Programming with C

Terry Marris November 2010

12 Modules

In the previous chapter we created some useful functions. Now we collect these functions into a module.

12.1 Header File

The header file, named *utility.h* is shown below.

```
/* utility.h: specifies some useful function prototypes */

#ifndef UTILITY
#define UTILITY

/* getString: reads a string from the keyboard, returns its length */
int getString(char string[], int maxLength);

/* getInt: returns the integer entered at the keyboard */
int getInt();

/* getDouble: returns the value of type double entered
   at the keyboard */
double getDouble();

/* doubleEquals: returns 1 if a and b are close enough to equal,
    (as determined by tolerance) otherwise returns 0 */
int doubleEquals(double a, double b, double tolerance);

#endif
```

The list of function prototypes are enclosed within

```
#ifndef UTILITY
#define UTILITY

...

#endif
```

Suppose we have several C files that each *#include*s *utility.h*. The first inclusion finds that *UTILITY* has not been defined. So, it is subsequently defined and the function prototypes up to the #endif are included in the compilation. Subsequent inclusions find that *UTILITY* has already been defined, and so compilation skips over the function prototypes to the *#endif*. This mechanism prevents multiple definitions of the same elements. The name *UTILITY* is chosen by the programmer.

The header file tells the applications programmer what he or she needs to know in order to use the functions, as in *testutilty.c* shown below.

12.2 Application

The application here is very simple: to check out functions specified in *utility.h*.

```
/* testutility.c: tests functions defined in utility.h */

#include <stdio.h>
#include "utility.h"

int main()
{
  char name[15];
  int age;
  double height;

  printf("Name? ");
  getString(name, 15);
  printf("Age in years? ");
  age = getInt();
  printf("Height in metres? ");
  height = getDouble();

  printf("Data entered: name = %s, age = %d, height = %0.2f\n",
          name, age, height);
  return 0;
}
```

Isn't that brilliant? We can use *getString()*, *getInt()* and *getDouble()* without knowing, without caring how they are implemented. All we need to know is what the functions do. KISS or what? (KISS - keep it small and simple.) We don't need to clutter up our program listings with these functions if we do not want to.

Notice that *stdio.h* is surrounded by angle brackets, < and >. These instruct the compiler to look for *stdio.h* in the system's directory or folder. But the quotation marks surrounding the *utility.h* instruct the compiler to look in the current working directory. This means that *utility.h* and the program that includes it should be in the same folder.

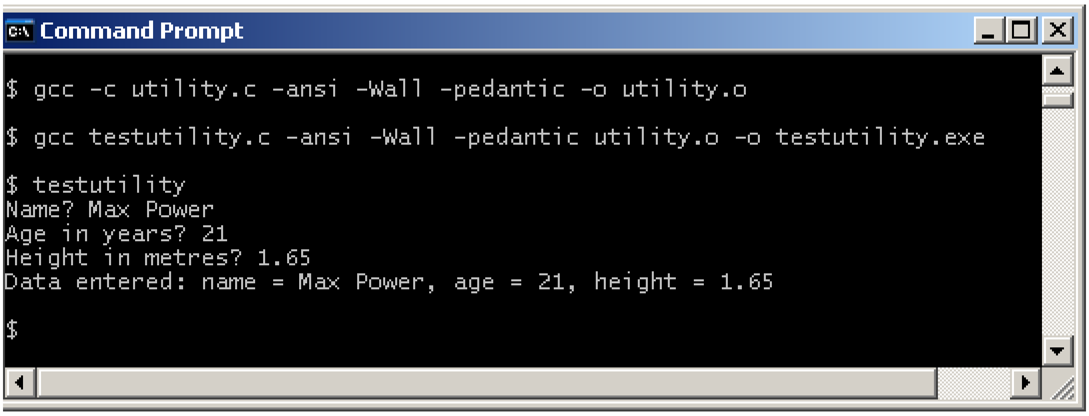
12.3 Implementation

The implementation of *getString()*, *getInt()* and *getDouble()* are shown in *utility.c* below.

```
/* utility.c: specifies some useful functions */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "utility.h"

/* getString: reads a string from the keyboard, returns its length */
int getString(char string[], int maxLength)
{
  char c;
  int i = 0;

  while ((c = getchar()) != '\n') {
    if (i < maxLength - 1) {
      string[i] = c;
      i++;
    }
  }
  string[i] = '\0';
  return i;
}

/* getInt: returns the integer entered at the keyboard */
int getInt()
{
  char string[10];

  getString(string, 10);
  return atoi(string);
}

/* getDouble: returns the value of type double entered
   at the keyboard */
double getDouble()
{
  char string[308];

  getString(string, 308);
  return atof(string);
}

/* doubleEquals: returns 1 if a and b are close enough to equal,
    (as determined by tolerance) otherwise returns 0 */
int doubleEquals(double a, double b, double tolerance)
{
  double difference = fabs(a - b);
  return difference < tolerance;
}
```

We compile *utility.c*, but do not, cannot run it because there is no *main()* function.

12.4 Linking and Executing



```
$ gcc -c utility.c -ansi -Wall -pedantic -o utility.o

$ gcc testutility.c -ansi -Wall -pedantic utility.o -o testutility.exe

$ testutility
Name? Max Power
Age in years? 21
Height in metres? 1.65
Data entered: name = Max Power, age = 21, height = 1.65

$
```

The first line:

gcc -c utilty.c -ansi -Wall -pedantic -o utility.o

-c because we want to compile the file, but not link it to form an executable program.
-o utility.o because we want the output to be an object file. It is this object file that we link in with out programs, like this:

gcc testutility.c -ansi -Wall -pedantic utility.o -o testutility.exe

Notice that *utility.o* is placed immediately before the *-o*.

The details may be different on your computer system.

Exercise 12.1

1. Implement *utility.h* and *utility.c*, and test *testutility.c*, as shown above.

We have seen how to compile and include separate units.

Next we examine pointers.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988