
UNIT- II: Control Flow**:** Statements and Blocks, if, switch statements, **Loops**: while, do-while, for, break and continue, go to and Labels.
Arrays and Strings**:** Introduction, One- dimensional arrays, Declaring and initializing Arrays, Multidimensional arrays, Strings, String Handling Functions.

UNIT-II

1. What are different types of 'if' statements available in c? Explain them.

(Or)

Describe all the variants of if-else statement with clear syntax and examples.

A. There are 4 if statements available in C:

1. Simple if statement.
2. if…else statement.
3. Nested if…else statement.
4. else if ladder

1. Simple if statement

Simple if statement is used to make a decision based on the available choice. It has the following form:

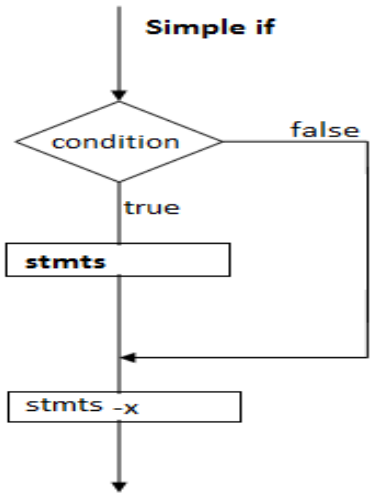
Syntax:

```
if ( condition )
{
  Stmt block;
}
Stmt-x;
```

In this syntax,

- If is the keyword. <*condition*> is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses '(' and ')'.
- The *stmt block* can be a simple statement or a compound statement or a null statement.
- *Stmt-x* is any valid C statement.

The flow of control using simple if statement is determined as follows:



Whenever simple if statement is encountered, first the condition is tested. It returns either true or false. If the condition is false, the control transfers directly to stmt-x with out considering the stmt block. If the condition is true, the control enters into the stmt block. Once, the end of stmt block is reached, the control transfers to stmt-x

Example Program:

PROGRAM FOR IF STATEMENT:

```
#include<stdio.h>
#include<conio.h>
int main()
 {
 int age;
 printf("enter age\n");
 scanf("%d",&age);
 if(age>=55)
 printf("person is retired\n");
getch();
```

}

2. If—else statement

if…else statement is used to make a decision based on two choices. It has the following form:

syntax:

if(condition)

{

True Stmt block;

}

Else

{

False stmt block;

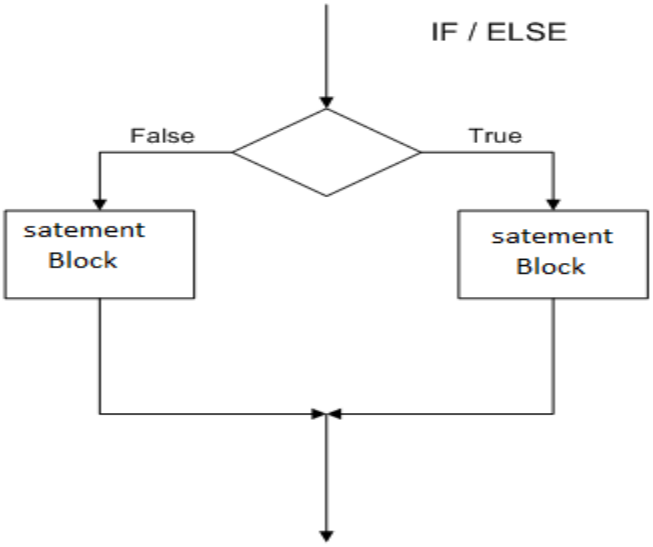
}

Stmt-x;

In this syntax,

- If and else are the keywords.
- *<condition>* is a relational expression or logical expression or any expression that returns either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- The *true stmt block* and false stmt block are simple statements or compound statements or null statements.
- *Stmt-x* is any valid C statement.

The flow of control using if...else statement is determined as follows:



Whenever if...else statement is encountered, first the condition is tested. It returns either true or false. If the condition is true, the control enters into the true stmt block. Once, the end of true stmt block is reached, the control transfers to stmt-x without considering else-body.

If the condition is false, the control enters into the false stmt block by skipping true stmt block. Once, the end of false stmt block is reached, the control transfers to stmt-x.

Example Program:

PROGRAM FOR IF ELSE STATEMENT:

```
#include<stdio.h>
#include<conio.h>
int main()
 {
 int age;
 printf("enter age\n");
 scanf("%d",&age);
 if(age>=55)
 printf("person is retired\n");
```

```
else
printf("person is not retired\n");
getch();
}
```

3. Nested If—else statement

Nested if…else statement is one of the conditional control-flow statements. If the body of if statement contains at least one if statement, then that if statement is called as "Nested if…else statement". The nested if…else statement can be used in such a situation where at least two conditions should be satisfied in order to execute particular set of instructions. It can also be used to make a decision among multiple choices. The nested if…else statement has the following form:

Syntax:

```
If(condition1)
{
If(condition 2)
{
 Stmt1
}
Else
{
Stmt 2
}
}
Else
{
Stmt 3;
```

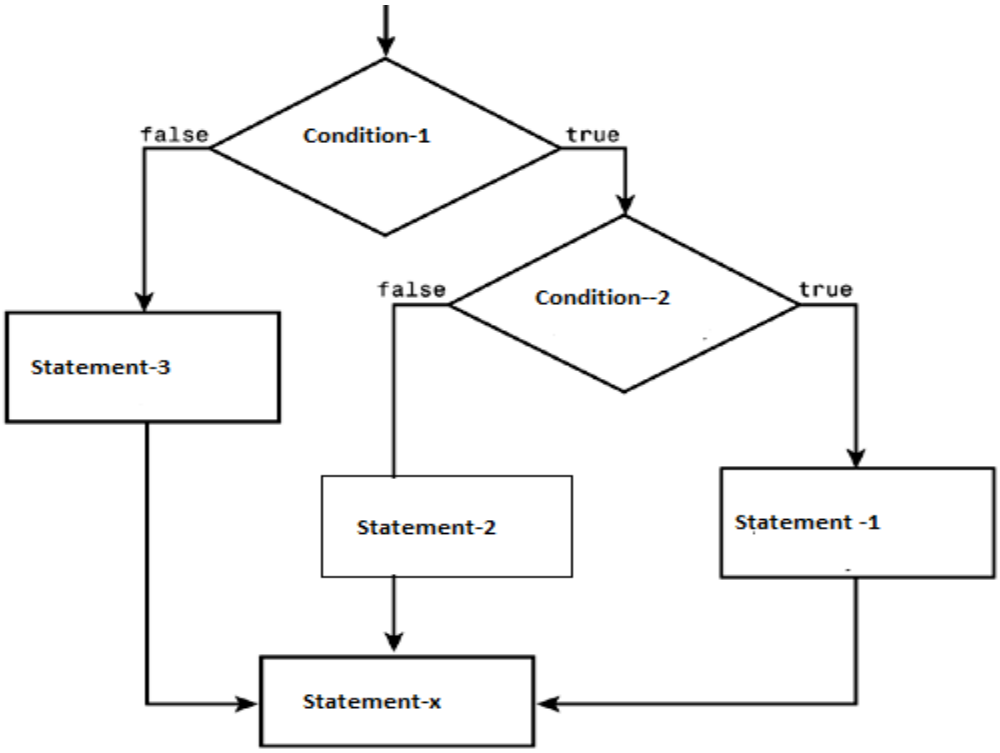
}

Stmt-x;

In this syntax,

- if and else are keywords.
- *<condition1>,<condition2> ... <condition>* are relational expressions or logical expressions or any other expressions that return true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *If-body* and *else-body* are simple statements or compound statements or empty statements.
- *Stmt-x* is a valid C statement.

