

CL-1002

LAB - 08

Programming

Multiple Dimension Array

Fundamentals

(2D,3D) Functions, Strings

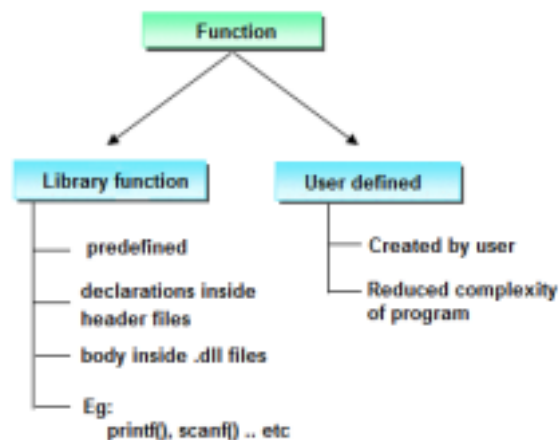
NATIONAL UNIVERSITY OF COMPUTER AND EMERGING
SCIENCES
Spring 2022

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 {}. 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.

TYPES OF FUNCTIONS IN C PROGRAMMING

Depending on whether a function is defined by the user or already included in C compilers, there are two types of functions in C programming

1. Standard library functions
2. User defined functions



Functions Name Description	
Printf()	Print data
Scanf()	Read data
Getchar()	Read a single a character
Sqrt()	Calculate the square
Pow()	Calculate the power
Fopen()	Open the specified file

BENEFITS OF USING FUNCTIONS

1. It provides modularity to your program's structure.
2. It makes your code reusable. You just have to call the function by its name to use it, wherever required.
3. In case of large programs with thousands of code lines, debugging and editing becomes easier if you use functions.
4. It makes the program more readable and easy to understand.

HOW USER-DEFINED FUNCTION WORKS?

Function prototype

Function call

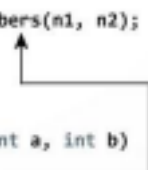
Example: User

```
#include <stdio.h>

int addNumbers(int a, int b);

int main()
{
    ... ..
    sum = addNumbers(n1, n2);
    ... ..
}

int addNumbers(int a, int b)
{
    ... ..
    return result;
}
```

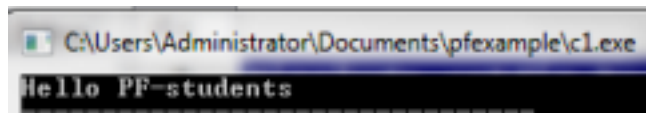


The diagram shows a call stack. In the `main` function, the line `sum = addNumbers(n1, n2);` has an arrow pointing down to the `addNumbers` function definition. Another arrow points from the `return result;` line of the `addNumbers` function back up to the line in `main` where the function was called. A box labeled `sum = result` is positioned next to the return statement in `addNumbers`.

Function definition

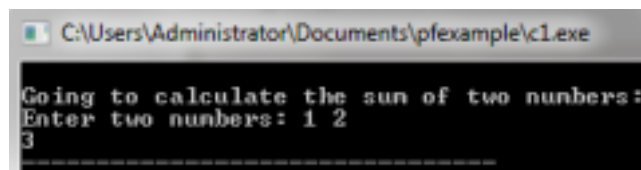
Here is an example one to print "Hello pf- students". To perform this task, a user-defined function `printName ()` is defined.

```
#include <stdio.h>
void printName ();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("PF-students");
}
```



Here is an example two to add two integer numbers. To perform this task, a user-defined function `sum ()` is defined.

