number.

```
# include <stdio.h>
main()
{  int n, rev=0;
   printf (" Enter the number to be reversed:");
   scanf (" %d", &n);
   while (n!=0)
   {     rev = rev*10 + n%.10;
         n   = n/10;
   }
   printf (" The reverse is %.d", rev);
}
```

---

break & continue statement :

08/02/16.

→ The break statement in c programming has the following two uses:

i) When the break statement is encountered inside a loop, the loop is immediately terminated, and program control resumes at the next statement following the loop.

ii) It can be used to terminate a case in the switch statement.

Syntax: break;

Nested - for loop

```
for ( init ; condition ; increment )
{
    for ( ; ; )
    {
        statement ;
    }
  statement ;
}
```

(inner loop.)

iii) If we are using nested loops, i.e. one loop inside another loop, the break statement will stop the execution of inner loop and start executing the next line of code after the block.

→ The continue statement in C programming works somewhat like the break statement, but instead of forcing termination, however, continue statement forces the next iteration of the loop to take place skipping any code in between.

Syntax: continue;

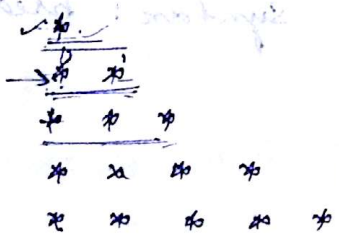
goto statement :

→ A 'goto' statement in C programming provides an unconditional jump from the goto to a labelled statement in the same function.

(It is discouraged ; being unconditional , it disturbs the flow control of a program).

---

⑧ WAP to print a following pattern :



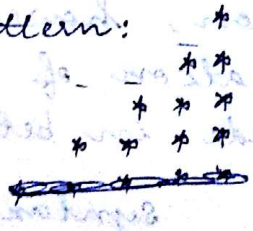
```
# include <stdio.h>
main()
{
   int i, j ;
   for ( i=0 ; i<5 ; i++)
   {
      for ( j=0 ; j <= i ; j++)
      {
         printf ("* ");
      }
   printf ("\n");
   }
}
```

i → no. of rows
j → no. of * corresponding to each i.

0 1 2 3 4
i=0 j=i
0 1 2 3

---

⑨ WAP to print pattern:



```
# include <stdio.h>
main()
{  int i, j, k ;
```

i → rows
j → spaces in decreasing order
k → * corresponding to each i in increasing order

```
for (i=0 ; i<5 ; i++)
{
    for ( j=5; j >= i ; j--)
    {
        printf ("\t");
    }
    for ( k=0 ; k <= i ; k++) ;
    {
        printf ("*");
    }
    printf ("\n");
}
}
```

---

⑨ WAP to print the pattern of Floyd's triangle.

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
# include <stdio.h>
main()
{
    int i,j;
    for (i=0 ; i<5 ; i++)
    {
        for( j=0 ; j <i ; j++)
        {
            printf ("%d ",j);
        }
        printf ("\n");
    }
}
```

10/02/15

Arrays :

→ An array is a fixed-sized sequence collection of elements of same datatype i.e. it is simply a grouping of similar type of data.

eg. marks of a class

datatype marks [60]

name of array

size of array

Datatypes

Derived	Fundamental	User defined
- Arrays	int, float, char,	- Structure
- functions	double	- Union.
- pointers		

Arrays

- One dimensional
- two dimensional
- Multi dimensional.

- accessing
- manipulating
- processing.

→ Since arrays provides a convenient structure for representing data, it is classified as one of the data structures in C.

One dimensional array :—

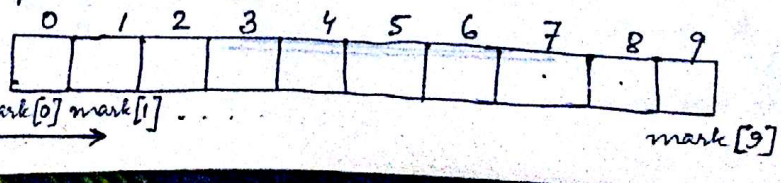
A list of items can be given one variable name using only one subscript and such a variable is called one-dim. array.

(represents either row or a column).

Declaration : datatype variablename [size];

eg. int marks [10];

Memory representation :



accessing — third element : mark[2].

manipulation — increase mark of third element : mark[2] + 10;

eg. char name[10] = "NITSILCHAR";

N	I	T	S	I	L	C	H	A	\0

for string, the termination is by default set as null character \0.

Initialization:

int mark[5] = {10, 20, 30, 40, 50};

datatype arrayname[size] = { list of values };

Accessing of array elements:

```
int marks[10] = {          };
for (int i = 0; i <= 9; i++)
{printf ("%d", marks[i]); }
```

(8) WAP to find out average of ten integer values.

```
#include <stdio.h>
main()
{   int i, sum=0; float avg;
    int values[10];
    printf ("Enter the values:");
    for (i=0; i<=9; i++)
    scanf ("%d", &values[i]);
    {
    sum = sum + values[i];
    }
    avg = sum/10;
    printf ("%f", avg);
}
```

sum = sum + values[i]

17/02/16.

Operations using Array :

(1) Traversing an array.

a) Searching → linear / sequential
b) Accessing → binary searching.
c) Sorting → bubble sort
→ selection sort
→ insertion sort

Linear search :

```
# include <stdio.h>
main()
{
    int i, num, value[6];
    value[6] = {10, 20, 15, 5, 8, 12}; scanf("%d", &n
    for (i=0; i<6; i++)
    {
        if (value[i] ==
    scanf ("%d", &value[i]);
    {
        if (value[i] == num)
        {
            printf ("%d element is found");
            break;
        }
        if (i==6)
        {
            printf ("element is not found");
        }
    }
```

Binary search :

The list of elements must be in sorted order.

sorting + Searching.

Bubble sort :

