UNIT – 2 CONTROL STATEMENTS, ARRAYS

SELECTION STATEMENTS:(Branching Statements)

A statement is a an instruction that causes an action to be performed when executed. The „C" statements ends with a semicolon(;).

A *selection statement* is a control statement that allows choosing between two or more execution paths in a program. Most modern programming languages include one-way, two-way and n-way (multiple) selectors.

a) An **if statement** with no else is a one-way selector
b) An **if-else statement** is a two-way selector
c) A **switch or case statement** is an n-way selector

a) IF- Statement

It is the basic form where the if statement evaluate a test condition and direct program execution depending on the result of that evaluation.

Syntax:

```
If (Expression)
{
Statement 1;
Statement 2;
}
```

Where a statement may consist of a single statement, a compound statement or nothing as an empty statement. The Expression also referred so as test condition must be enclosed in parenthesis, which cause the expression to be evaluated first, if it evaluate to true (a non zero value), then the statement associated with it will be executed otherwise ignored and the control will pass to the next statement.

Example:

```
if (a>b)
{
printf ("a is larger than b");
}
```

b)IF-ELSE Statement:

An if statement may also optionally contain a second statement, the **``else clause,''** which is to be executed if the condition is not met.

Syntax if(condition)

```
{
statement_block_1;
}
else
{
statement_block_2;
}
```

<u>Program:</u>

/* C program to check if a number entered by user is even or odd using if statement*/

```
#include<stdio.h>
void main()
```

```
{
int a;
printf("Enter a number\n");
scanf("%d",&a);
 if(a%2==0)
printf("The entered number is even\n");

else
printf("The entered number is odd\n");
 getch();
}
```

Output: Enter a number 4The entered number is even

c)NESTED-IF- Statement:

It is always legal in C programming to nest if-else statements, which means you can use one if or else if statement inside another if or else if statement(s).

Syntax:

```
if( boolean_expression 1)
{
   expression 1 is true
   { statement 1;
   }
    else if(boolean_expression 2)
   {   statement 2;
}
   Else
   {    statement3;
   }
```

Example:-

```
#include <stdio.h>
void main ()
{
int a = 100;
int b = 200;
if( a == 100 )
{
if( b == 200 )
{
printf("Value of a is 100 and b is 200\n" );
} }
printf("Exact value of a is : %d\n", a );
printf("Exact value of b is : %d\n", b );
getch();
}
```

4. Switch Case

This is another form of the multi way decision. It is used to execute the code from multiple conditions.

Syntax:

```
switch(expression)
{
 case value1:
//code to be executed;
break;
case value2:
//code to be executed;
break;
......
......
default:
code to be executed if all cases are not matched;
}
```

Program:

```
#include<stdio.h>
void main()
{
int number=0;
printf("enter a number:");
scanf("%d",&number);
 switch(number)
{
case 10: printf("number is equals to 10");
            break;
case 50: printf("number is equal to 50");
            break;
case 100:printf("number is equal to 100");
             break;
default: printf("number is not equal to 10, 50 or 100");
}
getch();
}
```

Looping Statements:-

The Iterative statements or Looping statements are statements that are executed repeatedly until a condition is satisfied. The *loops in C language* are used to execute a block of code or a part of the program several times. In other words, it iterates (repeats) a code or group of code many times.

Types of C Loops

There are three types of loops in C language that is given below:

a) do-while Loop
b) while Loop
c) for Loop

a)DO WHILE LOOP

To execute a part of program or code several times, we can use do-while loop of C language. The code given between the do and while block will be executed until condition is true.In do while loop, statement is given before the condition, so statement or code will be ***executed at least one time***. In other words, we can say it is executed 1 or more times. It is better if you have to execute the code at least once.

The general syntax of a do `while` loop is

```
do
{
//code to be executed
}
while(condition);
```

Program:

There is given the simple program of c language do while loop where we are printing the table of 1.

Example:

```
#include <stdio.h>
void main()
{
int i=1;
do
{
printf("Hello");
i++;
}
while(i<=10);
getch();
}
```

Semicolon is compulsory in do while loop

b)WHILE LOOP:

The while loop in C language is used to iterate the part of program or statements many times.In while loop, condition is given before the statement. So it is different from the do while loop. It can execute the statements 0 or more times. When use while loop in C. The C language while loop should be used if number of iteration is uncertain or unknown.

