Modular Programming with Functions

Functions or modules are sets of statements that typically perform an operation or that compute a value.

To maintain simplicity and readability in longer and more complex problem solutions, we develop programs that use a main function plus additional functions instead of using one long main function. By separating a solution into a group of modules, each module is simpler and easier to understand, thus adhering to the basic guidelines of structured programming as discussed earlier. We follow the process “divide and conquer.”

Once is a module is written and tested, it can be used in other problems also. The last property is called reusability.

The use of modules (called modularity) often reduces the overall length of the program. The use of modules that have been written to accomplish specific tasks supports the concept of abstraction. By using abstraction, we can reduce the development time of software as we increase its quality.

Advantages of using modules.
- A module can be written and tested separately from other parts of the solution, thus module development can be done in parallel for large projects.
- A module is a small part of the solution; thus, testing it separately is easier.
- Once a module is tested carefully, it does not need to be re-tested before it can be used in new problem solutions.
- The use of modules usually reduces the length of a program, making it more readable.
- The use of modules promotes the concept of abstraction which allows the programmer to “hide” the details in modules. This allows us to use modules in a functional sense without being concerned about the specific details.

Programmer-defined functions

The execution of a program always begins with the main function. Additional functions are called or invoked, when the program encounters function names. These additional functions must be defined in the file containing the main function or in another available file or library of files. If the function is a system library file, such as the Standard C library, it is often called a library function, other functions are called programmer-written or programmer-defined functions.

Example

\[ f(x) = \frac{\sin(x)}{x}. \]  

L’Hopital’s theorem can be used to prove that sinc(0)=1.

/* This program prints 21 values of the sinc function in the interval */
/* [a, b] using a programmer-defined function */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
main()
{
    /* declare variables and function prototypes */

    int k;
    double a, b, x_incr, new_x;
    double sinc(double x);

    /* get the interval endpoints from the user */

    printf("Enter endpoints a and b, (a<b): \n");
    scanf("%lf %lf", &a, &b);
    x_incr = (b-a)/20;

    /* compute and print table of sinc(x) values */

    printf("x and sinc(x) \n");
    for (k=0; k <= 20; k++)
    {
        new_x = a + k*x_incr;
        printf("%f %f \n", new_x, sinc(new_x));
    }
    
    
    
    
    
}

/* This function evaluates the sinc function */

double sinc(double x)
{
    if (fabs(x) < 0.0001)
        return 1.0;
    else
        return sin(x) / x;
}

In the following program we include the statements to perform the computation of sinc(x) in the main function.

/* This program prints 21 values of the sinc function in the interval */
/* [a, b] using computations within the main function */
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

main()
{
    /*declare variables*/
    int k;
    double a, b, x_incr, new_x, sinc_x;

    /*get the interval endpoints from the user*/
    printf("Enter endpoints a and b, (a<b): \n");
    scanf("%lf %lf", &a, &b);
    x_incr = (b-a)/20;

    /*compute and print table of sinc(x) values*/
    printf("x and sinc(x) \n");
    for (k=0; k <= 20; k++)
    {
        new_x = a + k*x_incr;
        if (fabs(new_x) < 0.0001)
            sinc_x = 1.0;
        else
            sinc_x = sin(new_x)/new_x;
        printf("%.2f %.2f \n", new_x, sinc_x);
    }
}

A function consists of a definition statement followed by declarations and statements. The first part of the definition statement defines the type of value that is returned by the function; if the function does not return a value, the type is void. The function name and parameter list follow the return-type. Thus, the general form of the function is:

return-type function_name(parameter declarations)
{
    declarations;
    statements;
}

All functions should include a return statement, which has the general form:

return expression;
A `void` function does not return a value and it has the following general form:

```c
void function_name(parameter declarations)
```

The `return` statement in a `void` function is:

```c
return;
```

The function reference in the `sinc` example is a `call-by-value` reference or a `reference-by-value`. In general a C function cannot change the value of an actual parameter.

**Parameter list**

`Formal parameters` are called the parameters defined by the definition statement of a function. Any statement that references the function must include values that correspond to the parameters; these are called `actual parameters`.

The definition statement is:

```c
double sinc(double x);
```

and the statement from the `main` program that references the function is:

```c
printf("%f %f \n", new_x, sinc(new_x));
```

The variable `x` is the formal parameter.
The variable `new_x` is the actual parameter.

Actual parameter

```
new_x  5.0
```

Formal parameter

```
x    5.0
```

**Storage class and scope**

`Local variables` are defined within a function and include the formal parameters and any other variables declared in the function. A local variable has a value when its function is being executed, but its value is not retained when the function is completed.

`Global variables` are defined outside the `main` function and any other programmer-defined functions and they can be accessed by any function within the program. However to reference a global or an external variable, a declaration within the function must include the keyword `extern` before the type designation to tell the program to look outside the function for the variable.
The automatic storage class is used to represent local variables by default. The external storage class is used to represent global variables. The static storage class is used to specify that the memory for a variable should be retained during the entire program execution.

Keyword static