Question #1:

Consider a structure defined as follows:

```
struct item
{
     char name[50];
     int price;
};
```

Let's say an array of 100 items (numbered 0 through 99) is declared as follows:

```
struct item items[100];
```

(a) Write a statement that will assign the integer 25 to the price of the first item in the array.
(b) Write a statement that will assign the string "jukebox" to the name of the last item in the array.

You can assume that all appropriate header files have been included at the top of the file.

Question #2:

You compile and run the following program:

```
#include <stdio.h>

int x = 10;

int func1(int *);
int func2(int);

int main(void)
{
     int y = 40;
     int *p1 = &y;

     y = func1(p1);
     printf("x = %d, y = %d, *p1 = %d\n", x, y, *p1);

     return 0;
}

int func1(int *p1)
{
     int y = 20;

     *p1 = 30;
     printf("x = %d, y = %d, *p1 = %d\n", x, y, *p1);
     p1 = &y;
     x = func2(*p1);
     printf("x = %d, y = %d, *p1 = %d\n", x, y, *p1);

     return (*p1 + 50);
}

int func2(int y)
{
     y = y - 10;
     return (y + 30);
}
```

What gets displayed to standard output?

Question #3:

The following are the definitions concerning nodes in a linked list:

```
typedef struct node
{
     int x;
     struct node *next;
} NODE;

typedef NODE *PNODE;
```

Write a function called "display_backwards" that accepts, as a parameter, a pointer "pList" which points to the head of a linked list. The function should display the integers stored in the nodes in backwards order (i.e. the integer in the final node should be displayed first, and the integer in the first node should be displayed last). Every integer should be displayed on its own line. The function should not actually manipulate the list itself (i.e. you should not actually reverse the pointers). The function should not return anything.

HINT: This may be easiest with recursion. If the list is empty, do nothing. Otherwise, display the integers in the remainder of the list in backwards order, then display the integer in the current node.

Question #4:

Write a function called "check_strings" that accepts two strings as parameters. The function should return 1 if the two strings are the exact reverse of each other, and it should return 0 otherwise. For example, if the two strings are "Hello World!" and "!dlroW olleH", the function should return 1, but if the first of these strings is the same and the second is anything else, it should return 0. You can assume that the appropriate header files have been included at the top of the file.

Question #5:

Write a function called "second_largest" which takes, as parameters, an array of positive integers and also an integer representing the size of the array. The function should return the value of the second largest integer in the array. Your function should not actually change the contents of the array in any way. You can assume that the size of the array is at least two, and that all of the integers in the array are greater than 0.

HINT: You can do this with a single pass through the array. Keep track of the largest number seen so far and the second largest number seen so far at all times.