Syntax:

```
while(condition)
{
//code to be executed
      }
```

Example:

```
#include <stdio.h>
 void main()
{
int i=1;

while(i<=10)
{
printf("Hello");
 i++;
}
getch();
}
```

c)FOR LOOP

The for loop in C language is also used to iterate the statement or a part of the program several times, like while and do-while loop. But, we can initialize and increment or decrement the variable also at the time of checking the condition in for loop. Unlike do while loop, the condition or expression in for loop is given before the statement, so it may execute the statement 0 or more times.

Syntax of for loop :-

```
for(initialization;condition;incr/decr)
{
//code to be executed
}
```

<u>Example of for loop ;</u>

```
#include <stdio.h>
void main()
{
int i;
clrscr();
for(i=1;i<=10;i++)
{
 printf("Hello”);
}
 getch();
}
```

<u>OTHER CONTROL STATEMENTS</u>

1. BREAK STATEMENT

The break statement in C language is used to break the execution of loop (while, do while and for) and switch case. „break‟ is a keyword used to terminate the loop or to exit from switch statement. In case of inner loops, it terminates the control of inner loop only. There can be two usage of C break keyword:

1. With switch case
2. With loop

Syntax:
```
jump-statement;
break;
```

The jump statement in c break syntax can be while loop, do while loop, for loop or switch case.

Example of C break statement with loop

```
#include <stdio.h>
void main()
{
int i=1;
//initializing a local variable
clrscr();
//starting a loop from 1 to 10
for(i=1;i<=10;i++)
{
```

```
printf("%d \n",i);
if(i==5)
{                    //if value of i is equal to 5, it will break the loop
break;
}
}
getch();
}
```

2 . CONTINUE STATEMENT

The continue statement in C language is used to continue the execution of loop (while, do while and for). It is used with if condition within the loop.In case of inner loops, it continues the control of inner loop only.

Syntax: jump-statement;
continue;

The jump statement can be while, do while and for loop.

Example of continue statement in c

```
#include <stdio.h>
void main()
{
int i=1;                    //initializing a local variable
clrscr();                   //starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5)
{                           //if value of i is equal to 5, it will continue the loop
continue;
}
printf("%d \n",i);
}                           //end of for loop
getch();
}
```

3. GOTO STATEMENT

A ***goto*** statement in C programming language provides an unconditional jump from the goto to a labeled statement in the same function.

NOTE: Use of goto statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify.

Syntax: goto label;

```
...
...
label:    statement;
```

Here label can be any plain text except C keyword and it can be set anywhere in the C program above or below to goto statement.

Program:

```
include <stdio.h>
void main()
```

```
{
int num;
clrscr();
 printf("www.");
goto x;
y: printf(".ac");
goto z;
x:printf("MLRITM");
goto y;
 z:printf(".in");
 getch();
}
```

Output:

```
www.MLRITM.ac.in
```

I/O: Simple input and output with scanf and printf, formatted I/O, stdin, stdout,stderr:

When we say **Input**, it means to feed some data into a program. An input can be given in the form of a file or from the command line. C programming provides a set of built-in functions to read the given input and feed it to the program as per requirement.

When we say **Output**, it means to display some data on screen, printer, or in any file. C programming provides a set of built-in functions to output the data on the computer screen as well as to save it in text or binary files.

The Standard Files

C programming treats all the devices as files. So devices such as the display are addressed in the same way as files and the following three files are automatically opened when a program executes to provide access to the keyboard and screen.

Standard File	**File Pointer**	**Device**
Standard input	stdin	Keyboard
Standard output	stdout	Screen
Standard error	stderr	Your screen

The file pointers are the means to access the file for reading and writing purpose. This section explains how to read values from the screen and how to print the result on the screen.

Formatted I/O : The scanf() and printf() Functions

The **int scanf(const char *format, ...)** function reads the input from the standard input stream **stdin** and scans that input according to the **format** provided.

The **int printf(const char *format, ...)** function writes the output to the standard output stream **stdout** and produces the output according to the format provided.

