FUNCTION DEFINITION

A function is a part of a program that is self-contained and performs a particular task. It can be thought of as a black box in which many things (arguments) may go into the box and only one thing (return value) may come out. Functions are defined by providing a return type, a function name, formal parameters, open curly brace, C code, and a closing curly brace. A simple function definition is shown below:

```
int SomeFunctionName(int a, char b, float c, int d)
{
      /* C Code Goes Here */

      return someIntValue;
}
```

In the example above, the return type is `int`. This means that the function must return an integer value to the calling function. The descriptive function name is called `SomeFunctionName(),` and the formal parameters (not descriptive to save space) are called `a, b, c,` and `d,` respectively. Note: In this example, I defined four formal parameters. That was arbitrary. A function can have any number of formal parameters in its definition. The function in the example above may be called in such a manner:

```
int status, w, x, y, z;

/* ... some C code may go here */

status = SomeFunctionName(w, (char)x, (float)y, z);
```

The return value of the call to `SomeFunctionName()` is copied to the integer `status`. The integer variables `w, x, y, and z` are arguments that are copied into the formal parameters `a, b, c, d`. Copying the arguments into the formal parameters is known as *Call-By-Value* parameter passing. Any action that `SomeFunctionName()` performs that changes the formal parameters do ***not*** change the arguments. This is because the arguments are copied into the formal parameters and both exist in separate memory space.

Functions save you from duplicating the same code over and over should you have to do a certain task many times in your program. Rather than duplicating the code, you simply call the function in the areas where the task is to be performed. This also reduces the risk of error should the procedure to perform the task ever change. A change to the procedure that performs the task is confined to one area of code (the function itself) when using functions and multiple areas of code when replicating code throughout your program.

Functions are also modular in that they are self-contained. Thus, a function can exist in a separate source file all by itself, or ideally with other functions that perform similar tasks, or helper tasks to functions contained within. The source file can be compiled, along

with other source files, and linked together with all other object code to create one executable image. This approach is the approach that is used in the industry, and it is the approach that we will use this point onward in the course.

FUNCTION DECLARATION (aka PROTOTYPE)

A function must be defined or declared before it is actually used in the program. That is, the compiler needs to know information about the function such as its return type and the formal arguments that it accepts. The function definition defines all of the above. Therefore, if a function definition appears in a source file prior to its use by another function somewhere else in the source file, there is no compiler problem. However, if the function is used is prior to its definition, a compiler error will result.

Furthermore, we said that it is ideal for a function to exist in a separate source file with other function definitions that perform similar tasks or are helper functions to the functions contained within the same source file. So, how do we avoid a compiler problem if a function that exists in one source file needs to be used by a calling function in another source file? Another question might be: how do we avoid a compiler error if a function in one source file is called before its definition in the same source file? Both questions have the same answer: A function declaration must be provided.

A function declaration provides information about the function to the compiler regarding the prototype of the function. That is, the function's return type, the function's name, and the function's argument types. The formal parameter names in a function declaration are optional. A semi-colon completes the declaration. For example, the declaration for the function `SomeFunctionName()` is shown below.

```
int SomeFunctionName(int a, char b, float c, int d);
```

OR

```
int SomeFunctionName(int, char, float, int);
```

Both function declarations are acceptable to the compiler. The compiler uses the declaration to build the object code of the source file under compilation. The linker then resolves the function call at link time when building the binary executable image.

A function declaration may exist in a C source file or a C header file. It is preferable to put the function declaration in a header file. For example, when we use the `printf()` function, we must include the `stdio.h` header file. This header file contains the function declaration for the `printf()` (and other I/O) function(s).

HEADER FILES

A header file is a file that typically contains function declarations, symbolic constants, macros, externals, etc. It normally does not contain actual C code, although it is legal to put C code in a header file. We will use header files for function declarations, symbolic constants, and macros in this class. The header file usually has a name that is identical to its source-file counterpart. For example, if there is a source file called `calculate.c`, there should be a header file called `calculate.h` that contains all the function declarations, symbolic constants, and macros used in `calculate.c`.

