UNIT-1

Selection Statements:

These are also called as decision making statements. These statements change the flow of execution depending on a given logical condition.

Types of decision making statements are:

(i) Simple if statement
(ii) if-else statement
(iii) Nested if-else statement
(iv) else-if ladder
(v) Switch statement

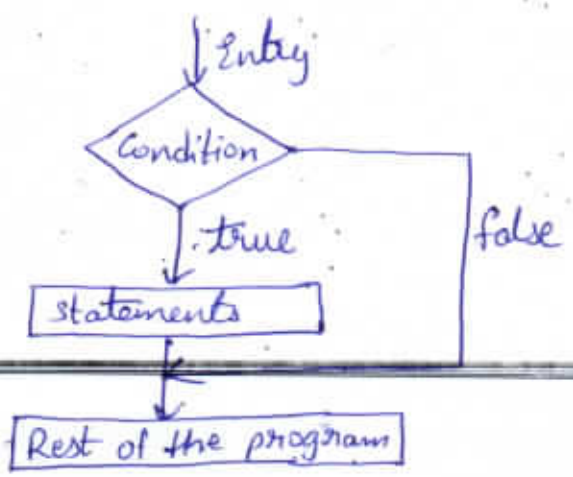
① Simple if statement:

* "if" statement executes only when the condition is true.
* It does nothing when the condition is false.
* It has only one option.

Syntax:

```
if (condition)
{
  statements;
}
```

flowchart:



Program:

```
main()
{
 int num;
printf (" Enter a number:");
scanf (" %d ", &num);
if (num > 0)
  {
   printf (" You entered %d \n", num);
  }
printf(" Simple if is easy");
}
```

output 1

```
Enter a number : 2
You entered 2
Simple if is easy
```

output 2

```
Enter a number : -1
Simple if is easy
```

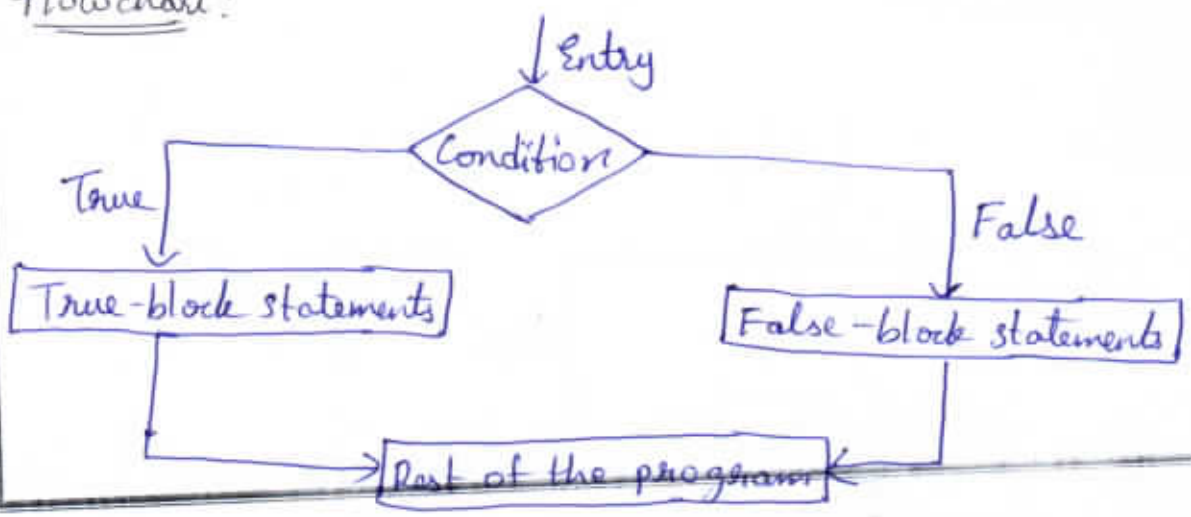
② if - else statement:

In if-else either True-block or False-block is executed and not both.