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\n Calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers: ");
    scanf("%d %d",&a,&b);
    return a+b;
}
```



}

FUNCTION PROTOTYPE

A function prototype is simply the declaration of a function that specifies function's name, Parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

SYNTAX OF FUNCTION PROTOTYPE

returnType functionName (type1 argument1, type2 argument2,...);

In the above example, `int sum(int a, int b);` is the function prototype which provides following information to the compiler:

1. name of the function is `addNumbers()`
2. return type of the function is `int`
3. two arguments of type `int` are passed to the function

The function prototype is not needed if the user-defined function is defined before the `main()` function.

CALLING A FUNCTION

Control of the program is transferred to the user-defined function by calling it.

SYNTAX OF FUNCTION CALL

functionName (argument1, argument2, ...);

In the above example, function call is made using `sum(n1,n2);` statement inside the

`main()`. **FUNCTION DEFINITION**

Function definition contains the block of code to perform a specific task i.e. in this case, adding two numbers and returning it.

SYNTAX OF FUNCTION DEFINITION

`returnType functionName(type1 argument1, type2 argument2,`

`{`

//body of the function

`}`

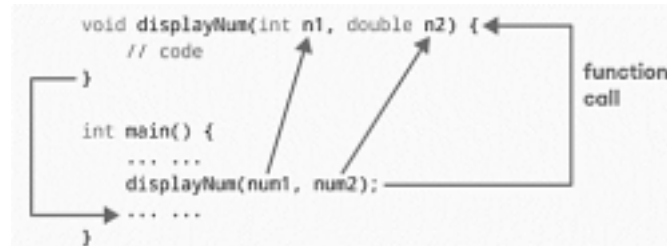
When a function is called, the control of the program is transferred to the function

definition. And, the compiler starts executing the codes inside the body of a function.

PASSING ARGUMENTS TO A FUNCTION

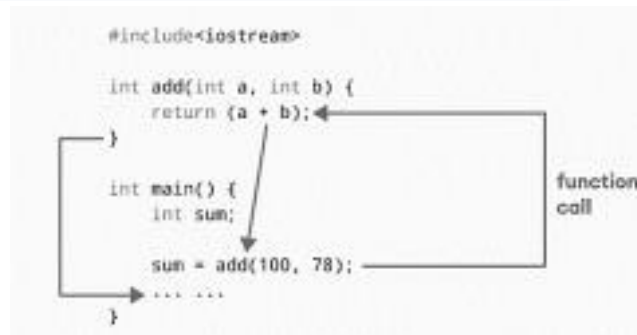
In programming, argument refers to the variable passed to the function. The parameters a and b

accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.



RETURN STATEMENT

The return statement terminates the execution of a function and returns a value to the calling function. The program control is transferred to the calling function after return statement. we have the data type int instead of void. This means that the function returns an int value.



SYNTAX OF RETURN STATEMENT

return (expression);

USER DEFINED HEADER FILE

The purpose to understand and learn header file is that, it also contains specific function define in it. You are required to call its header and can use its defined function in your program. Meanwhile header file serves two purposes.

- You include them in your program to supply the definitions and declarations you need to invoke system calls and libraries.
- Your own header files contain declarations for interfaces between the source files of your program. Each time you have a group of related declarations and macro definitions all or most of which are needed in several different source files.

Follow the step to create your header file:

Suppose we want to make header for sum function.

1. Make a header file with .h extension and give it unique name e.g sumfile-> sumfile.h
2. Define your program in header extension file.
a. `int add(int a,int b){return(a+b);}`
3. Make source file of c, where your main program is set.
a. `#include<stdio.h>`
b. `#include "sumfile.h" // don't use '<>', instead of it use ""`.
c. `void main()`
d. `{ int num1 = 10, num2 = 10, num3;`
e. `num3 = add(num1, num2);`
f. `printf("Addition of Two numbers : %d", num3);}`
4. Keep .h file path directory same as source file.

5. In the above program the 'add' function is basically called from the header file of sumfile.h Which we have explicitly defined.

INTRODUCTION TO STRINGS

- String is a series of characters treated as a single unit.
- A string may include letters, digits and various special characters such as +, -, *, / and \$.
- String literals, or string constants in C are written in double quotation marks.

STRING DECLARATION AND INITIALIZATION

A string in C is implemented as an array, so declaring a string variable is the same as declaring an array of type **char**.

EXAMPLE:

```
char var[9] = {'F','A','S','T',' ','U','N','I'};  
OR  
char string_var[9] = "FAST UNI";
```

The variable string_var will hold strings from 0 to 8 characters long.

NULL CHARACTER ('\0')

- Null character marks the end of a string.
- All of C's string handling functions simply ignore whatever is stored in the cells following the null character.
- When defining a character array to contain a string, the array must be large enough to store the string and its terminating null character.

MEMORY REPRESENTATION

F	A	S	T		U	N	I	\0
0	1	2	3	4	5	6	7	8

STRING INPUT/OUTPUT LIBRARY

INPUT/OUTPUT WITH PRINTF AND SCANF

- A string can be read using the %s placeholder in the scanf function. However, it has a limitation that the strings entered cannot contain spaces and tabs.

```
#include<stdio.h>

int main()
{
    char sentence[80];

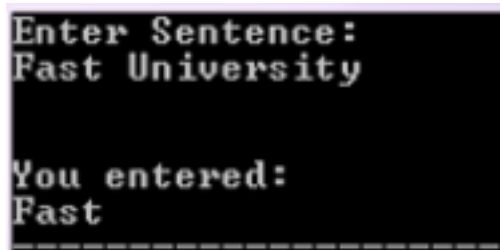
    printf("Enter
    Sentence:\n");

    scanf("%s
    ",sentence);

    printf("\n\nYou
    entered:\n");

    printf("%s",
    sentence)