If a function in one source file needs to use a function in a different source file, the programmer should only need to include the header file where the function to be used is declared. This is done as follows:

```
#include "SomeFileName.h"
```

Notice the header file is included using " " and not < > as is done for header files such as `stdio.h`. The reason for this is that the compiler will look in its library header file directory (location of this directory depends on the compiler) to find the `stdio.h` file. If you used < > for your header file, the compiler would look for your header file in the wrong directory. If you use " " for your header file, then the compiler looks in the same directory that the compilation occurs. That is, the compilation should occur in the same directory as the source and header files. Otherwise, a path must be provided.

SEPARATE COMPILATION

A common way of designing a robust, modular program is to use separate source and header files in a reasonable manner. All homework, as well as the project, from this point onward will require that you use separate compilation. The function `main()` should always exist in its own source file with a name that is descriptive of the program's overall objective. All other functions should exist in a separate source file, and each of the source files should have a corresponding header file. If several functions perform a similar task, they should be put in the same source file. In addition, helper functions should be put in the same source file that contains the functions that they help.

Source files containing functions that call functions in other source files should include the header file containing the function declarations. The same is true should they use symbolic constants or macros. If the header file is not included, a compiler error will result.

PROGRAMMING EXAMPLE

The following program illustrates the method of separate compilation. I have put boxes around the code to show you to which file the code belongs.

Source File: example.c

```
#include <stdio.h>
#include <stdlib.h>

#include "colors.h"
#include "numbers.h"

/* Declarations*/
int GetChoice(void);

/*-------------------------------------------------------------
 * Function Name: main()
 *
 * Description:   This program illustrates separate compilation
 *                by calling functions that are defined in
 *                source files.  The user chooses which function
 *                to call by menu-selection.  The result is the
 *                display of either a random number or random
 *                color.
 *
 * Inputs:        None
 *
 * Outputs:       None
 *
 * Return:        Status
 *-------------------------------------------------------------*/
int main(void)
{
   int num;
   int choice;

   printf("\n\tWELCOME TO THE COLOR / NUMBER PROGRAM\n");

   printf("\n\tEnter a Seed for Random Number Generator: ");
   scanf("%d", &num);
   srand(num);

   do
   {
      /* Display menu and get user's choice */
      choice = GetChoice();

      /* Get a random number */
      num = rand();
```

Source File: example.c (Continued)

```
      switch(choice)
      {
         case 0:
            printf("\nExiting...\n");
         break;

         case 1:
            DisplayColor(num);
         break;

         case 2:
            DisplayNumber(num);
         break;

         default:
            printf("\nProgram Error.\nExiting...\n");
            choice = 0; /* Exit the do-while loop */
         break;
      }
   } while(choice != 0);

   return 0;
}


/*-------------------------------------------------------------
 * Function Name: GetChoice()
 *
 * Description:   This function will display a menu to the user
 *                and get the user's desired input
 *
 * Inputs:        None
 *
 * Outputs:       None
 *
 * Return:        User's Menu Selection
 *-------------------------------------------------------------*/
int GetChoice(void)
{
   int choice;

   do
   {
      /* Display a menu and get the user's selection */
      printf("\n\tSELECTION MENU\n");
      printf("\n\t0 - Exit the Program");
      printf("\n\t1 - Display a Color");
      printf("\n\t2 - Display a Number\n");

      printf("\n\tEnter Choice: ");
      scanf("%d", &choice);
```

Source File: example.c (Continued)

```
      /* Validate the range of the user's selection */
      if( (choice < 0) || (choice > 2) )
            printf("\nERROR.  Please re-enter choice...\n");
   } while( (choice < 0) || (choice > 2) );

   /* We have a valid choice at this point, so return it */
   return choice;
}
```

Header File: colors.h

```
#ifndef _COLORS_H_
#define _COLORS_H_

#define TOTAL_COLORS    6

void DisplayColor(int randomNumber);
int GetColorChoice(int randomNumber);

#endif /* _COLORS_H_ */
```