Syntax:

```
if (condition)
  {
   statements of true-block
  }
else
  {
   statements of false block
  }
```

flowchart:



Program:

```
main()
{
  int num;
  printf("Enter a number:");
  scanf("%d", &num);
  if (num % 2 == 0)
    printf("%d is an even number", num);
  else
    printf("%d is an odd number", num);
}
```

output 1:
Enter a number: 4
4 is an even number

output 2:
Enter a number: 5
5 is an odd number

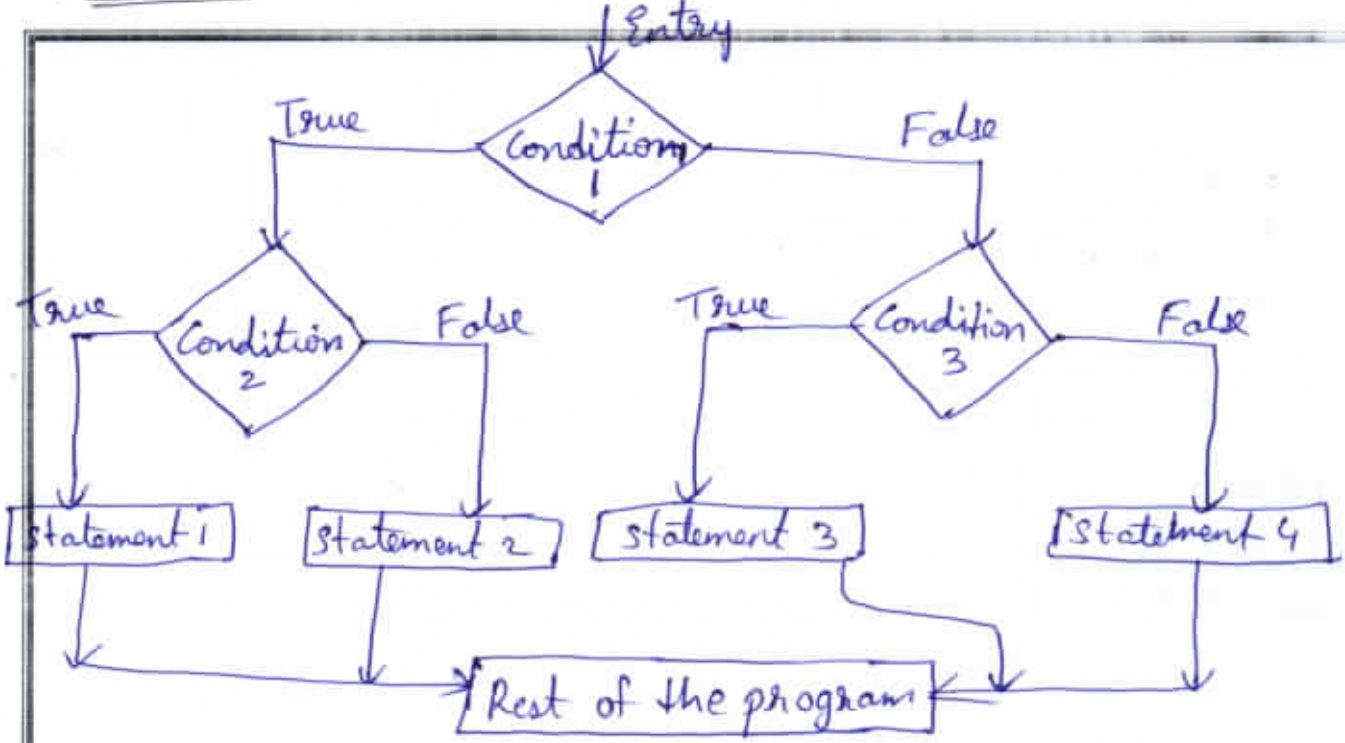
③ Nested "if-else" statement

Using of one if-else statement in another if-else statement is called as nested if-else statements.

