**21CSS101J Programming and Problem Solving (PPS)**

**UNIT 3**

**String Basics - String Declaration and Initialization**

**String Basics:**

1. An array of characters is commonly called as strings.
2. They are used by programmers to manipulate texts or sentences.
3. In real time, we use a lot of applications which processes words and sentences to find pattern in user data, replace it , delete it , modify it etc.
4. They are widely used in spell checkers, spam filters, intrusion detection system, search engines, plagiarism detection, bioinformatics, digital forensics and information retrieval systems etc.

**String Declaration:**

A C String is a simple array with char as a data type. [‘C’ language](https://www.guru99.com/c-programming-tutorial.html) does not directly support string as a data type. Hence, to display a String in C, you need to make use of a character array.

The general syntax for declaring a variable as a String in C is as follows,

char string\_variable\_name [array\_size];

The classic Declaration of strings can be done as follow:

char string\_name[string\_length] = "string";

The size of an array must be defined while declaring a C String variable because it is used to calculate how many characters are going to be stored inside the string variable in C. Some valid examples of string declaration are as follows,

char first\_name[15]; //declaration of a string variable

char last\_name[15];

The above example represents string variables with an array size of 15. This means that the given C string array is capable of holding 15 characters at most. The indexing of array begins from 0 hence it will store characters from a 0-14 position. The C compiler automatically adds a NULL character ‘\0’ to the character array created.

**String Initialization:**

Following example demonstrates the initialization of Strings in C,

char first\_name[15] = "ANTHONY";

char first\_name[15] = {'A','N','T','H','O','N','Y','\0'}; // NULL character '\0' is required at end in this declaration

char string1 [6] = "hello";/\* string size = 'h'+'e'+'l'+'l'+'o'+"NULL" = 6 \*/

char string2 [ ] = "world"; /\* string size = 'w'+'o'+'r'+'l'+'d'+"NULL" = 6 \*/

char string3[6] = {'h', 'e', 'l', 'l', 'o', '\0'} ; /\*Declaration as set of characters ,Size 6\*/

In string3, the NULL character must be added explicitly, and the characters are enclosed in single quotation marks.

**String Functions: gets(), puts(), getchar(),putchar(), printf()**

**getchar() & putchar() functions**

The getchar and putchar functions are used for taking character input from the user and printing the character as output.

**The getchar() function**

* The getchar() function reads a character from the terminal and returns it as an **integer**.
* This function reads only a single character at a time.

Here is the syntax for the getchar() function:

int getchar(void);

**The putchar() function**

* The putchar() function displays the character passed to it on the screen and returns the same character.
* This function too displays only a single character at a time.

Here is the syntax for the putchar() function:

int putchar(int character);

#include <stdio.h>

void main( )

{

int c;

printf("Enter a character");

/\*

Take a character as input and

store it in variable c

\*/

c = getchar();

/\*

display the character stored

in variable c

\*/

putchar(c);

}

Output:

Enter a character: Studytonight

S

**gets() & puts() functions**

The gets and puts functions are used for taking string input and giving string output.

**The gets() function**

The gets() function reads a line of text from **stdin**(standard input) into the buffer pointed to by str [pointer](https://www.studytonight.com/c/pointers-in-c.php), until either a terminating newline or EOF (end of file) occurs.

Here is the syntax for the gets() function:

char\* gets(char\* str);

**The puts() function**

The puts() function writes the string str with a newline character ('\n') at the end to **stdout**. On success, a non-negative value is returned.

Here is the syntax for the gets() function:

int puts(const char\* str);

#include <stdio.h>

void main()

{

/\* character array of length 100 \*/

char str[100];

printf("Enter a string: ");

gets(str);

puts(str);

getch();

return 0;

}

Output:

Enter a string: Studytonight

Studytonight

**Printf()**

printf function is used to print output on the screen.  This function is a part of the C standard library “**stdio.h**” and it can allow formatting the output in numerous ways.

**Syntax of printf:**

*printf(” format String”, Arguments);*

**Here,**

* **Format String:**It is a string that specifies the output. It may also contain a format specifier to print the value of any variable such as character and an integer value.
* **Arguments:**These are the variable names corresponding to the format specifier.

**Built-inString Functions: atoi, strlen, strcat, strcmp**

### strcpy()

This is the string copy function. It copies one string into another string.

**Syntax:**

strcpy(string1, string2);

The two parameters to the function, string1 and string2, are strings. The function will copy the string string1 into the string 1.

### strcat()

This is the string concatenate function. It concatenates strings.

