Looping Strategies and Complex Data Types

- Looping Statements

  While, for, do while

- Jumps in Loops
- Arrays: Declaration and initialization of 1D and 2D Array
- I/O of Arrays
- Strings: Declaration and initialization of Character Array
- I/O of Strings, Operations on Strings
- String-handling Functions

Looping statements:

They are used when a statement or group of statements are required to be executed repeatedly.

They are 3 looping statements used in 'c' :

1. while loop
2. do…while loop
3. for loop

while statement (Entry Controlled Loop)

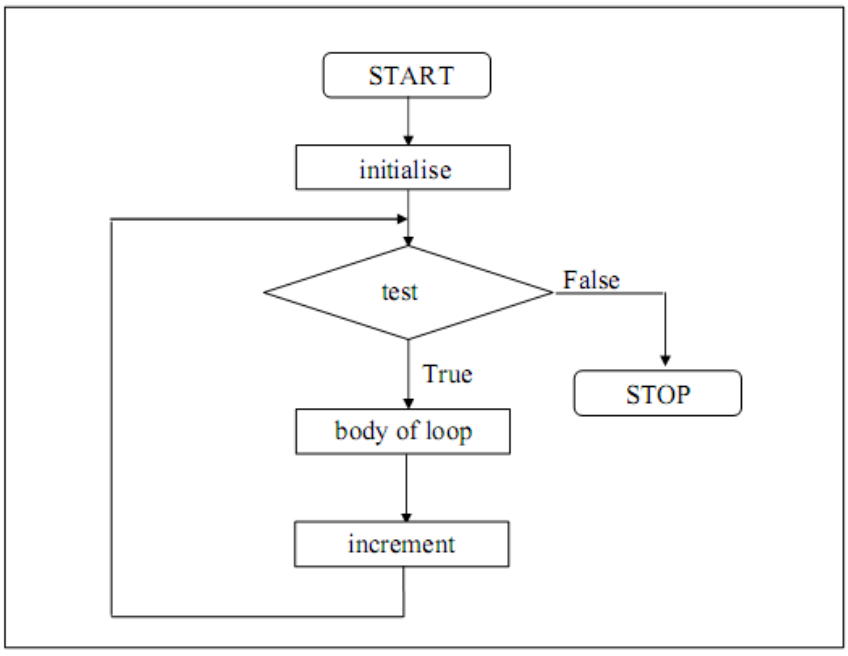
Purpose : It is used when the number of iteration to be performed are not known in advance.

It is ***Entry-controlled loop***, as test condition is checked before the loop is executed. Also known as **pre-test** loop.

Syntax:

```
while ( test condition )
{
        body of the loop
}
```

Flow chart of While loop:



Description:

- Initially test condition is evaluated.
- If the condition is **true** then the body of loop is executed and the execution continues as long as it remains true.
- If the condition is **false** then loop is terminated and control is transferred out of the loop and next-statement after the body of the loop is executed.
- Body of loop may consist of one or more statements.
- Braces are needed only if loop body consists to two or more statements.
- Body of the loop may not be executed at all if the condition is not satisfied at the very first attempt.

Example: Write a C program to display 1 to N number using while loop.

```
void main()
{
        int i ;
        i=1 ;
        while ( i<=3 )
        {
                printf("\n i= %d", i);
                i++;
        }
}
```

