Module 6 – Managing C Programs

Tutorial Questions

Objectives
To consolidate your understanding of arrays, pointers and functions, and their combined use and to review storage classes.

Activities
1. Review of Pointers, Arrays, Functions

   Write a function that accepts an array of the following type as an argument, and computes the sum and average of the numbers stored in the array.

   ```
   #define SampleSize 10
   int rainfall[SampleSize];
   ```

   You will have to determine an appropriate function heading (function interface) to this purpose.

2. Why doesn't the following program print the same string twice?

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

int main()
{
   char *source = "duplicate message";
   char *p1, *p2, destination[50];
   p1 = source;

   puts(p1);
   p2 = destination;
   while(*p2++ = *p1++)
      ;
   puts(p2);

   return EXIT_SUCCESS;
}
```

   Hint: You may find it useful to draw diagrams representing these variables, and deskcheck this code.

3. What is printed?

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

int main()
{
   char words[5][8] = { "the", "cat", "in", "the", "hat"};
   char *wp[5];
   int i;
```

```
      for(i=0; i<5; i++)
      {
         wp[i] = words[i];
      }

      puts(*wp);
      puts(*(wp+2));
      puts(*(wp+3)+1);
      putchar(**(wp+1));

      return EXIT_SUCCESS;
   }
```

Hint: Again, you may find it useful to draw diagrams representing these variables, and deskcheck this code -- particular attention needs to be placed on identifying the _type_ of the object being dealt with.

4. Function Declarations, Storage Classes

   At some later time you are encouraged to test the following on your favourite C compiler. For now, however, try and answer the questions away from a proven test environment.

Problem One
The following program should generate several warnings suggesting that functions f() and g() have conflicting, multiple declarations. What might this imply, and how does one eliminate the warnings? Ignore the fact that the program does not do a great deal!

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

int main()
{
   g();
   return EXIT_SUCCESS;
}

double g()
{
   return f();
}

double f()
{
   return g();
}
```

Problem Two
Write a function that returns the number of times it has been called during program execution. The function must not have any parameters. (*Hint: Use the static storage class, appropriately.*)

Problem Three
Given the definitions

```
struct cntr {
int counter;
char display[3][10];
}
typedef struct cntr COUNTER_TYPE;
```

Develop a separate source translation unit (a separately compilable source) that allows client code to manipulate counters of type COUNTER_TYPE. The functionality that should be available to the client code must include: counter increment (by one or more), counter decrement (by one or more), and counter reset (to 0 or a positive integer).

An invocation of any of the above functions must result in a display of the counter. The display version of the counter is stored in the display member of the struct definition provided. This display version is created by a separate function, which must not be accessible to client code. Finally, any operation that causes an overflow (> 999) or underflow (< 0) should result in a "bad" counter value of -1 being stored, with the display showing three asterisks. The following shows examples of a good number and a bad number displayed by the counter.

```
     GOOD         BAD
  ---------- ----------
    | 234 |     | *** |
  ---------- ----------
```

The overflow and underflow checking should also be carried out by a separate, non-accessible function.