**Syntax:**

strcat(string1, string2);

The two parameters to the function, string1 and string2 are the strings to be concatenated. The above function will concatenate the string string2 to the end of the string string1.

### strlen()

This is the string length function. It returns the length of the string passed to it as the argument.

**Syntax:**

strnlen(string1)

The parameter string1 is the name of the string whose length is to be determined. The above function will return the length of the string string1.

### strcmp()

This is the string compare function. It is used for string comparison.

**Syntax:**

strcmp(string1, string2);

The above function will return 0 if strings string1 and string2 are similar, less than 0 if string1<string2 and greater than 0 if string1>string2.

### Example:

The following example demonstrates how to use the above string functions:

#include <iostream>

#include <cstring>

using namespace std;

int main() {

char name1[10] = "Guru99";

char name2[10] = "John";

char name3[10];

int len;

strcpy(name3, name1);

cout << "strcpy( name3, name1) : " << name3 << endl;

strcat(name1, name2);

cout << "strcat( name1, name2): " << name1 << endl;

len = strlen(name1);

cout << "strlen(name1) : " << len << endl;

return 0;

}

**Output:**

[String Functions in C++](https://www.guru99.com/images/2/030720_0458_CStringsst5.png)

**Atoi():**

The C library function **int atoi(const char \*str)** converts the string argument **str** to an integer (type int).

## Declaration

Following is the declaration for atoi() function.

int atoi(const char \*str)

## Parameters

* **str** − This is the string representation of an integral number.

## Return Value

This function returns the converted integral number as an int value. If no valid conversion could be performed, it returns zero.

## Example

The following example shows the usage of atoi() function.

#include <stdlib.h>

#include <string.h>

int main () {

int val;

char str[20];

strcpy(str, "98993489");

val = atoi(str);

printf("String value = %s, Int value = %d\n", str, val);

strcpy(str, "tutorialspoint.com");

val = atoi(str);

printf("String value = %s, Int value = %d\n", str, val);

return(0);

}

Output:

String value = 98993489, Int value = 98993489

String value = tutorialspoint.com, Int value = 0

## ****strrev()****

|  |
| --- |
| **Program to demonstrate the use of strrev() function in C.** |
| **#include<stdio.h>**  **#include<string.h>**  **int main()**  **{**  **char name[20]=”Rahul”;**  **strrev(name);**  **printf("name=%s",name);**  **return(0);**  **}** |
| **Output** |
| **name=luhaR** |

strrev() function is used to reverse the characters stored in a string value.

## ****strcpy()****

strcpy() is one of the most popular String functions in c language . This function is used store a value in a string variable

Syntax :

**strcpy(Str2,Str1);**

**Str2** is the string variable to store the value.

**Str1**is the string value to be stored in string variable **Str2.**

We can’t use assignment operator = to assign a value to string variable. So we need to use strcpy() function to do that.

**char name[20];**  
**strcpy(name,”Amit”);**

|  |
| --- |
| **Program to demonstrate the use of strcpy() function in C.** |
| **#include<stdio.h>**  **#include<string.h>**  **int main()**  **{**  **char name[20];**  **strcpy(name,”Rahul”);**  **printf("name=%s",name);**  **return(0);**  **}** |
| **Output** |
| **name=Rahul** |

## ****strstr() String Function:****

the strstr() string function is used to find the first occurrence of a substring in another string. This function returns a pointer to the element in string where substring begins. It returns NULL if no occurrence found. The general syntax of strstr() string function is as follows:

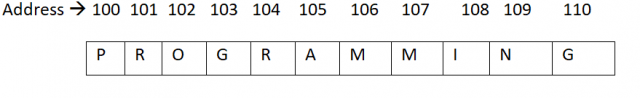
Var= strstr(string, substring); Where

String: it represents a string.

Substring: it represents a substring.

Var: it represents pointer of char type where result is to be stored.4

For example, string “PROGRAMMING” is shown below with supposed memory addresses for each character.

[](https://www.electroniclinic.com/wp-content/uploads/2020/11/strstr-String-Function-figure1.png)

The function strstr(“PROGRAMMIGN”,  “GRAM”) will return pointer to “G”(i.e. memory address 103)  because substring “GRAM” begins at this location.

### ****Programming example: write a program that explains the concept of strstr string function:****

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | #include <iostream>  #include <string.h>  #include <stdio.h>  using namespace std;   int main()  {      char st1[] = "programmingdigest";      char \*pstr;      pstr = strstr(st1, "ing");      cout<<pstr<<endl;      cout<<pstr-st1<<endl;      } |

Output:

Largest

8

## ****strtok() String Function:****

the strtok() string function is used to split a string into tokens. It means that this function divides the string into smaller string based on the given character. A sequence of calls to this function split the string into tokens, which are sequences of contiguous characters separated by any of the characters that are part of delimiters. The general syntax of this function is as follows;

strtok(string, delimiter); where

string: it represents the string to be broken into tokens.

