UNIT-3

Arrays, Functions, Storage classes, Strings

C 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.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

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

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.

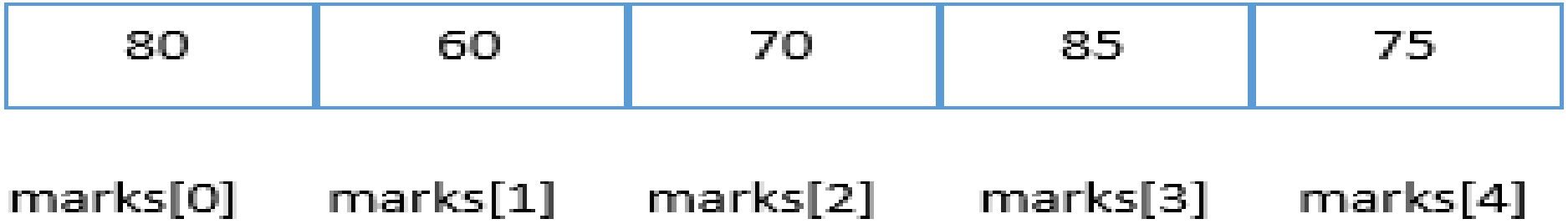
```
int marks[5];
```

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.

```
1.marks[0]=80;//initialization of array
2.marks[1]=60;
3.marks[2]=70;
4.marks[3]=85;
5.marks[4]=75;
```



Initialization of Array

C Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60};
```

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Example of 1-D array

```
#include<stdio.h>
int main(){
int i;
int marks[5]={20,30,40,50,60};//declaration and init
ialization of array
 //traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
return 0;
}
```

Output:

```
20
30
40
50
60
```

C Array: Reading values into an array and displaying array contents

Example of 1-D array

```
#include<stdio.h>
int main(){
int i;
int marks[5];//declaration and initialization of array
printf(“enter the elements of an array\n”);
for(i=0;i<5;i++)
{
scanf(“%d”, &marks[i]);  //reading values
}
printf(“The elements of an array are\n”);
for(i=0;i<5;i++)
{
printf("%d \n",marks[i]);  //displaying values
}
return 0;
}
```

Output:

```
enter the elements of an array
20
30
40
50
60
The elements of an array are
20
30
40
50
60
```

Two Dimensional Array in C

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 in C

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

```
data_type  array_name[rows][columns];
```

```
int twodimen[4][3];
```

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

Initialization of 2D Array in C

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.

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

Two-dimensional array example in C

```
#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}};
//traversing 2D array
for(i=0;i<4;i++){
 for(j=0;j<3;j++){
   printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
 }//end of j
}//end of i
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
```

C Functions

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure* or *subroutine* in other programming languages.

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

Function Aspects

There are three aspects of a C function.

•**Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

•**Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

•**Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

SN	C function aspects	Syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

The syntax of creating function in c language is given below:

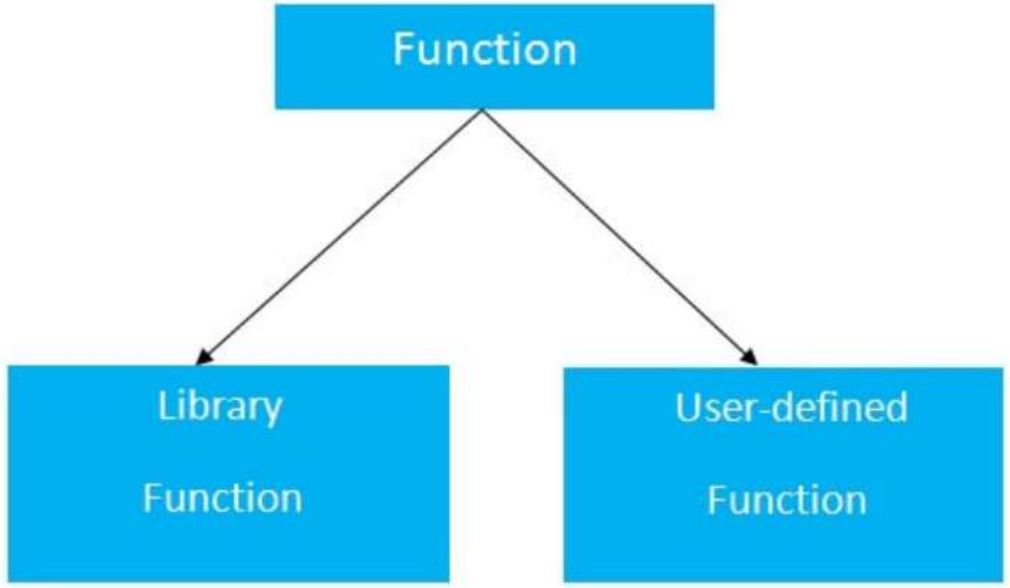
```
return_type function_name(data_type parameter...)
{
//code to be executed
}
```

Types of Functions

There are two types of functions in C programming:

1.Library Functions: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.

2.User-defined functions: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.



Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

Example without return value:

