# Lecture – 11

# C Programming Functions

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

A function is a block of statements that performs a specific task. Suppose you are building an application in C language and in one of your program, you need to perform a same task more than once. In such case you have two options –

a) Use the same set of statements every time you want to perform the task
b) Create a function to perform that task, and just call it every time you need to perform that task.

Using option (b) is a good practice and a good programmer always uses functions while writing codes in C.

## 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

There are two types of functions in C programming:

- Standard library functions
- User defined functions

### 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.

## User-defined functions

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

Depending upon the complexity and requirement of the program, you can create as many user-defined functions as you want.

# How user-defined function works?

```
#include <stdio.h>

void functionName()

{

    ... .. ...

    ... .. ...

}



int main()

{

    ... .. ...

    ... .. ...
```

```
    functionName();


    ... .. ...

    ... .. ...

}
```

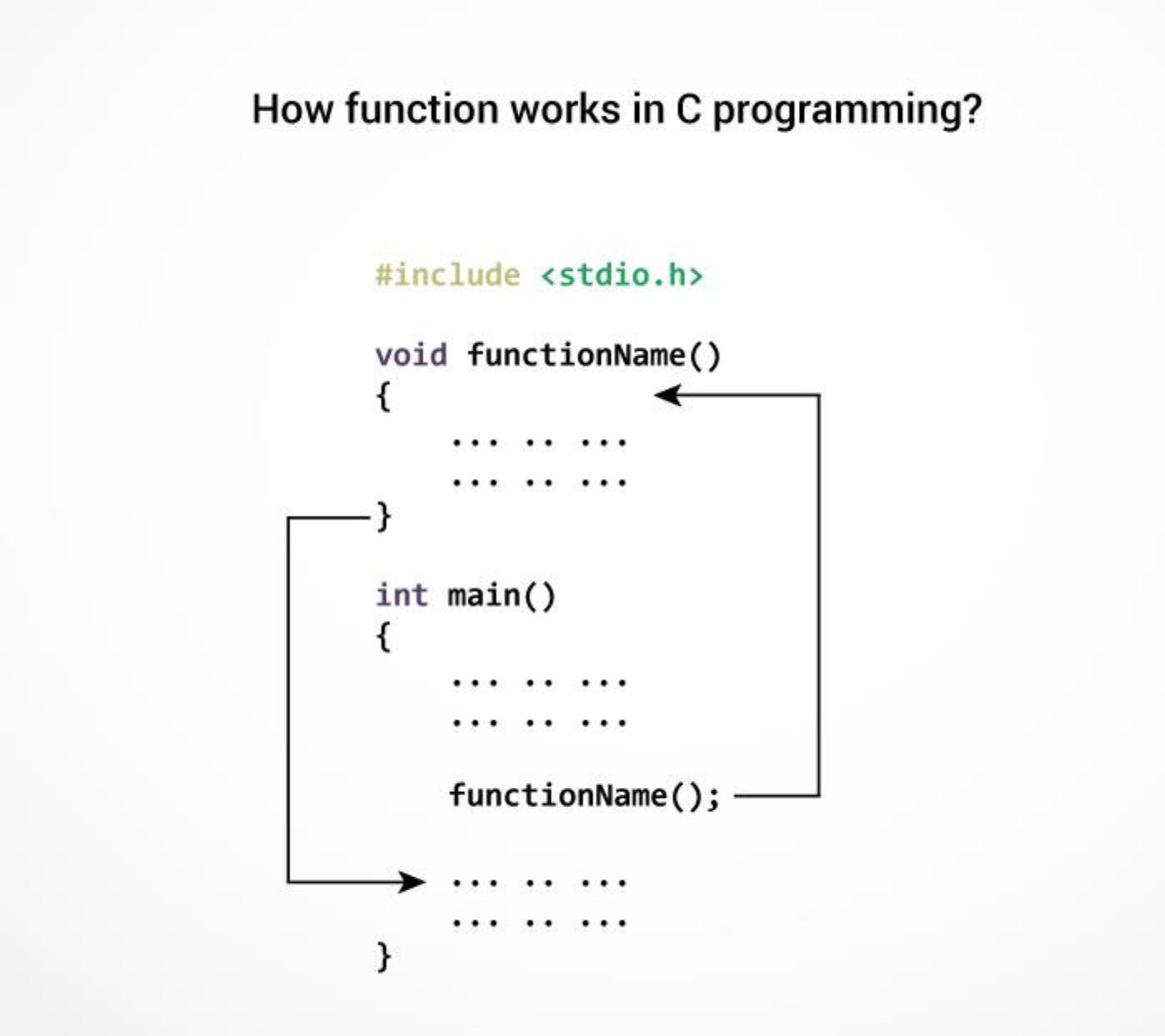
The execution of a C program begins from the `main()` function.

