PROGRAMMING IN C

UNIT-2

Iteration:

While loop:

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax:

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

Do while loop:

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

Syntax:

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

For loop:

The for loop in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

Syntax:

```
for(Expression 1; Expression 2; Expression 3){
//code to be executed
}
```

Infinitive for loop:

To make a for loop infinite, we need not give any expression in the syntax. Instead of that, we need to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.

```
#include<stdio.h>
void main ()
{
   for(;;)
   {
      printf("welcome to javatpoint");
   }
}
```

Nested Loops:

C supports nesting of loops in C. Nesting of loops is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.

Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define 'while' loop inside a 'for' loop.

Syntax:

```
Outer_loop
{
   Inner_loop
  {
       // inner loop statements.
  }
     // outer loop statements.
}
```

Break statement:

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

Syntax:

```
//loop or switch case

Break;
```

Example:

```
#include<stdio.h>

#include<stdlib.h>

void main ()

{

   int i;

   for(i = 0; i<10; i++)

   {

      printf("%d ",i);

      if(i == 5)

      break;

   }

   printf("came outside of loop i = %d",i);

}
```

Continue statement:

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

For the for loop, continue statement causes the conditional test and increment portions of the loop to execute. For the while and do...while loops, continue statement causes the program control to pass to the conditional tests.

Syntax:

```
continue;
```

Example:

```
#include <stdio.h>

int main () {

   /* local variable definition */
   int a = 10;

   /* do loop execution */
   do {

      if( a == 15) {
         /* skip the iteration */
         a = a + 1;
         continue;
      }

      printf("value of a: %d\n", a);
      a++;

   } while( a < 20 );

   return 0;
}
```

OUTPUT:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

Array:

An array is defined as the collection of similar type of data items stored at contiguous memory locations.

Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

Properties of Array:

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantage of C Array:

1. **Code Optimization**: Less code to the access the data.
2. **Ease of traversing**: By using the for loop, we can retrieve the elements of an array easily.
3. **Ease of sorting**: To sort the elements of the array, we need a few lines of code only.
4. **Random Access**: We can access any element randomly using the array.

Disadvantage of C Array:

1. **Fixed Size**: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaration of C Array:

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

data_type array_name[array_size];

Now, let us see the example to declare the array.

int marks[5];

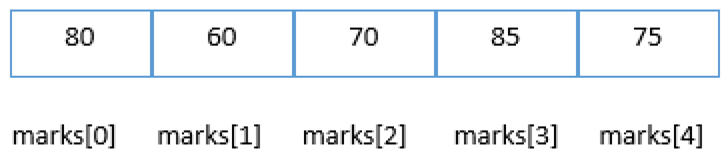
Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array:

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

marks[0]=80;

```
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



Initialization of Array

Array example:

```
#include<stdio.h>
int main(){
int i=0;
int marks[5];//declaration of array
marks[0]=80;//initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}//end of for loop
return 0;
}
```

Output:

```
80
60
```

```
70
85
75
```

Two-Dimensional Array:

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

Declaration of two dimensional Array:

The syntax to declare the 2D array is given below.

data_type array_name[rows][columns];

Consider the following example.

int twodimen[4][3];

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array:

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

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

Two-dimensional array example:

```
#include<stdio.h>

int main(){

int i=0,j=0;

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

for(i=0;i<4;i++){

 for(j=0;j<3;j++){

   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);

 }

}
```

return 0;

}

Output:

```
arr[0][0]                                   =                                   1
arr[0][1]                                   =                                   2
arr[0][2]                                   =                                   3
arr[1][0]                                   =                                   2
arr[1][1]                                   =                                   3
arr[1][2]                                   =                                   4
arr[2][0]                                   =                                   3
arr[2][1]                                   =                                   4
arr[2][2]                                   =                                   5
arr[3][0]                                   =                                   4
arr[3][1]                                   =                                   5
arr[3][2] = 6
```

Strings:

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

Let's see the example of declaring string by char array in C language.

char ch[9]={'C', 'o', 'd', 'e', 'c', 'h', 'a', 'm', 'p','\0'};

As we know, array index starts from 0, so it will be represented as in the figure given below.

While declaring string, size is not mandatory. So we can write the above code as given below:

char ch[]={'C', 'o', 'd', 'e', 'c', 'h', 'a', 'm', 'p', '\0'};

We can also define the string by the string literal in C language. For example:

char ch[]="Codechamp";

In such case, '\0' will be appended at the end of the string by the compiler.

Difference between char array and string literal:

- We need to add the null character '\0' at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.
- The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

String Example in C

```
#include<stdio.h>

#include <string.h>

int main(){

  char ch[10]={'C', 'o', 'd', 'e', 'c', 'h', 'a', 'm', 'p',  '\0'};

   char ch2[10]="Codechamp";

   printf("Char Array Value is: %s\n", ch);

   printf("String Literal Value is: %s\n", ch2);

 return 0;

}
```

Output:

```
Char Array Value is: Codechamp
String Literal Value is: Codechamp
```

Traversing String:

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.

Hence, there are two ways to traverse a string.

1. By using the length of string
2. By using the null character.

String-handling functions:

1. strlen() function:

The strlen() function returns the length of the given string. It doesn't count null character '\0'.

```
#include<stdio.h>

#include <string.h>

int main(){

char ch[20]={'C', 'o', 'd', 'e', 'c', 'h', 'a', 'm', 'p', '\0'};

  printf("Length of string is: %d",strlen(ch));

 return 0;

}
```

Output:

```
Length of string is: 10
```

2. strcpy() function:

The strcpy(destination, source) function copies the source string in destination.

```
#include<stdio.h>

#include <string.h>

int main(){

 char ch[20]={'C', 'o', 'd', 'e', 'c', 'h', 'a', 'm', 'p', '\0'};

  char ch2[20];

  strcpy(ch2,ch);

  printf("Value of second string is: %s",ch2);

 return 0;

}
```

Output:

```
Value of second string is: Codechamp
```

3. strcat() function:

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

```
#include<stdio.h>

#include <string.h>

int main(){

  char ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};

   char ch2[10]={"Codechamp", '\0'};

   strcat(ch,ch2);

   printf("Value of first string is: %s",ch);

 return 0;

}
```

Output:

```
Value of first string is: helloCodechamp
```

4. strcmp() function:

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using *gets()* function which reads string from the console.

```
#include<stdio.h>

#include <string.h>

int main(){

  char str1[20],str2[20];

  printf("Enter 1st string: ");

  gets(str1);//reads string from console

  printf("Enter 2nd string: ");

  gets(str2);

  if(strcmp(str1,str2)==0)

      printf("Strings are equal");

  else
```

```
    printf("Strings are not equal");
 return 0;
}
```

Output:

```
Enter                1st                 string:                Codechamp
Enter                2nd                 string:                Codechamp
Strings are equal
```