Delimiter: it represents the delimiter to be used for scanning.

**Programming example: write a program that splits the given strings using strtok string function:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | #include <iostream>  #include <string.h>  #include <stdio.h>   using namespace std;   int main()  {      char \*str1 = "china, India, USA, Uk";      char \*str2 = "Pakistan#Japan#Sweden#Turkey";      char \* pch;      cout<<"splitting string1"<<endl;      pch=strtok(str1,",");      while(pch !=NULL)      {          cout<<pch<<endl;          pch = strtok(NULL, ",");              }      cout<<"\nSplitting string2"<<endl;      pch=strtok(str2,"#");      while(pch !=NULL)      {       cout<<pch<<endl;          pch=strtok(NULL,"#");      }  } |

# sprintf()

sprintf stands for "string print". In [C programming language](https://www.javatpoint.com/c-programming-language-tutorial), it is a file handling function that is used to send formatted output to the string. Instead of printing on console, sprintf() function stores the output on char buffer that is specified in sprintf.

### Syntax

**int** sprintf(**char** \*str, **const** **char** \*format, ...)

### Parameter values

The sprintf() function accepts some parameter values that are defined as follows -

**str:** It is the pointer to an array of char elements where the resulting string is stored. It is the buffer to put the data in.

**format:** It is [C string](https://www.javatpoint.com/c-strings) that is used to describe the output along with placeholders for the integer arguments to be inserted in the formatted string. It is said to the string that contains the text to be written to buffer. It consists of characters along with the optional format specifiers starting with %.

### Example1

This is a simple example to demonstrate the use of sprintf() function in C. Here, we are using multiple arguments with the sprintf() function.

1. #include <stdio.h>
2. int main()
3. {
4. char buffer[50];
5. int a = 15, b = 25, res;
6. res = a + b;
7. sprintf(buffer, "The Sum of %d and %d is %d", a, b, res);
8. printf("%s", buffer);
9. return 0;
10. }

**Output:**

The Sum of 15 and 25 is 40

**sscanf():**

sscanf() is used to read formatted input from the string.

**Syntax:**

int sscanf ( const char \* s, const char \* format, ...);

**Return type:** Integer

**Parameters:**

**s:** string used to retrieve data

**format:** string that contains the type specifier(s) arguments contains pointers to allocate storage with appropriate type.There should be at least as many of these arguments as the

number of values stored by the format specifiers.

On success, the function returns the number of variables filled. In the case of an input failure, before any data could be successfully read, EOF is returned.

|  |
| --- |
| // C program to illustrate sscanf statement  #include <stdio.h>   int main ()  {  // declaring array s      char s [] = "3 red balls 2 blue balls";      char str [10],str2 [10];      int i;      // %\*s is used to skip a word      sscanf (s,"%d %\*s %\*s %\*s %s %s", &i, str, str2);      printf ("%d %s %s \n", i, str, str2);  return 0;  } |

**Output:**

3 blue balls

**Function:**

A **function in C**is a set of statements that when called perform some specific task. It is the basic building block of a C program that provides modularity and code reusability. The programming statements of a function are enclosed within **{ } braces**, having certain meanings and performing certain operations. They are also called subroutines or procedures in other languages.

**Syntax of Functions in C**

The syntax of function can be divided into 3 aspects:

1. **Function Declaration**
2. **Function Definition**
3. **Function Calls**

**Function Declarations**

In a function declaration, we must provide the function name, its return type, and the number and type of its parameters. A function declaration tells the compiler that there is a function with the given name defined somewhere else in the program.

**Syntax**

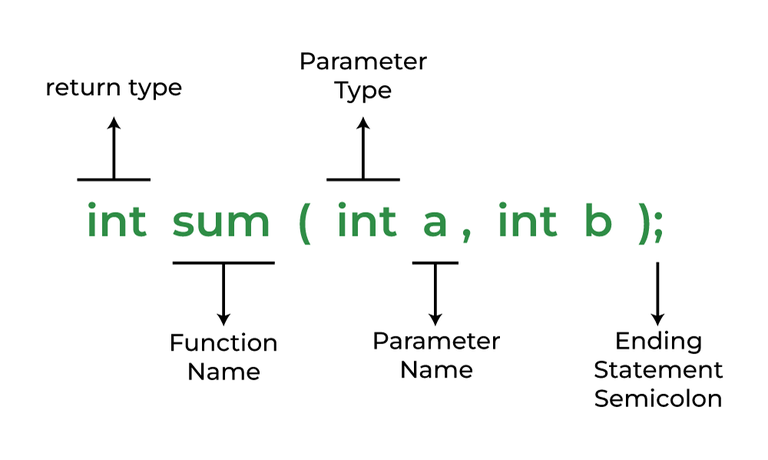
return\_type **name\_of\_the\_function** (*parameter\_1*, *parameter\_2*);

The parameter name is not mandatory while declaring functions. We can also declare the function without using the name of the data variables.