Output

```
i=1
i=2
i=3
```

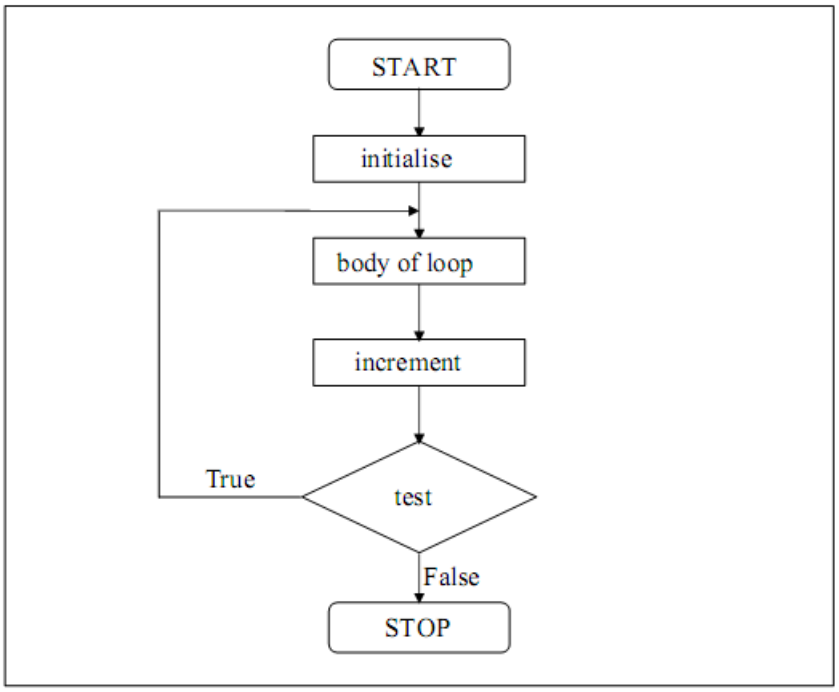
do... while statement (Exit – Controlled Loop)

Purpose : Is used to execute a statement or block of statements at least once even if the condition is false initially.

It is ***Exit-controlled loop***, as test condition is checked at the bottom of the loop, thus body of the loop is executed at least once.Also known as **post-test** loop.

Syntax:

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



Description:

- First body of loop is executed and then condition is evaluated.
- At the end of the loop , the test-condition in the while statement is evaluated.
- If the condition is **true** then the body of loop is executed and the execution continues as long as it remains true.

- If the condition is **false** then loop is terminated and control is transferred out of the loop and next-statement after the body of the loop is executed.

Example: Write a C program to display 1 to N number using do…while loop.

```
void main()
{
        int  i ;
        i=1 ;
        do
        {
                printf("\n i=%d",i);
                i++;
        }while ( i<=3 ) ;
        getch();
}
```

Output

Enter any positive number :3

i=1

i=2

i=3

Differentiate between Entry and Exit Controlled Loop

	Entry Controlled Loop	**Exit Controlled Loop**
1	While loop is known as entry controlled loop	Do...While is known as exit controlled loop
2	Also known as pre-test loop.	Also known as post-test loop.
3	It is ***Entry-controlled loop***, as test condition is checked before the loop is executed.	It is as ***Exit-controlled loop***, as test condition is checked at the bottomof the loop, thus body of the loop is executed at least once.

4	**Syntax:**
	while (test -condition)
	{
	body of the loop
	};
		Syntax:
		do
		{
		body of the loop
		} while (test - condition) ;

5	**Example :**
	i=1;
	while (i<=3)
	{
	printf("\n %d",i);
	i++;
	};
	Output:
	1
	2
	3
		Example :
		i=1;
		do
		{
		printf("\n %d",i);
		i++;
		}while(i<=3);
		Output:
		1
		2
		3

for loop statement

The for loop is useful while executing a statement a fixed number of times.

Syntax:

```
for(initialization ; test -condition ; updation)
{
        body of the loop
}
```

Description:

- The *initialization* of the control variable is done first.
- If the *test condition* is true, the body of loop is executed.
- The control variable is (*updated*) incremented or decremented.
- Once again the control variable is tested against the test condition and the whole process is repeated as long as the control variable fails to satisfy the test condition.

Example: Write a C program to display 1 to N number using for loop.

```
void main()
{
       int n, i ;
       clrscr();

       for( i=1 ; i<=3 ; i++)
              printf("\n i=%d",i);
       getch();
}
```

Output

```
Enter any positive number :3
i=1
i=2
i=3
```

break statement

Syntax:

break ;

Description: A break statement terminates the execution of loop and the control is transferred to the statement immediately following the loop.

It exits only from a single loop containing it.