Syntax:

```
if (condition 1)
{
  if (condition 2)
    statement 1;
  else
    statement 2;
}
else
{
  if (condition 3)
    statements 3;
  else
    statements 4;
}
```

Flowchart:



program:

```
main()
{
 int x,y,z;
 printf("Enter three numbers:");
 scanf("%d %d %d",&x,&y,&z);
 printf("\nLargest of three numbers is:\n");
 if (x>y)
 {
   if (x>z)
     printf("x=%d",x);
   else
     printf("z=%d",z);
 }
 else
 {
   if (z>y)
     printf("z=%d",z);
   else
     printf("y=%d",y);
 }
}
```

output1

Enter three numbers: 1 2 3
Largest of three numbers is:
z = 3

output2

Enter three numbers: 3 2 1
largest of three numbers is:
x = 3

④ "else-if" ladder:-

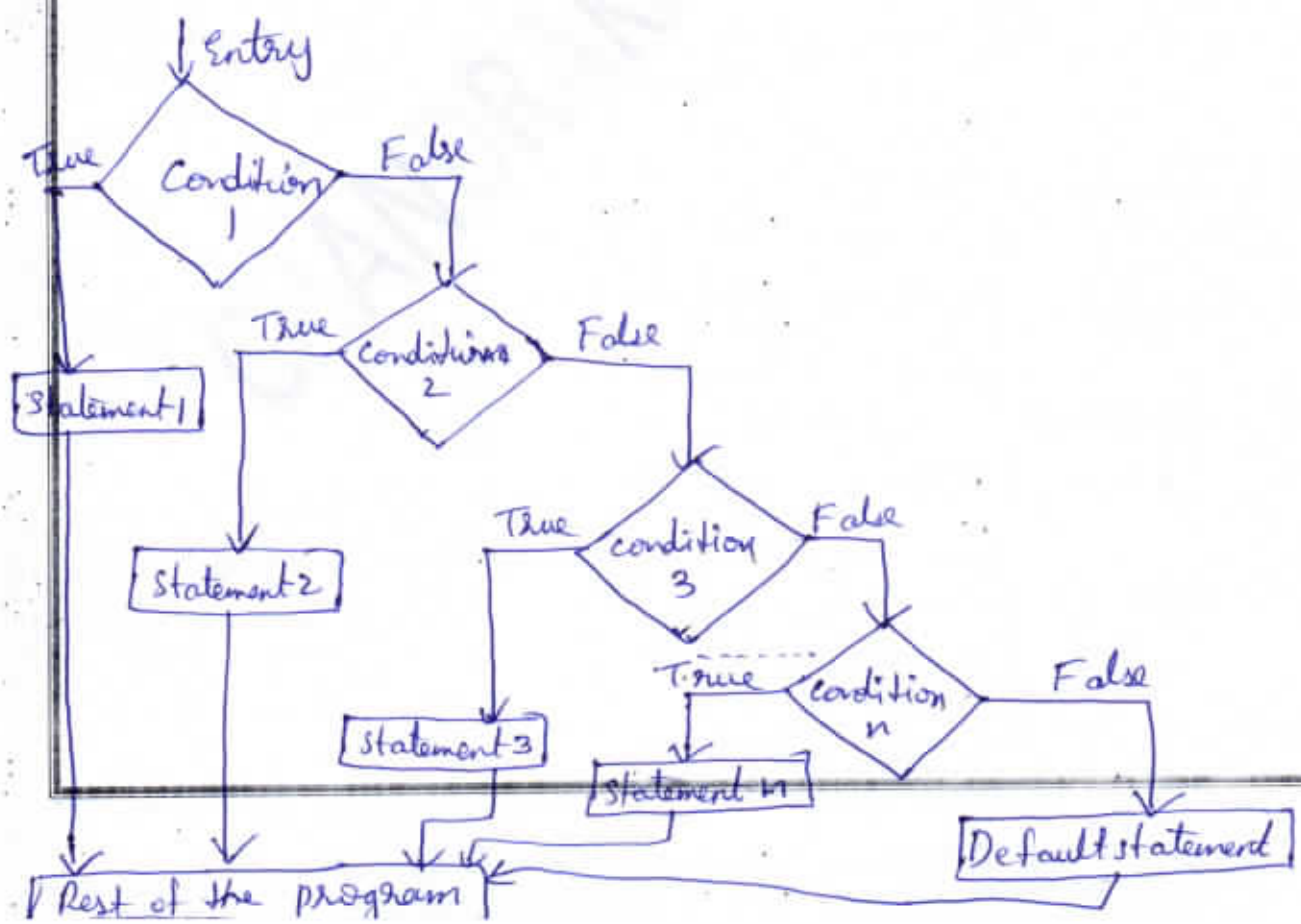
This is another way of putting 'if's' together when multiple decisions are involved.