The flow of control using Nested if...else statement is determined as follows:



Whenever nested if…else statement is encountered, first <condition1> is tested. It returns either true or false.

If condition1 (or outer condition) is false, then the control transfers to else-body (if exists) by skipping if-body.

If condition1 (or outer condition) is true, then condition2 (or inner condition) is tested. If the condition2 is true, if-body gets executed. Otherwise, the else-body that is inside of if statement gets executed.

Example Program:

PROGRAM FOR NESTED IF… ELSE STATEMENT:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
printf("enter three values\n");
scanf("%d%d%d",&a,&b,&c);
if(a>b)
{
if(a>c)
printf("%d\n",a);
else
printf("%d\n",b);
}
else
{
if(c>b)
printf("%d\n",c);
else
printf("%d\n",b);
```

```
}
getch();
}
```

4. Else—if Ladder

Else if ladder is one of the conditional control-flow statements. It is used to make a decision among multiple choices. It has the following form:

Syntax:

```
If(condition 1)
{
 Statement 1;
}
Else if(condition 2)
{
 Statement 2;
}
Else if(condition 3)
{
 Statement 3;
}
Else if(condition n)
{
 Statement n;
}
Else
```

{

Default Statement;

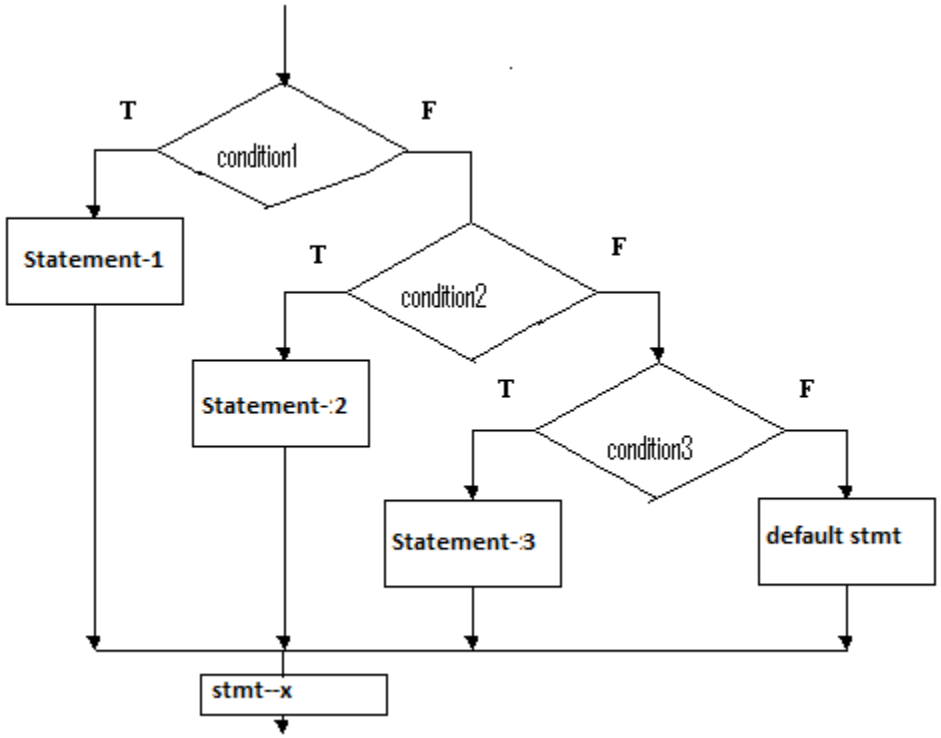
}

Stmt- x;

In this syntax,

- if and else are keywords. There should be a space between else and if, if they come together.
- *<condition1>,<condition2>….<condtionN>* are relational expressions or logical expressions or any other expressions that return either true or false. It is important to note that the condition should be enclosed within parentheses (and).
- *Statement 1, statement 2,statement 3……,statement n* and *default statement* are either simple statements or compound statements or null statements.
- *Stmt-x* is a valid C statement.

The flow of control using else--- if Ladder statement is determined as follows:



Whenever else if ladder is encountered, condition1 is tested first. If it is true, the statement 1 gets executed. After then the control transfers to *stmt-x*.

If *condition1* is false, then *condition2* is tested. If *condition2* is false, the other conditions are tested. If all are false, the default stmt at the end gets executed. After then the control transfers to *stmt-x*.

If any one of all conditions is true, then the body associated with it gets executed. After then the control transfers to *stmt-x*.

Example Program:

PROGRAM FOR THE ELSE IF LADDER STATEMENT:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int m1,m2,m3,avg,tot;

printf("enter three subject marks");

scanf("%d%d%d", &m1,&m2,&m3);

tot=m1+m2+m3;

avg=tot/3;

if(avg>=75)

{

printf("distinction");

  }

else if(avg>=60 && avg<75)

{

printf("first class");   }

else if(avg>=50 && avg<60)

{

printf("second class");
```

```
  }
else if (avg<50)
{
printf("fail");
 }
getch();    }
```

2. Explain the switch statement with Example program.

switch statement:

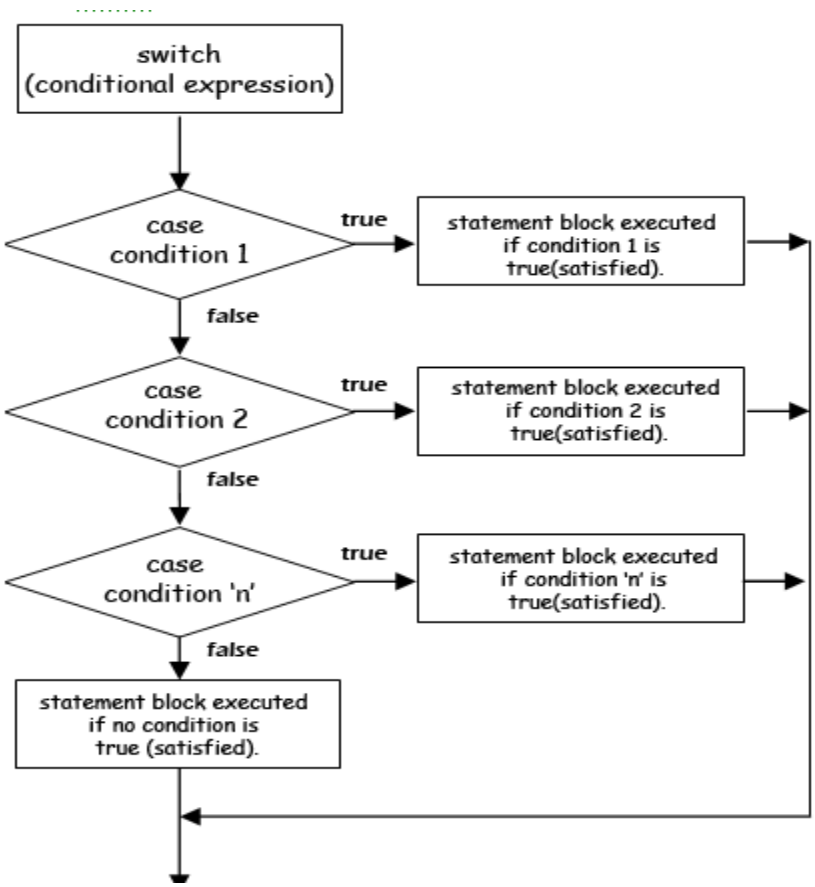
switch statement is one of decision-making control-flow statements. Just like else if ladder, it is also used to make a decision among multiple choices. switch statement has the following form:

```
switch(<exp>)
{
    case <exp-val-1>: statements block-1
                      break;
     case <exp-val-2>: statements block-2
                      break;
        case <exp-val-3>: statements block-3
                      break;
                :
                :
    case <exp-val-N>: statements block-N
                      break;
    default: default statements block
}
```