It is also used with switch case statement, break denotes end of particular case and transfers control out of switch statement.

Example: Following C program display sum positive integer numbers.

```
#include<stdio.h>
#include<conio.h>
void main()
{
int  a;

        for (i=0;i<=5;i++)
        {
                if ( i==3)
                {
                        break;
                        printf("i = %d",i);
                 }
}
```

Output:

i =1

i =2

Continue statement

Syntax:

```
continue ;
```

Description: A continue statement skips the statements after it and transfers the control back to the loop statement.

Continue statement will skip only a single loop containing it.

Example: Following C program display sum positive integer numbers.

```
void main()
{
int  a;

        for (i=0;i<=5;i++)
        {
                if ( i==3)
                {
                        continue;
                        printf("i = %d",i);
                 }
}
```

Output:

i =1

i =2

i = 4

i = 5

Differentiate between Break and Continue

	Break	**continue**
1	A break statement terminates the execution of loop and the control is transferred to the statement immediately following the loop.	A continue statement skips the statements after it and transfers the control back to the loop statement.
2	Syntax: break ;	Syntax: continue ;
3	Break statement will exit only a single loop containing it.	Continue statement will skip only a single loop containing it.
4	Break statement is also used with switch case statement, break denotes end of particular case and transfers control out of switch statement.	Continue statement is not used in switch statement.

5	Example
	void main()
	{
	int a;
	for (i=0;i<=5;i++)
	{
	if (i==3)
	{
	break;
	printf("i=%d",i);
	}
	getch();
	}
	Output:
	i =1
	i =2
		Example
		void main()
		{
		int a;
		for (i=0;i<=5;i++)
		{
		if (i==3)
		{
		continue;
		printf("i = %d",i);
		}
		getch();
		}
		Output:
		i =1
		i =2
		i = 4
		i = 5

Methods of Structured Programming

Large program are difficult to understand. Hence the large programs are written using three logic structures:

1. Sequence
2. Selection
3. Iteration

A **sequence** construct is a set of instructions serially performed without any looping. The normal flow of control is from top to bottom and from left to right unless there is an iteration construct. Thus the first instruction in the program is executed first, then control passes to the next instruction that sequence.

A **selection** construct involves branching but the branches join before the exit point. The selection is required and the flow of control depends on the selection. For example, if statement, if…else statement and switch statement in C language.

An **iteration** construct contains loops in it but has only one exit point. This construct allows repeated execution of a set of codes without repeating codes in the program. It saves the trouble of writing the same codes again and again in the program. For example, while loop, do..while loop and for loop in C language.

Array :

Definition: An array is a fixed –size sequenced collection of elements of same data type.

Types of Array:

1. One-dimensional Array
2. Two-dimensional Array

[1]. One-dimensional Array

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

Array declaration or Array definition :

The syntax for defining a one-dimensional array is

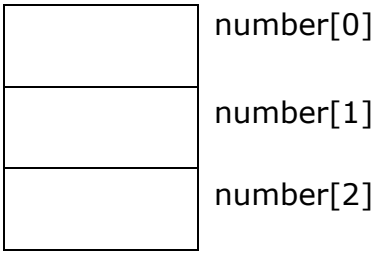
Datatype ArrayName [ArraySize] ;

Here,

datatype : any valid data type of C language such as int, char, float etc. ArrayName : variable name of an array.

The ArraySize indicates the maximum number of elements the array can hold and it is an integer value.

Eg:



The complete set of values is referred to as an ***array*** and the individual values are called **elements**.

Accessing array elements:

Once a one-dimensional array variable is defined, its elements can be accessed by using an index value.

Syntax : ArrayName [index]

The minimum value of index is ***zero*** and the maximum value of index is ***size – 1.***

A **index number** (or **subscript)** in brackets after the array name.

