LECTURE 04

Functions

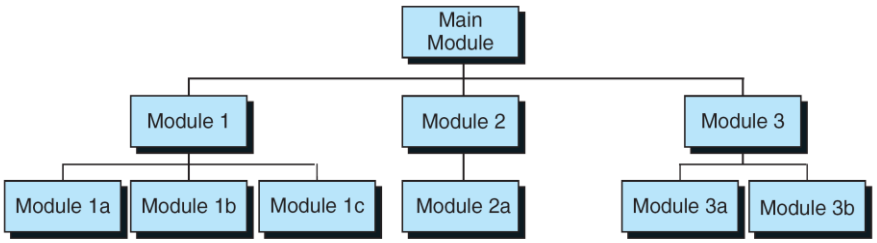
Outcomes

1. To understand the basic function designs and working principals.
2. To run simple C program using user-defined and standard library functions.

Contents

1. Introduction to Functions

In programming, it is easy for us if we have one a simple problem to be solved for example addition of two numbers. As we consider larger and larger problems, you will discover that it is possible to understand all thousands of code lines in a fast manner. Therefore, in good programming practice, we have to reduce those code lines into many sub-module. Lets say we have to calculate addition, multiplication, and division of two integers, we can break this three calculations into three sub-module for instance module 1 (addition), module 2 (multiplication), and module 3 (division). These modules can communicate each other by sending or receiving data between them. In C programming, we called this module as functions and the structure of the above example is illustrated in the next figure.



Notice that **the main module (function) is always one and only one** in C programming. The main function in C code is started with `int main() {` and ended with `return 0; }`. Module 1, 2, and 3 can have more other sub module if you want. So what is the important of using functions? The major advantages are:

1. As already mentioned, the problems can be divided into understandable and manageable steps.
2. Functions provide a way to reuse code that is required. For instance, if you have many 5 pairs of number to compute addition, we could write the code to compute addition five times, but this would take a lot of effort. Also, if we needed to change calculation of addition, we would have to find all five places and change each of them.

2. User-Defined Functions

I'm using a same example as above to describe user-defined functions. We have three operations to be done which are addition, multiplication, and division. We also have two integers as input and three results of the operations. We will write C program as usual without using functions which is as follow:

```
/*C Program to illustrate user-defined functions
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
int main ()
{
      int firstNumber;
      int secondNumber;
      int resultAdd;
      int resultMultiply;
      float resultDivision;
      printf("Enter first number:\n");
      scanf("%d", &firstNumber);
      printf("Enter second number:\n");
      scanf("%d", &secondNumber);
      printf("Result of Addition: %d\n",(firstNumber+secondNumber));
      printf("Result of Multiplication: %d\n",(firstNumber*secondNumber));
      printf("Result of Division: %f\n",(firstNumber/secondNumber));
      return 0;
}
```

I hope you understand the above program since it is a very easy example. Now, lets change this code using user-defined functions. In this case, we will have four functions: (i) main function, (ii) addition function, (iii) multiplication function, and (iv) division function. There are three main things that you have to remember when declaring a function. **(i) function prototype, (ii) function call, and (iii) function definition**. Function prototype is a declaration of function. It always situated after header file (e.g. `#include<stdio.h>`) and before `int main ()`. Function call is responsible to call the respective function in `int main()` function. Finally, function definition is contents (e.g. computation) of the function. Below is the code using functions based on the above example.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 int addition(int x, int y);       //function prototype for addition
```

```
4 int multiplication(int x, int y); //function prototype for multiplication
5 float division(float x, float y); //function prototype for division
6 int main ()
7 {
8     int firstNumber;
9     int secondNumber;
10    int resultAdd;
11    int resultMultiply;
12    float resultDivision;
13    printf("Enter first number:\n");
14    scanf("%d", &firstNumber);
15    printf("Enter second number:\n");
16    scanf("%d", &secondNumber);
      //function call for addition
17    resultAdd = addition(firstNumber,secondNumber);
      //function call for multiplication
18    resultMultiply = multiplication(firstNumber,secondNumber);
      //function call for division
19    resultDivision = division(firstNumber,secondNumber);
20    printf("Addition=%d\nMultiplication=%d\nDivision=%f",resultAdd,
21    resultMultiply,resultDivision);
22    return 0;
23 }
24 int addition(int x, int y)       //function definition for addition
25 {
26    return(x+y);
27 }
28 int multiplication(int x, int y)  //function definition for multiplication
29 {
30    return(x*y);
31 }
32 float division(float x, float y) //function definition for division
33 {
34    return(x/y);
35 }
```

The general format for a prototype is:

return-type function_name (var_type var1, ..., var_type varN);

Lets compare with our code in line 3 to line 5.