Next-statement;

In this syntax,

- switch, case, default and break are keywords.
- *<exp>* is any expression that should give an integer value or character value. In other words, it should never return any floating-point value. It should always be enclosed with in parentheses (and). It should also be placed after the keyword switch.
- *<exp-val-1>, <exp-val-2>, <exp-val-3>…. <exp-val-N>* should always be integer constants or character constants or constant expressions. In other words, variables can never be used as *<exp-val>*. There should be a space between the keyword case and *<exp-val>*. The keyword case along with its *<exp-val>* is called as a case label. *<exp-val>* should always be unique; no duplications are allowed.
- *Statements block-1, statements-block-2, statements block-3… statements block-N* and *default statements block* are simple statements, compound statements or null statements. It is important to note that the statements blocks along with their own case labels should be separated with a colon (:).
- The break statement at the end of each statements block is an optional one. It is recommended that break statement always be placed at the end of each statements block. With its absence, all the statements blocks below the matched case label along with statements block of matched case get executed. Usually, the result is unwanted.
- The statement block and break statement can be enclosed with in a pair of curly braces { and }.
- The default along with its statements block is an optional one. The break statement can be placed at the end of default statements block. The default statements block can be placed at any where in the switch statement. If they placed at any other places other than at end, it is compulsory to include a break statement at the end of default statements block.
- *Next-statement* is a valid C statement.



Whenever, switch statement is encountered, first the value of <*exp*> gets matched with case values. If suitable match is found, the statements block related to that matched case gets executed. The break statement at the end transfers the control to the *Next-statement*.

If suitable match is not found, the default statements block gets executed. After then the control gets transferred to *Next-statement*.

Example Program:

```
#include<stdio.h>

void main()

{

 int a,b,c,ch;

 clrscr();

   printf("\nEnter the choice:");
```

```
 scanf("%d",&ch);
 switch(ch)
 {
  case 1: printf("\nEnter two numbers :");
        scanf("%d%d",&a,&b);
        c=a+b;
        break;
  case 2: printf("\nEnter two numbers :");
        scanf("%d%d",&a,&b);
        c=a-b;
        break;
  case 3: printf("\nEnter two numbers :");
        scanf("%d%d",&a,&b);
        c=a*b;
        break;
  case 4: printf("\nEnter two numbers :");
        scanf("%d%d",&a,&b);
        c=a/b;
        break;
  case 5:return;
 }
printf("\nThe result is: %d",c);
getch();
}
```

---

3. Write in detail about different types of loop statements in C.

While:

The simplest of all the loopiung structures in c is the while statement.The basic format of the while statement is:

Syntax:

While(condition)

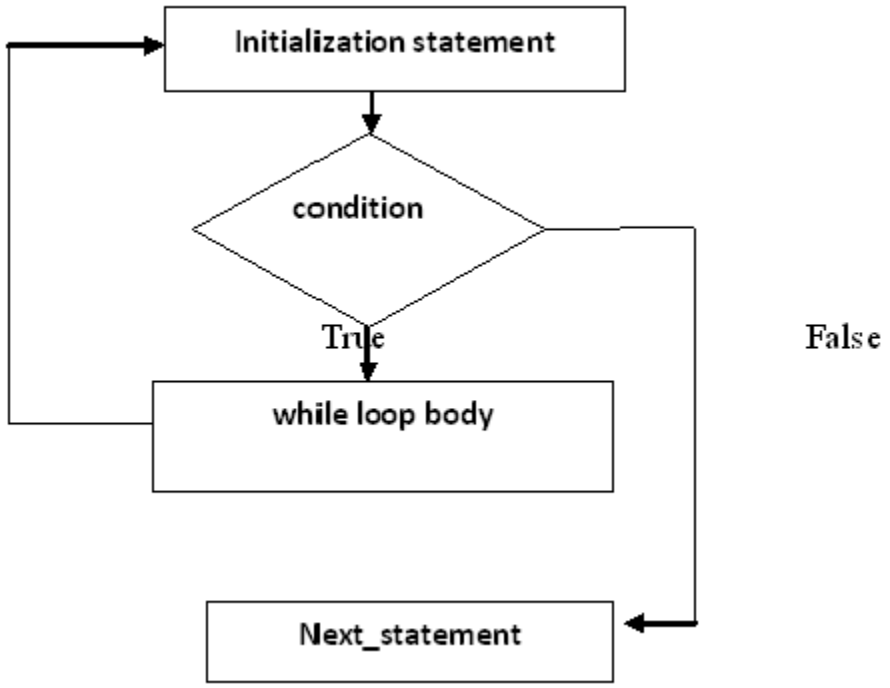
{

Statements;

}

The while is an entry –controlled loop statement.The condition is evaluated and if the condition is true then the statements will be executed.After execution of the statements the condition will be evaluated and if it is true the statements will be executed once again.This process is repeated until the condition becomes false and the control is transferred out of the loop .On exit the program continues with the statement immediately after the body of the loop.

Flow chart:



Example Program:

WHILE: PRINT N NATURAL NUMBERS

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i,n;
printf("enter the range\n");
 scanf("%d",&n);
 i=1;
 while(i<=n)
 {
```

