

Function

Function: A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. Functions are used to perform certain actions and they are important for reusing code. Define the code once and use it many times.

Syntax:

```
Void myFunction()  
{  
  //Code to be executed  
}
```

Example Explained:

My function is the name of function

Void means the function doesn't have any return value.

Types of Functions

There are two types of functions in C programming:

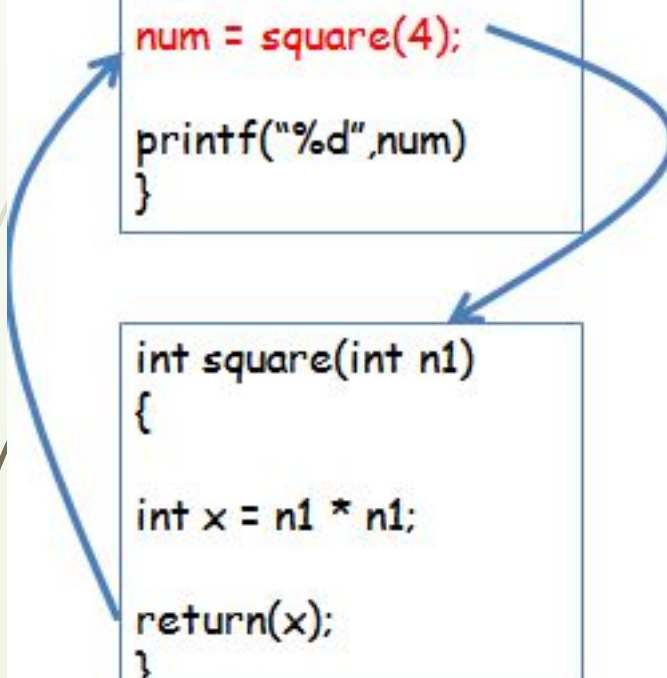
Library Functions: are the functions which are declared in the C header files such as `scanf()`, `printf()`, `gets()`, `puts()`, `ceil()`, `floor()` etc.

User-defined functions: There are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code

Calling a Function

```
void main()
{
    int num;

    num = square(4);
    printf("%d",num)
}
```



A blue arrow originates from the `num = square(4);` line in the `main` function and points to the `int square(int n1)` function definition. Another blue arrow originates from the `return(x);` line in the `square` function and points back to the line immediately following `num = square(4);` in the `main` function.

```
int square(int n1)
{
    int x = n1 * n1;

    return(x);
}
```

Understanding Flow

1. Firstly Operating System will call our main function.
2. When control comes inside main function , execution of main starts (i.e execution of C program starts)
3. Consider Line 4 :

```
num = square(4);
```

4. We have called a function `square(4)`. [See : [How to call a function ?](#)].
5. We have passed "4" as parameter to function.

Note : Calling a function halts execution of the current function , it will execute called function , after execution control returned back to the calling function.

6. Function will return 16 to the calling function.(i.e main)
7. Returned value will be copied into variable.
8. `printf` will gets executed.
9. main function ends.
10. C program terminates.

Example: Calling a Function

```
#include <stdio.h>

/* function returning the max between two numbers */
int max(int num1, int num2)
{
    /* local variable declaration */
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}

void main ()
{
    /* local variable definition */
    int a = 100;
    int b = 200;
    int ret;
    /* calling a function to get max value */
    ret = max(a, b);
    printf( "Max value is : %d\n", ret );
}
```

Type of Function call

While calling a function, there are two ways that arguments can be passed to a function:

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

Call by value

Consider the function swap() definition as follows.

```
/* function definition to swap the values */  
void swap(int x, int y)  
{  
    int temp;  
    temp = x; /* save the value of x */  
    x = y;    /* put y into x */  
    y = temp; /* put temp into y */  
}
```

Call by reference

To pass the value by reference, argument pointers are passed to the functions just like any other value.

So accordingly you need to declare the function parameters as pointer types as in the following function **swap()**, which exchanges the values of the two integer variables pointed to by its arguments.

```
/* function definition to swap the values */  
void swap(int *x, int *y)  
{  
    int temp;  
    temp = *x; /* save the value of x */  
    *x = *y;   /* put y into x */  
    *y = temp; /* put temp into y */  
}
```

Global and local variables

Local variable:

- A local variable is a variable that is declared inside a function.
- A local variable can only be used in the function where it is declared.