**Example**

int **sum**(int *a*, int *b*);

int **sum**(int , int);



*Function Declaration*

**Function Definition**

The function definition consists of actual statements which are executed when the function is called (i.e. when the program control comes to the function).

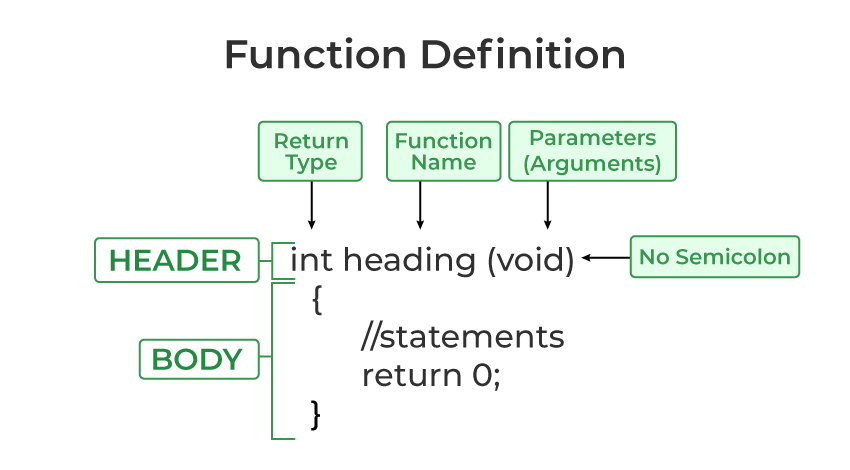
A C function is generally defined and declared in a single step because the function definition always starts with the function declaration so we do not need to declare it explicitly. The below example serves as both a function definition and a declaration.

return\_type **function\_name** (para1\_type *para1\_name,* para2\_type *para2\_name*)

{

*// body of the function*

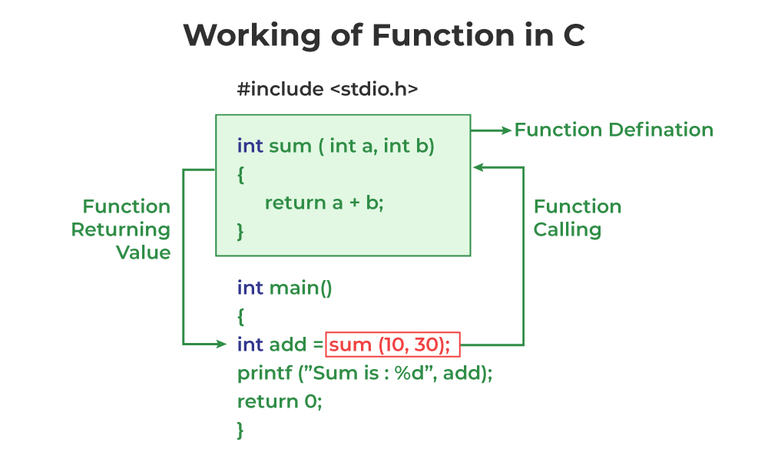
}



*Function Definition in C*

**Function Call**

A function call is a statement that instructs the compiler to execute the function. We use the function name and parameters in the function call.



*Working of function in C*

**Example of C Function**

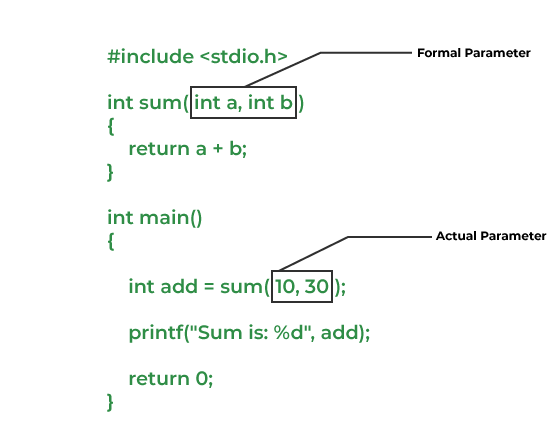
|  |
| --- |
| // C program to show function  // call and definition  #include <stdio.h>   // Function that takes two parameters  // a and b as inputs and returns  // their sum  int sum(int a, int b)  {    return a + b;  }   // Driver code  int main()  {    // Calling sum function and    // storing its value in add variable    int add = sum(10, 30);   printf("Sum is: %d", add);    return 0;  } |