The individual elements can be assigned as follows:

number[0]=35

number[1]=56

number[2]=67

This will cause array to store as follows:

number[0]	35
number[1]	56
number[2]	67

These elements can be used just like other 'c' variables.

x=number[0];

number[4]=number[0]+number[2];

Example: The following are some valid array definition statements.

```
int rollno[50] ;            // integer array of size 50
char   name[25] ;           // character array of size 25
int height[10], group[10] ;          // defines two integer array a and b
```

Example : Consider an integer array **rollno** of ten elements.

```
int rollno[10];
```

To display a value of **first** element of the array **rollno** :

```
printf("%d", rollno[0]);
```

To display a value of **fifth** element of the array **rollno**.

```
printf("%d", rollno[4]);
```

To display all the elements of the array **rollno**.

```
for(int i=0 ; i<10 ; i++)
    printf("%d",rollno[i]);
```

One Dimensional Array initialization (Compile time):

The one-dimensional array can be initialized at the point of their declaration as follow.

```
Datatype    ArrayName[ size ] = { List of values separated by comma } ;
```

There are following different ways to initialize one-dimensional array.

Case-1: If the number of values in the list is same as the size of an array.

```
int rollno[ 5 ] = { 341, 342, 343, 344, 345 } ;
```

In above declaration, size of an array is 5. In this case, the first element of array rollno is initialized with 341, second with 342 and so on.

Case-2: If the number of values in the list is less than the size of an array.

```
int rollno [ 5 ] = { 341, 342, 343 } ;
```

This statement will initialize the first three elements are 341, 342 and 343 and remaining two elements to zero.

Case-3: The array size may be omitted.

```
int counter[  ] = { 1,1,1,1,1} ;
```

In such cases, the compiler assumes the array size to be equal to the number of elements enclosed within the curly braces. Hence, in above statement, the size of an array is 5.

Case-4: Initialize using static keyword.

```
static int    a[50] ;
```

This statement initialized all elements of an array to zero.

Initialization of character array:

Case 1 : char name[ ] = {'K', 'r', 'i', 's', 'h', 'n' , 'a', '\0' } ;

Case 2 : char name[ ] = "Krishna";

Case 1 and case 2 : This statement declares the name array of eight characters, initialized with string "Krishna" ending with the null character.

If the number of values in the list is less than the size of an array.

char city [5] = { 'A', 'B'} ;

This statement will initialize the first two elements are initialized to 'A' and 'B' and remaining three elements to **NULL**.

Following statements are invalid and illegal in 'C':

If we have more initializers than the declared size,

int num [4] = {1,2,3,4,5} ;

String:-**A sequence of characters is called string**. It can be used for storing and manipulating text such as words, names and sentences. A string is an array of characters.

Declaring string variable:

The general form: char StringName [size] ;

StringName : valid variable name

size : indicates the maximum number of elements the string can hold and it is integer value.

Initializing string variable:

The string is initialized when they are declared as follow.

char name[ ] = {'K', 'r', 'i', 's', 'h', 'n' , 'a', '\0' } ;

or

char name[9] = "Krishna";

or

char name[ ] = "Krishna";

When the compiler assigns a character string to a character array, it automatically appends a null character ('\0') at the end of the string.

One Dimensional Array initialization (Run time):

In one dimensional Array elements can be initialized at run time using two methods.

1. **Using Assignment statement**

Example 1 :

```
int a[10];
for (i=0;i<10;i++)
{
        if (i<5)
                a[i]=0;
        else
                a[i]=1;
}
```

First five elements will be initialized with 0 and
remaining five will be initialized with 1

2. **Using scanf statement**

Example 2 :

```
int a[5];
for(i=0;i<5;i++)
{
        printf("Enter element a[%d] :",i);
        scanf("%d",&a[i]);
}
```

Example of one dimensional array

Write a program to read and print elements of one dimensional array

```
void main()
{

        int a[5],n, i ;
        printf ("Enter total elements in an array: ");
        scanf("%d",&n);

        for(i=0; i<n ; i++)
        {
                printf("Enter  element  %d : " , i+1);
                scanf("%d", &a[i]);
        }
```

```
        for(i=0; i<n ; i++)
                printf("Element  %d : %d : " , i+1, a[i]);
        getch();
}
```

Two-Dimensional Arrays

There could be situations where a table of values will have to be stored.
We can think of table as a matrix consisting of rows and columns.