```
# include <stdio.h>
main()
{
    int i, j, k, array [10]
    for ( i=0 ; i<=9 ; i++ )
    {
      for (j=0, j<10-i ; j++)
      scanf ("%d", & array [i])
    }
    for (i=0 ; i<9 ; i++)
    {
        for (j=0; j<10-i ; j++)
        {
            if (array[j] > array [j+1])
            {
                k = array [j] ;
                array [j] = array [j+1] ;
                array [j+1] = k ;
            }
        }
```

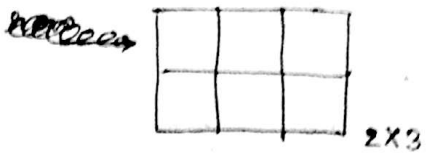
19/02/

Two dimensional array :—

Declaration :

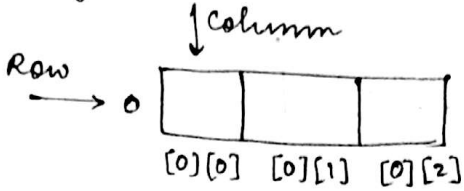
datatype arrayname [size of row][size of column];

eg int a [2][3] ;



Initialization :

Memory representation :



Initialization (Three ways to initialize)

(i) int a [2][3] = { 0, 0, 0, 1, 1, 1 } ; ←

(ii) = { {0,0,0}, {1,1,1} } ; ←

{

(iii) {0,0,0} ;

{1,1,1} ; ←

}

eg: int a [2][3] = { 1, 2, 3, 4, 5 }

No 6th element.

1	2	3
4	5	0 ← 6th element is declared 0.

⑨ WAP to perform addition of two matrix.

```
# include <stdio.h>
main ()
{   int a [20][20] ;
    int b [20][20] ;
    int c [20][20] ;
    int i, j, p, q ;
```

```
printf ("Enter the number of rows & columns: ");
scanf ("%d %d", &p, &q);
printf ("Enter the elements of first matrix: ");
for (i=0; i<p; i++)
   for (j=0; j<q; j++)
   scanf ("%d", a[i][j]);
printf ("Enter the elements of second matrix: ");
for (i=0; i<p; i++)
   for (j=0; j<q; j++)
   scanf ("%d", b[i][j]);
printf ("The resultant matrix is: ");
for (i=0; i<p; i++)
   for (j=0; j<q; j++)
   {
      c[i][j] = a[i][j] + b[i][j];
      printf ("%d \t", c[i][j]);
   }
   printf ("\n");
}
```

---

⑨ WAP to print the transpose of a matrix.

```
#include <stdio.h>
main()
{
   int a[10][11], b[11][10], i, j, p, q;
   printf ("Enter the number of rows & columns: ");
   scanf ("%d %d", &p, &q);
   printf ("Enter the elements of first matrix: ");
   for (i=0; i<p; i++)
      for (j=0; j<q; j++)
      scanf ("%d", a[i][j]);
```

```
printf (" The transpose is : ");
for (i=0 ; i<p ; i++)
    for (j=0 ; j<q ; j++)
    {
        b[i][j] = a[j][i]
        printf (" %d\t ", b[i][j]);
    }
    printf ("\n");
}
```

24/02/16

Strings

→ Strings are arrays of character terminated by '\0'.

Declaration : char stringname [size] ;

Initialization : char stringname [12] = "NIT SILCHAR";

= { 'N', 'I', 'T', ' ', 'S', 'I', 'L'

Functions used for reading a string
- (i) scanf ( )
- (ii) gets ( )
- (iii) getch ( )

```
scanf ("%s", string[i]);
```
↑ No '&' is used

& string [0] → one character in the string

%s reads the string as a whole

→ Although the syntax of 'scanf' function is well known and easy to use, the main disadvantage with this function is that the function terminates as soon as it finds a blank space.

→ 'gets()' is a simple function that overcomes the drawbacks of 'scanf()'. The 'gets' function takes the starting address of the string which will hold the input. The string inputted using 'gets()' automatically terminates with a '\0' character

→ 'getch()' : A string can also be read by using getch() or getchar() repeatedly to read a sequence of single characters unless a terminating character is entered.

Functions used in writing a string
- (i) printf()
- (ii) puts()
- (iii) putch()

printf ("%s\n", string); ⟹ entire string

printf ("%12.3s\n", string); ⇒ first 3 char. (right justified)

printf ("%12.0s\n", string); → no output.

printf ("%-12.3s\n", string); ⇒ first 3 char. (left justified.)

-12 or 12 is for the justification whether left or right.

⑨ WAP to find the length of a string.

```
#include<stdio.h>
main()
{
    char str[100];
    int i=0, length;
    printf("Enter the string:");
    gets(str);
    while(str[i]!='\0')
    i++;
    length=i;
    printf("Length of string=%d", length);
}
```

i →	0	1	2	3
	N	I	T	\0

Length = 3.

⑨ WAP to concatenate two strings.

```
#include <stdio.h>
main()
{
    char a[50], b[50], c[100];
    int i=0, j=0;
    printf("Enter the first string:");
    gets(a);
    printf("Enter the second string:");
    gets(b);
    while(a[i]!='\0')
    {
        c[j]=a[i];
        i++;
        j++;
    }
    i=0;
    while(b[i]!='\0')
    {
        c[j]=b[i];
        i++;
        j++;
    }
    c[j]='\0';
    printf("The concetenated string is:");
    puts(c);
}
```

NIT

SILCHAR

07/03/16.

Functions :

A function is a group of statements that together performs a certain task. It is a self contained block of statements that performs coherent tasks of some kind.

Advantages →

1. Functions avoids rewritting the same code again and again, that is, it ensures reusability.
2. Separating the code into the modular functions also makes the program easier to understand, design and debug.

Defining a function →

A function definition provides the actual body of the function.

General form:

return-type function-name (parameter list) { body }

- return-type: defines the data type of the value that the function will return.
- function-name: unique identifier
- body: operations.

Parameter list : When a function is called or invoked we pass a value to the parameter, This value is also referred as actual parameter or argument.

eg for maximum of two numbers.

