

User Defined Functions



Introduction

- User-defined functions in C language are defined by the programmer to perform specific operations.
- To define a functions the following components should be known:
 - Function Prototype
 - Function Definition
 - Function calling
 - Call by value
 - Call by reference
 - Return statement

Example: User Defined Function

- Here is an example to add two integers.
- To perform this task, we have created an user-defined-addNumbers()

```
#include <stdio.h>
int addNumbers(int a, int b);           // function prototype

int main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);           // function call
    printf("sum = %d",sum);

    return 0;
}

int addNumbers(int a, int b)           // function definition
{
    int result;
    result = a+b;
    return result;                      // return statement
}
```

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

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

Function Prototype

- In the example, `int addNumbers(int a, int b);` is the function prototype which provides the following information to the compiler:
 - ❖ name of the function is `addNumbers()`
 - ❖ return type of the function is `int`
 - ❖ 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.

Function Definition

- Function definition contains the block of code to perform a specific task.
- In our example, adding two numbers and returning it.

- Syntax

```
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.

Function Calling

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

```
functionName(argument1, argument2, ...);
```

- In our example, the function call is made using `addNumbers(n1, n2);` statement inside the `main()` function.

Passing Arguments to a Function

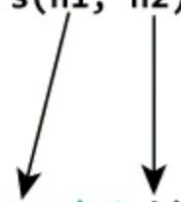
- In programming, argument refers to the variable passed to the function. In the above example, two variables n1 and n2 are passed during the function call called actual parameters.
- The parameters a and b accepts the passed arguments in the function definition. These arguments are called formal parameters of the function.

```
#include <stdio.h>

int addNumbers(int a, int b);

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

int addNumbers(int a, int b)
{
    ... ..
    ... ..
}
```

A diagram with two arrows pointing downwards. The first arrow starts from the variable 'n1' in the function call 'addNumbers(n1, n2);' inside the 'main()' function and points to the parameter 'a' in the function definition 'int addNumbers(int a, int b)'. The second arrow starts from the variable 'n2' in the same function call and points to the parameter 'b' in the function definition.

Passing Arguments to a Function

- The type of arguments passed to a function and the formal parameters must match, otherwise, the compiler will throw an error.
- If n1 is of char type, a also should be of char type. If n2 is of float type, variable b also should be of float type.
- A function can also be called without passing an argument.

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 the return statement.
- In the example, the value of the result variable is returned to the main function. The sum variable in the main() function is assigned this value.
- Syntax

```
return (expression);
```

```
return a;  
return (a+b);
```

Return Statement

- The type of value returned from the function and the return type specified in the function prototype and function definition must match.

```
#include <stdio.h>

int addNumbers(int a, int b);

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

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

sum = result

Function calling types

➤ Function calling

- ❖ Call By Value

- ❖ Call By Reference

Function calling – Call by value

- In call by value, a copy of the value is passed to the function and changes that are made to the function are not reflected back to the values.
- Actual and formal arguments are created in different memory locations.

Function Calling – Call by value

Example

- Values aren't changed in the call by value since they aren't passed by reference

Output

Values of x and y before swap are: 10, 20

Values of x and y after swap are: 10, 20

```
#include <stdio.h>

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int x = 10, y = 20;
    printf("Values of x and y before swap are: %d, %d\n", x, y);
    swap(x, y);
    printf("Values of x and y after swap are: %d, %d", x, y);
    return 0;
}
```

Function Calling – Call by reference

- In a call by Reference, the address of the argument is passed to the function, and changes that are made to the function are reflected back to the values.

Output

Values of x and y before swap are: 10, 20

Values of x and y after swap are: 20, 10

```
#include <stdio.h>

void swap(int* a, int* b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int x = 10, y = 20;
    printf("Values of x and y before swap are: %d, %d\n", x, y);
    swap(&x, &y);
    printf("Values of x and y after swap are: %d, %d", x, y);
    return 0;
}
```


Practice Problems

- Write a function to print all Armstrong numbers between given interval using function.

Input

Input lower limit: 1

Input upper limit: 1000

Output

Armstrong numbers between 1 to 1000 are:

1, 153, 370, 371, 407

- Write a program to find largest number in an array using a function.
- Write a program to convert a string from upper case to lowercase using a function.
- Write a program to concatenate two strings using library function.

References URL:

- https://www.w3schools.com/c/c_functions.php
- <https://medium.com/@infinator/c-programming-for-beginners-more-on-functions-fe6251f59000>
- <https://medium.com/@infinator/c-programming-for-beginners-string-functions-69be524972e9>
- <https://www.geeksforgeeks.org/c-functions/>

THANK YOU