It is used to execute one code from multiple conditions.

Syntax:

```
if (condition 1)
    statement1;
else if (condition2)
    statement2;
else if (condition3)
    statement3;
    =
    =
else if (condition n)
    statement n;
else
    default statement;
```

Flowchart:



Program:

```
main()
{
 int a,b, c, d;
 printf(" Enter values of a, b, c, d:");
 scanf ("%d %d %d %d", &a, &b, &c, &d);
 printf ("\nThe largest number is: \n");
 if((a>b) && (a>c) && (a>d))
   printf ("a = %d ", a);
 else if( (b>a) && (b>c) && (b>d) )
    printf (" b = %d ", b);
 else if( (c>a) && (c>b) && (c>d))
    printf (" c = %d", c);
 else
    printf (" d = %d", d);
}
```

output1:

```
Enter values of a,b,c,d: 1 2 3 4
The largest number is
   d = 4
```

output 2:

```
Enter values of a,b,c d: 4 3 2 1
The largest number is:
   a = 4
```

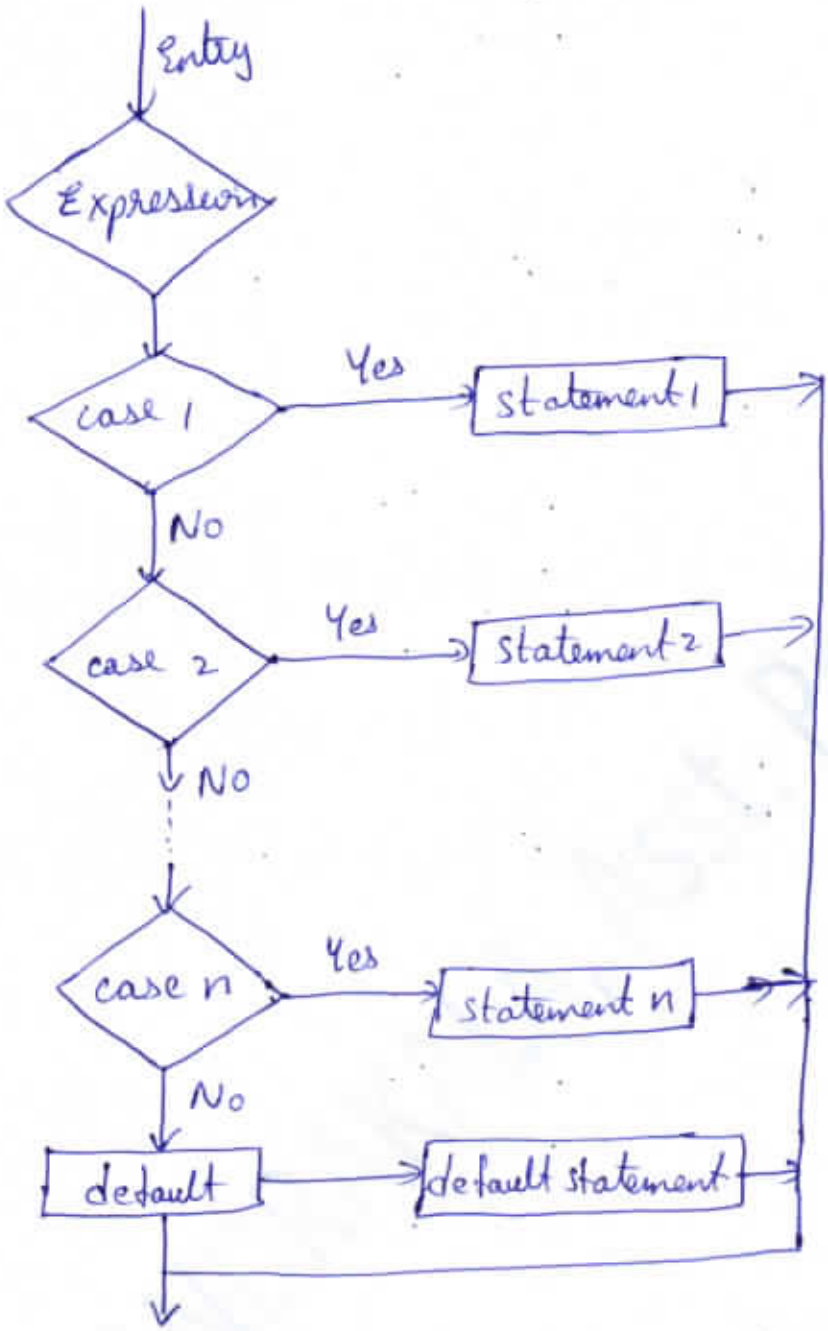
⑤ Switch statement:

