1. Suppose your program contains #include <string> and the following type definitions:

struct Box

{

string name;

int number;

Box \*next;

};

typedef Box\* BoxPtr;

What is the output produced by the following code?

BoxPtr head;

head = new Box;

head->name = "Sally";

head->number = 18;

cout << (\*head).name << endl;

cout << head->name << endl;

cout << (\*head).number << endl;

cout << head->number << endl;

The following program outputs Sally twice and 18 twice. The program displays the values using both a dot notation method cout << (\*head).name << endl; and arrow notation

cout << head->name << endl;

2. Suppose that your program contains the type definitions and code given in Exercise 1. That code creates a node that contains the string "Sally" and the number 18. What code would you add to set the value of the member variable next of this node equal to NULL?

head->name = "";

3. Given the following structure definition:

struct ListNode

{

string item;

int count;

ListNode \*link;

};

ListNode \*head = new ListNode;

Give code to assign the string "Wilbur's brother Orville" to the member variable item of the variable to which head points.

ListNode l;

head->item = "Willbur's brother Orville";

4. Write type definitions for the nodes and pointers in a linked list. Call the node type NodeType and call the pointer type PointerType. The linked lists will be lists of letters.

typedef NodeType \*pointerType;

Pointer pointerType;

5. A linked list is normally referred to via a pointer that points to the first node in the list, but an empty list has no first node. What pointer value is normally used to represent an empty list?

The pointer value null is normally used to represent an empty list.

6. Suppose your program contains the following type definitions and pointer variable declarations:

struct Node

{

double data;

Node \*next;

};

typedef Node\* Pointer;

Pointer p1, p2;

Suppose p1 points to a node of the above type that is in a linked list. Write code that will make p1 point to the next node in this linked list. (The pointer p2 is for the next exercise and has nothing to do with this exercise.)

p1 = p1->next;

7. Suppose your program contains type definitions and pointer variable declarations as in Exercise 6. Suppose further that p2 points to a node of the above type that is in a linked list and which *is not the last node on the list*. Write code that will delete the node *after* the node pointed to by p2. After this code is executed, the linked list should be the same, except that there will be one less node in the linked list.

(*Hint:* You may want to declare another pointer variable to use.)

Node \*currentPtr = p2->next;

delete currentPtr;

8. Suppose your program contains the following type definitions and pointer variable declarations:

class Node

{

public:

Node(double theData, Node\* theLink)

: data(theData), next(theLink){}

Node\* getLink( ) const { return next; }

double getData( ) const { return data; }

void setData(double theData) { data = theData; }

void setLink(Node\* pointer) { next = pointer; }

private:

double data;

Node \*next;

};

typedef Node\* Pointer;

Pointer p1, p2;

Suppose p1 points to a node of the above type that is in a linked list. Write code that will make p1 point to the next node in this linked list. (The pointer p2 is for the next exercise and has nothing to do with this exercise.)

p1 = p1->next

9. Suppose your program contains type definitions and pointer variable declarations as in Exercise 8. Suppose further that p2 points to a node of the above type that is in a linked list and which is not the last node on the list. Write code that will delete the node *after* the node pointed to by p2. After this code is executed, the linked list should be the same, except that there will be one less node in the linked list.

(*Hint:* You may want to declare another pointer variable to use.)

ListNode \*currentPtr = p2->next;

delete currentPtr;

10. Choose an ending to the following statement, and explain:

For a large array and a large list holding the same type objects, inserting a new object at a known location into the middle of a linked list compared to insertion in an array is

a. more efficient.

b. less efficient.

c. about the same.

Less efficient because all other elements in the linkedList must be pushed up or down 1 to accommodate the new object, or the list will have to be sorted after the insert. However, when inserting into an array, the element will be held in the array if there is space in the specified part of the array which is more efficient as it will not affect the rest of the elements.

A way to make this insertion more efficient from the list point of view is to use pointers to minimise the amount of elements that have to be shifted.