Exercise 14 – Pointers and Structures

Objective

The major objective is to practice passing structures by reference.

Reference Material

This is based mainly on the *Pointers and Structures* chapter. The practical session is located in the following directory:

Windows Directory:	**c:\qacprogex\ptrstruc**
Windows Solution directory:	**c:\qacprogex\ptrstruc\solution**
Linux Directory:	**/home/user1/qacprg/PTRSTRUC**
Linux Solution directory:	**/home/user1/qacprg/PTRSTRUC/Solution**

Overview

The first three questions are standalone. The optional questions involve linked-list manipulation, and delegates with no prior experience in link lists are advised not to attempt them. However, you could benefit from studying the solutions..

Practical Outline

1. Open the Visual Studio Solution **prt_card.sln**, and take a look at the code template provided in **prt_card.c**. Write a function with the following prototype:

   ```
   void print_card(struct Card *);
   ```

   `print_card` prints out the values stored in the `suit` and `index` fields of the variable pointed to by its parameter. Use the examples in the notes as a guide.

2. Open the Visual Studio Solution **find_ace.sln**, and take a look at the code template provided in **find_ace.c**. Write a function that has the following prototype:

   ```
   const struct Card * ace(struct Card *, int);
   ```

   `ace` is passed a pointer to a `struct Card`, i.e. the start of an array of `struct Cards`, together with the size of this array. It should return a pointer to the first ace in this array or `NULL` if no is ace found.

3. Open the Visual Studio Solution **3of_kind.sln**, and take a look at the code template provided in **3of_kind.c**. Write a function called `three` that takes a pointer to a `struct Card` and the size of an array. It reports back if the array, which represents a hand of cards, contains three of a kind, i.e. three 2s or three Jacks, etc. It returns the index of the card, i.e. 2 or 11(for Jack) or -1 if the hand does not contain three of a kind. The prototype of this function is:

```
int three(struct Card *, int);
```

Optional:

4. Open the Visual Studio Solution **prt_list.sln**, and take a look at the code template provided in **prt_list.c**. The program defines the following struct template:

```
struct Node
{
     struct Node * next;  /* pointer to next in list */
     int i_val;           /* value stored at this node */
};
```

The program declares a number of `Node` variables, linked together statically to form a simple linked list:

```
struct Node n1 = { NULL, 40 };
struct Node n2 = { &n1,  32 };
struct Node n3 = { &n2,  8  };
struct Node n4 = { &n3,  -2 };
struct Node * head_of_list = &n4;
```

Write a function called `print_list` that has the following prototype:

```
void print_list(struct Node *first_node);
```

This function should print out the integer value stored at each node. It should do this by traversing the list through the next field of each node. The output in this example should be:

```
-2 8 32 40
```

If you are familiar with recursion, use this technique to write a function `reverse_print` that prints out the list in the opposite order, i.e.:

```
40, 32, 8 -2
```

5. Open the Visual Studio Solution **ins_list.sln**, and take a look at the code template provided in **ins_list.c**. The program extends the linked list example from the previous question, allowing new nodes to be inserted into the list dynamically.

   The nodes of this list are ordered by the value in the `i_val` field. Write a function called `insert_node` which inserts any new node into this list at the point that maintains this ordering. It should have the following prototype:

```
struct Node *
insert_node(struct Node *, struct Node *);
```

   The node pointed to by `new_node`, `new_first` and `new_last` should be inserted into the list. This function should return a pointer to the `struct Node`, which is currently at the head of the list.