Lab 6

Variable Scope, Lifetime and Arrays

Objectives

- To work with variable scope and lifetime
- To experience side-effects, and the evils of global variables
- To work with arrays
- To prepare for Assignment #3

Getting started

Answers to all of the **Questions** and **Exercises** throughout this lab should be put into a single document (Word or rtf or text file). Be sure to label your answers under appropriate headings.

A. Variable scope

Exercise 1: Consider the following piece of code: (Do ***not*** write any code for this exercise!)

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

int foo (int x);
int x = 0;

int main()
{
     int a, b;
     scanf("%d", &a);
     b = foo(a);
     printf("%d %d %d \n", b , a, x);
     system("PAUSE");
     return 0;
}
int foo( int b )
{
     int x = 5;
     return (b + x);
}
```

Questions:

- Circle all the local variables
- Underline all the global variables
- List all the function parameters:
    - Formal parameters:
    - Actual parameters (arguments):
- Suppose we enter value 5 for variable a; trace the code segment above (i.e. draw a trace table) to determine the output - what is it? Explain why.

Exercise 2: The following piece of code intends to compute the sum from n (0<n<nMax) to nMax. For example, if nMax is 3, then the program is supposed to output the following:

The sum from 0 to 3 is 6
The sum from 1 to 3 is 6
The sum from 2 to 3 is 5

But the program uses **global variables** inappropriately and thus causes *unintended side-effects.*

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

void sumInts(); /* function prototype */
/* defining global variables */
int n = 0;
int nMax = 0;
int sum = 0;
int main()
{
     int n;

     printf( "Enter a small integer number (say < 20): \n" );
     scanf("%d", &nMax);

     for (n=0; n<nMax; n++)
     {

        sumInts();
        printf( "The sum from %d to %d is " , n , nMax);
        printf("%d \n",sum );
     }
     return 0;
}
```

```
void sumInts()
{
      while (n <= nMax)
      {
      sum = sum + n;
      n++;
      }
}
```

Questions:

(Note: Except for the side-effects, the code is basically correct, so don't change its purpose!)

- Why does it not do what is obviously intended?
- Re-write this code to perform the intended task correctly without any global variables. Test your program in a new C project; copy and paste your final version into your document file.

B. Lifetime of Variables

Consider the following code segment:

```
int foo()
{
         int f = 0;
         f++;
         return f;
}
int main()
{
        int a, i=0;
        while (i < 5)
        {
              a = foo();
              printf("%d ", a);
              i++;

        }

        return 0;
}
```

Questions:

- What is the output of this code?
- Change the code segment so that it outputs "1 2 3 4 5" using global variables. (Test in a C project, then copy final version into document file)
- Change the code segment so that it outputs "1 2 3 4 5" using only local variables. You must add a parameter to function foo(). (Test in a C project, then copy final version into document file)
- What is the lifetime of local variables and global variables?

C. Arrays and Prep for Assignment #3

An array is a built-in data structure in C used to store a linear sequence of values. Later in the course we will look at sophisticated array processing algorithms, but in the meantime, you will need to use a very simple integer array in Assignment 3. The following exercises are meant to introduce the simple array you will use in Assignment 3 and to implement some of the functions that will be useful when you do Assignment 3. In addition, these exercises demonstrate an important point -- all arrays in C are passed by reference (address)! We will discuss more on this later in class.

You can create an array of any type in C, but you must specify its size (the number of array elements) when you declare the array. You should define a global constant to hold the size of an array and use that constant to declare the array. (Why is it OK to use global constants when it is NOT OK to use global variables?)

Assignment #3 will be implementing a simplified version of Black Jack. In order to play the game, we must have a deck of cards. The deck of cards can be represented as an array of integers. Assume that the cards are sorted in the following way: spades from the ace to the king, hearts from the ace to the king, diamonds from the ace to the king and clubs spades from the ace to the king. We can assign numbers 0-12 to sorted spades, 13-25 to sorted hearts, 26-38 to sorted diamonds and 39-51 to sorted clubs. We can see for now that the array needs to have at least 52 elements (values). But during the play of the game, we also need to know how many cards are left in the deck, because at any time when the deck is empty, we need to reshuffle the deck. We can give the array one more element, making it’s size as 53, and let the first 52 elements to store the cards, the very last to store the number of cards left in the deck.

We would require the following declarations:

```
#define DECK_SIZE 53

int deck[DECK_SIZE ]; //Put this in the beginning of the main function.
```

Download testDriver.c program. For each of the exercises below,

- **Add the new function to the bottom of testDriver.c**
- **Add the function prototype to the function prototypes section near the top of the file**
- **Add a case constant for the new function to the case constants sections below the function prototypes**
- **Add a new menu item to the GetTestChoice() function**
- **Add a new test case to main to test your function**

Exercise 1 . Write a void function **InitializeDeck** which takes an integer array as a parameter and assign values to the array elements (from index 0 to 52) in the following way:

First 13 cards should be spades, then hearts, diamonds and clubs, as described above. Assign 52 to the very last element, meaning that the deck is full.

Exercise 2. Write a function RandomNumber which returns a random number in the range 0 to 51.

Exercise 3. Write a void function **ShuffleDeck**, which takes an integer array as a parameter and shuffle the values in the array in the following way: (Please note: you will need to call the function defined in Exercise 2 to complete this exercise.)

Start with the last card and generate a random number between 0 and 51. Swap the last card with the card that is in the position determined by the random number. Generate a random number between 0 and 50 and swap the card which is in that position and the card that is in the 50th position. Generate a random number between 0 and 49 and swap the appropriate cards. Do this until you reach the 1st card.

The following exercises are not required for the lab, but will be helpful for Assignment 3.

Exercise 4 . Write a void function **DisplayCard**, which takes an integer [0-51] (representing a card) as a parameter and displays the card (suit and rank) based on the following:

Divide the integer by 13 to get the suit of the card and look at the integer modulo 13 to get the rank of the card.

Exercise 5 . Write a void function **DisplayDeck**, which takes an integer array (a deck of cards) as a parameter and displays the cards stored in the array. You need to call the function defined in Exercise 4 to complete this exercise.

Exercise 6. Whenever a player (human or computer) draws a card from the deck, we need to know the score (face value) of that card. Write a function **ScoreCard**, which takes an integer (a card) as a parameter and returns the score of the card.