Declaration of two dimensional array:

datatype arrayname[row_size][column_size];

Example:

- **Declaration of integer type 2-dim array:**

int arrayname[row_size] [column_size];

- **Declaration of float type 2-dim array:**

float arrayname[row_size] [column_size];

- **Declaration of character type 2-dim array:**

char arrayname[row_size] [column_size];

Accessing 2 dimensional array elements

- Array index ranges from zero to maximum size of array minus one.
- the first index indicates the row and the second index indicates the column within that row.

	Column 0	Column 1	Column 2
Row 0	[0][0]	[0][1]	[0][2]
Row 1	[1][0]	[1][1]	[1][2]
Row 2	[2][0]	[2][1]	[2][2]

Fig : index of 3 * 3 array

12	45	56
54	43	78
10	0	8

Fig : representation of values in memory of 3 * 3 array

Initializing Two-Dimensional Arrays (Compile Time)

➔ Two-dimensional array of **integer type** is initialized by following declaration:

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

➔ Two-dimensional array of **float type** is initialized by following declaration:

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

Here initializes the elements of the first row to zero and the second row to one. The initiazation is done row by row.

➔ The above statement can be equivalently written as following by surrounding the elements of each row by braces:

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

➔ It is also initialized in form of a matrix as shown below:

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

                };
```

➔ Another form:

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

➔ If the values are missing in an initialize, they are automatically set to zero.

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

                };
```

It will initialize the first two elements of the first row to one, the first element of the second row to two, and all other elements to zero.

➔ When all the elements are to be initialized to zero, the following short-cut may be used:

```
int table[3][5]={{0},{0},{0}};
```

Note: same way instead of integer type you can declare and initialize float and character type 2 dimensional array with float and char keyword.

Two Dimensional Array initialization (Run time):

In two dimensional array elements can be initialized at run time.

Syntax : **arrayname[row index][column index]=value;**

Example :

1. Using Assignment statement

```
int  a[3][3] , i , j;
for (i=0;i<3;i++)
{
        for (j=0;j<3;j++)
        {
                if (i==j)
                        a[i][j]=1;
                else
                        a[i][j]=0;
                printf("%d", a[i][j]);
        }
        printf("\n");
}
```

Diagonal elements will be initialized with 0 and remaining elements will be initialized with 1.

Output:

1 0 0
0 1 0
0 0 1

Example 2 : Using scanf statement (Run time initialization)

```
int  a[3][3] , i , j;
for (i=0 ; i<3 ; i++)
{
        for (j=0 ; j<3 ; j++)
        {
                printf("enter element a[%d][%d] : ",i,j);
                scanf("%d",&a[i][j]);
        }

}
```

Write a program to read and print elements of two dimensional array

```
void main()
{

        int a[5],n, i ,j,r,c;
        printf (“Enter rows in an array: “);
        scanf(“%d”, &r);

        printf (“Enter columns in an array: “);
        scanf(“%d”, &c);

        for(i=0 ; i<r ; i++)
        {
                for(j=0; j<c ; j++)
                {
                        printf(“Enter  element  a[%d] [%d] : ” , i  , j);
                        scanf(“%d”, & a[i][j]);
                }
        }


        for(i=0 ; i<r ; i++)
        {
                for(j=0; j<c ; j++)
                {
                        printf(“%d %d ” ,a[i][j]);
                 }
                printf(“\n”);
        }

getch();
}
```

What is String:-

A sequence of characters is called string. It can be used for storing and manipulating text such as words, names and sentences. A string is an array of characters.

How to declare string :

The general form:

char StringName [size] ;

StringName : valid variable name

size : indicates the maximum number of elements the string can hold and it is integer value.

How to initialize string :

The string is initialized when they are declared as follow.

```
char    name[ ] = {'K', 'r', 'i', 's', 'h', 'n' , 'a', '\0' } ;
```

or

```
char    name[9] = "Krishna";
```

or

```
char    name[ ] = "Krishna";
```

When the compiler assigns a character string to a character array, it automatically appends a null character ('\0') at the end of the string.

