SEVEN SIMPLE STEPS OF PROGRAMMING

It is a wise idea for a new programmer to develop good coding habits right away. Although it may appear to be a little extra work up front, it will save you countless hours of frustration in the future. There are seven steps to programming which, if followed, should help in your development of good coding habits and, hopefully, result in robust programming solutions. These steps are described below.

PROGRAMMING STEP	WHAT TO DO
1. Define the Program Objectives	Determine exactly what it is the program is supposed to do and write it down. Usually there will be more than one objective.
2. Design the Program	Write pseudo-code and/or create flowcharts that provide the solution to the program objectives.
3. Write the Code	Convert the pseudo-code and/or flowcharts into C code that is syntactically correct.
4. Compile and Link the Program	Compile (fix errors if necessary) and Link (fix errors if necessary). This builds a binary executable image than runs on your PC.
5. Run the Program	Run the executable image.
6. Test and Debug the Program	Supply test data for you program while it is running and validate the objectives. If there are errors with respect to the solution of the program objectives, debug the program and retry.
7. Maintain and Modify the Program	New requirements may be added to your program at a future date. New data may suggest that your program doesn't solve the original objectives. Your program will need to be maintained and modified accordingly. This is a good reason to develop good coding habits immediately.

ANATOMY OF A C PROGRAM

The C Programming Language provides a set of keywords, data and operators that are used for making up C statements within the C language. There are five types of C statements in the C Language: declaration, assignment, function call, control, and NULL.

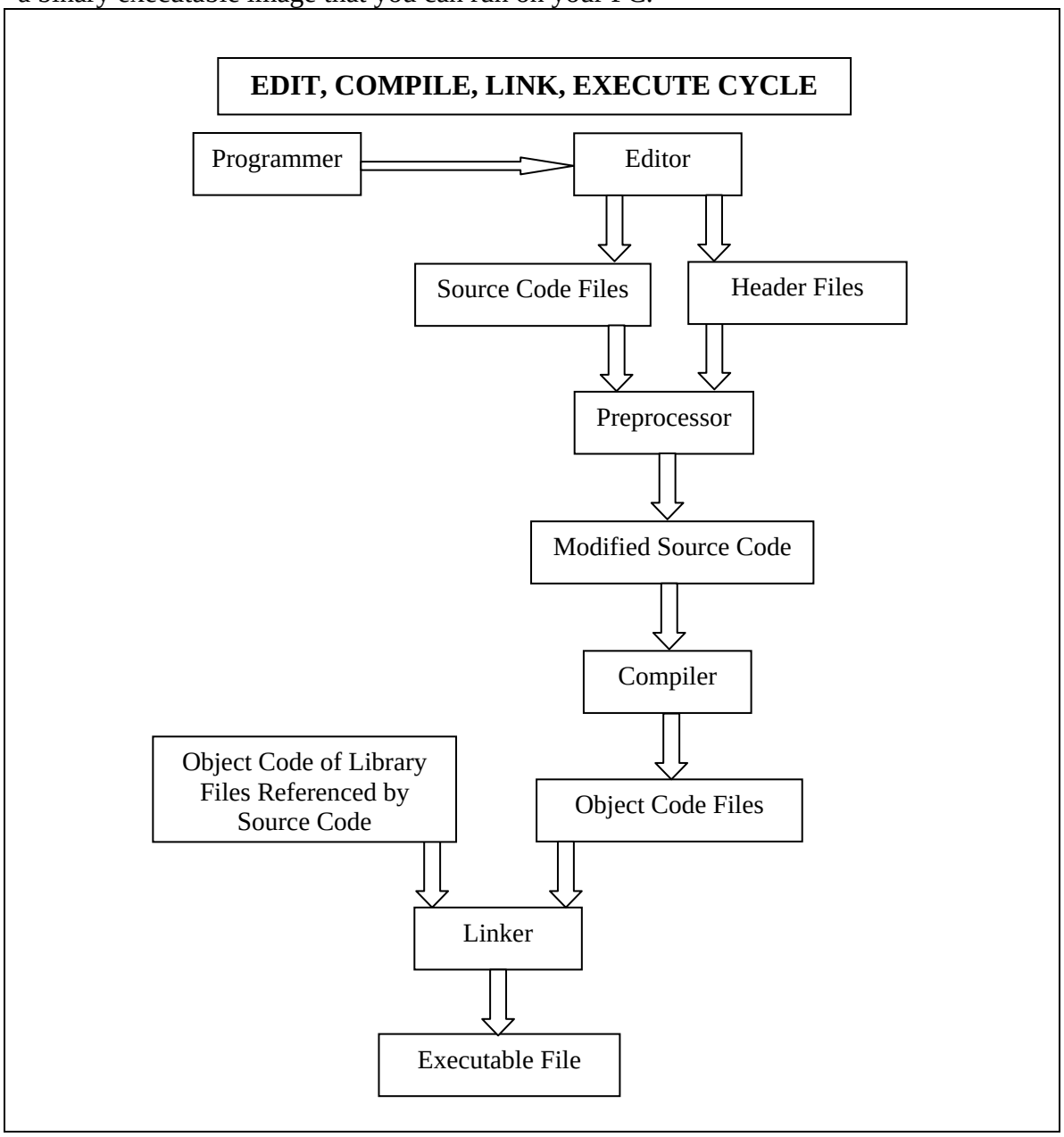
Functions are the building blocks of C Programming. Functions are made up of C statements. A typical C program may contain many user-defined functions, each with its own unique name. However, one of these functions must be called `main(),` and `main()` is the function that is always called first when a program is executed.