Header File: numbers.h

```
#ifndef _NUMBERS_H_
#define _NUMBERS_H_

#define NUMBER_MAX  100

void DisplayNumber(int randomNumber);
int CreateNumberInRange(int randomNumber);

#endif /* _NUMBERS_H_ */
```

Source File: colors.c

```
#include <stdio.h>

#include "colors.h"

/*-------------------------------------------------------------
 * Function Name: DisplayColor()
 *
 * Description:   This function will display a primary or secondary
 *                color in a random fashion.
 *
 * Inputs:        randomNumber - A random number
 *
 * Outputs:       None
 *
 * Return:        None
 *-------------------------------------------------------------*/
void DisplayColor(int randomNumber)
{
   int colorChoice;

   colorChoice = GetColorChoice(randomNumber);

   switch(colorChoice)
   {
      case 0:
            printf("\nPrimary Color: RED\n");
      break;

      case 1:
            printf("\nPrimary Color: YELLOW\n");
      break;

      case 2:
            printf("\nPrimary Color: BLUE\n");
      break;

      case 3:
            printf("\nSecondary Color: ORANGE\n");
      break;

      case 4:
            printf("\nSecondary Color: VIOLET\n");
      break;

      case 5:
            printf("\nSecondary Color: GREEN\n");
      break;

      default:
      printf("\nCOLOR ERROR.\nReturning to Calling Function...\n");
      return;
   }
}
```

Source File: colors.c (Continued)

```
/*---------------------------------------------------------------
 * Function Name: GetColorChoice()
 *
 * Description:   This function will return an integer number
 *                that is within the range of the total number
 *                of colors to display.
 *
 * Inputs:        randomNumber - A random number
 *
 * Outputs:       None
 *
 * Return:        An integer within the range of the total number
 *                of colors.
 *---------------------------------------------------------------*/
int GetColorChoice(int randomNumber)
{
      return randomNumber % TOTAL_COLORS;
}
```

Source File: numbers.c

```
#include <stdio.h>

#include "numbers.h"

/*---------------------------------------------------------------
 * Function Name: DisplayNumber()
 *
 * Description:   This function will display a number between
 *                zero and NUMBER_MAX
 *
 * Inputs:        randomNumber - A random number
 *
 * Outputs:       None
 *
 * Return:        None
 *---------------------------------------------------------------*/
void DisplayNumber(int randomNumber)
{
   int numberInRange;

   numberInRange = CreateNumberInRange(randomNumber);
   printf("\nThe number is: %d\n", numberInRange);
}


/*---------------------------------------------------------------
 * Function Name: CreateNumberInRange()
 *
 * Description:   This function create a number in the range of
 *                zero to NUMBER_MAX
 *
 * Inputs:        randomNumber - A random number
 *
 * Outputs: None
 *
 * Return:        An integer within the range of zero to
 *                NUMBER_MAX
 *---------------------------------------------------------------*/
int CreateNumberInRange(int randomNumber)
{
      return randomNumber % (NUMBER_MAX + 1);
}
```

Sample Compilation (Using Borland C++ Compiler):

```
C:\BC5\BIN>bcc example.c colors.c numbers.c
Borland C++ 5.2 Copyright (c) 1987, 1997 Borland
International
Mar 19 1997 17:29:40
example.c:
colors.c:
numbers.c:
Turbo Link  Version 7.1.32.2. Copyright (c) 1987, 1996
Borland International

C:\BC5\BIN>
```

Sample Run:

```
C:\BC5\BIN>example

        WELCOME TO THE COLOR / NUMBER PROGRAM

        Enter a Seed for Random Number Generator: 323

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 1

Secondary Color: VIOLET

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 1

Primary Color: RED

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 2
```

Sample Run (Continued):

```
The number is: 62

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 2

The number is: 39

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 1

Secondary Color: ORANGE

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 2

The number is: 4

        SELECTION MENU

        0 - Exit the Program
        1 - Display a Color
        2 - Display a Number

        Enter Choice: 0

Exiting...

C:\BC5\BIN>
```