It is used to select one option from several options.

Syntax:

```
switch (expression)
 {
  case value1 : statement 1;
                break;
  case value2 : statement 2;
                break;
      =
      =
  case value n : statement n;
                 break;
  default      : default statement;
                 break;
 }
```

Flowchart:


Entry
Expression
case 1
Yes
statement 1
No
case 2
Yes
statement 2
No
case n
Yes
statement n
No
default
default statement

Program:

```
main()
{
  char choice;
  printf("Enter any vowel:");
  scanf("%c", &choice);

  switch(choice)
  {
  case 'a': printf("Your character is a");
            break;
  case 'e': printf("Your character is e");
            break;
  case 'i': printf("Your character is i");
            break;
  case 'o': printf("Your character is o");
            break;
  case 'u': printf("Your character is u");
            break;
  default : printf("Not a vowel");
            break;
  }
}
```

output1

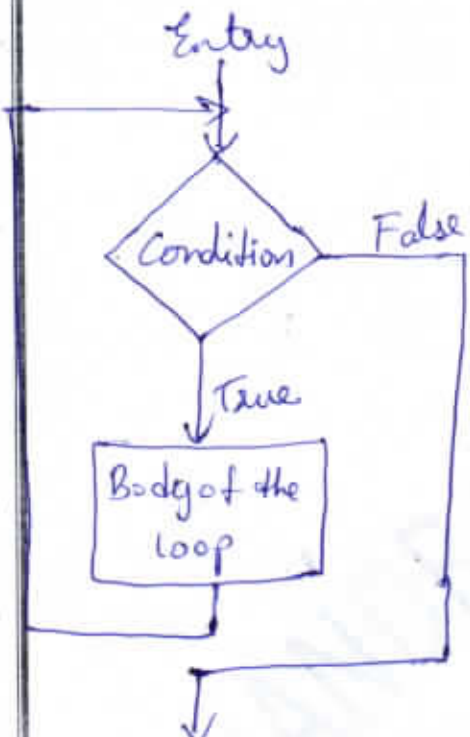
```
Enter any vowel: a
Your character is a
```

output2

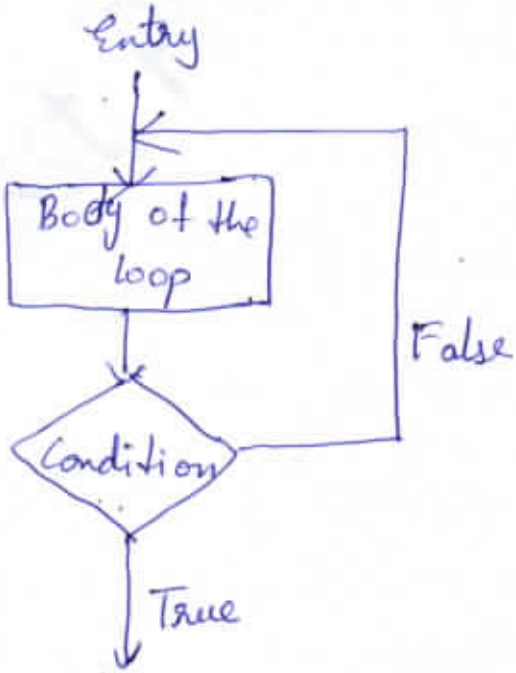
```
Enter any vowel: h
Not a vowel
```

Iteration statements:

* These are also called as loop control statements.
* A loop is defined as a block of statements which are repeatedly executed for certain number of times.
* Depending on the position of control statement in the loop, a control structure is classified as
  (i) entry controlled loop
  (ii) exit controlled loop



(a) Entry control



(b) Exit control

* Types of loop control statements are:
  (i) for loop
  (ii) while loop
  (iii) do-while loop

① for loop:- It is an entry-controlled loop.