```
void hello(){
printf("hello c");
}
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

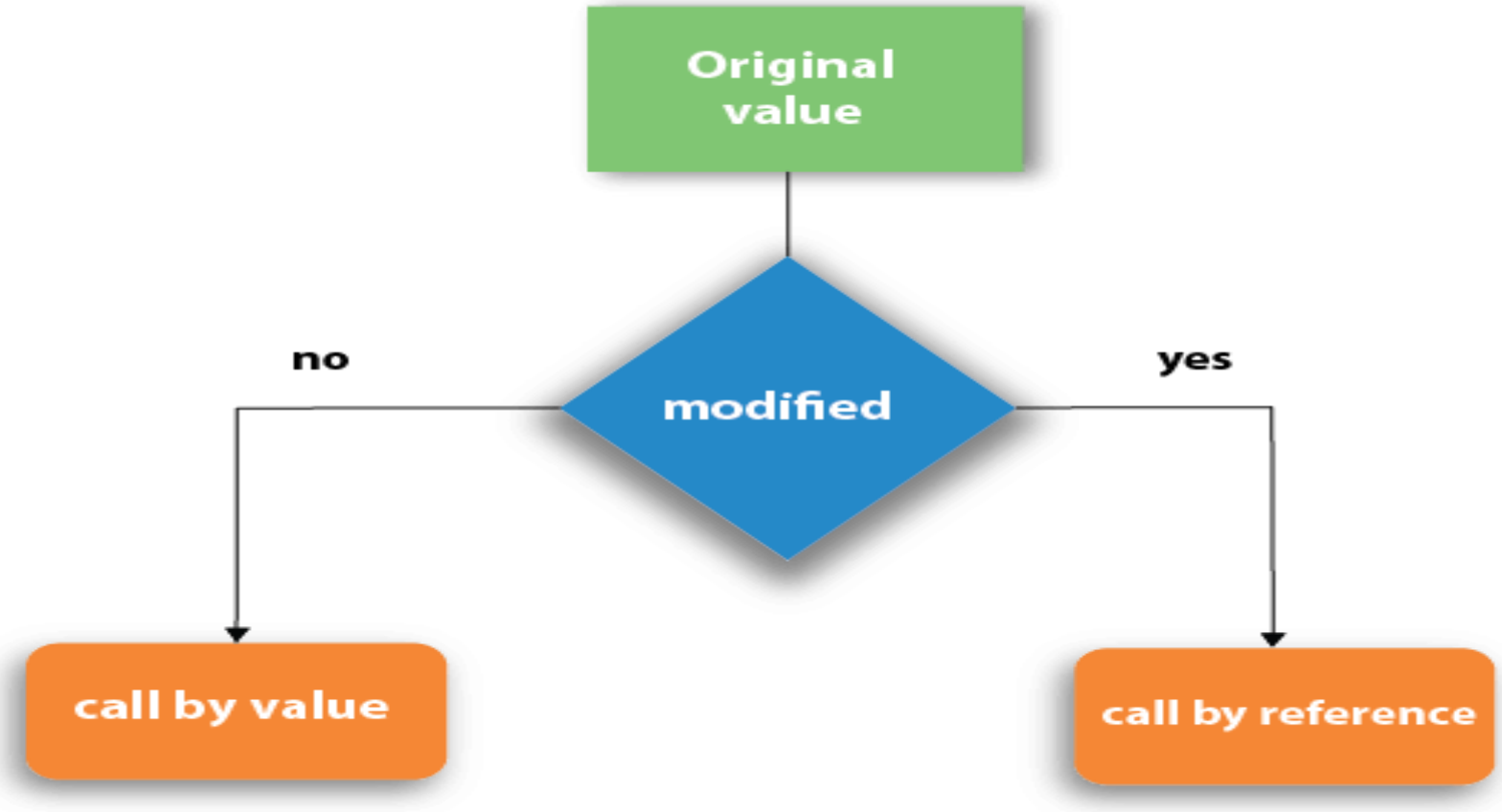
Let's see a simple example of C function that returns int value from the function.