In addition to functions and statements, a typical C program may contain preprocessor directives (`#include`, `#define, ...`) that are handled by the compiler prior to the actual compilation. Exactly how the preprocessor handles them depends on the type of preprocessor directive. These will be discussed later in the semester.

The functions, preprocessor directives, and statements comprise a C program that can be compiled and linked in order to create an executable image that can be run by the program user.

EDIT, COMIPLE, LINK, EXECUTE CYCLE

The edit, compile, link, execute cycle is shown in the figure below. These are the steps that a programmer follows from inception to completion of the executable image. The programmer uses his editor of choice and edits source code file(s) and header file(s) that contain the solution to the program objectives. When compiling, the preprocessor processes all pre-processor directives, creates a modified source code file(s) based on the result of the preprocessing, and compiles the modified source code file(s). The result of the compile is a binary object file that is linked together with object code of the library files referenced by all source code file(s). The linker takes the object code and produces a binary executable image that you can run on your PC.



C PROGRAMMING EXAMPLES

```
/* --- Hello to the World --- */
#include <stdio.h>
int main(void)
{
      printf("Hello World!\n");

      return 0;
}
```

Sample Run,

```
Hello World!
Press any key to continue
```

```
/* --- Determine if Water Will Freeze --- */
#include <stdio.h>
int main(void)
{
      int temp;

      printf("Enter the current temperature in degrees Fahrenheit: ");
      scanf("%d", &temp);

      if(temp <= 32)
            printf("Water will freeze at this temperature.\n");
      else
            printf("Water will not freeze at this temperature.\n");

      return 0;
}
```

Sample Runs,

```
Enter the current temperature in degrees Fahrenheit: 68
Water will not freeze at this temperature.
Press any key to continue

Enter the current temperature in degrees Fahrenheit: 32
Water will freeze at this temperature.
Press any key to continue

Enter the current temperature in degrees Fahrenheit: -2
Water will freeze at this temperature.
Press any key to continue
```