**Output**

Sum is: 40

**Passing Parameters to Functions (Actual and Formal Parameters)**

The data passed when the function is being invoked is known as the Actual parameters. In the below program, 10 and 30 are known as actual parameters. Formal Parameters are the variable and the data type as mentioned in the function declaration. In the below program, a and b are known as formal parameters.



We can pass arguments to the C function in two ways:

1. Pass by Value
2. Pass by Reference

**1. Pass by Value**

Parameter passing in this method copies values from actual parameters into formal function parameters. As a result, any changes made inside the functions do not reflect in the caller’s parameters.

**Example:**

|  |
| --- |
| // C program to show use  // of call by value  #include <stdio.h>   void swap(int var1, int var2)  {    int temp = var1;    var1 = var2;    var2 = temp;  }   // Driver code  int main()  {    int var1 = 3, var2 = 2;    printf("Before swap Value of var1 and var2 is: %d, %d\n",            var1, var2);    swap(var1, var2);    printf("After swap Value of var1 and var2 is: %d, %d",            var1, var2);    return 0;  } |

**Output**

Before swap Value of var1 and var2 is: 3, 2

After swap Value of var1 and var2 is: 3, 2

**2. Pass by Reference**

The caller’s actual parameters and the function’s actual parameters refer to the same locations, so any changes made inside the function are reflected in the caller’s actual parameters.

**Example:**

|  |
| --- |
| // C program to show use of  // call by Reference  #include <stdio.h>   void swap(int \*var1, int \*var2)  {    int temp = \*var1;    \*var1 = \*var2;    \*var2 = temp;  }    // Driver code  int main()  {    int var1 = 3, var2 = 2;    printf("Before swap Value of var1 and var2 is: %d, %d\n",            var1, var2);    swap(&var1, &var2);    printf("After swap Value of var1 and var2 is: %d, %d",            var1, var2);    return 0;  } |

**Output**

Before swap Value of var1 and var2 is: 3, 2

After swap Value of var1 and var2 is: 2, 3

**Advantages of Functions in C**

Functions in C is a highly useful feature of C with many advantages as mentioned below:

1. The function can reduce the repetition of the same statements in the program.
2. The function makes code readable by providing modularity to our program.
3. There is no fixed number of calling functions it can be called as many times as you want.
4. The function reduces the size of the program.
5. Once the function is declared you can just use it without thinking about the internal working of the function.

**Disadvantages of Functions in C**

The following are the major disadvantages of functions in C:

1. Cannot return multiple values.
2. Memory and time overhead due to stack frame allocation and transfer of program control.

**Passing an array to function:**

An array is an effective way to group and store similar data together. We are required to pass an array to function several times, like in merge or quicksort. An array can be passed to functions in C using pointers by passing reference to the base address of the array,

**passing an array to function example**

1. #include<stdio.h>
2. int minarray(int arr[],int size){
3. int min=arr[0];
4. int i=0;
5. for(i=1;i<size;i++){
6. if(min>arr[i]){
7. min=arr[i];
8. }
9. }//end of for
10. return min;
11. }//end of function
13. int main(){
14. int i=0,min=0;
15. int numbers[]={4,5,7,3,8,9};//declaration of array
17. min=minarray(numbers,6);//passing array with size
18. printf("minimum number is %d \n",min);
19. return 0;
20. }

**Output**

minimum number is 3

**Function parameter:**

Information can be passed to functions as a parameter. Parameters act as variables inside the function. Parameters are specified after the function name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma:

### Syntax

returnType functionName(parameter1, parameter2, parameter3) {  
  // code to be executed  
}  
The following function that takes a [string of characters](https://www.w3schools.com/c/c_strings.php) with **name** as parameter. When the function is called, we pass along a name, which is used inside the function to print "Hello" and the name of each person.

### Example

void myFunction(char name[]) {  
  printf("Hello %s\n", name);  
}  
  
int main() {  
  myFunction("Liam");  
  myFunction("Jenny");  
  myFunction("Anja");  
  return 0;  
}

Output:  
Hello Liam  
 Hello Jenny  
 Hello Anja