The **format** can be a simple constant string, but you can specify %s, %d, %c, %f, etc., to print or read strings, integer, character or float respectively. There are many other formatting options available which can be used based on requirements. Let us now proceed with a simple example to understand the concepts better −

```
#include <stdio.h>
int main( ) {

  char str[100];
  int i;

  printf( "Enter a value :");
  scanf("%s %d", str, &i);

  printf( "\nYou entered: %s %d ", str, i);

  return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then program proceeds and reads the input and displays it as follows −

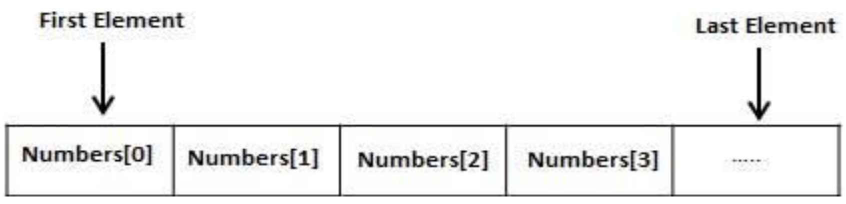
```
$./a.out
Enter a value : seven 7
You entered: seven 7
```

Arrays:

Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.

All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays

To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows −

type arrayName [arraySize];

This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type double, use this statement −

double balance[10];

Here *balance* is a variable array which is sufficient to hold up to 10 double numbers.

Initializing Arrays

You can initialize an array in C either one by one or using a single statement as follows −
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

i) <u>**One dimensional array:**</u>

One dimensional array is a linear list consisting of related and similar data items. In memory all the data items are stored in contiguous memory locations one after the other.
DECLARING ONE DIMENSIONAL ARRAYS:
To declare regular variables we just specify a data type and a unique name: **int number;**
To declare an array, we just add an array size.
For example: **int temp[5];** Creates an array of **5** integer elements.
For example: **double stockprice[31];** creates an array of **31** doubles.

SYNTAX FOR DECLARING ONE DIMENSIONAL ARRAYS
elementType arrayName [size];
where : **elementType:** specifies data type of each element in the array.
arrayName: specifies name of the variable you are declaring.
size: specifies number of elements allocated for this array.

Example:-
C PROGRAM TO READ AN ARRAY ELEMENTS AND FIND THE SUM OF THE ELEMENTS.
```
#include <stdio.h>
void main()
{
 int a[10];
int i, SIZE,
total=0;
printf("\n Enter the size of the array : ");
scanf("%d",&size);
for (i = 0; i < SIZE ; i++)
scanf("%d",&a[i]);
for (i = 0; i < SIZE ; i++)
total += a[i];
printf("Total of array element values is %d\n", total);
getch();
}
```

<u>2) TWO DIMENSIONAL ARRAYS</u>:

An array of arrays is called a two-dimensional array and can be regarded as a table with a number of rows and columns .

DECLARATION OF TWO DIMENSIONAL ARRAY:

We can declare an array in the c language in the following way.

dataType arrayName [rows][cols];

<u>Example</u>, **char names[3][4];**

In the above declaration

char(elementType) specifies type of element in each slot

names(arrayName) specifies name of the array

[3](rows) specifies number of rows

[4](cols) specifies number of columns

Ex;-An array may be initialized at the time of declaration:

```
char names [3][4] = { {„J‟, 'o', 'h', 'n'},
                      {„M‟, 'a', 'r', 'y'},
                       {„I‟, 'v', 'a', 'n'} };
```

//C Program to read and print the elements of the two dimensional array.

```
#include <stdio.h>
#define M   3 /* Number of rows */
#define N   4 /* Number of columns */
int main(void)
{
        int a [ M ] [ N ], i, j, sum = 0;
        for ( i = 0; i < M; ++i )  /* fill the array */
            for (j = 0; j < N, ++j )
                scanf (%d”, &a [i] [j]);
        for ( i = 0; i < M; ++i )
       {         /* print array values */
            for (j = 0; j < N, ++j )
                printf(“a [ %d ] [ %d ] = %d “, i, j, a[ i ] [ j ]);
                printf (“\n”);
        }
        for ( i = 0; i < M; ++i ) /* sum the array */
         for (j = 0; j < N, ++j )
                sum += a[ i ] [ j ]; printf(“\nsum =
        %d\n\n”); return 0;
                }
                }
```