Exercise 13 – Pointers and Arrays

Objective

The major objective is to practice using pointers to access and manipulate array elements.

Reference Material

This is based the *Pointers and Arrays* chapter and material from the *Arrays* practical exercise. The practical session is located in the following directory:

Windows Directory:	**c:\qacprogex\ptrarray**
Windows Solution directory:	**c:\qacprogex\ptrarray\solution**
Linux Directory:	**/home/user1/qacprg/PTRARRAY**
Linux Solution directory:	**/home/user1/qacprg/PTRARRAY/Solution**

Overview

Questions 1 and 2 manipulate arrays of `longs`. Questions 3, 4 and 5 are closely associated with questions 3 and 4 of the Arrays practical exercise. The optional question 6 deals with some manipulation of arrays of `doubles`, and the final two questions involve dynamic arrays.

Practical Outline

1. Open the Visual Studio Solution **larray.sln**, and take a look at the code template in **larray.c**. The program calls a function called `print_larray`, which prints an array of `long ints`. Complete the program using a pointer argument to the function.

2. Open the Visual Studio Solution **rev_prnt.sln**, and take a look at the code template in **rev_prnt.c**. The `main` function calls the function `rev_print_larray`, which prints an array of `long ints` in reverse order. Complete the program.

3. Open the Visual Studio Solution **slen.sln**, and take a look at the code template in **slen.c**. This is a copy of the solution to question 3 in the Arrays practical exercise (if you prefer, you can use your original version of this program, in the Visual Studio Solution **c:\qacprogex\arrays\slen.sln**).

   Rewrite the program so that the function `slen` uses pointer techniques to calculate the length of a string. It will now have the prototype:

```
int slen(char *);
```

In your solution, walk the pointer `s` up the array by 's++'ing. Test at each stage if `*s` is `'\0'`. You will have to count how many times you increment `s` before this terminator is found. You should try not to use the notation `s[i]` in your solution.

4. Open the Visual Studio Solution **scpy.sln**, and take a look at the code template in **scpy.c**. This is a copy of the solution to question 4 in the Arrays practical exercise (if you prefer, you can use your original version of this program, in the Visual Studio Solution **c:\qacprogex\arrays\scpy.sln**).

   Rewrite the program so that the function `scpy` uses pointer techniques to perform the copying. It will now have the prototype:

```
void scpy(char *, char *);
```

5. In the previous two questions, you have written the functions `slen` and `scpy` using pointer techniques. These two functions will now be brought together into the same program; open the Visual Studio Solution **scpy_ret.sln**, and take a look at the code template in **scpy_ret.c**. Notice that the `slen` and `scpy` functions do indeed appear in this program.

   Your task is to modify the `scpy` function, so that it returns a `char *` rather than `void`. The return value is the starting address of the destination string, which is what the standard library function `strcpy` does. Consult the on-line help on this function.

   After making this change, the code fragment:

```
char str[MAXLENGTH];
scpy(str, "Hello Again");
printf("%d\n", slen(str));
```

   could be changed to:

```
char str[MAXLENGTH];
printf("%d\n", slen(scpy(str, "Hello Again")));
```

Optional:

6. Open the Visual Studio Solution **avearray.sln**, and take a look at the code template in **avearray.c**. The `main` function appears as follows:

```
int main(void)
{
     double da[SIZE];
     double ave = 0.0;

     scandarr(da, SIZE);        /* take SIZE doubles from */
                                /* keyboard into the array da */
```

```
        ave =  avedarr(da, SIZE);      /* and return the average */
        printf("The average value is %f\n", ave);

        return 0;
}
```

Complete the program by providing the `scandarr` and `avedarr` functions.

7. Open the Visual Studio Solution **rev_arr.sln**, and take a look at the code template in **rev_arr.c**. This program contains a variety of functions (`print_array`, `copy_array` and `comp_array`) for manipulating an array of `int`s.

   Design, code and test another function called `rev_array`, which has the prototype:

   ```
   int rev_array(int *, int);
   ```

   The function's first argument points to the array to be 'reversed'. The return is a boolean, i.e. true if the array has been reversed or false if it is a palindrome (indicating that it is the same in either order and hence does not need to be reversed).

   Note: You will be expected to use `malloc()` to allocate a temporary array to process the reversing.

8. Open the Visual Studio Solution **dynlarr.sln**, which contains an empty code template in **dynlarr.sln**. Using `calloc` (or `malloc`), create a dynamic array of `long`s, the size of which is entered from the keyboard. Before allocating from the heap, ensure that the user has not entered too large a number (e.g. larger that `INT_MAX`).

   Assign the cardinal values `1..size` to the array, and display it (cut-and-paste your `print_larray` function from earlier?). When you have finished with the array, make sure that you `free` the appropriate pointer.