```
int max ( int a, int b )
{   int result ;
    if ( a>b)
    .....
    .....
    .....
}
```

```
#include <stdio.h>
void message();  /* function prototype declaration */
void main()
{
    message();  /* function call */
    printf("After calling function");
}
void message()  /* function definition */
{
    printf("I am the called function");
}
```

Step 1 → Step 2

→ When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed, it returns the program control back to the main program, to the statement next from where it is called.

```
#include <stdio.h>
void function_name() {
    .....
    .....
}
int main() {
    .....
    .....
    function_name();
    .....
    .....
}
```

Step 1 → Step 2

⑤ WAP to add two integers using functions. 09/03/16.

```
#include <stdio.h>
int add (int a, int b) ;  // function declaration
main()
{
    int num1, num2, sum;
    printf (" Enter the numbers");
    scanf (" %d %d", &num1, &num2);
    sum = add (num1, num2) ;  // function call
    printf (" sum = %d", sum);
}
int add (int a, int b)  // function definition
{
    int total;  → local variable (declared inside
    total = a+b;                a function body).
    return total;
}
```

---

⑥ differentiate between actual parameter and formal parameter in C.

→ Formal parameters are place holders for values of actual parameter.

→ Formal parameters are declared/defined during function declaration.

→ Actual parameters are always passed & variables that are declared and passed as arguments to procedure by caller.

→ The 'return' statement is used to terminate the execution of a function and returns control to the calling function.

Q ~~WAP to perform~~

→ Passing parameters to the function :

(i) Call by value
(ii) Call by reference.

When a function is called, the calling function may have to pass some values to the called function. Basically, there are two ways in which parameters can be passed, to the called function.

(i) Call by value in which values of the variable are passed by the calling function to the called function.

(ii) Call by reference in which address of the variables are passed instead of values.

Q WAP to perform swapping of two numbers using functions. (call by value).

```
# include <stdio.h>
int swap (int x, int y);
main()
{                          ← actual parameters
    int a, b;
    printf ("Enter the two numbers");
    scanf ("%d %d", &a, &b);
    a = a+b;
    b = a-b;
    a = a-b;
    swap (a,b);
    printf ("a=%d b=%d", a, b);
}
```

```
int swap (int X, int Y)
                    ← formal parameters
{   int z;
    z = X;
    X = Y;
    Y = z;
    printf ("X = %d  Y = %d", X, Y); // optional
    return (X, Y);
}
```

→ Actual parameters are the actual source of information in actual parameter, the caller program supplies the data to the called function in the form of actual variables.

---

(8) WAP to find factorial of a number using functions.

```
# include <stdio.h>
int fact (int x);
main()
{   int a, fact;
    printf ("Enter a number");
    scanf ("%d", &a);
 fact = fact (a);
    printf ("Factorial of %d = %d", a, fact);
}
int fact (int x)
{
    int i, n, fact = 1;
    for (i = 1; i <= n; i++)
    {
        fact = fact * i;
    }
    return fact;
}
```

/* prog to find largest of three numbers using function */

```
#include <stdio.h>
int largest (int x, int y, int z);
main()
{
    int a, b, c;
    printf ("Enter the three numbers : ");
    scanf ("%d %d %d", &a, &b, &c);
  l = largest (a, b, c);
    printf ("%d is the largest", l (largest));
}
int largest (int x, int y, int z)
{
    if (x > y && x > z)
    printf ("%d is the largest", x);
    else if (y > x && y > z)
    printf ("%d is the largest", y);
    else
    printf ("%d is the largest", z);
    return largest;
}
```

Recursion:

```
main()
{   printf ("Hello World!");
    main();
}
```

output :— Hello World! (infinite no. of times).

* A process where function calls itself is called recursion.

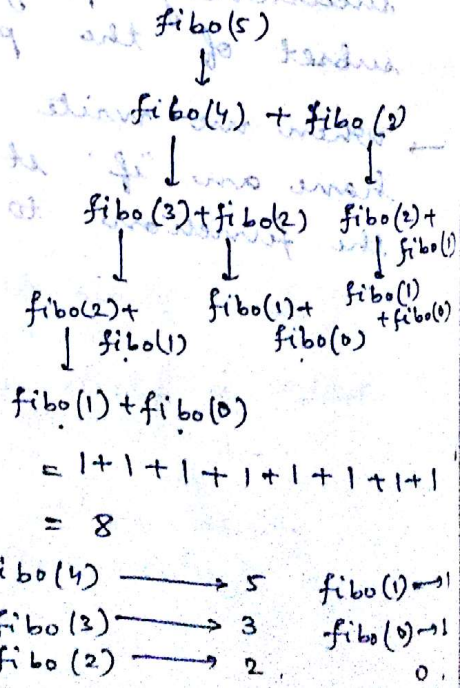
(An 'if' statement is reqd. while using recursive functions : to terminate).

(8) WAP to print Fibonacci series using recursion.

```
#include <stdio.h>
int fibo(int);
main()
{
    int n;
    printf("Enter the no. of terms ....: ");
    scanf("%d", &n);
    f=fibo(n);
    printf("The fibonacci series
    for(i=0; i<n; i++)
        Preintf("%d\n", fibo(i));
}
int fibo(int n)
{
    if (n==0)
      { return 0; }
    if (n==1)
      { return 1; }
    return fibo(n-1)+(n-2);
}
```

14/03/16.

```
int fibo(n)
{
    if (n<2)
    return 1;
    else
    return fibo(n-1)+(n-2);
}
```



0 1 1 2 3 5 8.

⑧ WAP to find the sum of n natural numbers using recursion.