Global variable:

- A global variable is a variable that is declared outside **all** functions.
- A global variable can be used in all functions.

Global and local variables

```
#include <stdio.h>


// Global variables
int A;
int B;

int Add()
{
    return A + B;
}

void main()
{
    int answer; // Local variable
    A = 5;
    B = 7;
    answer = Add();
    printf("%d\n", answer);
    return 0;
}
```

- As you can see two global variables are declared, **A** and **B**. These variables can be used in `main()` and `Add()`.
- The local variable **answer** can only be used in `main()`.

Parameters and Arguments



```
#include<stdio.h>

void myFunction(char name[])
{
    Printf(“The Code is Executed”, name);
}

int main()
{
    myFunction(“Sabbir”);
    myFunction(“Sakil”);
    myFunction(“Habib”);
    return 0;
}
```

Multiple Parameter

```
#include<stdio.h>

Void myFunction(char name[], int age )
{
printf("The Code is Executed", name, age);
}


int main()
{
myFunction("Sabbir", 28, "Teaching");
myFunction("Sakil", 26);
myFunction("Habib", 26);
return 0;
}
```

Return Value

```
#include<stdio.h>

int myFunction(int x )
{
    return 5+x;
}

int main()
{
    printf ( “Value Pass”, myFunction(3);
    return 0;
}
```



SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc.
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.
8	stdarg.h	Variable argument functions are defined in this header file.
9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions.
11	locale.h	This file contains locale functions.
12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.

Recursion

- ❑ Recursion is the process of repeating items in a self-similar way. . Recursion is the technique of making a function call itself. Same applies in programming languages as well where if a programming allows you to call a function inside the same function that is called recursive call of the function as follows..

```
void recursion()  
{  
    recursion(); /* function calls itself */  
}  
  
int main()  
{  
    recursion();  
}
```

- ❑ The C programming language supports recursion, i.e., a function to call itself.
- ❑ But while using recursion, programmers need to be careful to define an exit condition (base condition) from the function, otherwise it will go in infinite loop.
- ❑ Recursive function are very useful to solve many mathematical problems like to calculate factorial of a number, generating Fibonacci series, etc.

Write a C program to add a range of numbers together using recursion

```
#include <stdio.h>
```

```
int sum(int k);
```

```
int main() {  
    int result = sum(10);  
    printf("%d", result);  
    return 0;  
}
```

```
int sum(int k) {  
    if (k > 0) {  
        return k + sum(k - 1);  
    } else {  
        return 0;  
    }  
}
```

Example Explained

When the `sum()` function is called, it adds parameter `k` to the sum of all numbers smaller than `k` and returns the result. When `k` becomes 0, the function just returns 0. When running, the program follows these steps:

```
10 + sum(9)
10 + ( 9 + sum(8) )
10 + ( 9 + ( 8 + sum(7) ) )
...
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + sum(0)
10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 + 0
```

Since the function does not call itself when `k` is 0, the program stops there and returns the result.

Write a C program to find the factorial of a number using recursion

```
#include <stdio.h>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want
to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
```

```
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}
```

We can understand the above program of the recursive method call by the figure given below:

```
return 5 * factorial(4) = 120
└─ return 4 * factorial(3) = 24
    └─ return 3 * factorial(2) = 6
        └─ return 2 * factorial(1) = 2
            └─ return 1 * factorial(0) = 1
```

javaTpoint.com

$1 * 2 * 3 * 4 * 5 = 120$

Fig: Recursion

Lab Task:

Write a C program to check a number Even or Odd using function

```
#include <stdio.h>
char* odd_even(int i);
int main()
{
    int number;
    printf("Enter an integer : ");
    scanf("%d", &number);
    printf("Result : %s", odd_even(number));
    return 0;
}
char* odd_even(int number)
{
    if (number%2 == 0)
    {
        return "YOUR NUMBER IS EVEN NUMBER";
    }
    else
    {
        return "YOUR NUMBER IS ODD NUMBER";
    }
}
```