Example with return value:

```
int get(){
return 10;
}
```

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., *call by value* and *call by reference*.



Call by value in C

•In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.

•In call by value method, we can not modify the value of the actual parameter by the formal parameter.

•In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.

•The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

Swapping the values of the two variables

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
    swap(a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameters do not change by changing the formal parameters in call by value, a = 10, b = 20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

Call by reference in C

•In call by reference, the address of the variable is passed into the function call as the actual parameter.

•The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.

•In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Swapping the values of the two variables using call by reference

```
#include <stdio.h>
void swap(int *, int *); //prototype of the function
int main()
{
   int a = 10;
   int b = 20;
   printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a and b in main
   swap(&a,&b);
   printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual parameters do change in call by r
eference, a = 10, b = 20
}
void swap (int *a, int *b)
{
   int temp;
   temp = *a;
   *a=*b;
   *b=temp;
   printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal parameters, a = 20, b = 10
}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

Difference between call by value and call by reference in c

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.
In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```

Output

```
Enter the number whose factorial you want to calculate?5
factorial = 120
```

We can understand the above program of the recursive method call by the figure given below:

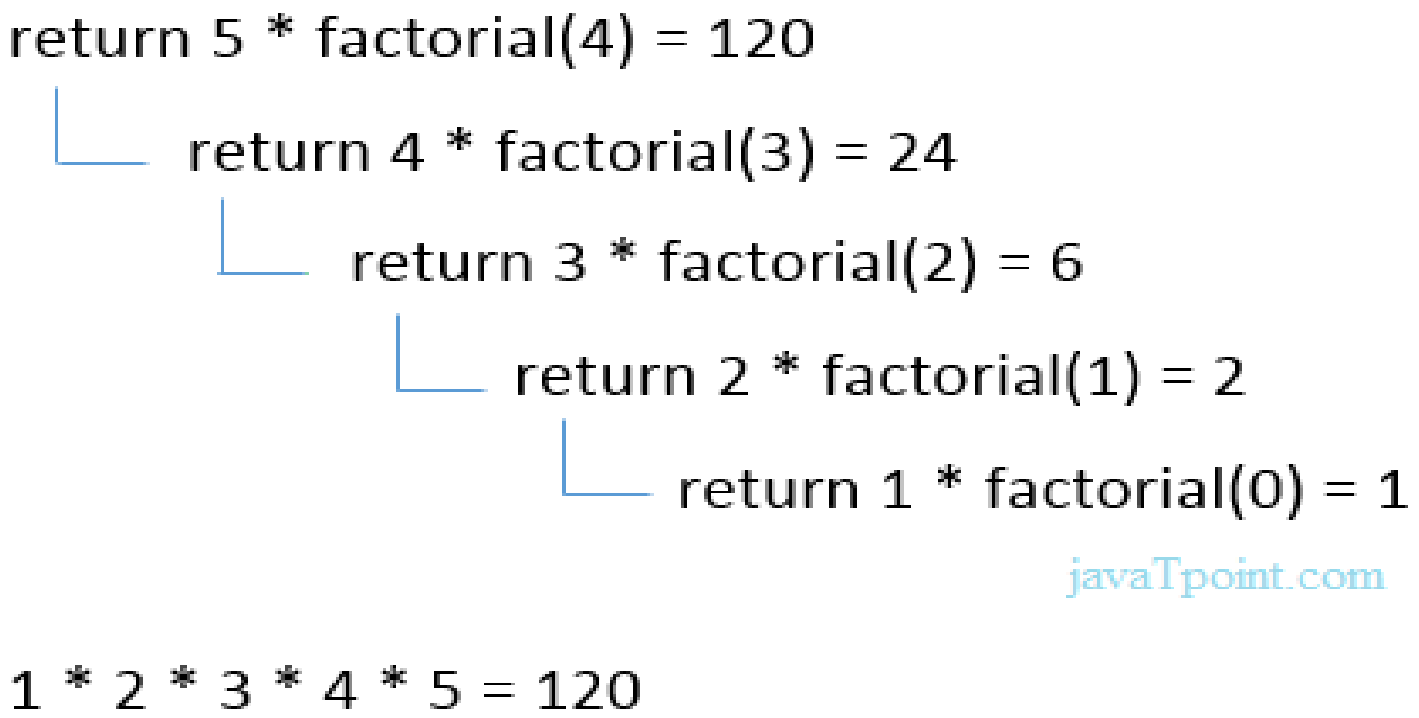


Fig: Recursion

Storage Classes in C

Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C

- Automatic
- External
- Static
- Register

Storage Classes	Storage Place	Default Value	Scope	Lifetime
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of the main program Maybe declared anywhere in the program
static	RAM	Zero	Local	Till the end of the main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within the function

Automatic

•Automatic variables are allocated memory automatically at runtime.

•The visibility of the automatic variables is limited to the block in which they are defined.

•The scope of the automatic variables is limited to the block in which they are defined. The automatic variables are initialized to garbage by default.

•The memory assigned to automatic variables gets freed upon exiting from the block.

•The keyword used for defining automatic variables is auto.

•Every local variable is automatic in C by default.

Example 1

```
#include <stdio.h>
int main()
{
int a; //auto
char b;
float c;
printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.
return 0;
}
```

Output:

garbage garbage garbage

Example 2

```
#include <stdio.h>
int main()
{
int a = 10,i;
printf("%d ",++a);
{
int a = 20;
for (i=0;i<3;i++)
{
printf("%d ",a); // 20 will be printed 3 times since it is the local value of a
}
}
printf("%d ",a); // 11 will be printed since the scope of a = 20 is ended.
}
```

Output:

11 20 20 20 11

Static

- The variables defined as static specifier can hold their value between the multiple function calls.
- Static local variables are visible only to the function or the block in which they are defined.
- A same static variable can be declared many times but can be assigned at only one time.
- Default initial value of the static integral variable is 0 otherwise null.
- The visibility of the static global variable is limited to the file in which it has declared.
- The keyword used to define static variable is static.

Example 1

```
#include<stdio.h>
static char c;
static int i;
static float f;
static char s[100];
void main ()
{
printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.
}
```

Output:

```
0 0 0.000000 (null)
```

Example 2

```
#include<stdio.h>
void sum()
{
static int a = 10;
static int b = 24;
printf("%d %d \n",a,b);
a++;
b++;
}
void main()
{
int i;
for(i = 0; i< 3; i++)
{
sum(); // The static variables holds their value between multiple function calls.
}
}
```

Output:

10 24 11 25 12 26

Register

•The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.

•We can not dereference the register variables, i.e., we can not use &operator for the register variable.

•The access time of the register variables is faster than the automatic variables.

•The initial default value of the register local variables is 0.

•The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler’s choice whether or not; the variables can be stored in the register.

•We can store pointers into the register, i.e., a register can store the address of a variable.

Example 1

```
#include <stdio.h>
int main()
{
register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.
printf("%d",a);
}
```

Output:

0

Example 2

```
#include <stdio.h>
int main()
{
register int a = 0;
printf("%u",&a); // This will give a compile time error since we can not access the address of a register variable.
}
```

Output:

error: address of register variable ?a? requested

External

•The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.

•The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.

•The default initial value of external integral type is 0 otherwise null.

•We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.

•An external variable can be declared many times but can be initialized at only once.

•If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.

Example 1

```
1.#include <stdio.h>
2.int main()
3.{
4.extern int a;
5.printf("%d",a);
6.}
```

Output

undefined reference to `a’

Example 2

```
#include <stdio.h>
int a;
int main()
{
extern int a = 0; // this will show a compiler error since we can not use extern and initializer at same time
printf("%d",a);
}
```

Output

compile time error

Example 3

```
#include <stdio.h>
int main()
{
extern int a; // Compiler will search here for a varia
ble a defined and initialized somewhere in the pogr
am or not.
printf("%d",a);
}
int a = 20;
```

Output

20

C 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.

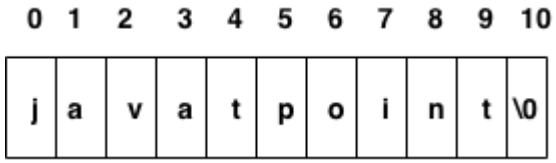
1.By char array

2.By string literal

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

char ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\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[]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

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

```
char ch[]="javatpoint";
```

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

Difference between char array and string literal

There are two main differences between char array and 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.

C gets() and puts() functions

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

C gets() function

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Reading string using gets()

```
#include<stdio.h>
void main ()
{
    char s[30];
    printf("Enter the string? ");
    gets(s);
    printf(“\nYou entered %s",s);
}
```

Output

```
Enter the string? javatpoint is the best
 You entered javatpoint is the best
```

C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

Let's see an example to read a string using gets() and print it on the console using puts().

```
#include<stdio.h>
#include <string.h>
int main(){
char name[50];
printf("Enter your name: ");
gets(name); //reads string from user
printf("Your name is: ");
puts(name);  //displays string
return 0;
}
```

Output:

```
Enter your name: Sorry
Your name is: Sorry
```

C String Functions

There are many important string functions defined in "string.h" library.

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.

C String Length: 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]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
   printf("Length of string is: %d",strlen(ch));
 return 0;
}
```

Output:

Length of string is: 10

C Copy String: strcpy()

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

```
#include<stdio.h>
#include <string.h>
int main(){
 char ch[20]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
   char ch2[20];
   strcpy(ch2,ch);
   printf("Value of second string is: %s",ch2);
 return 0;
}
```

Output:

```
Value of second string is: javatpoint
```

C String Concatenation: strcat()

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]={'c', '\0'};
   strcat(ch,ch2);
   printf("Value of first string is: %s",ch);
 return 0;
}
```

Output:

```
Value of first string is: helloc
```

C Compare String: strcmp()

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: hello
Enter 2nd string: hello
Strings are equal
```

C Reverse String: strrev()

The strrev(string) function returns reverse of the given string. Let's see a simple example of strrev() function.

```
#include<stdio.h>
#include <string.h>
int main(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nReverse String is: %s",strrev(str));
 return 0;
}
```

Output:

```
Enter string: javatpoint
String is: javatpoint
Reverse String is: tnioptavaj
```