```
# include <stdio.h>
int sum (int);
main()
{
    int n, i;
    printf (" Enter the number of terms:");
    scanf (" %d", &n);
    for (i=1; i<=n; i++)
    printf (" %d\n", sum(n));
}
int sum (int)
{
    if (n == 1)
    return 1;
    else
    return n + sum (n-1);
}
```

n%10 + sum(n/10)
3

n = n%10 + n/10

n = 5
sum(123) 5 + sum(4)
4 + sum (3)
3 + sum 2
2 + sum 1
1
sum. 5+4+3+2+1 = 15.

123
3 +

123
10

3 + 2 + 1

123
1 + 2 + 3

sum(12) 12
2 + sum(1)
+ 1

10 | 123
3

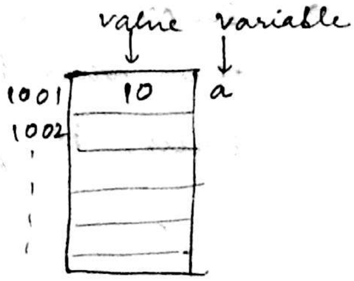
16/03/16.

Pointers :

→ A pointer is a variable that represents location/ address of a data item such as a variable or an array element.

→ A pointer is a derived datatype in C, which contain memory addresses ~~and~~ as their values.

int a = 10;
*p = 1001;



Advantages

① Pointers are more efficient in handling array and data tables.

② Pointers can be used ~~with~~ to return multiple values from a function via function arguments.

(3) It allows C to support dynamic memory allocation (DMA). (allocate memory or increase the size during run time).

(4) Pointers reduce the length and complexity of program which leads to reduction of the program execution time.

eg. int sum 18

sum ← variable
18 ← value
1001 ← address

→ Since memory addresses are simply numbers they can be assigned to some variables that can be stored in memory like any other variables.

& sum = *p → pointer variable.

Such variables that holds the memory address are called <u>pointer variables</u>.

So, a pointer variable is nothing but a variable that contains an address which is the location of another variable in memory.

variable	value	address
sum	→ 18	1001
p	→ 1001	5000

→ Since value of the variable 'p' is the address of the variable 'sum', 'p' points to variable 'sum'. Thus, it gets the name 'pointer'.

(a) WAP to print the memory address of a variable along with its value.

```
#include <stdio.h>
main()
{
    int a = 10;
    printf ("%d is stored in %u", a, &a);
}
```

Declaration of pointer variable:

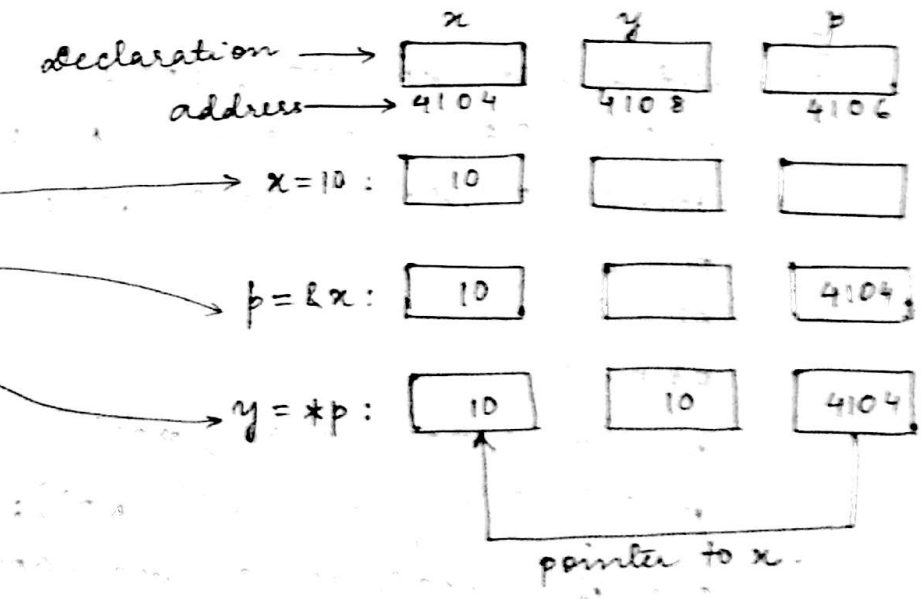
General form: data type *ptr_name;

eg. int *p;

→ * tells the compiler the variable name (ptr_name) is a pointer variable.

→ A pointer name always needs a memory location.

eg.
```
int x, y;
int *p;
x = 10;
p = &x;
y = *p;
```



28/03/16

⑦ WAP to add two integers using pointers.

```
#include <stdio.h>
main()
{
    int a, b, *p1, *p2, sum;
    printf ("Enter two numbers: ");
    scanf ("%d %d", &a, &b);
    p1 = &a;
    p2 = &b;
    sum = *p1 + *p2;
    printf ("sum = %d", sum);
}
```

---

⑧ WAP to find largest among three numbers using pointers.

```
#include <stdio.h>
main()
{
    int a, b, c, *p1, *p2, *p3;
    printf ("Enter three numbers: ");
    scanf ("%d %d", &a, &b, &c);
    p1 = &a;
    p2 = &b;
    p3 = &c;
    if (*p1 > *p2 && *p1 > *p3)
    printf ("largest = %d", *p1);
    else if (*p2 > *p1 && *p2 > *p3)
    printf ("largest = %d", *p2);
    else
    printf ("largest = %d", *p3);
}
```

---

⑦ WAP to swap two numbers, using pointers.

```
#include <stdio.h>
main()
{
    int a, b, *p1, *p2;
    printf (" Enter two numbers : ");
    scanf (" %d %d", &a, &b);
    p1 = &a;
    p2 = &b;
    *p1 = *p1 + *p2;
    *p2 = *p1 - *p2;
    *p1 = *p1 - *p2;
    printf (" After swapping a = %d , b = %d", *p1, *p2);
}
```

30/3/16.

⑧ WAP to swap two numbers, using call by reference.

