

Lecture 8

Functions

What is Function?

- A function is a block of code that performs a specific task.

Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:

- create a circle function
 - color function
- A fragment of code that accepts zero or more *argument values*, produces a *result value*, and has zero or more *side effects*.
- Dividing complex problem into small components makes program easy to understand and use.

Why Function?

- Break longer jobs into conceptually smaller jobs which are precisely defined.
- C program generally consists of many small functions.
- A piece of a code which repeats at many places can be written only once and used again and again.
- Debugging and maintenance is easy.

Function cont.

- Functions
 - Modularize a program
 - All variables defined inside functions are local variables
 - Known only in function defined
 - Parameters
 - Communicate information between functions
 - Local variables
- Benefits of functions
 - Divide and conquer
 - Manageable program development
 - Software reusability
 - Use existing functions as building blocks for new programs
 - Abstraction - hide internal details (library functions)
 - Avoid code repetition

Types of function

- 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
- Standard library functions: examples, printf(), scanf(), abs(int), sqrt(float), etc
- User defined functions: examples, main(), anyFunction(), addTwoNumbers(int, int) etc.

Standard library functions

- The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.
- These functions are defined in the header file. When you include the header file, these functions are available for use. For example:
 - The `printf()` is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "`stdio.h`" header file.
- There are other numerous library functions defined under "`stdio.h`", such as `scanf()`, `fprintf()`, `getchar()` etc. Once you include "`stdio.h`" in your program, all these functions are available for use.

Standard library functions cont.

`#include <math.h>`

- `sin(x)` // radians
- `cos(x)` // radians
- `tan(x)` // radians
- `atan(x)`
- `atan2(y, x)`
- `exp(x)` // e^x
- `log(x)` // $\log_e x$
- `log10(x)` // $\log_{10} x$
- `sqrt(x)` // $x \geq 0$
- `pow(x, y)` // x^y
- ...

`#include <stdio.h>`

- `printf()`
- `fprintf()`
- `scanf()`
- `sscanf()`
- ...

`#include <string.h>`

- `strcpy()`
- `strcat()`
- `strcmp()`
- `strlen()`
- ...

User-defined function

- As mentioned earlier, C allow programmers to define functions. Such functions created by the user are called user-defined functions.
- Function definition format

```
return-value-type function-name( parameter-list )  
{  
    declarations and statements  
}
```

- **Function-name:** any valid identifier
- **Return-value-type:** data type of the result (default `int`)
 - `void` – indicates that the function returns nothing
- **Parameter-list:** comma separated list, declares parameters
 - A type must be listed explicitly for each parameter unless, the parameter is of type **`int`**

User-defined function cont.

- Function definition format (continued)

```
return-value-type function-name( parameter-list )  
    {  
        declarations and statements  
    }
```

- Definitions and statements: function body (block)

- Variables can be defined inside blocks (can be nested)
- Functions can not be defined inside other functions

- Returning control

- If nothing returned
 - **return;**
 - **or, until reaches right brace**
- If something returned
 - **return** *expression* ;

How user-defined function works?

- The execution of a C program begins from the **main()** function.
- When the compiler encounters `functionName();` inside the main function, control of the program jumps to ***void functionName()***
- *And, the compiler starts executing the codes inside the user-defined function.*
- *The control of the program jumps to statement next to `functionName();` once all the codes inside the function definition are executed.*

```
#include <stdio.h>

void functionName()
{
    ... ..
    ... ..
}

int main()
{
    ... ..
    ... ..

    functionName();

    ... ..
    ... ..
}
```

User-defined function cont.

- Basically a function has the following characteristics:

1. *Named with unique name* .
2. *Performs a specific task* - Task is a discrete job that the program must perform as part of its overall operation, such as sending a line of text to the printer, sorting an array into numerical order, or calculating a cube root, etc.
3. *Independent* - A function can perform its task without interference from or interfering with other parts of the program.
4. *May receive values from the calling program (caller)* - Calling program can pass values to function for processing whether directly or indirectly (by reference).
5. *May return a value to the calling program* – the called function may pass something back to the calling program.

User-defined function cont.

- Function mechanism
 - C program does not execute the statements in a function until the function is called.
 - When it is called, the program can send information to the function in the form of one or more arguments although it is not a mandatory.
 - Argument is a program data needed by the function to perform its task.
 - When the function finished processing, program returns to the same location which called the function.

User-defined function example

```
#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 of function prototype

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

Function prototype

```
#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
}
```

In the above example, `int addNumbers(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.

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.

Calling a function

- Control of the program is transferred to the user-defined function by calling it.
- Syntax of function call
 - functionName(argument1, argument2, ...);

```
#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
}
```

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

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 function call and it is called **actual parameter** of the function.
- The parameters *a* and *b* accepts the passed arguments in the function definition. These arguments are called **formal parameters** of the function.

Passing arguments to a function

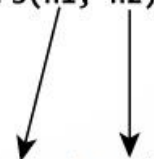
How to pass arguments to a function?

```
#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 from the arguments 'n1' and 'n2' in the function call 'addNumbers(n1, n2);' within the 'main' function to the parameters 'a' and 'b' in the function definition 'int addNumbers(int a, int b)'. The arrow from 'n1' points to 'a', and the arrow from 'n2' points to 'b'.

- The data type of arguments passed to a function and the formal parameters must match, otherwise the compiler throws 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 return statement.

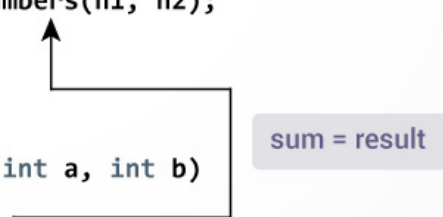
Return statement of a Function

```
#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

- In the above example, the value of variable result is returned to the variable sum in the main() function.

Return Statement

- Syntax

```
return (expression);
```

Example:

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

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

Functions: Example

- Prime Numbers

- Print a table as follows

1*	2*	3*	4	5*	...	10
11*	12	13*	14	15	...	20

upto 100. All primes are starred.

- The main function prints the nice looking table. And it is not about finding primes.
 - The function isPrime(n) is expected to return 1(true) if n is a prime and 0(false) otherwise.

```
#include <stdio.h>

main()
{
    int i;

    for( i = 1; i <= 100; i++ ) {
        printf("%d", i);
        if ( isPrime(i) )
            printf("*");
        if ( i % 10 == 0 )
            printf("\n");
        else
            printf("\t");
    }
}
```

Functions: Example

- Prime Numbers
 - A rather simple implementation of isPrime function is given here.
 - The function has same structure as main function we saw before, except that int before the name of the function and return statement.

```
int isPrime(int num)
{
    int    i;

    if ( num < 3 ) return 1;

    for (i=2; i<=(int) sqrt(num);i++ )
    {   if ( num % i == 0 ) break; }

    if ( num % i ) return 1;

    return 0;
}
```