It works where number of iterations of the loop is known before the loop is entered.

Syntax:

```
for (initialization ; condition ; increment/decrement)
{
  body of the loop
}
```

program:

```
main()
{
 int n;
 for (n=1; n<=5; n++)
 {
  printf ("%d\n",n);
 }
}
```

output

1
2
3
4
5

② while loop: It is an entry controlled loop.

* If the condition is true, then the body of the loop is executed.
* The process is repeated until the condition is false.
* It is the simplest of all looping structures.

Syntax:

```
initialization;
while (condition)
  {
    body of the loop
  }
```

program:

```
main()
{
 int n;
 n=1;
 while (n<=5)
 {
  printf("%d \n",n);
  n++;
 }
}
```

output
1
2
3
4
5

③ do-while loop: It is an exit control loop statement.

* Here condition is checked at the end of the loop.
* Loop will execute atleast once even if condition is false.

Syntax:

```
do
 {
  body of the loop
 }
 while (condition);
```

program:

```
main()
{
 int n;
 n=1;
 do
 {
  printf("%d \n", n);
  n++;
 }
 while(n<=5);
}
```

output
1
2
3
4
5

Jump Statements:

These are also called as unconditional statements.

These statements are used to interrupt the normal flow of program.

Types of jump statements are:

(i) Break statement
(ii) Continue statement
(iii) Goto statement
(iv) Return statement

① Break statement:-

* It is used inside loop or switch statement.
* When break is find inside a loop, then the loop is aborted & continue next statements after loop.

Syntax:

```
Jump statement;
break;
```

program:

```
main()
{
 int a;
 for(a=1 ; a<=10 ; a++)
 {
  printf ("hello\n");
  if (a>=5)
  break;
 }
 printf ("End of program");
}
```

output

```
hello
hello
hello
hello
hello
End of program
```

④ Continue statement:

* ~~It is used inside a loop~~
* Loop doesnot terminate when a continue statement is used, but remaining loop statements are skipped.

Syntax:

```
Jumpstatement;
  Continue;
```

Program:

```
main()
{
int a;
for(a=0; a<=3; a++)
 {
   printf("Hello \n");
   continue;
   printf("World");
 }
}
```

output

```
Hello
Hello
Hello
```

⑤ Goto Statement:

* It is a statement which jumps from one point to another point within a function.
* Goto requires a label in order to identify the place where the branch is to be made.
* A label is any valid variable name followed by a colon (:)

Syntax:

```
goto label;
  -----
  -----
label:
  statements;
```

program:

```
main()
{
 printf (" Hi \n");
 printf (" Hello \n");
 goto last;
 printf (" Welcome \n");
 printf (" World \n");
 last :
 printf(" End of program");
}
```

output:

```
Hi
Hello
End of program
```

(4) Return statement:

* It terminates the execution of a function & returns control to the calling function.
* It can also return a value to the calling function.

Syntax:

`return;` or `return expression;`

program:

Ex1:

```
int main()
{
 int a,b,sum;
 printf("Enter two numbers:\n");
 scanf("%d %d", &a, &b);
 sum = a+b;
 printf(" %d + %d = %d", a, b, sum);
 return 0;
}
```

output:

```
Enter two numbers:
5 6
5 + 6 = 11
```

Ex2:
```
void main()
{
int sum;
sum = addDigits();
printf("Sum is %d", sum);
return;
}
addDigits()
{
int sum, digit;
sum = 0;
for (digit = 0; digit <= 5; ++digit)
{
sum += digit;
}
return sum;
}
```

output

Sum is 15

Expression Statements

* Most of the statements in a C program are expression statements.
* An expression statement consists of an expression followed by a semicolon.