```
#include <stdio.h>
int swap (int *p1, int *p2);
main()
{
    int a, b, *q1, *q2;
    printf (" Enter two numbers : ");
    scanf (" %d %d", &a, &b);
    q1 = &a ;
    q2 = &b ;
    swap (*q1, *q2) ;
    printf (" a = %d  b = %d", *q1, *q2);
}
int swap (int *p1, int *p2)
{
    *p1 = *p1 + *p2 ;
    *p2 = *p1 - *p2;
    *p1 = *p1 - *p2;
    return (*p1, *p2);
}
```

---

(1) Pointer arithmetic :

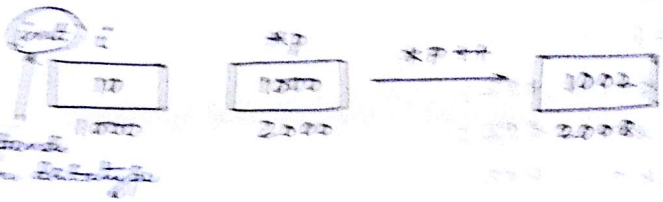
eg: *p + 2

*p - 2

*p1 + *p2

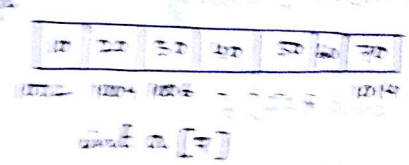
(2) Pointer increment / decrement :

eg:



— Incrementing pointer is generally used in array because we have continuous memory in array and we access the contents of next memory location.

eg:



int a[7]

*p++

New add. = 1002 + 1 X 2

= 1004.

— Incrementing pointer variable depends on datatype of the pointer variable.

*p++
++*p
*p+1
} results the same (Add 1).

General formula :

New value of pointer var. = Current address + i * size of datatype

*p + 3

1002

New address = 1002 + 3X2

= 1008

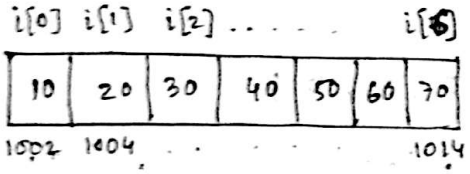
eg. int main()
{
    int *ptr = (int *) 1000; // ⟹ ptr is a pointer var. which holds a value 1000.
    ptr = ptr + 1;
    printf ("New value of ptr %u", ptr);
}

// : The line 2 will store 1000 in the ptr. variable considering 1000 is memory location for any of the integer variable.

OUTPUT : 1000 + 1 × 2 = 1002.

If p1 and p2 are both pointers to the same array, then p2 - p1 will give the number of elements between p1 and p2.

eg. a



eg
int main()
{
    float *ptr1 = (float *) 1000;
    float *ptr2 = (float *) 2000;
    printf ("Difference = %d", ptr2 - ptr1);
}

$$\text{No. of elements} = \frac{2000 - 1000}{\text{size of datatype}}$$

$$= \frac{1000}{4} = 250.$$

06/04/16.

Pointers and Arrays:

int A[5] =

0	1	2	3	4
5	6	10	12	15
400	402	404	406	408

{ 5, 6, 10, 12, 15 }

base address.

eg.

```
int *p = A
print p ; //400
print *p ; // 5
print *(p+2) ; //10
```

$p = A = \&A[0] = 400$

→ If we declare p as an integer pointer, then we can make the pointer p to point to the array A by following assignment

$p = A$

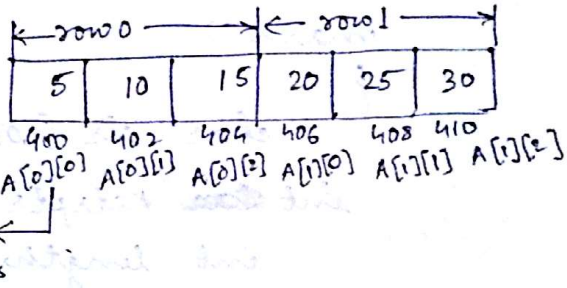
which is equivalent to address of A[0] i.e. &A[0].

④ WAP using pointers to compute the sum of elements in an array.

```
# include <stdio.h>
main()
{
    int A[5], sum=0, i, *p;
    for (i=0; i<5; i++)
    A[5] = { 5,6,10,12,15 };
    p = A = &A[0] ; // initialization of base address
    for (i=0; i<5 ; i++)
        sum = sum + *p ;
    printf ("%d", sum);
}
```

2-D arrays

```
int A[2][3] = { 5, 10, 15,
                20, 25, 30 };
```



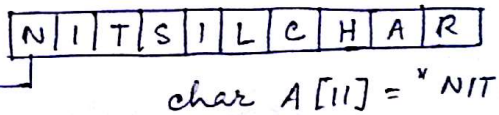

```
p = A = &A[0][0]

print A ;    // 400
print *A ;   // 400
print A+1 ;  // 402
print *A+1 ; // 402
print *(*A+1) ; // 10
```

*(A[0]+1) = *(402)

07/04/16.

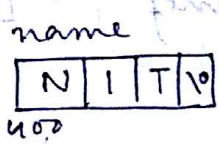
Pointers and Strings:

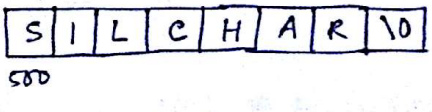


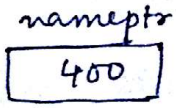

```
char A[11] = "NITSILCHAR";
cptr = A = A[0] ;
cptr++ or cptr+2 ..
```

Since we are discussing strings which are made up of characters we will be using pointer to character instead of char *s. However pointer only hold an address. They cannot hold all the characters in a character array. This means that when we use pointer to character char *s to keep a string, the character array containing the string must already exist.

eg.
```
char name[ ] = "NIT"
char name1[ ] = "SILCHAR"
char *nameptr ;
nameptr = name;
```

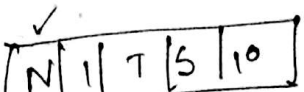






(9) WAP using pointers to find the length of a string.

