csc2404 Practical

Introduction to Structures

Week 3

Structures

An array is a block of contiguous memory containing data of the same type and a *structure* is a block of contiguous memory containing data of different types.

Reading A : Chapter 11 of "An Introduction to the C Programming Language and Software Design" by *Tim Bailey* (Can ignore the last section on Unions)
or
Chapter 20 of "The GNU C Programming Tutorial" by *Mark Burgess & Ron Hale-Evens* (Can ignore section 20.3 on Unions)
and
Chapter 5 of "A Tutorial on Pointers and Arrays in C" by *Ted Jensen*

Exercise 1: Explain why the following structure is **illegal**:

```
struct Point {
    int x;
    int y;
    struct Point p;
};
```

Explain why the following structure is **legal**:

```
struct Point {
    int x;
    int y;
    struct Point *p;
};
```

Prove the above statements by declaring the above structures in a simple program and trying to compile it.

Link-Lists

Unlike an array where every item can be accessed directly since the memory an array uses is contiguous—items in a list must be accessed in a particular order since the items are scattered throughout memory. The most common method to implement a list is a linked-list—where each item is linked to the next item in the list—as each item on the list contains the address of the next item on the list. Each item must be read to find the address of the next item in the list.

Code: linklists.tgz

Exercise 2: Compile and run the link-list code. Study the `linklists.c` file.

Exercise 3: Doubly linked-lists as well as having a pointer to the next item on the list also have a pointer to the previous item on the list. This means that the list can be traversed in either direction—starting from the start or from the end.

Modify the code so it uses a doubly-linked list by adding the following element to the `_contact_details` structure:

```
struct _contact_details *prev;
```

The functions `add()`, and `delete()` will also have to be modified.

Stack

A stack is an ordered data list that uses the last in, first out (LIFO) principle for adding and removing items (consider a stack of plates). This means that the last item placed onto the stack is the first item removed.

Code: stack.tgz

Exercise 4: Compile and run the stack code. Study the `stack.c` file.

Exercise 5: The stack in the code has been implemented using an array. How would you implement the stack using a linked list?

Compare the stack code and the link-list code. The link-list `add()` function—where is it adding a new item to the list? The link-list `delete()` function—which item is it deleting from the list? How does the link-list operations compare to the stack operations?

Queue

A queue is an ordered data list that uses that first in, first out (FIFO) principle for adding and removing items (consider a queue of people at a ticket office). This means that items are removed in the order they where added.

Code: queue.tgz

Exercise 6: Compile and run the queue code. Study the `queue.c` file.

Exercise 7: The queue is implemented using a linked-list—explain why it would not be a good idea to implement a queue using an array.

Hash Table

Code: hash.tgz

Exercise 8: Compile and run the hash code. Study the `hash.c` file and in particular the `hash()` function.

Code: StaffList.txt.gz

Exercise 9: Use the `StaffList.txt` to load the hash table with over two thousand entries. Use the `count` command to see how the contacts are distributed in the hash table.

The hash function used in the example code is not particularly good—though adequate for our example and by the fact that the key string we are using is not a truly random string (given name-family name).

Explain what constitutes a good hash function.

Exercises

Ex. 10: Answer the exercises at the end of Chapter 2 of the Study Book.

Ex. 11: All the above codes use `strcpy` to individually copy each item in the `ContactDetails` structure—a structure is always stored in contiguous memory so it would be far better to copy the entire structure (reduce the potential number of system calls from four to one)

Replace the four `strcpy` lines that copy each element of the structure to one line that copies the whole structure. Hint: see `memcpy`, you will also need to use `sizeof(ContactDetails)`.