```
printf("%d",i);
i=i+1;
}
getch();
}
```

do-while statement:

It is one of the looping control statements. It is also called as *exit-controlled looping control statement*. i.e., it tests the condition after executing the do-while loop body.

The main difference between "while" and "do-while" is that in "do-while" statement, the loop body gets executed at least once, though the condition returns the value false for the first time, which is not possible with while statement. In "while" statement, the control enters into the loop body when only the condition returns true.

Syntax:

Initialization statement;

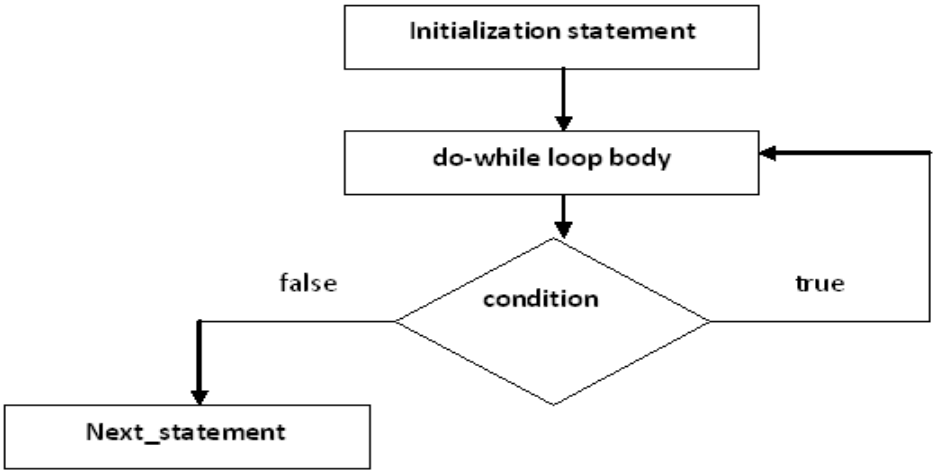
do

{ statement(s);

} while(<condition>);

next statement;

While and do are the keywords. <condition> is a relational expression or a compound relational expression or any expression that returns either true or false. initialization statement, statement(s) and next_statement are valid 'c' statements. The statements with in the curly braces are called as do-while loop body. The updating statement should be included with in the do-while loop body. There should be a semi-colon (;) at the end of while(<condition>).



Whenever "do-while" statement is encountered, the initialization statement gets executed first. After then, the control enters into do-while loop body and all the statements in that body will be executed. When the end of the body is reached, the condition is tested again with the updated loop counter value. If the condition returns the value false, the control transfers to next statement with out executing do-while loop body. Hence, it states that, the do-while loop body gets executed for the first time, though the condition returns the value false.

Example Program:

DO WHILE:

```
#include<stdio.h>

#include<conio.h>

int main()

{

int i,n;

printf("enter the range\n");

 scanf("%d",&n);

 i=1;

do

 {
```

```
printf("%d\n",i);
i=i+1;
}
while(i<=n);
getch();
}
```

for statement:

It is one of the looping control statements. It is also called as *entry-controlled looping control statement*. i.e., it tests the condition before entering into the loop body. The syntax for "for" statement is as follows:

Syntax:

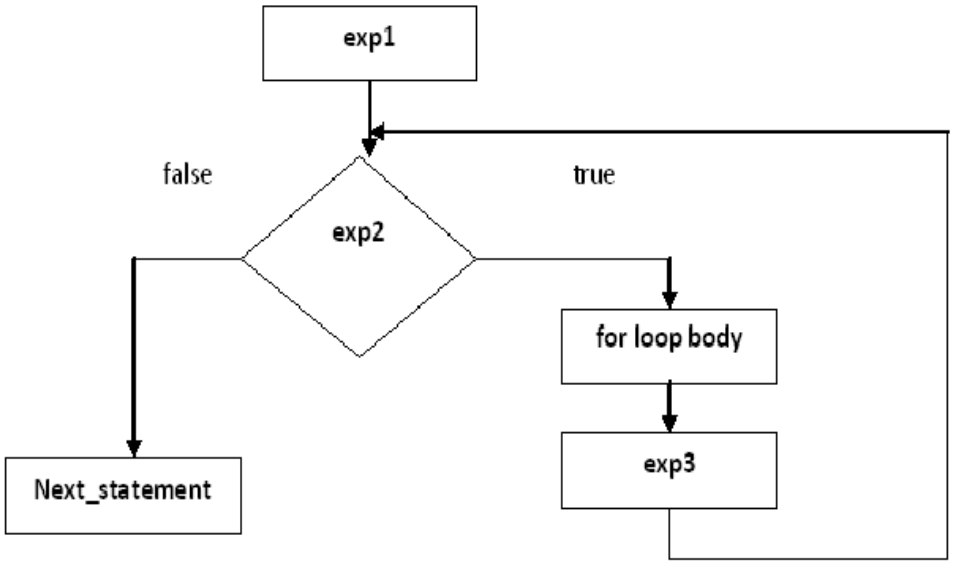
for(exp1;exp2;exp3)

for-body

next_statement;

In this syntax,

for is a keyword. exp1 is the initialization statement. If there is more than one statement, then those should be separated with commas. exp2 is the condition. It is a relational expression or a compound relational expression or any expression that returns either true or false. The exp3 is the updating statement. If there is more than one statement, then those should be separated with commas. exp1, exp2 and exp3 should be separated with two semi-colons. exp1, exp2, exp3, for-body and next_statement are valid 'c' statements. for-body is a simple statement or compound statement or a null statement.



Whenever "for" statement is encountered, first exp1 gets executed. After then, exp2 is tested.

If exp2 is true then the body of the loop will be executed otherwise loop will be terminated.

When the body of the loop is executed the control is transferred back to the for statement after evaluating last statement in the loop.now exp3 will be evaluated and the new value is again tested .if it satisfies body of the loop is executed .This process continues till condition is false.

Example Program:

FOR LOOP:

```
#include<stdio.h>

#include<conio.h>

void main()

{

int i,n;

printf("enter the value");

scanf("%d",&n);

for(i=1;i<=n;i++)

{
```

```
printf("%d\n",i);
}
getch();
}
```

4. Explain Jumping control-flow statements.

(Or)

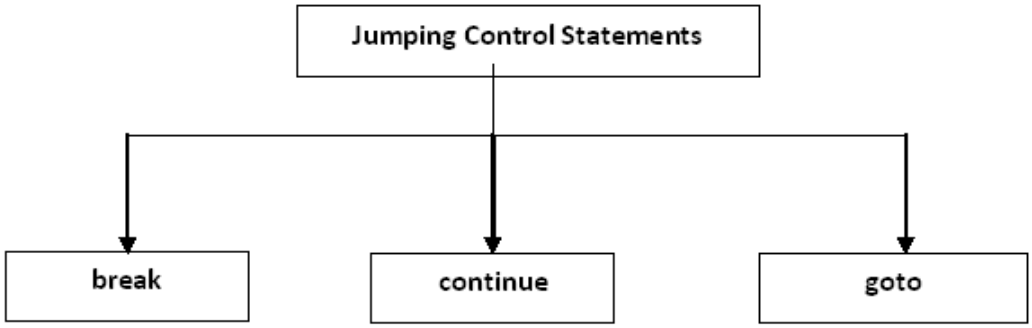
Explain the differentiate between the following with example?

a) Break statement

b) Continue statement

c) Goto statement

Jumping control-flow statements are the control-flow statements that transfer the control to the specified location or out of the loop or to the beginning of the loop. There are 3 jumping control statements:



1**. break statement**

The "break" statement is used with in the looping control statements, switch statement and nested loops. When it is used with the for, while or do-while statements, the control comes out of the corresponding loop and continues with the next statement.

When it is used in the nested loop or switch statement, the control comes out of that loop / switch statement within which it is used. But, it does not come out of the complete nesting.

The syntax for the "break" statement is:

```
break;
```

In this syntax, break is the keyword. The following diagram shows the transfer of control when break statement is used:

Any loop

{

statement_1;

statement_2;

:

break;

:

}

next_statement

Example Program:

BREAK STATEMENT:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int i;
for(i=1; i<=10; i++)
{
if(i==6)
break;
printf("%d",i);
```

}

getch();