```
#include <stdio.h>
main()
{
    char str[100] = "NITSILCHAR";
    int *strptr = str;
    int length;
    while (*strptr != '\0')
    strptr ++;
    length = *strptr - str;
    printf ("Length = %d", length);
}
```




```
{
char str[100] = "...."   400
int length;
char *strptr;
strptr = str;
while (*strptr != '\0')
  strptr ++;
length = strptr - str;
printf ("Length = %d", length);
}
```

*strptr = str 400

*strptr - str =

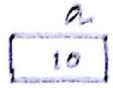
```
{char *strptr;
int length;
```

```
char s[100];
char *p;
p = s;
while (*p != '\0')
  p++;
length = p - s
```

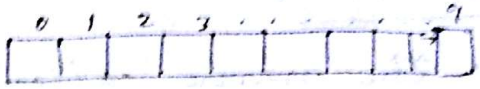
Dynamic memory allocation (DMA):

Static memory allocation: (done during compile time).

int a = 10;



int a[10] = {...};



dynamic memory allocation: (done during run time).

functions:
1. malloc
2. calloc
3. realloc
4. free

(cast type: type of data that has to be returned)

→ Memory allocation done during the run time of a program is known as DMA.

→ DMA allows a program to obtain memory space while running or to release space when no space is required.

03/04/16.

Malloc 1):

→ The name 'malloc' stands for memory allocation

→ The function 'malloc' reserves a block of memory of specified size and returns a pointer of first byte of allocated space.

Syntax: ptr = (cast-type *) malloc * (size of-byte)

eg. ptr = (int *) malloc (100 * sizeof (int));

100*2

(200);

↳ (a int-type block of memory of size 100. & area 200)

Here, ptr is a pointer of datatype. The malloc function returns a pointer to an area of memory with size of byte-size;

If space is insufficient, allocation fails.
And, returns a NULL pointer.

Calloc():

→ The name 'calloc' stands for contiguous allocation.

→ The only diff. b/w malloc() and calloc() is that malloc allocates single block of memory whereas calloc allocates multiple blocks of memory, each of same size and sets all bytes to 0.

Syntax: ptr = (cast_type *) calloc (n, element size);

This statement will allocate a contiguous space in memory for an array of n elements.

eg ptr = (float *) calloc (25, sizeof (float));

Realloc():

→ The name 'realloc' stands for reallocation.

→ If the previously allocated memory is insufficient or more than sufficient, then we can change memory size previously allocated using 'realloc' function.

Syntax: ptr=realloc (ptr, newsize);

free():

→ Dynamically allocated memory with either 'calloc' or 'malloc' function does not get return on its own. (does not release).

→ The user must use free() function explicitly to release space.

Syntax: free (ptr);

This statement cause the space in memory pointer by ptr to be de-allocated.

(8) WAP to find sum of n elements entered by user and allocate memory dynamically using malloc function.

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int n, sum=0, *p
    printf ( " Enter number of elements : ");
    scanf (" %d",&n);
Line7:  p = (int *) malloc ( n * sizeof (int)); // allocate memory dynamically
                                  sizeof function
    if ( p == NULL)
    {
        printf (" Memory insufficient ");
    }
    else
    {
    printf (" Enter the elements : ");
    for ( i=0 ; i<n ; i++)
    {
        scanf ( " %d ",p+i) ;
        sum = sum + *(p+i) ;
    }
    printf ( " Sum = %d", sum);
    free(p);
    }
}
```

Line7 : p = (int *) calloc (n, sizeof(int)); // using calloc function.

Structures and Unions:

02/04/16

Structure:

A structure is a constructed datatype used in C for packing / clubbing data of different types.

In other words, a structure is a convenient tool for handling a group of logically related data items.

eg. Student record

```
{ char name [20] ;
  int roll-no [10] ;
  float marks ;
};
```

Difference b/w Arrays & structures:

(i) An array is a collection of related data elements of similar type.
Whereas, structure can have elements of different datatype.

(ii) An array is a derived datatype, whereas a structure is an user-defined datatype.

(iii) Array behaves like a built-in data type i.e. declare an element and use it.
But, in structure we have to design and declare before it is used.

Defining a structure:

```
struct student_record
{
    char name [20];
    int roll no [ ];
    float marks;
};
```

→ 'struct' to hold name, defined element

→ Structu structu

Generall

② Design

stru {

}

Declarati

st {

→ The

The

are

Scanned by CamScanner

→ 'struct' is a keyword, that declares a structure to hold the details of three data-fields: name, roll no. and marks. These data-fields defined inside a structure, are called structure elements or members.

→ Structure tag is the name given to the structure. Here,: student_record

General syntax:

```
struct structure_tag
{
    datatype member 1;
    datatype member 2;
    .....
    .....
};
```

② Design a structure for storing the bank details.

```
struct bank_record
{
    char name [20];
    int account_no;
    float balance;
};
```

Declaration of structure variable

```
struct student_record
{
    char name [20];
    int roll_no;
    float marks;
};
struct student_record student1, student2;
```

→ The members of structures are not variables. They donot occupy any memory unless they are associated with structure variables such as student 1

Structure initialization :

(I)

```
main()
{
    struct info
    {
        int weight;
        float height;
    }
    student1 = {60, 160.5};
    student2 = {50, 172.5};
}
```

(II)

```
main()
{
    struct std_record
    {
        char name[20];
        int roll no;
    };
    struct std_record std1 = {"XYZ", 100};
    struct std_record std2 = {"PQR", 101};
}
```

Accessing structure :

11/04/16.