```
3 int addition(int x, int y);        //function prototype for addition
4 int multiplication(int x, int y); //function prototype for multiplication
5 float division(float x, float y); //function prototype for division
```

I'm naming the **function_name** as `addition`, `multiplication` and `division` respectively. You can name this **function_name** with any name that you like. For addition and multiplication, I'm using `int` as **return-type** and **`var_type`** because the `return` value to `main` function is `int`. It is different to division function where I'm using `float` since the division result sometimes give floating number. In **`( var_type var1, ..., var_type varN )`**, we put two variables `int x` and `int y` for `addition` and `multiplication` functions and `float x` and `float y` for `division` function since we have two input integer value to be process. You can put the same name as in `main` function which is `int firstNumber` and `int secondNumber`, but I want it to in different name so that you can differentiate where is variable in `main` function and user-defined function. The function call is in line 17 to line 19 as follows:

```
      //function call for addition
17    resultAdd = addition(firstNumber,secondNumber);
      //function call for multiplication
18    resultMultiply = multiplication(firstNumber,secondNumber);
      //function call for division
19    resultDivision = division(firstNumber,secondNumber);
```

We declared `resultAdd` and `resultsMultiply` as `int`. So what does this mean? This mean that the `return` value of addition and multiplication will be hold by `resultAdd` and `resultsMultiply` respectively. The syntax `(firstNumber,secondNumber)` is referred to two inserted values `int firstNumber;` and `int secondNumber;` are send to each function. I put `resultsMultiply` as `float` since sometimes the result for division is in floating number. While calling a function, there are two ways in which arguments can be passed to a function: call by value and call by reference. For this class we only use **call by value**. This method copies the actual value of an argument into the formal parameter of the function as explains above. Meanwhile, call by reference copies the address of an argument into the formal parameter. This will involves the use of pointer and memory address which you will learn later on programming for engineer subject.

Finally, the function definition is in line 24 to 35. You can see at first line in each definition line 24, 28, and 32 are same with function prototype with has been declared first before `int main()`. If we want to change the function definition header, we also have to change the function prototype and vice versa.

The most important thing in function definition is the `return` statement must only contain one value. You cannot `return` many values in one function. To explain how this function work, we use only one function example as below.

```
/*C Program to illustrate user-defined functions – with return value
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdio.h>
#include<stdlib.h>
#include <stdio.h>
int add(int p,int q);
int main ()
{
      int a,b,c;
      printf("Enter two numbers\n");
      scanf("%d %d",&a,&b);
      c=add(a,b);
      printf("Sum of %d and %d is %d",a,b,c);
      return 0;
}
int add(int p,int q)
{
      int result;
      result=p+q;
      return(result);
}
```

The following table presents the explanations of the above C codes.

Codes	Variable	Value	Explanations

`int main ()`
`{`
`int a,b,c;`
`}`
	`a`	10	Lets say user inserted 10 (for `a`) at `scanf("%d %d",&a,&b);`

	`b`	5	Lets say user inserted 5 (for `b`) at `scanf("%d %d",&a,&b);`
	`c`	15	`c=add(a,b);` `c` is calling `add` function which send two values `a` and `b` to `add` function

`int add(int p,int q)`
`{`
`int result;`
`}`
	`p`	10	This `p` value is holding `a` value that receive from `add(a,b)`. `p` is declare in `add` function

	`q`	5	This `q` value is holding `b` value that receive from `add(a,b)`. `q` is declare in `add` function
	`result`	15	`result=p+q; = result=10+5=15`. The `result` value is then return to `main` function to variable `c`

The above examples are user-defined functions with `return` value. Another type of user-defined functions is without `return` value. In this case, we are using `void` for function `return-type`. Below is the user-defined functions without `return` value. I'm using the same example as the above example.