}

2.**continue statement**

A continue statement is used within loops to end the execution of the current iteration and proceed to the next iteration. It provides a way of skipping the remaining statements in that iteration after the continue statement. It is important to note that a continue statement should be used only in loop constructs and not in selective control statements. The syntax for continue statement is:

```
continue;
```

where continue is the keyword. The following diagram shows the transfer of control when continue statement is

used:

Any loop

{

statement_1;

statement_2;

:

continue;

:

}

next_statement

Example Program:

CONTINUE STATEMENT:

#include<stdio.h>

```
#include<conio.h>
int main()
{
int i, sum=0, n;
for(i=1; i<=10; i++)
{
printf("enter any no:");
scanf("%d",&n);
if(n<0)
continue;
else
sum=sum+n;
printf("%d\n",sum);
}
getch();
}
```

3. **goto statement**

The goto statement transfers the control to the specified location unconditionally. There are certain situations where goto statement makes the program simpler. For example, if a deeply nested loop is to be exited earlier, goto may be used for breaking more than one loop at a time. In this case, a break statement will not serve the purpose because it only exits a single loop.

he syntax for goto statement is:

```
label:
        {
        statement_1;
```

statement_2;

:

}

:

goto label;

In this syntax, goto is the keyword and label is any valid identifier and should be ended with a colon (:).

The identifier following goto is a statement label and need not be declared. The name of the statement or label can also be used as a variable name in the same program if it is declared appropriately. The compiler identifies the name as a label if it appears in a goto statement and as a variable if it appears in an expression.

If the block of statements that has label appears before the goto statement, then the control has to move to backward and that goto is called as backward goto. If the block of statements that has label appears after the goto statement, then the control has to move to forward and that goto is called as forward goto.

Example Program:

PROGRAM FOR GOTO STATEMENT:

```
#include<stdio.h>
#include<conio.h>
void main()
{
clrscr();
printf("www.");
goto x;
y:
printf("expert");
```

```
goto z;

x:

printf("c programming");

goto y;

z:

printf(".com");

getch();

}
```

5. What are the differences between break statement and Continue statement?

The break statement will immediately jump to the end of the current block of code. The continue statement will skip the rest of the code in the current loop block and will return to the evaluation part of the loop. In a do or while loop, the condition will be tested and the loop will keep executing or exit as necessary. In a for loop, the counting expression (rightmost part of the for loop declaration) will be evaluated and then the condition will be tested. Take the

following example:

```
int i;
for (i=0;i<10;i++)
{

if (i==5)
continue;
printf("%d",i);
if (i==8)
break;
}
```

This code will print 1 to 8 except 5.

Continue means, whatever code that follows the continue statement WITHIN the loop code block will not be exectued and the program will go to the next iteration, in this case, when

the program reaches i=5 it checks the condition in the if statement and executes 'continue', everything after continue, which are the printf statement, the next if statement, will not be executed.

Break statement will just stop execution of the look and go to the next statement after the loop if any. In this case when i=8 the program will jump out of the loop. Meaning, it wont continue till i=9,10.

break is also used in switch-case statements to delimit the different cases.

6. What are the differences between break statement and exit ?

Break	exit()
1) It is used to come out of loop or switch or block in which it is placed.	1) It is used to come out of entire program.
2) It is a keyword. Its definition has already defined by language developers.	2) It is a pre-defined function that is available in process.h or stdlib.h header files. It takes an argument: 0 or 1. exit(0) means a clean exit without an error message. exit(1) means an abrupt exit with an error message.

7. What are the differences between while and do-while statements ?

The main difference between the while and do-while loop is in the place where the condition is to be tested.

In the while loops the condition is tested following the while statement then the body gets executed. Where as in do-while, the condition is checked at the end of the loop. The do-while loop will execute at least one time even if the condition is false initially. The do-while loop executes until the condition becomes false.

while	do-while
1) It is an entry-controlled control-flow statement. That is, it tests condition before the body gets executed.	1) It is an exit-controlled control-flow statement. That is, it tests condition after the body gets executed.

2) The body gets executed only when the condition is true.	2) The body gets executed at least once though the condition is false for the first time.

8. What is an array? How to declare and initialize arrays? Explain with examples

Array:-

An array is defined as an ordered set of similar data items. All the data items of an array are stored in consecutive memory locations in RAM. The elements of an array are of same data type and each item can be accessed using the same name.

Declaration of an array:- We know that all the variables are declared before the are used in the program. Similarly, an array must be declared before it is used. During declaration, the size of the array has to be specified. The size used during declaration of the array informs the compiler to allocate and reserve the specified memory locations.

Syntax:- data_type array_name[n];

where, n is the number of data items (or) index(or) dimension.

0 to (n-1) is the range of array.

Ex: int a[5];

float x[10];

Initialization of Arrays:-

The different types of initializing arrays:

1. At Compile time
   - (i) Initializing all specified memory locations.
   - (ii) Partial array initialization
   - (iii) Initialization without size.
   - **(iv)** String initialization.
2. At Run Time

1. Compile Time Initialization

We can initialize the elements of arrays in the same way as the ordinary variables when they are declared. The general form of initialization of arrays is

Type array-name[size]={ list of values};

(i) Initializing all specified memory locations:- Arrays can be initialized at the time of declaration when their initial values are known in advance. Array elements can be initialized with data items of type int, char etc.

Ex:- int a[5]={10,15,1,3,20};

During compilation, 5 contiguous memory locations are reserved by the compiler for the variable a and all these locations are initialized as shown in figure.

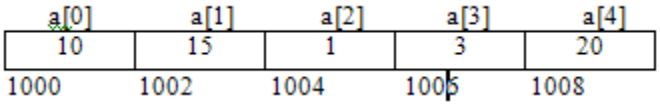


Fig: Initialization of int Arrays

Ex:-

int a[3]={9,2,4,5,6};//error: no. of initial vales are more than the size of array.

(ii) Partial array initialization:- Partial array initialization is possible in c language. If the number of values to be initialized is less than the size of the array , then the elements will be initialized to zero automatically.

Ex:-

int a[5]={10,15};

Eventhough compiler allocates 5 memory locations, using this declaration statement; the compiler initializes first two locations with 10 and 15, the next set of memory locations are automatically initialized to 0's by compiler as shown in figure.

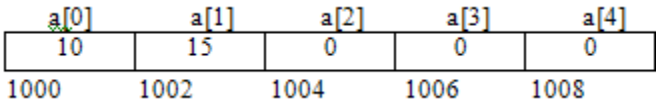
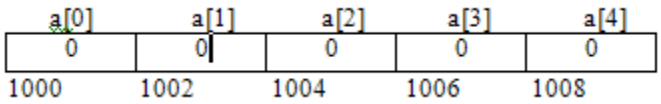


Fig: Partial Array Initialization

Initialization with all zeros:-

Ex:-

int a[5]={0};



(iii) Initialization without size:- Consider the declaration along with the initialization.

Ex:-

char b[]={'C','O','M','P','U','T','E','R'};

In this declaration, eventhough we have not specified exact number of elements to be used in array b, the array size will be set of the total number of initial values specified. So, the array size will be set to 8 automatically. the array b is initialized as shown in figure.