→ using a dot (.) operator.

syntax : structure_varname.member_name

eg.

```
main()
{
    struct vehicle
    {
        char vname[20];
        char color[10];
        int wheels;
    }
    v1 = {"NANO", "RED", 4};
    printf("Vehicle name is %s", v1.vname);
    printf("Vehicle color is %s", v1.color);
    printf("No. of wheels is %d", v1.wheels);
}
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
    struct student
    {
        char name[30];
        int rollno;
        float marks;
    };
    struct student std;
    printf("Enter name ");
    gets(std.name);
    printf("Enter roll no: ");
    scanf("%d", &std.rollno);
    printf("Enter marks ");
    scanf("%f", &std.marks);
    printf("\nDisplaying student details");
    printf("Name is %s", std.name);
    printf("Roll no is %d", std.rollno);
    printf("marks is %f", std.marks);
}
```

Array of structure :

```
main()
{
    struct student
    {
        int sub1;
        int sub2;
        int sub3;
    };
    struct marks student[3] =
      { {50,60,70}, {55,75,70}, {60,75,80} };
}
```

⑨ WAP to read and display all students in cla

```
#include <stdio.h>
main()
{
    struct student
    {
        char name[30];
        int roll_no;
        float marks;
    };
    struct student std[60];
    int n, i;
    printf("No. of students: ");
    scanf("%d", &n);
    for (i=0; i<n; i++)
    {
        printf("Enter name: ");
        gets(std[i].name);
        printf("Enter roll no: ");
        scanf("%d", std[i].roll_no);
        printf("Enter marks:");
        scanf("%f", std[i].marks);
    }
    for (i=0; i<n; i++)
    {
        printf("Name: %s", std[i].name);
        printf("Roll No. %d", std[i].roll_no);
        printf("Marks: %d", std[i].marks);
    }
}
```

Nested

```
struct
{
};
struc
{
};
```

OR

⑨ def
(empl
basic
#
ma
{

Nested structure (structure within structure).

```
struct DOB
{   int day, month, year;
};
struct student
{   int roll-no;
    char name[50];
    float fees;
    struct DOB;
};
```

(OR)

```
struct student
{   int roll-no;
    char name[50];
    float fees;
    struct DOB
    {   int day, month, year;
    };
};
```

(1) Define a nested structure with a name (employee details) fields : name, department, basic pay, TA, HRA.

```
# include<stdio.h>
main()
{   struct employee details
    {   char name[30];
        char department[100];
        struct salary
        {   int basic pay;
            int TA, HRA;
        }
    }
```

Union :

→ almost same as structure.
→ difference is in the storage members.

eg.
```
char name[20];
char dept[10];
int salary;
```

For structure, memory = 20+10+2 = 32 bytes.
For union, memory = highest individual size = 20 bytes.

12/04/16.

File Management in C

→ File is a place on disk where group of related data is stored. eg. C programs.

Common functions

fopen(): creates new file for use / open an existing file for use.
fclose(): closes a file which had been open for use.
getc(): allows to read characters.
putc(): write a character to a file.
fprintf(): writes a set of values to a file.
fscanf(): reads a set of values from a file.

Defining and opening a file

① file name
② data structure
③ purpose

→ file name is a string of characters that make up a valid file name for the OS.
→ The datastructure of a file is defined as FILE in the library of std i/o function. Therefore all files should be declared as FILE before their use.

eg. FILE
fp =
Here,
fp
a
e

Types of
r
w
a

w+/r

a+

FILE

p1
p2

fo
fo

⑤ WAP
{

eg. FILE *fp;

fp = fopen ("filename", "mode"); → points to the beginning of file

Here, variable fp is a pointer to the type 'file'. fp contains all information about file. It is a communication link between system and a program.

Types of mode :

r ⟶ read only mode

w ⟶ write only

a ⟶ appending : it opens a file for appending (adding) data.

w+/r+ ⟶ allows us to open a file for both writing and reading.

a+ ⟶ it is same as apart from reading and writing.

```
FILE *p1, *p2;
    p1 = fopen ("data", "r");
    p2 = fopen ("results", "w");
    ....
    ....
    fclose (p1);
    fclose (p2);
```

18/4/16.

⑤ WAP to read & write a file using getc & putc.

```
{   FILE *fp;
    char c;
    fp = fopen ("test", "w");
    while ((c=getch()) != EOF)   // EOF - end of file
    {
        putc (c, fp);
    }
    fclose (fp);
```

```
    fp = fopen ( "test","r");
    while ( (c=getc(fp)) != EOF)
    {printf ( "%c", c);}
    fclose (fp);
}
```

⑨ WAP to count the number of characters in a file.

```
{
    FILE *p;
    char c;
    int count = 0;
    p = fopen ("test", "w");
    if (p == NULL)
    {   printf ("ERROR! File can't be created");
    }
    while ((c = getchar()) != EOF)
    {   putc (c, p);
    }
    fclose (p);
    p = fopen ("test","r");
    while ((c= getc (p)) != EOF)
    {
        count ++;
        printf ("%c", c);
    }
    printf ("Number of characters = %d", count);
    fclose (p);
}
```

⑧ WAP

{

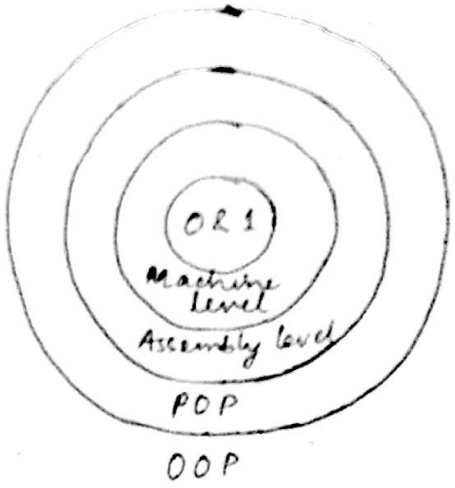
}

(2) WAP to copy a file from one to another.
(Source file → Destination file)

```
{
    FILE *p1, *p2;
    char c;
    p1 = fopen("test1", "r");
    p2 = fopen("test2", "w");
    while ((c = getc(p1)) != EOF)
    {
        putc(c, p2);
    }
    fclose(p2);
    p2 = fopen("test2", "r");
    while ((c = getc(p2)) != EOF)
    {
        printf("%c", c);
    }
    fclose(p2);
    fclose(p1);
}
```

Object Oriented programming (OOP):

(i) OOP v/s POP (Procedural oriented programming)
(C++) (C)



POP Top-down approach

OOP Bottom up approach

(ii) Basic concepts of OOP
- classes & objects.
- inheritance
- encaptulation
- abstraction.
- polymorphism.