C PROGRAMMING EXAMPLES - CONTINUED

```
/* Determine the equation of a line given two points */
#include <stdio.h>
int main(void)
{
      float x1, y1, x2, y2; /* Two points on a line */
      float slope, yIntercept;

      printf("Enter x1 and y1: ");
      scanf("%f%f", &x1, &y1);

      printf("Enter x2 and y2: ");
      scanf("%f%f", &x2, &y2);

      /* Calculate the slope */
      slope = (y2 – y1) / (x2 – x1);

      /* Calculate the Y-Intercept using the first point */
      yIntercept = y1 – (slope * x1);

      printf("Equation of Line: y = %.1fx + %.1f\n",slope, yIntercept);

      return 0;
}
```

Sample Runs,

```
Enter x1 and y1: 1 3
Enter x2 and y2: 4 8
Equation of Line: y = 1.7x + 1.3
Press any key to continue

Enter x1 and y1: -3 9
Enter x2 and y2: 2 -8
Equation of Line: y = -3.4x + -1.2
Press any key to continue

Enter x1 and y1: -6 4
Enter x2 and y2: 12 4
Equation of Line: y = 0.0x + 4.0
Press any key to continue
```

***** REVIEW THE POWERPOINT PRESENTATION ON CODING GUIDELINES *****

The following example represents good coding style. At this point of the semester, do not pay so much attention to the unfamiliar C statements presented in the code. These statements will be covered in great detail later in the semester. Rather, pay attention to the "style" of the code. Each function has a function description header that gives information relating to the function name, function description, function input, function output, and function return. Note the use of white space throughout the program. Also note the use of proper indenting and commenting. Look at the names of the variables and functions – they are *descriptive*!!!

All homework that you pass in should use a coding style similar to the example below. Homework that does not use good coding style will have a 5-point deduction regardless of whether or not the program works. Remember, it is important to develop good coding habits right away. It is equally important to be consistent in your coding habits regardless of the complexity of the problem.

```
#include <stdio.h>
#define NUM_STUDENTS 10

/*---------------------------------------------------------------------
 * Function Name: calculateAverage()
 *
 * Description:   This function will calculate the average of an
 *                array of integers.
 *
 * Input:         numbersArray     - Array of ints from which we will
 *                                   calculate the average
 *                numberOfElements – Number of ints in numbersArray
 *
 * Output:        None.
 *
 * Return:        The average of numbersArray
 *---------------------------------------------------------------------*/
float calculateAverage(const int numbersArray[], int numberOfElements)
{
      int sum = 0, counter;

      /* Sum up the elements of the array */
      for(counter=0; counter < numberOfElements; counter++)
            sum += numbersArray[counter];

      /* Return the average of the array */
      return (float) sum / numberOfElements;
}
```

```
/*---------------------------------------------------------------------
 * Function Name: main()
 *
 * Description:   This function will take user-supplied test scores
 *                and determine the average grade of the test.
 *
 * Input:         None.
 *
 * Output:        None.
 *
 * Return:        Status of function call
 *---------------------------------------------------------------------*/
int main(void)
{
      int testScores[NUM_STUDENTS];
      int counter;
      float testAverage;

      printf("PROGRAM TO CALCULATE AVERAGE TEST SCORE\n\n");

      /* Get the test scores. */
      for(counter=0; counter < NUM_STUDENTS; counter++)
      {
            printf("\tEnter grade #%02d: ", counter+1);
            scanf("%d", &testScores[counter]);
      }

      /* Determine the average grade and display it */
      testAverage = calculateAverage(testScores, NUM_STUDENTS);
      printf("\n\tTest Average = %.1f%%\n\n", testAverage);

      return 0;
}
```

Sample Run,

```
PROGRAM TO CALCULATE AVERAGE TEST SCORE

        Enter grade #01: 83
        Enter grade #02: 79
        Enter grade #03: 91
        Enter grade #04: 88
        Enter grade #05: 67
        Enter grade #06: 75
        Enter grade #07: 81
        Enter grade #08: 84
        Enter grade #09: 78
        Enter grade #10: 96

        Test Average = 82.2%

Press any key to continue
```