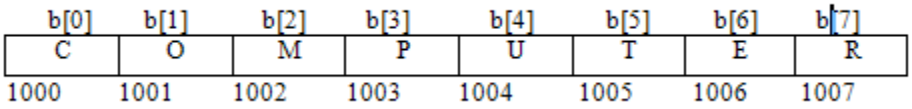


Fig: Initialization without size

Ex:- int ch[]={1,0,3,5} // array size is 4

(iv) Array initialization with a string: - Consider the declaration with string initialization.

Ex:-

char b[]="COMPUTER";

The array b is initialized as shown in figure.

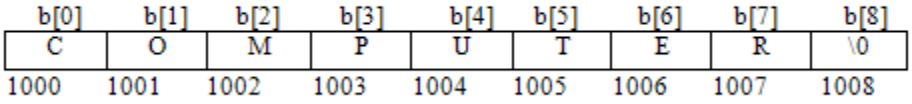


Fig: Array Initialized with a String

Eventhough the string "COMPUTER" contains 8 characters, because it is a string. It always ends with null character. So, the array size is 9 bytes (i.e., string length 1 byte for null character).

Ex:-

char b[9]="COMPUTER"; // correct

```
char b[8]="COMPUTER";          // wrong
```

2. Run Time Initialization

An array can be explicitly initialized at run time. This approach is usually applied for initializing large arrays.

Ex:- scanf can be used to initialize an array.

```
    Int x[3];
Scanf("%d%d%d",&x[0],&x[1],&x[2]);
```

The above statements will initialize array elements with the values entered through the key board.

(Or)

```
For(i=0;i<100;i=i+1)
{
  If(i<50)
Sum[i]=0.0;
Else
Sum[i]=1.0;
}
```

The first 50 elements of the array sum are initialized to 0 while the remaining 50 are initializede to 1.0 at run time.

9. How can we declare and initialize 2D arrays? Explain with examples.

Two-dimensional arrays of fixed length

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns,. These are well suited to handle the table of data. In 2-D array we can declare an array as :

Declaration:-

Syntax:

```
Data_type array_name[row_size][column_size] = {{list of first row elements},
      {list of second row elements},….
      {list of last row elements}};
```

Ex:- int arr[3][3];

where first index value shows the number of the rows and second index value shows the no. of the columns in the array. We will learn about the 2-D array in detail in the next section, but now emphasize more on how these are stored in the memory.

Initialization:-

int arr[3][3] = {{ 1, 2, 3},{4, 5, 6},{7, 8, 9}};

Where first index value shows the number of the rows and second index value shows the no. of the columns in the array.

10. Explain how two dimensional arrays can be used to represent matrices.

(or)

Define an array and how the memory is allocated for a 2D array?

These are stored in the memory as given below.

- **Row-Major order Implementation**
- **Column-Major order Implementation**

In Row-Major Implementation of the arrays, the arrays are stored in the memory in terms of the row design, i.e. first the first row of the array is stored in the memory then second and so on. Suppose we have an array named arr having 3 rows and 3 columns then it can be stored in the memory in the following manner :

int arr[3][3];

arr[0][0]	arr[0][1]	arr[0][2]
arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Thus an array of 3*3 can be declared as follows :

arr[3][3] = { 1, 2, 3,

4, 5, 6,

7, 8, 9 };

and it will be represented in the memory with row major implementation as follows :

1	2	3	4	5	6	7	8	9

In Column-Major Implementation of the arrays, the arrays are stored in the memory in the term of the column design, i.e. the first column of the array is stored in the memory then the second and so on. By taking above eg. we can show it as follows :

arr[3][3] = { 1, 2, 3,

4, 5, 6,

7, 8, 9 };

and it will be represented in the memory with column major implementation as follows :

1	4	7	2	5	8	3	6	9

Two-dimensional arrays of variable length

An array consisting of two subscripts is known as two-dimensional array. These are often known as array of the array. In two dimensional arrays the array is divided into rows and columns,. These are well suited to handle the table of data. In 2-D array we can declare an array as :

Declaration:-

Syntax:

Data_type array_name[row_size][column_size];

Initialization :-

int arr[3][3] ;

Where first index value shows the number of the rows and second index value shows the no. of the columns in the array.
These are stored in the memory as given below.

arr[0][0]	arr[0][1]	arr[0][2]

arr[1][0]	arr[1][1]	arr[1][2]
arr[2][0]	arr[2][1]	arr[2][2]

Ex:-

To initialize values for variable length arrays we can use scanf statement & loop constructs.

```
for (i=0 ; i<3; i++)
  for(j=0;j<3;j++)
    scanf("%d",&arr[i][j]);
```

11. Define multi-dimensional arrays? How to declare multi-dimensional arrays?

Multidimensional arrays are often known as array of the arrays. In multidimensional arrays the array is divided into rows and columns, mainly while considering multidimensional arrays we will be discussing mainly about two dimensional arrays and a bit about three dimensional arrays.

Syntax:

```
Data_type arrat_name[size1][size2][size3]------[sizeN];
```

In 2-D array we can declare an array as :

```
int arr[3][3] = { 1, 2, 3,
                  4, 5, 6,
                  7, 8, 9
                };
```

where first index value shows the number of the rows and second index value shows the no. of the columns in the array. To access the various elements in 2-D we can access it like this:

```
printf("%d", a[2][3]);
/* its output will be 6, as a[2][3] means third element of the second row of the array */
```

In 3-D we can declare the array in the following manner :

```
int arr[3][3][3] =

        { 1, 2, 3,
          4, 5, 6,
          7, 8, 9,

          10, 11, 12,
          13, 14, 15,
          16, 17, 18,

          19, 20, 21,
          22, 23, 24,
          25, 26, 27
        };
```

/* here we have divided array into grid for sake of convenience as in above declaration we have created 3 different grids, each have rows and columns */

If we want to access the element the in 3-D array we can do it as follows :

```
printf("%d",a[2][2][2]);
```

/* its output will be 26, as a[2][2][2] means first value in [] corresponds to the grid no. i.e. 3 and the second value in [] means third row in the corresponding grid and last [] means third column */

Ex:-

```
Int arr[3][5][12];
```

```
Float table[5][4][5][3];
```

Arr is 3D array declared to contain 180 (3*5*12) int type elements. Similarlly tabnle is a 4D array comntaining 300 elements of float type.

