Functions, Part 1 of 3

Topics
- Using Predefined Functions
- Programmer-Defined Functions
- Using Input Parameters
- Function Header Comments

Reading
- Sections 5.1 - 5.8

Review of Structured Programming
- Structured programming is a problem solving strategy and a programming methodology that includes the following guidelines:
  - The program uses only the sequence, selection, and repetition control structures.
  - The flow of control in the program should be as simple as possible.
  - The construction of a program embodies top-down design.

Review of Top-Down Design
- Involves repeatedly decomposing a problem into smaller problems.
- Eventually leads to a collection of small problems or tasks each of which can be easily coded.
- The function construct in C is used to write code for these small, simple problems.
Functions

- A C program is made up of one or more functions, one of which is `main()`. Execution always begins with `main()`, no matter where it is placed in the program. By convention, `main()` is located before all other functions.
- When program control encounters a function name, the function is called (invoked).
  - Program control passes to the function.
  - The function is executed.
  - Control is passed back to the calling function.

Sample Function Call

```c
#include <stdio.h>

int main ( )
{
    printf is the name of a predefined function in the stdio library
    printf ("Hello World\n");
    return 0;
}
```

This statement is known as a function call

Functions (cont.)

- We have used three predefined functions so far:
  - `printf`
  - `scanf`
  - `getchar`
- Programmers can write their own functions.
- Typically, each module in a program’s design hierarchy chart is implemented as a function.
- C function names follow the same naming rules as C variables.
Sample Programmer-Defined Function

```c
#include <stdio.h>

void printMessage ( void ) ;

testMessage ( ) ;

void main ( )
{ noextra ;

void printMessage ( void )
{ noextra ;

print ( "A message for you:\n\nHave a nice day!\n\n\" ) ;

print ( "Hello universe!\n\n\" ) ;

print ( "Have a nice day!\n\n\" ) ;
}
}

void printMessage ( void )
{ noextra ;

print ( "A message for you:\n\nHave a nice day!\n\n\" ) ;

print ( "Hello universe!\n\n\" ) ;

print ( "Have a nice day!\n\n\" ) ;
}

Examining printMessage

#include <stdio.h>

void printMessage ( void ) ;

testMessage ( ) ;

void main ( )
{ noextra ;

void printMessage ( void )
{ noextra ;

print ( "A message for you:\n\nHave a nice day!\n\n\" ) ;

print ( "Hello universe!\n\n\" ) ;

print ( "Have a nice day!\n\n\" ) ;
}
}

The Function Prototype

- Informs the compiler that there will be a function defined later that:
  - returns this type
  - has this name
  - takes these arguments

- Needed because the function call is made before the definition -- the compiler uses it to see if the call is made properly
The Function Call

- Passes program control to the function
- Must match the prototype in name, number of arguments, and types of arguments

```c
void printMessage(void);
int main()
{
    printMessage();
    return 0;
}
```

The Function Definition

- Control is passed to the function by the function call. The statements within the function body will then be executed.
- After the statements in the function have completed, control is passed back to the calling function, in this case `main()`. Note that the calling function does not have to be `main()`.

```c
void printMessage(void)
{
    printf("A message for you:
    Have a nice day!
\n\n\n");
}
```

General Function Definition Syntax

```c
<function type> <function name> (<parameter1>, ..., <parameter n>)
{
    <variable declaration(s)>
    <statement(s)>
}
```

- If there are no parameters, either `functionName()` OR `functionName(void)` is acceptable.
- There may be no variable declarations.
- If the function type (return type) is void, a return statement is not required, but the following are permitted:
  ```c
  return ; OR return();
  ```
void printMessage (int int)
int main ()
{
    int num;
    printf ("Enter an integer:\n");
    scanf ("%d", &num);
    printMessage (num);  // one argument of type int matches the one formal parameter of type int
    return 0;
}
void printMessage (int int)
{
    int i;
    for (i = 0; i < int; i++)
    {
        printf ("Have a nice day!\n");
    }
}

Final “Clean” C Code
#include <stdio.h>
void printMessage (int int)
int main ()
{
    int num;  // number of times to print message
    printf ("Enter an integer:\n");
    scanf ("%d", &num);
    printMessage (num);
    return 0;
}

Final “Clean” C Code (cont.)
/**************************************************************************
** printMessage - prints a message a specified number of times
** Inputs: counter - the number of times the message will be printed
** Outputs: None
**************************************************************************
void printMessage ( int int)
{
    int i;  // loop counter
    for (i = 0; i < int; i++)
    {
        printf ("Have a nice day!\n");
    }
Good Programming Practice

- Notice the function header comment before the definition of function printMessage.
- This is a good practice and is required by the 104 C Coding Standards.
- Your header comments should be neatly formatted and contain the following information:
  - function name
  - function description (what it does)
  - a list of any input parameters and their meanings
  - a list of any output parameters and their meanings
  - a description of any special conditions