Functions

1 Introduction

A ***function*** is a block of code that does a certain task. Modularizing your program into functions simplifies your program into chunks of codes. Functions can help you avoid repetition of code blocks in your program.

1.1 Function Definition

The general form of a function in C is as follows:

```
<return_type> <function_name>([<formal_parameter_list>]) {
    <function_body>
}
```

where `return_type` is the datatype of the value being returned by the function,
`function_name` is the name of the function,
`formal_parameter_list` is the list of arguments that the function expects, and
`function_body` is the series of commands the function will execute once called.

The return_type, function_name, and parameter_list comprises the ***function signature***.

EXAMPLE: A function that adds two numbers:

```
int sum(int a, int b) {
    int sum;
    sum = a + b;
    return sum;
}
```

1.2 Function Call

To use a function you defined, you must invoke it by calling the function. The syntax of a function call is as follows:

```
<function_name> ( <actual_parameter_list> );
```

where `function_name` is the name of the function to be called, and
`actual_parameter_list` is the list of parameters to be passed to the function.

We can call the function in the previous example by typing sum(4, 5); or sum(3,4);

However, since the function returns a value, we must make sure that the returned value is assigned to a variable. In this case, we will put the function call in an assignment statement. Below is the usual syntax:

```
variable = <function_name> ( <actual_parameter_list> );
```

where `variable` is where the returned value will be assigned,
`function_name and actual_parameter_list` is just like in the previous definition.

So the function defined above must be called like in the function call below:

```
answer = sum(4, 5);
```

1.3 Function Prototype

In C, each function should be defined in the lines before a function call, just like in Python.

```
#THIS WOULD NOT WORK
print(getSum(4,5))

def getSum(a, b):
    return a + b
```

```
#THIS WOULD WORK
   def getSum(a, b):
       return a + b

    print(getSum(4,5))
```

However, in C, you can define a function even after a function call as long as you declare it first.

To declare a function, you only need to type the function's signature followed by a semicolon. So to declare the previously defined example function, you need to type:

```
int sum(int a, int b);
```

Another way of declaring a function is by typing its function signature but omitting the variable names of the parameters, just like in the example below:

```
int sum(int, int);
```

TIPS ON USING FUNCTIONS

1. A non-void function can only return AT MOST one value.
2. The receiving variable of the return value of the function must be of the same type, i.e., if the return type of a function is int, then the variable catching the return value must also be int.
3. The number and order of actual parameters in the function call must correspond to the formal parameter list of the definition of the called function.
4. All local variables must be declared at the start of the function to avoid errors.
5. All functions must be defined/written before the function that uses them. If necessary to place the definitions after the functions that use them, function prototypes may be used.
6. A return statement automatically halts the execution of the function and returns the corresponding value it must return.

2 Variable Scoping

In a C program, we have what we call local and global variables.

Local variables are variables that can only be accessed inside its own block-meaning inside its own {}. Formal parameters are also local variables in the function where they belong.

Global variables, on the other hand, are variables that can be accessed by all functions. It is declared outside the scope of any function (i.e., there is no {} surrounding it).

NOTE: Global variables are powerful but can cause confusion in your program so it's best to avoid them.