Syntax:

```
Expression statement ;
```

Examples:

① A = 10;
② printf("Hello");
③ sum = a+b;
④ display(x,y);

program:

```
int main()
{
    int a, b, sum;
    printf("Enter two numbers:");
    scanf("%d %d", &a, &b);
  sum= add(a, b);
    printf("\nSum is: %d", sum);
    return 0;
}
int add (int p, int q)
{
  int sum;
  sum = p+q;
  return(return);
}
```

output:

```
Enter two numbers:
2
3
Sum is: 5
```

Block statements:

* It is also called as compound statement.
* A block of statements is a group of statements which is treated as a single statement by a compiler.
* Blocks begin with '{' and end with '}'.
* Blocks can be nested inside of other blocks.
* Blocks allow multiple statements to be used wherever a single statement can normally be used.
* They are extremely useful when you need a set of statements to execute together.

program:

(Same as above example program)

Arrays:

* An array is a group of similar data items that share a common name. (or)
* An array is a collection of homogeneous elements in a single variable.

Types of arrays are:
(i) One-dimensional array
(ii) Two-dimensional array
(iii) Multi-dimensional array.

One-dimensional array:

It is also called as single dimensional array, or single-subscripted variable.

Syntax:

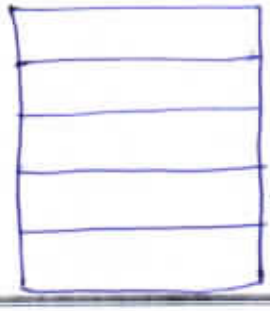
```
datatype variablename[size];
```

Declaration of arrays:

Ex: `int number[5];`

Here we are representing a set of 5 numbers. They are {35, 40, 20, 57, 19}, by a arrayname "number".

The computer reserves five storage locations:



The values to array elements are assigned as follows.

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

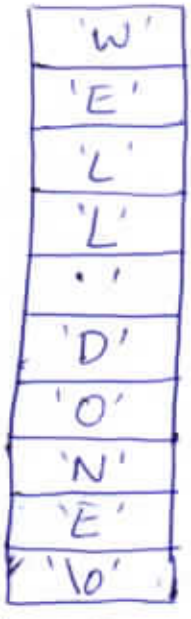
This would cause the array number to store values as:

number[0]	35
number[1]	40
number[2]	20
number[3]	57
number[4]	19

Ex2: `char name[10];`

`name = "WELL DONE";`

Each character of the string is treated as an element of an array name & is stored in the memory as follows:



For the character string, an additional null character is seen. When declaring character arrays, we must always allow one extra element space for null character ('\0').

Initialization of arrays:

* After an array is declared, it must be initialized. Otherwise, it will contain garbage values.
* It can be initialized in the same way as the ordinary variables can be declared.

Syntax:

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

Examples:

① static int number[3] = { 0, 0, 0 };

② static int number[4] = { 5, 8, 9 };

* Here only 3 numbers of elements are initialized.
* So, the remaining element will be set to zero.

③ static int counter[] = { 1, 1, 1, 1 };

* Here the size is omitted.
* So, the compiler allocates enough space for all initialized elements.

④ static char name[] = { 'J', 'o', 'h', 'n' };

Note: Static keyword could be omitted, when an ANSI compiler is used.

Drawbacks:

* There is no convenient way to initialize only selected elements.
* There is no shortcut method for initializing a large number of array elements like the one available in FORTRAN language.

Program:

```
int main()
{
 int marks[10], i, n, sum=0, average;
 printf("Enter n value:");
 scanf("%d", &n);
 for(i=0; i<n; ++i)
 {
  printf("Enter number %d:", i+1);
  scanf("%d", &marks[i]);
  sum += marks[i];
 }
 average = sum/n;
 printf("Average = %d", average);
 return 0;
}
```

output:

```
Enter n value : 5
Enter number 1: 45
Enter number 2 : 35
Enter number 3 : 38
Enter number 4 : 31
Enter number 5 : 49
Average = 39
```

Two-dimensional arrays:

* There could be a situation where table of values will have to be stored, in that case, we use two-dimensional arrays.
* It is also called as two-subscripted variable.