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.




Remember, function name is an identifier and should be unique.

This is just an overview on user-defined function. Visit these pages to learn more on:

- User-defined Function in C programming
- Types of user-defined Functions

## Advantages of user-defined function

1. The program will be easier to understand, maintain and debug.

2. Reusable codes that can be used in other programs
3. A large program can be divided into smaller modules. Hence, a large project can be divided among many programmers.

# Types of User-defined Functions in C Programming

For better understanding of arguments and return value from the function, user-defined functions can be categorized as:

- Function with no arguments and no return value
- Function with no arguments and a return value
- Function with arguments and no return value
- Function with arguments and a return value.

The 4 programs below check whether an integer entered by the user is a prime number or not. And, all these programs generate the same output.

## Example #1: No arguments passed and no return Value

```
#include <stdio.h>

void checkPrimeNumber();

int main()
{
    checkPrimeNumber();    // no argument is passed to prime()
    return 0;
}
```

```c
// return type of the function is void becuase no value is returned from the function
void checkPrimeNumber()
{
    int n, i, flag=0;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
        {
            flag = 1;
        }
    }
    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);
}
```

The `checkPrimeNumber()` function takes input from the user, checks whether it is a prime number or not and displays it on the screen.

The empty parentheses in `checkPrimeNumber();` statement inside the `main()` function indicates that no argument is passed to the function.

The return type of the function is `void`. Hence, no value is returned from the function.

# Example #2: No arguments passed but a return value

```c
#include <stdio.h>
int getInteger();

int main()
{
    int n, i, flag = 0;

    // no argument is passed to the function
    // the value returned from the function is assigned to n
    n = getInteger();

    for(i=2; i<=n/2; ++i)
    {
        if(n%i==0){
            flag = 1;
            break;
        }
    }

    if (flag == 1)
        printf("%d is not a prime number.", n);
    else
        printf("%d is a prime number.", n);

    return 0;
}

// getInteger() function returns integer entered by the user
```

```
int getInteger()
{
    int n;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    return n;
}
```

The empty parentheses in `n = getInteger();` statement indicates that no argument is passed to the function. And, the value returned from the function is assigned to `n`.

Here, the `getInteger()` function takes input from the user and returns it. The code to check whether a number is prime or not is inside the `main()` function.

# Example #3: Argument passed but no return value

```
#include <stdio.h>
void checkPrimeAndDisplay(int n);

int main()
{
    int n;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    // n is passed to the function
    checkPrimeAndDisplay(n);

    return 0;
```

```
}


// void indicates that no value is returned from the function
void checkPrimeAndDisplay(int n)
{
    int i, flag = 0;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0){
            flag = 1;
            break;
        }
    }
    if(flag == 1)
        printf("%d is not a prime number.",n);
    else
        printf("%d is a prime number.", n);
}
```

The integer value entered by the user is passed to `checkPrimeAndDisplay()` function.

Here, the `checkPrimeAndDisplay()` function checks whether the argument passed is a prime number or not and displays the appropriate message.

# Example #4: Argument passed and a return value

```
#include <stdio.h>
int checkPrimeNumber(int n);


int main()
```

```
{
    int n, flag;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    // n is passed to the checkPrimeNumber() function
    // the value returned from the function is assigned to flag variable
    flag = checkPrimeNumber(n);

    if(flag==1)
        printf("%d is not a prime number",n);
    else
        printf("%d is a prime number",n);

    return 0;
}

// integer is returned from the function
int checkPrimeNumber(int n)
{
    /* Integer value is returned from function checkPrimeNumber() */
    int i;

    for(i=2; i <= n/2; ++i)
    {
        if(n%i == 0)
            return 1;
    }

    return 0;
```

```
}
```

The input from the user is passed to `checkPrimeNumber()` function.

