FUNCTIONS

CMSC 21: Modular Programming Review

Prepared by: Martee Aaron Lim Manalang

The best way to develop a large program is to divide it into several smaller program modules. In C these modules are called functions. Segmenting a program into manageable chunks is a very important aspect to programming. Some of the reasons for doing this are the following:

- Dividing the program into a number of separate functions allows each function to be written and tested separately. This greatly simplifies the process of getting the total program to work.

- Several separate functions are easier to handle and understand than one huge function.

How do we define functions in C programs?

```
[return type] function_name ( [parameter list] )
{
            declarations;
            statement;
            ...
            statement;
}
```

On the right is a sample program that uses **more than one function** aside from the **`main() function`**. You'll see how we *define functions* and how we write *function calls*.

function calls can be made not only from the **`main()`** function but from ANY function.

Also, a function can be called ANY number of times.

```
#include <stdio.h>

float radius; //global variable(s)

void compute_circ();
void compute_area();
float square(float);

main(){
   printf("Input radius: ");
   scanf("%f", &radius);

   compute_circ(); //function call
   compute_area(); //function call
}

void compute_circ(){
   float circ; //local variable(s)
   circ = 2 * 3.14 * radius;
   printf("circumference = %f \n", circ);
}

void compute_area(){
   float area; //local variable(s)
   area = 3.14 * square(radius);
   printf("area = %f \n", area);
}

float square(float x){
   return x*x;
}
```

"A function should do only one thing".

Not all functions need to take arguments or return a value. **`void`** in the above example indicates that a function does not return anything. Also, if a function does not take any parameter(s), its parameter list is empty, it can contain the keyword void but that style is now out favor.

A function that does not return any value

```
void printHello()
{
       printf("hello");
}
```

A function that returns a value

```
int computeSum( int x, int y )
{
        int intSum;
        intSum = x + y;
        return intSum;
}
```

IMPLEMENTING A FUNCTION

A program with function declaration/prototype

```
#include <stdio.h>

int computeTwice(int );     }  Function Prototype

main( )

{

     int intNum = 13;

     int intA = 1;

     int intB = 2;

     intA = computeTwice(intA);             }  Function Calls

     intB = computeTwice(intB + intNum);

}


/* This function computes the double of a number */

int computeTwice (int intNum)

{

    int intResult = intNum * 2;          }  Function Definition

    return intResult;

}
```

Things to notice:

- The expression passed to a function by its caller is called the actual parameter - such as "intA" and "intB + intNum" (see Line 9). The parameter local to the function is called the formal parameter - such as the "intNum" in "int computeTwice(int intNum)" (see Line 13).

- Parameters are passed "by value". The value in the actual parameter is copied into the function's formal parameter just before the function begins executing. So, any modification to the arguments does not affect the values of the variables used in the function call.

- The variables local to the function computeTwice(), intNum and intResult, only exist temporarily while computeTwice() is executing. In the same way, as variables local to main and other functions exist only when function main() is executing.

- Function declarations/prototypes go before the main function.

- Function definitions go after the main function.

LOCAL VARIABLES vs. GLOBAL VARIABLES

Local Variables	Global Variables
• Parameters and variables declared inside the definition of a function. • Exist only inside the function body. • Once the function returns, the variables no longer exist!	• Variables declared outside any of the function definition. • Any function can access/change global variables.

Note: Global variables become hard to manage when you begin to work with multiple functions. Thus, it is a good practice to declare variables only in the scope where they are needed.

Practice Exercise:

Write a function celsius() to convert degrees Fahrenheit to degrees Celsius. (The conversion formula is °C = 5/9 * (°F - 32). Write a function fahrenheit() to convert degrees Celsius to degrees Fahrenheit. (The conversion formula is °F =(9/5 * C) + 32).

References:

- http://www.aspfree.com/c/a/Code-Examples/Programming-in-C/6/
- **C How to Program** by Deitel & Deitel
- CMSC 21 handouts prepared by Ms. Joycee P. Centeno
- CMSC 11 handouts by Maverick C. Crisostomo