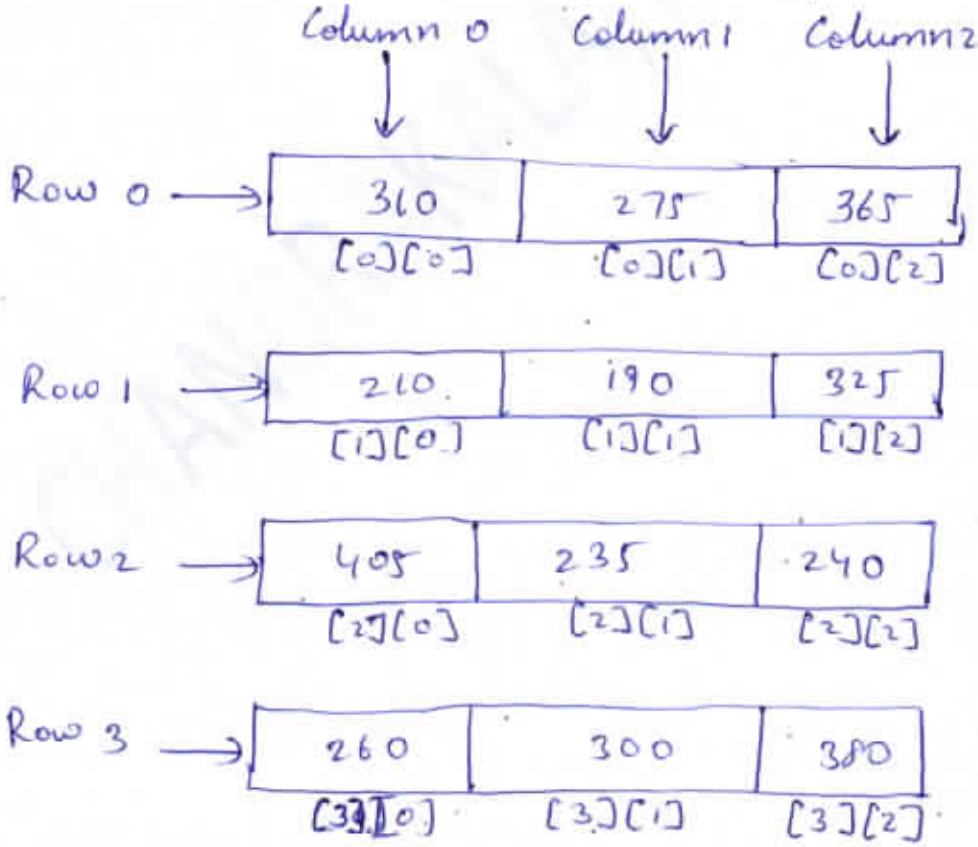
Syntax:

```
datatype arrayname[rowsize][columnsize];
```

Declaration:

```
int table[4][3];
```

The above two dimensional array is stored in memory as follows.



Initializing two-dimensional arrays:

Syntax:

```
static datatype arrayname[rowsize][col_size] = { list of elements };
```

Examples:

① `static int table[2][3] = {0,0,0,1,1,1};`

* initializes elements of first row to zero and the elements of second row to one.

② `static int table[2][3] = { {0,0,0}, {1,1,1} };`

* here elements of each row are surrounded by braces.

③
```
static int table[2][3] = {
                           {0,0,0},
                           {1,1,1}
                         };
```

Note: If the values are missing in initializer, they are automatically set to zero.

program:

```
void main()
{
 int i,j, row, col, mat[10][10];
 printf("Enter the order of the matrix:\n");
 scanf("%d %d", &row, &col);
 printf("\n Enter the elements of matrix:\n");
  for(i=0; i<row; i++)
   for(j=0; j<col; j++)
 scanf("%d", &mat[i][j]);
```

```
printf("\n The elements in the matrix are : \n");
    for(i=0; i<row; i++)
    {
      for(j=0; j<col; j++)
      {
        printf("%d", mat[i][j]);
        printf("\t");
      }
      printf("\n");
    }
    getch();
}
```

output:

```
Enter the order of the matrix:
 3  3
Enter the elements of matrix:
 1
 2
 3
 4
 5
 6
 7
 8
 9
The elements in the matrix are:
 1    2    3
 4    5    6
 7    8    9
```

Multi-dimensional arrays:

Syntax:

```
datatype arrayname[size1][size2][size3]....[sizen];
```

Examples:

```
int survey[3][5][12];
float table[5][4][5][3];
```