The `checkPrimeNumber()` function checks whether the passed argument is prime or not. If the passed argument is a prime number, the function returns 0. If the passed argument is a non-prime number, the function returns 1. The return value is assigned to `flag` variable.

Then, the appropriate message is displayed from the `main()` function.

## Which approach is better?

Well, it depends on the problem you are trying to solve. In case of this problem, the last approach is better.

A function should perform a specific task. The `checkPrimeNumber()` function doesn't take input from the user nor it displays the appropriate message. It only checks whether a number is prime or not, which makes code modular, easy to understand and debug.

# Some Examples:

## #Example: C Program to Display Prime Numbers between Intervals Using Function

Example to print all prime numbers between two numbers (entered by the user) by making a user-defined function.

```
#include <stdio.h>


int checkPrimeNumber(int n);
int main()
{
    int n1, n2, i, flag;
```

```
    printf("Enter two positive integers: ");
    scanf("%d %d", &n1, &n2);
    printf("Prime numbers between %d and %d are: ", n1, n2);

    for(i=n1+1; i<n2; ++i)
    {
        // i is a prime number, flag will be equal to 1
        flag = checkPrimeNumber(i);

        if(flag == 1)
            printf("%d ",i);
    }
    return 0;
}

// user-defined function to check prime number
int checkPrimeNumber(int n)
{
    int j, flag = 1;

    for(j=2; j <= n/2; ++j)
    {
        if (n%j == 0)
        {
            flag =0;
            break;
        }
    }
    return flag;
```

```
}
```

**Output**

```
Enter two positive integers: 12

30

Prime numbers between 12 and 30 are: 13 17 19 23 29
```

**User defined function:** A function is a self contained block of statements that perform a coherent task of some kind. Every C program can be though of as a collection of these functions.

```
return_type fname (arguments);

 int main ()
{
        …
        …
        return 0;
}
return_type fname (arguments)
{
        …
        …
}
```

**Advantages of function:**

- Reduce redundancy of logic
- Easy to debug
- Easy to write.

Example 1:

```
int sum(int p, int q);

int main()
{
        int x =1, y=2, z ;
        z = sum(x,y);
        printf("%d",z);
        return 0;
}
int sum(int p, int q)
{
        return p+q;
}
```

3

Example 2:

```
void one(void);
void two(void);


int main()
```

```
{
        one();
        two();
        return 0;
}
void one()
{
        printf("In One\n");
        two();
}

void two()
{
        printf("In Two\n");
}
```

```
In One
In Two
In Two
```

Example 3:

```
void swap(int p, int q);

int main()
{
        int x =2, y=3 ;
        printf("Before swap x = %d and y = %d\n",x,y);
        swap(x,y);
        printf(After swap x = %d and y = %d",x,y);
        return 0;
}
void swap(int p, int q)
{
        int temp;
        temp = p;
        p = q ;
        q = temp;
}
```

```
Before swap x = 2 and y = 3
After swap x=2 and y = 3
```

Example 4:

```
void swap(int x, int y)
{
        int temp;
        temp = x;
        x = y ;
        y = temp;
}
```

```
Before swap x = 2 and y = 3
After swap x=2 and y = 3
```

```
int main()
{
```

```
        int x =2, y=3 ;
        printf("Before swap x = %d and y = %d\n",x,y);
        swap(x,y);
        printf(After swap x = %d and y = %d",x,y);
        return 0;
}
```

Variable x and y are not same within main() and swap() because of local. declaration. If we declare variables globally then x and y variable will be recognized by both main() and swap(). In this case, output will be :

```
Before swap x = 2 and y = 3
After swap x=3 and y = 2
```

Example: 5

```
void square(int n)
{
        printf("The square of %d is %d\n", n, n*n);
}

int main()
{
        int index;
        for(index=1; index<=5; index++)
                square(index);
        return 0;
}
```

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
```

Example: 6

```
void even(int p)
{
        int x;
        for (x =1; x<=p ; x ++)
                if(x % 2 = = 0)
                        printf(" %d\n", x);
}

int main()
{
        int n;
        printf("Please enter a number");
        scanf("%d",&n);
        even(n);
        return 0;
}
```

```
Please enter a number 10
2
4
6
8
10
```