12. Write a program to perform matrix addition.

/*ADDITION OF TWO MATRICES*/

```
#include<stdio.h>

void main()

{int i,j;
```

```
int a[2][2]={1,2,3,4};
int b[2][2]={1,2,3,4};
int c[2][2];
clrscr();
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
printf("%2d",a[i][j]);
}
printf("\n\n");
}
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{printf("%d",b[i][j]);
}
printf("\n\n");
}
Printf("\n The addition of given two matrices is\n");
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
```

```
c[i][j]=a[i][j]+b[i][j];
}
}
for(i=0;i<2;i++)
{
for(j=0;j<2;j++)
{
Printf("%d",c[i][j]);
getch();
}
```

OUTPUT:

1 2

3 4

1 2
3 4

The addition of given two matrices is

2 4
6 8

13. Write a program to perform matrix multiplication

/*MATRIX MULTIPLICATION*/

```
#include<stdio.h>
#include<conio.h>
```

```
void main()
{
int a[3][3],b[3][3],c[3][3],i,j,k;
clrscr();
printf("\n enter the matrix elements");
for(i=0;i<3;i++);
{
for(j=0;j<3;j++);
{
scanf("%d",&a[i][j]);
}
}
printf("\n a matrix is\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",a[i][j]);
}
printf("\n");
}
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
scanf("%d\t",&b[i][j]);
}
}
printf("\n b matrix is\n");

for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",b[i][j]);
}
printf("\n");
}
```

```
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
c[i][j]=0;
for(k=0;k<3;k++)
{
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
}
}
printf("\n The multiplication of given two matrices is\n");
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
printf("%d\t",c[i][j]);

}
printf("\n");
}
getch();
}
```

OUTPUT:

enter the matrix elements

1 1 1

1 1 1

1 1 1

2 2 2

2 2 2

2 2 2

A matrix is:

1 1 1

1 1 1

1 1 1

B matrix is:

2 2 2

2 2 2

2 2 2

The multiplication of given two matrices is

6 6 6

6 6 6

6 6 6

14. Write program to find largest of the given 3 integer numbers.

Refer QNO 1 if-else statement ex program

15. Define C string? How to declare and initialize C strings with an example?

C Strings:-

In C language a string is group of characters (or) array of characters, which is terminated by delimiter \0 (null). Thus, C uses variable-length delimited strings in programs.

Declaring Strings:-

Since strings are group of characters, generally we use the structure to store these characters is a character array.

Syntax:- char string_name[size];

The size determines the number of characters in the string name.

Ex:- char city[10];

Char name[30];

Initializing strings:-

There are several methods to initialize values for string variables.

Ex:- char str[[6]="HELLO";

H	E	L	L	O	\0

Ex:- char month[]="JANUARY";

J	A	N	U	A	R	Y	\0

Ex:- char city[8]="NEWYORK";

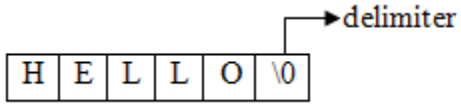
Char city[8]={'N','E','W','Y','O','R','K','\0'};

The string city size is 8 but it contains 7 characters and one character space is for NULL terminator.

Storing strings in memory:-

In C a string is stored in an array of characters and terminated by \0 (null).

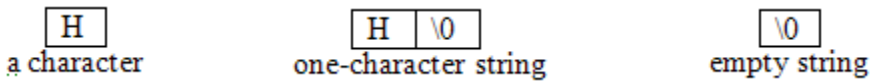
Ex:-



A string is stored in array, the name of the string is a pointer to the beginning of the string.

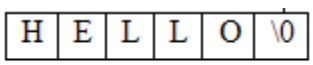
The character requires only one memory location.

If we use one-character string it requires two locations. The difference shown below,



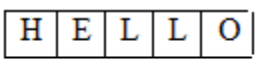
The difference between array & string shown below,

Ex:-



string

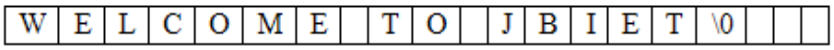
Ex:-



array

Because strings are variable-length structure, we must provide enough room for maximum-length string to store and one byte for delimiter.

Ex:- char str[20];



Strings in Arrays

Why do we need null?

A string is not a datatype but a data structure. String implementation is logical not physical. String implementation is logical not physical. The physical structure is array in which the string is stored. The string is variable-length, so we need to identify logical end of data in that physical structure.

String constant (or) Literal:-

String constant is a sequence of characters enclosed in double quotes. When string constants are used in C program, it automatically initializes a null at end of string.

Ex:- "Hello" "Welcome" "Welcome to C Lab"

16. Explain about the string Input/ Output functions with example?

Reading and Writing strings:-

C language provides several string functions for input and output.

String Input/Output Functions:-

C provides two basic methods to read and write strings. Using formatted input/output functions and using a special set of string only functions.

Reading strings from terminal:-

(a) **Using formatted input/output**:- scanf can be used with %s format specification to read a string.

Ex:- char name[10];

scanf("%s",name);

Here don't use '&' because name of string is a pointer to array. The problem with scanf, %s is that it terminates when input consisting of blank space between them.

Ex:- NEW YORK

Reads only NEW (from above example).

(b) **Special set of functions**:-

(1) **getchar()**:- It is used to read a single character from keyboard. Using this function repeatedly we may read entire line of text and stored in memory.

Ex:- char ch='z';

ch=getchar();

(2) **gets()**:- It is more convenient method of reading a string of text including blank spaces.

Ex:- char line[100];

gets(line);

Writing strings on to the screen:-

(1) **Using formatted output functions**:- printf with %s format specifier we can print strings in different formats on to screen.

Ex:- char name[10];

printf("%s",name);

Ex:- char name[10];

printf("%0.4",name);

J	A	N	U

/* If name is JANUARY prints only 4 characters ie., JANU */

Printf("%10.4s",name);

						J	A	N	U

printf("%-10.4s",name);

J	A	N	U						

(2) **Using special set of functions**:-

(a) **putchar()**:- It is used to print one by one character.

Ex:- putchar(ch);

(b) **puts()**:- It is used to print strings including blank spaces.

Ex:- char line[15]="Welcome to lab";

puts(line);

17. Explain about the following string handling functions with example programs.

(i) strlen ( ) (ii) strcpy (iii) strcmp ( ) (iv) strcat [15M]

(i) strlen( )

C supports a number of string handling functions. All of these built-in functions are aimed at performing various operations on strings. All of these built-in functions are defined in the header file **string.h**.

strlen() function: This function is used to find the length of the string excluding the NULL character. In other words, this function is used to count the number of characters in a string. Its syntax is as follows:

```
int strlen(string);
```

Example: char str1[ ] = "WELCOME";

int n;

n = strlen(str1);

/* A program to calculate length of string by using strlen() function*/

```
#include<stdio.h>
#include<string.h>
main()
{
    char string1[50];
    int length;
    printf("\n Enter any string:");
    gets(string1);
    length=strlen(string1);
    printf("\n The length of string=%d",length);
}
```

(ii) strcpy( )

This function is used to copy one string to the other. Its syntax is as follows:

```
strcpy(string1,string2);
```

where string1 and string2 are one-dimensional character arrays.

This function copies the content of string2 to string1. E.g., string1 contains master and string2 contains madam, then string1 holds madam after execution of the strcpy (string1,string2) function.

Example: char str1[ ] = "WELCOME";

char str2[ ] ="HELLO";

strcpy(str1,str2);

/* A program to copy one string to another using strcpy() function */

```
#include<stdio.h>
#include<string.h>
main( )
{
   char string1[30],string2[30];
   printf("\n Enter first string:");
   gets(string1);
   printf("\n Enter second string:");
   gets(string2);
   strcpy(string1,string2);
   printf("\n First string=%s",string1);
   printf("\n Second string=%s",string2);
}
```

(iii) **strcmp ( )**

This function compares two strings character by character (ASCII comparison) and returns one of three values {-1,0,1}. The numeric difference is '0' strings are equal otherwise if it is negative string1 is alphabeticallly above string2 or if it is positive string2 is alphabeticallly above string1.

