Chapter 6: User Defined Functions

Overview of Chapter

Lesson 1: Introduction to Functions

A function is a block of code that accepts in an input and produces an output. Breaking down a lengthy piece of code into functional units would make the code modular thus easier to maintain and debug. Since chapter 2, functions are already introduced, such as the main function itself and the built in math functions. For this chapter, you would create your own function, with parameters and usability according to specification to address the problem.

Learning Outcome(s)

- To write a function value returning and void function
- To test the usability of the function by calling it on function main.

Writing Functions in C

To write a function, you must specify the return type, function name, arguments (or parameters), then the function body. In general the function is written as:

```
returnDataType functionName (arg1, arg2, argN)
{
//function body
}
```

A function can have no arguments, or would not return anything or could be both. If a function would not return anything you would use the void keyword as datatype (i.e. void basically means nothing), for example:

```
void printRevCount(int numOfTimes)
{
      for (numOfTimes;numOfTimes>0;numOfTimes--)
      {
      printf("Hello!\n");

      }
}
```

When a function would have a return statement, you must specify the return data type and may or may not place void on the argument (parameter) block. For example:

```
int doSum (void)
{
      int i=0, sum=0;
      for (i=0;i<5;i++)
      {
            sum+=i;
      }
  return sum;
}
```

However, the most practical way of writing a function is that it would have a return data type and argument(s). For instance you would take the sum of factors from an argument (i.e. if argument is 5, the sum of factors is 6 (1+5)):

```
int sumOfFactors(int x)
{
      int i=1, sum=0;
      for (i=1;i<=x;i++)
      {
            if (x%i==0)
            {
               sum+=i;
            }
      }
   return sum;
}
```

Using Functions in C

When a function is located after the function main, it is important that you place a function prototype (i.e. a function without a function body). Then when invoking (calling) a function, you need to indicate the function name, the arguments (if there are any). The complete example of the sum of factors with function prototype and function body is shown as:

```
#include <stdio.h>
//function prototype
int sumOfFactors(int x);
int main()
{
    int var=0, x=5;
    //invoking function
    var = sumOfFactors(x);
    printf("Sum of Factors %d: %d ",x,var);
}
int sumOfFactors(int x)
{
      int i=1, sum=0;
    for (i=1;i<=x;i++)
    {
            if (x%i==0)
        {
            sum+=i;
        }
    }
    return sum;
}
```

Lesson 2: Arguments and Return Values

Arguments are the inputs of a function. They are the data necessary for the function to operate. Meanwhile the return values are the output. This are the result after the function finished executing.

Learning Outcome(s)

- To perform pass by value and pass by reference to a function
- To implement a function with a return value.

Integral Variables and Arrays as Arguments

When passing integral variables, the usual method, is to a pass it by value. It is also important to take note that the actual parameter (i.e. arguments of the invoking function) is similar to the formal parameter (i.e. arguments of the invoked function). For example:

```
        int z=5;
         sumXY (4, z);
//copied to   ↓   ↓
int sumXY(int x, int y)
{
      return x + y;
}
//the output would be 9 since x=4,y=z=5
```

However, when an array is passed into a function it is passed by reference. Since, it would be inefficient to duplicate another local copy of the array within the function body. For example:

```
      int a[5] = {4,5,6,7,8};
//… points to        ↑
void displayContent(int a [])
{
      int i=0;
      for (i=0;i<5;i++)
      {
            printf("%d,",a[i]);
      }
}
```

Return values

As noted on the previous sub-chapter, void function do not have any return value. Moreover, there is only one return value per function. Refer to the code below

```
//erroneous since a void function would not return any value
void foo()
{
      return 0;
}

//erroneous since only one return value per function
int foo()
{
```

```
	return 0,1;
}

//correct! one return value is indicated for a non-void function
int foo()
{
	return 1;
}
```