13/04/18

Characteristics of POP:

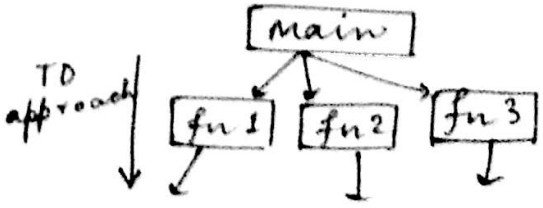
(i) Emphasis is given on doing things (algorithm or instructions).

(ii) Large programs are divided into smaller programs known as functions.

(iii) Most of the functions share global data

(iv) Data move openly around the system from function to function.

(v) It employs top-down approach in program design.



Characteristics of OOP :

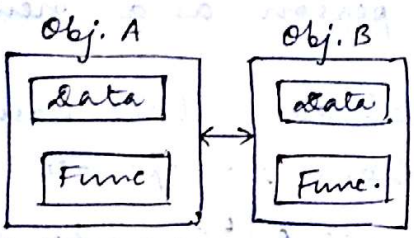
(i) Emphasis is given on data, rather than procedure. (object)

(ii) Programs are divided into what are known as objects.

(iii) Data structures are designed such that they characterize the objects.

(iv) Objects may communicate with each other through function.

(v) It follows bottom-up approach.



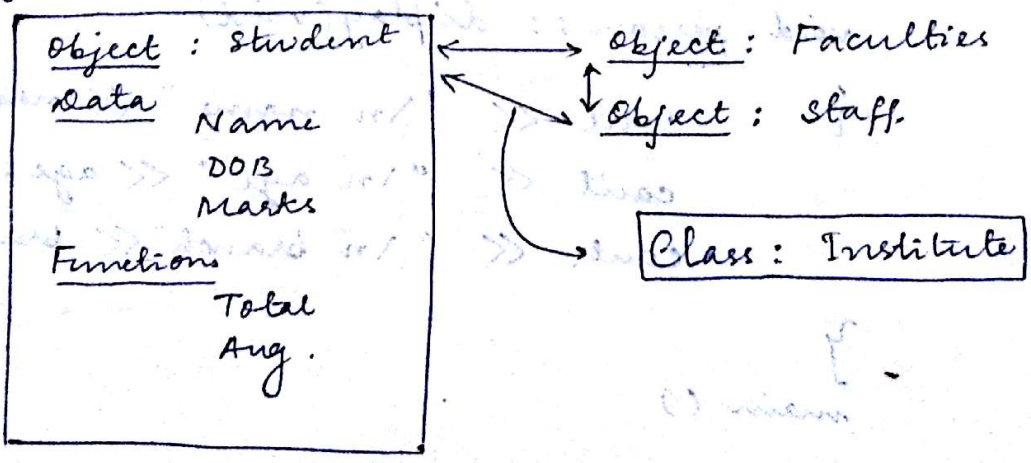
Object : Objects are the basic real time entities in an object-oriented system. They may represent a person, a place, an account or any other item that the program has to handle.

→ Objects take up space in the memory and have an associated address, like a structure in C.

Classes : A class is a collection of objects of similar type.

→ Objects are known as instances of class.

eg.



```
#include <iostream.h>
class person
{
    char name[30];
    int age;
    char branch;
    Public :
        void getdata(void);
        void display(void);
};
```

member functions

Access specifiers used in C++

① Public
② Private
③ Protected

→ The program define person as a new data of type class.

→ The class 'person' includes three basic data type items and 2 functions to operate on that data. These functions are called 'member functions'.

```
void person :: getdata (void)
{
    cout << "enter name";
    cin >> name;
    cout << "enter age";
    cin >> age;
    cout << "enter branch";
    cin >> branch;
}
```

scope resolution.

similar to printf & scanf in C.

console out
console in

cout
cin.

```
void person :: display(void)
{
    cout << "\n name" << name;
    cout << "\n age" << age;
    cout << "\n branch" << branch;
}
main()
```

```
{
    person p ;          → object
    p.getdata() ;
    p.display() ;
}
```

→ The main program uses person to declare variable of its type

→ As we know class variables are known as objects. Here 'p' is an object of type 'person'.

Inheritance :

→ The process by which objects of one class acquire the properties of another class is called inheritance.

→ It supports the concept of hierarchial classification.

For eg. the bird 'Robin' is a part of class 'flying bird', which is again a part of the class 'Bird'.

→ The principle behind this sort of division is that each derived class shares common characteristics with the class from which it is derived.

```
# include <stdio.h>
main()
{ char name ;
  int marks ;
  pf ("\n ---- std.")
  sf ("%d", &n)
  FILE *p;
  p = fopen("c:\\student.txt", "w");
  if (p == NULL)
     pf ("Error!");
     exit(1);
  for (i=0; i<n; ++i)
  { pf ("for student %d\n
        Enter name", i+1)
    sf ("%s", name);
    pf ("Enter marks");
    sf ("%d", marks);
    fprintf
    (p, "Name = %s
        \n Marks = %d",
        name, marks);
  fclose (p)
}
```

Count

```
{ FILE *p;
  char c;
  int count = 0;
  p = fopen ("test", "w");
  if (p == NULL)
  {  printf ("ERROR!");
  }
  while ( (c = getchar()) != EOF)
  {  putc (c, p);
  }
  fclose (p);
  p = fopen ("test", "r");
  while ( (c = getc(p)) != EOF)
  {
      count++;
      printf ("%c", c);
  }
  printf (" Number of characters = %d", count);
  fclose(p);
}
```

Copy

```
{ FILE *p, *q;
  char c;
  p = fopen ("Source", "r");
  q = fopen ("Destination", "w");
  while ( (c = getc(p)) != EOF)
  {     putc ( c, q);
  }
  fclose (q);
  q = fopen ("Destination", "r");
  while ( (c = getc(q)) != EOF)
  {     printf ("%c", c);
  }
  fclose (q);
}   fclose (p);
```