Its syntax is as follows:

```
int strcmp(string1,string2);
```

Example: char str1[ ] = "ROM";

char str2[ ] ="RAM";

strcmp(str1,str2);

(or)

strcmp("ROM","RAM");

/* A program to compare two strings using strcmp() function */

```
#include<stdio.h>
#include<string.h>
main()
{
   char string1[30],string2[15];
   int x;
   printf("\n Enter first string:");
   gets(string1);
   printf("\n Enter second string:");
   gets(string2);
   x=strcmp(string1,string2);
   if(x==0)
```

```
    printf("\n Both strings are equal");
  else if(x>0)
    printf("\n First string is bigger");
  else
    printf("\n Second string is bigger");
}
```

(iv) strcat ( )

This function is used to concatenate two strings. i.e., it appends one string at the end of the specified string. Its syntax as follows:

```
strcat(string1,string2);
```

where string1 and string2 are one-dimensional character arrays.

This function joins two strings together. In other words, it adds the string2 to string1 and the string1 contains the final concatenated string. E.g., string1 contains prog and string2 contains ram, then string1 holds program after execution of the strcat() function.

Example:

```
char str1[10 ] = "VERY";
    char str2[ 5] ="GOOD";
strcat(str1,str2);
```

/* A program to concatenate one string with another using strcat() function*/

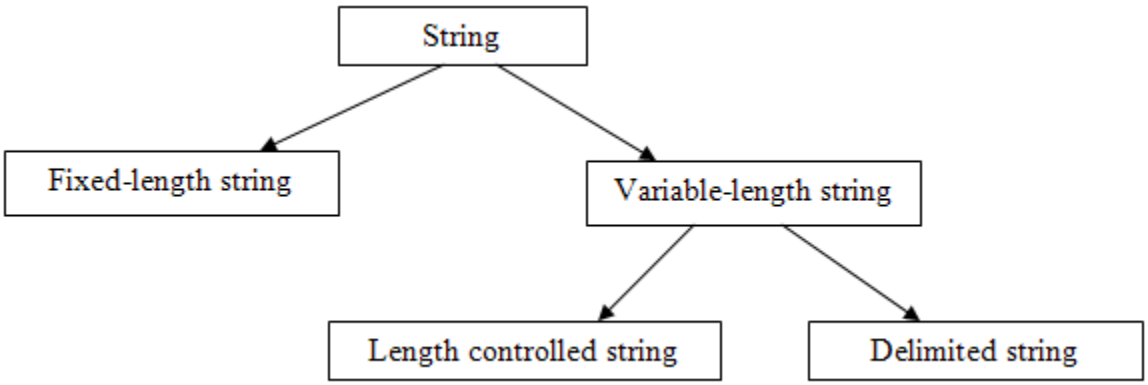
```
#include<stdio.h>
#include<string.h>
main()
{
  char string1[30],string2[15];
```

```
    printf("\n Enter first string:");
    gets(string1);
    printf("\n Enter second string:");
    gets(string2);
    strcat(string1,string2);
    printf("\n Concatenated string=%s",string1);
}
```

18. Write about storage representation of fixed and variable length format strings with example?

String concepts:-

In general a string is a series of characters (or) a group of characters. While implementation of strings, a string created in pascal differs from a string created in C language.



1. **Fixed-length strings:**
   When implementing fixed-length strings, the size of the variable is fixed. If we make it too small we can't store, if we make it too big, then waste of memory. And another problem is we can't differentiate data (characters) from non-data (spaces, null etc).

2. **Variable-length string:**
   The solution is creating strings in variable size; so that it can expand and contract to accommodate data. Two common techniques used,

(a) Length controlled strings:

These strings added a count which specifies the number of characters in the string.

Ex:-

5	H	E	L	L	O

(b) Delimited strings:

Another technique is using a delimiter at the end of strings. The most common delimiter is the ASCII null character (\0).

Ex:-

H	E	L	L	O	\0

19. How can we declare and initialize Array of strings in C? Write a program to read and display array of strings.

We have array of integers, array of floating point numbers, etc.. similarly we have array of strings also.

Collection of strings is represented using array of strings.

Declaration:-

char arr[row][col];

where,

arr - name of the array

row - represents number of strings

col - represents size of each string

Initialization:-

char arr[row][col] = { list of strings };

Example:-

char city[5][10] = { “DELHI”, “CHENNAI”, “BANGALORE”, “HYDERABAD”,
“MUMBAI” };

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

In the above storage representation memory is wasted due to the fixed length for all strings. More convenient way to initialize array of strings is as follows.

char *city[5] = { “DELHI”, “CHENNAI”, “BANGALORE”, “HYDERABAD”,
“MUMBAI” };

D	E	L	H	I	\0				
C	H	E	N	N	A	I	\0		
B	A	N	G	A	L	O	R	E	\0
H	Y	D	E	R	A	B	A	D	\0
M	U	M	B	A	I	\0			

Program to read and display array of strings

```
# include<stdio.h>

#include<conio.h>
```

```
void main( )
{
  char *city[5];
  int i;
 clrscr( );
  printf("enter 5 city names:");
  for( i=0; i<5; i++)
     scanf("%s",city[i]);
  for( i=0; i<5; i++)
    printf("city[%d] = %s",i,city[i]);
  getch( );
}
```

Output:-

Enter 5 city names: Hyderabad Mumbai Chennai Bangalore Delhi

City[0] = Hyderabad

City[1] = Mumbai

City[2] = Chennai

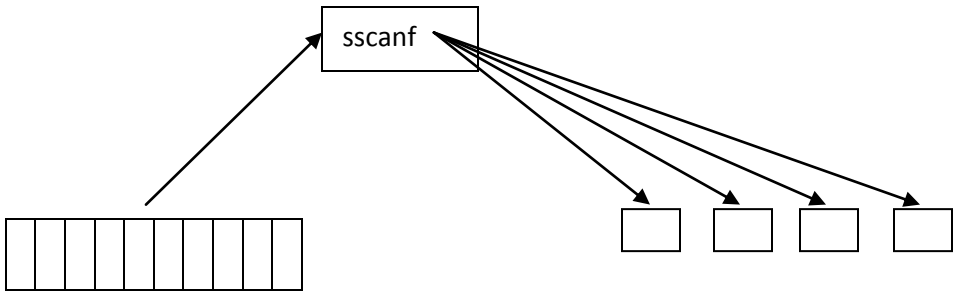
City[3] = Bangalore

City[4] = Delhi

20. Write the syntax and representations for the conversion of String to Data.

String to Data Conversion

The string scan function is called sscanf. This function scans a string as though the data were coming from a file.



String Variables

The function declaration is as follows:

Syntax:- int sscanf(char* str, "format string", variables);

where,

str - the string contains the data to be scanned. The string may be read from file or using gets function.

format string – format specifiers such as %d, %c, %f etc.

variables – list of variables in which the data will be converted

sscanf is a one-to-many function. It splits one string into many variables.

Ex: program for conversion of string to data.

```
#include<stdio.h>
 void main( )
 {
char str[ ]= " RAMU  985  98.5";
char name[10];
int tot_marks;
    float avg;
```

```
sscanf(str,"%s%d%f",name,&tot_marks,&avg);
printf("Name=%s\n",name);
printf("Total marks=%d\n",tot_marks);
printf("Average=avr%f\n",avg);
getch( );
}
```

Output:

Name = RAMU

Total marks = 985

Average = 98.5