    return 0;
}
```



```
Enter Sentence:
Fast University

You entered:
Fast
-----
```

In the above code if multiple words (separated by space) are entered, scanf() will only consider the first word as shown in the output attached.

INPUT/OUTPUT WITH GETS AND PUTS

To overcome the problem with scanf, C provides the **gets** function. It allows us to read a line of characters (including spaces and tabs) until the newline character is entered, i. e., the Enter key is pressed. A call to this function takes the following form:

gets(sentence);

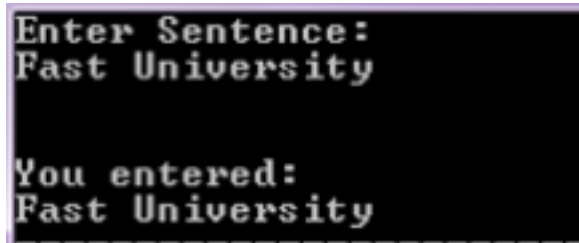
where, sentence is an array of char, i.e., a character string. The function reads characters entered from the keyboard until newline is entered and stores them in the argument string sentence, the newline character is read and converted to a null character (\0) before it is stored in s.

C provides another function named puts to print a string on the display. A typical call to this function takes the following form:

puts(sentence);

where, sentence is an array of characters, i.e., a character string. This string is printed on the display followed by a newline character.

```
#include<stdio.h>
int main()
{
    char sentence[80];
    printf("Enter Sentence:\n");
    gets(sentence);
    puts("\n\nYou entered:");
    printf("%s", sentence);
    return 0;
}
```



ARRAYS OF STRINGS

One string is an array of characters, an array of strings is a two-dimensional array of characters in which each row is one string.

An array of strings can be initialized at declaration in the following manner:

```
char day[7][10] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
                  "Saturday"};
```

STRINGS-FUNCTIONS

EXAMPLES

strlen function: size_t strlen(const char *str)

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[30] = "Programming Fundamentals";
    printf("Length of string str1: %d", strlen(str1));
    return 0;
}
```

strlen vs sizeof

strlen returns you the length of the string stored in an array, however **sizeof** returns the total allocated size assigned to the array. So if I consider the above example again, then the following statements would return the below values.

strlen(str1) returned value 23.

sizeof(str1) would return value 30 as the array size is 30 (see the first statement in main function).

strcmp function: int strcmp(const char *str1, const char str2)

- It compares the two strings and returns an integer value.

- if Return value < 0 then it indicates str1 is less than str2.
- if Return value > 0 then it indicates str2 is less than str1.
- if Return value = 0 then it indicates str1 is equal to str2.

```
#include <stdio.h>
#include <string.h>
int main () {
    char str1[15];
    char str2[15];
    int ret;
    strcpy(str1, "abcdef");
    strcpy(str2, "ABCDEF");
    ret = strcmp(str1, str2);
    if(ret < 0) {
        printf("str1 is less than str2");
    } else if(ret > 0) {
        printf("str2 is less than str1");
    } else {
        printf("str1 is equal to str2");
    }
    return (0);
}
```

strncmp function: int strncmp(const char *str1, const char *str2, size_t n)

It compares both the string till n characters or in other words it compares first n characters of both the strings.

```
#include <stdio.h>
#include <string.h>
int main () {
    char str1[15];
    char str2[15];
    int ret;
    strcpy(str1, "abcdef");
    strcpy(str2, "ABCDEF");
    ret = strncmp(str1, str2, 4);
    if(ret < 0) {
        printf("str1 is less than str2");
    } else if(ret > 0) {
        printf("str2 is less than str1");
        printf("str1 is equal to str2");
    }
    return(0);}
}
```

strcat function: char *strcat(char *str1, char *str2)

It concatenates two strings and returns the combined string.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strcat(s1,s2);
    printf("Output string after concatenation: %s", s1);
    return 0;
}
```

strncat function: char *strncat(char *str1, char *str2, int n)

It concatenates n characters of str2 to string str1.

```
#include <string.h>
int main()
{
    char s1[10] = "Hello";
    char s2[10] = "World";
    strncat(s1,s2, 3);
    printf("Concatenation using strncat: %s", s1);
    return 0; }
```

strcpy function: char *strcpy(char *str1, char *str2)

It copies the string str2 into string str1, including the end character (terminator char '\0').

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[20] = "string 1";
    char s2[50] = "string 2 : I will be copied into s1";
    /* this function has copied s2 into s1*/
    strcpy(s1,s2);
    printf("String s1 is: %s", s1);
    return 0; }
```

strchr function: char *strchr(char *str, int ch)

It searches string str for character ch

```
#include <stdio.h>
#include <string.h>
int main()
{
    char mystr[50] = "I am an example to use function strchr";
```

```
printf ("%s", strchr(mystr, 'f'));  
return 0; }
```

strstr function: char *strstr(char *str, char *srch term)

It is similar to strchr, except that it searches for string srchterm instead of a single char.

```
#include <stdio.h>  
#include <string.h>  
int main()  
{  
char inputstr[70] = "String Function in C at BeginnersBook.COM";  
printf ("Output string is: %s", strstr(inputstr, "Begi"));  
return 0; }
```