```
/*C Program to illustrate user-defined functions – without return value
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include<stdlib.h>
#include <stdio.h>
void add(int p,int q);
int main ()
{
    int a,b;
    printf("Enter two numbers\n");
    scanf("%d %d",&a,&b);
    add(a,b);
    return 0;
}
void add(int p,int q)
{
    int result;
    result=p+q;
    printf("Sum of %d and %d is %d",p,q,result);
    return;
}
```

As seen in the above codes, I have change `int add(int p,int q);` to `void add(int p,int q);` since we do not want to `return` any values back to `main` function. When we call `add(a,b);`, the values of `a` and `b` are send to `add` function. In `add` function, the values of `a` and `b` are pointed to `p` and `q` respectively. Finally the `result` is an addition of `p` and `q`. Since we are using `void`, the value of `result` cannot be send back to `main` function. Therefore, we print the results directly through this `add` function.

3. Standard Library Functions

C provides a rich collection of standard library functions whose definition have been written and ready to be used in your programs. For example, we have arithmetic/math functions which responsible to perform mathematical operations. Other than that, we have time functions which purposely to interact with system time routine. Following is the table for arithmetic and time function definition.

Functions	Description	Example
sqrt(x)	square root of *x*	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
exp(x)	exponential function e^x	exp(1.0) is 2.718282 exp(2.0) is 7.389056
log(x)	natural logarithm of *x* (base *e*)	log(2.718282) is 1.0 log(7.389056) is 2.0
log10(x)	logarithm of *x* (base 10)	log10(1.0) is 0.0 log10(10.0) is 1.0 log10(100.0) is 2.0
fabs(x)	absolute value of *x*	fabs(5.0) is 5.0 fabs(0.0) is 0.0 fabs(-5.0) is 5.0
ceil(x)	rounds *x* to the smallest integer not less than *x*	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
floor(x)	rounds *x* to the largest integer not greater than *x*	floor(9.2) is 9.0 floor(-9.8) is -10.0
pow(x, y)	*x* raised to power *y* (x^y)	pow(2, 7) is 128.0 pow(9, .5) is 3.0
fmod(x, y)	remainder of *x*/*y* as a floating point number	fmod(13.657, 2.333) is 1.992
sin(x)	trigonometric sine of *x* (*x* in radians)	sin(0.0) is 0.0
cos(x)	trigonometric cosine of *x* (*x* in radians)	cos(0.0) is 1.0
tan(x)	trigonometric tangent of *x* (*x* in radians)	tan(0.0) is 0.0
Commonly used arithmetic/math library functions.		

Functions	Description
setdate()	This function used to modify the system date
getdate()	This function is used to get the CPU time
clock()	This function is used to get current system time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
strftime()	This function is used to modify the actual time format

Functions	Description
mktime()	This function interprets tm structure as calendar time
localtime()	This function shares the tm structure that contains date and time information
gmtime()	This function shares the tm structure that contains date and time information
ctime()	This function is used to return string that contains date and time information
asctime()	Tm structure contents are interpreted by this function as calendar time. This time is converted into string.
Commonly used arithmetic/math library functions.	

To use these functions, we must include their function declarations. The function declarations for these functions are grouped together and collected in several header files. Instead of adding the individual declarations of each function, therefore, we simply include the headers at the top of our program. Below are the simple program using arithmetic functions. Notice that in each of program, we must put `include <math.h>` at the top of program. Then for square root computation, we just only call `sqrt(value)` function and this function will return the result of this computation. Same goes to second example using power function, we just only call `pow(value,value)` function and this function will return the result of this computation.

```
/*C Program to illustrate the use of arithmetic/math functions
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
   printf ("sqrt of 16 = %f\n", sqrt (16) );
   printf ("sqrt of  2 = %f\n", sqrt (2) );
   return 0;
}
```

```
sqrt of 16 = 4.000000
sqrt of 2 = 1.414214
```

```
/*C Program to illustrate the use of arithmetic/math functions
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
int main()
{
```

```
    printf ("2 power 4 = %f\n", pow (2.0, 4.0));
    printf ("5 power 3 = %f\n", pow (5, 3));
    return 0;
}
```

```
2 power 4 = 16.000000
5 power 3 = 125.000000
```

Below are the simple program using time functions. Notice that in each of program, we must put `include <time.h>` at the top of program. Then to get system time, we just only call `time()` function and this function will return the current system time. Same goes to second example using `time` and `difftime` functions, we just only call `time()` and `difftime()` function and these functions will return the system time and difference between two given times respectively. The `time()` function is giving the current system time in seconds.

```
/*C Program to illustrate the use of time functions
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main()
{
    int i;
    time_t CPU_time_1 = time(NULL);
    printf("CPU start time is : %d \n", CPU_time_1);
    for(i = 0; i < 1500000000; i++);
    time_t CPU_time_2 = time(NULL);
    printf("CPU end time is : %d", CPU_time_2);
}
```

```
CPU start time is : 1476166752
CPU end time is : 1476166756
```

```
/*C Program to illustrate the use of time functions
Written by: AFAN, FKP, UMP    Date: September 2016*/
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
int main()
{
    time_t begin,end;
```

```
    long i;
    begin= time(NULL);
    for(i = 0; i < 1500000000; i++);
    end = time(NULL);
    printf("for loop used %.f seconds to complete the execution\n",
           difftime(end, begin));
    return 0;
}
```

```
for loop used 5 seconds to complete the execution
```