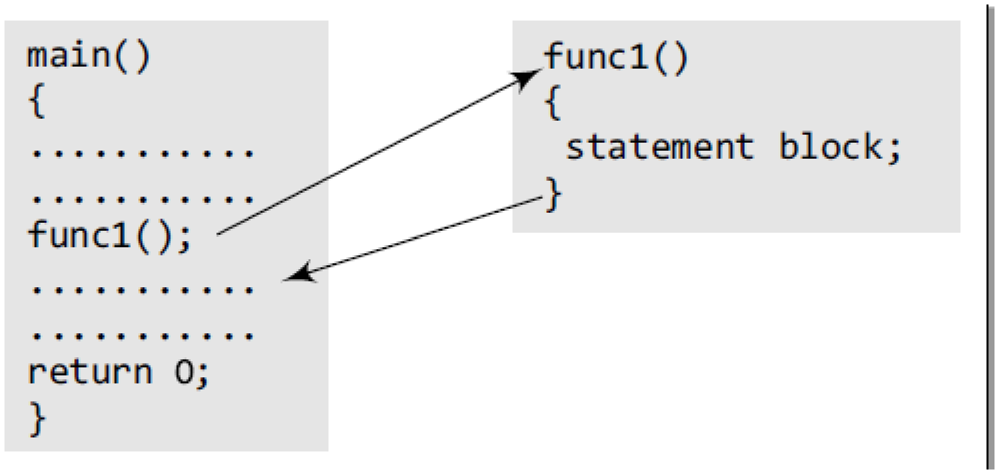
FUNCTIONS

- C enables its programmers to break up a program into segments commonly known as **functions**, each of which can be written more or less independently of the others. Every function in the program is supposed to perform a well-defined task. Therefore, the program code of one function is completely insulated from the other functions.
- Every function interfaces to the outside world in terms of how information is transferred to it and how results generated by the function are transmitted back from it. This interface is basically specified by the **function name**.



```
main()
{
...........
...........
func1();
...........
...........
return 0;
}
```

```
func1()
{
 statement block;
}
```

- ***main()*** calls a function named ***func1()***. Therefore, main() is known as the ***calling function*** and func1() is known as the ***called function***.
- The moment the compiler encounters a function call, the control jumps to the statements that are a part of the called function. After the called function is executed, the control is returned to the calling program.
- The main() function can call as many functions as it wants and as many times as it wants.
- Not only main(), any function can call any other function.

Why are Functions Needed.

- Dividing the program into separate well-defined functions facilitates each function to be written and tested separately. This simplifies the process of getting the total program to work.
- Understanding, coding, and testing multiple separate functions is easier than doing the same for one big function.
- If a big program has to be developed without using any function other than `main()`, then there will be countless lines in the `main()` function and maintaining that program will be a difficult task.
- All the libraries in C contain a set of functions that the programmers are free to use in their programs. These functions have been pre-written and pre-tested, so the programmers can use them without worrying about their code details. This speeds up program development, by allowing the programmer to concentrate only on the code that he has to write.
- Like C libraries, programmers can also write their own functions and use them from different points in the main program or any other program that needs its functionalities.
- When a big program is broken into comparatively smaller functions, then different programmers working on that project can divide the workload by writing different functions.

Definition of Terms.

- A function *f* that uses another function *g* is known as the *calling function,* and *g* is known as the *called function*.
- The inputs that a function takes are known as *arguments*.
- When a called function returns some result back to the calling function, it is said to *return* that result.
- The calling function may or may not pass *parameters* to the called function. If the called function accepts arguments, the calling function will pass parameters, else not.
- *Function declaration* is a declaration statement that identifies a function's name, a list of arguments that it accepts, and the type of data it returns.
- *Function definition* consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function.

Function Declaration.

- Before using a function, the compiler must know the number of parameters and the type of parameters that the function expects to receive and the data type of value that it will return to the calling program.
- The general format for declaring a function that accepts arguments and returns a value as result can be given as:
  return_data_type function_name(data_type variable1, data_type variable2,..);
- Note that the number of arguments and the order of arguments in the function header must be the same as that given in the function declaration statement.

Function call

The function call statement invokes the function. When a function is invoked, the compiler jumps to the called function to execute the statements that are a part of that function. Once the called function is executed, the program control passes back to the calling function. A function call statement has the following syntax:

```
function_name(variable1, variable2, ...);
```

The following points are to be noted while calling a function:

- Function name and the number and the type of arguments in the function call must be same as that given in the function declaration and the function header of the function definition.
- Names (and not the types) of variables in function declaration, function call, and header of function definition may vary.
- Arguments may be passed in the form of expressions to the called function. In such a case, arguments are first evaluated and converted to the type of formal parameter and then the body of the function gets executed.
- If the return type of the function is not `void`, then the value returned by the called function may be assigned to some variable as given below.

```
variable_name = function_name(variable1, variable2, ...);
```

```
#include <stdio.h>
int evenodd(int); //FUNCTION DECLARATION
int main(){
    int num, flag;
    printf("\n Enter the number : ");
    scanf("%d", &num);
    flag = evenodd(num); //FUNCTION CALL
    if (flag == 1)
        printf("\n %d is EVEN", num);
    else
        printf("\n %d is ODD", num);
    return 0;
}
    int evenodd(int a) // FUNCTION HEADER
{
    // FUNCTION BODY
    if(a%2 == 0)
        return 1;
    else
        return 0;
}
```

Exercise.

1. Ammend your calculator code to have individual functions handling its operation. These functions need to be callable by the main() function and return their response to main.
2. The Fibonacci sequence is a series of numbers where each number is the sum of the two preceding ones, usually starting with 0 and 1. The sequence starts like this: 0, 1, 1, 2, 3, 5, 8, 13, 21, and so on. Mathematically, the Fibonacci sequence can be defined recursively as follows: F(0) = 0

   F(1) = 1

   F(n) = F(n-1) + F(n-2) for n > 1

   Write a C program to generate the Fibonacci series up to a given number n.

3. Write a C program to check whether a given number n is a palindrome or not.
4. An Armstrong number (also known as narcissistic number, plenary number, or pluperfect digital invariant) is a number that is equal to the sum of its own digits each raised to the power of the number of digits. For example, 153 is an Armstrong number because:

   $1^3+5^3+3^3 = 153$

   Write a C program to check whether a given number n is an Armstrong number or not.

This assessment is to be completed and uploaded to github by midnight Friday 16th 2024. Happy Coding.