String handling functions:

To use these functions, the header file string.h must be included in the program using statement :

```
#include<string.h>
```

1. strlen – calculates length
2. strcmp – compares 2 strings
3. strcpy – copy content of one string to another
4. strcat – joins 2 strings
5. strrev – reverse a string
6. gets –reads a string
7. puts –writes a string

[1]. String Length:

Syntax	-	strlen(string);
Header file	-	string.h
Return Value	-	Integer
Description	-	Counts and returns number of characters in a string.

Example	-	char city[20];
		printf("Enter your city…");
		scanf("%s", city);
		len = strlen(city) ;
		printf("\n Length = %d", len);
		
		Output:
		Enter your city…Anand
		Length = 5

[2]. String Comparision:

Syntax	-	strcmp(s1, s2) Where s1 and s2 are string.
Header file	-	string.h
Return Value	-	Integer
Description	-	It is used to compares two string. It returns 0 (zero) value if both s1 equal to s2. Otherwise, returns ascii difference between first non-matching character.

Example	-	char s1[20], s2[20];
		int ans;
		printf("Enter string s1…");
		scanf("%s",s1);
		printf("Enter string s2…");
		scanf("%s",s2);
		ans = **strcmp**(s1,s2) ;
		if (ans = = 0)
		printf("\n Both strings are equal");
		if (ans > 0)
		printf("\n String1 is greater");
		if (ans < 0)
		printf("\n String2 is greater");
		<u>Output</u>:
		Enter string s1… book
		Enter string s2… book
		Both strings are equal

[3]. String Copy:

Syntax	-	strcpy(s1, s2) Where s1, s2 both are string.
Header file	-	string.h
Description	-	Copies the contents of s2 to s1. s2 may be character array or a string constant. s1 should be large enough to receive content of s2.

Example	-	char s1[20], s2[20];
		printf("Enter string…");
		scanf("%s",s2);
		strcpy(s1,s2) ;
		printf("\n Original string = %s & Copied string = %s", s2,s1);
		
		Output:
		Enter string… Hello
		Original string = Hello & Copied string = Hello

[4]. String Concatenation:

Syntax	-	strcat(s1, s2) Where s1, s2 both are string.
Header file	-	string.h
Description	-	Joins(concates) two strings together.s2 is appended to s1. It does by removing null character from s1 and placing s2 there.

Example	-	printf("Enter string s1…");
		scanf("%s",s1);
		printf("Enter string s2…");
		scanf("%s",s2);
		strcat(s1,s2) ;

		printf("\n Resultant string = %s ", s1); Output: Enter string s1… Hello Enter string s2… World Resultant string = HelloWorld

[5]. String reverse:

Syntax	-	strrev(s1) //Where s1 is string.
Header file	-	string.h
Description	-	It is used to reverse the contents of a string.

Example	-	char s1[20];
		printf("Enter string…");
		scanf("%s",s1);
		printf("\n Original string = %s", s1);
		strrev(s1) ;
		printf("\n Reverse string = %s", s1);
		Output:
		Enter string… Hello
		Original string = Hello
		Reverse string = olleH

[6]. gets:

Syntax	-	gets(s1) //where s1 is string
Header file	-	stdio.h
Description	-	It is used to read a string containing white spaces. It reads characters from the keyboard until new line character is encountered and then appends a null character to the string.

Example	-	printf("Enter string…");
		
		gets(s1);
		
		printf("\n string = %s", s1);
		
		Output:
		
		Enter string… Hello World
		
		string = Hello World

[6]. puts:

Syntax	-	puts(s1) //Where s1 is string
Header file	-	stdio.h
Description	-	It is used to print the value of the string and moves the cursor to the beginning of next line.

Example	-	printf("Enter string…");
		
		gets(s1);
		
		puts(s1);
		
		Output:
		
		Enter string… Hello World
		
		Hello World