LINKED LISTS

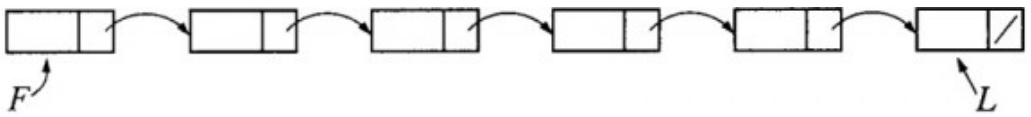
Short Answer

In this material, without any other specification, we assume that we are using a singly linked list, in which a node has two members, ***key*** *– a positive integer representing data and* ***next*** *– a pointer to the next element in the linked list.*

```
struct Node {
    int key;
    Node* next;
};
```

1. * What are the advantages and disadvantages of *linked lists* over *arrays*?
2. * What is a *singly linked list*? How is the end of a *singly linked list* usually signified?
3. * Describe basic operations of a *singly linked list*.
4. ** Deleting a node from a *linked list* requires two steps
   1) Remove the node from the list without breaking the links held by the next pointers.
   2) Delete the node from memory.

   Why is it impossible to use the delete operator to remove the node from memory, i.e., step 2 only?
5. ** Consider a *singly linked list*. Make a proper node definition for each of the following cases.
   - A node that contains a character
   - A node that contains the name of a person
   - A node that contains the name, date of birth, gender, and nationality of a person
   - A node that contains the ID, name, midterm, and final grades of a student
6. ** Consider the following *singly linked list*, in which `F` points to the first element in the list and `L` points to the last element in the list.

   Which of the following operations has its running time proportional to the length of the list? Briefly explain for each case.
   - Insert an element before the first element
   - Insert an element after the last element
   - Delete the first element
   - Delete the last element
7. ** Let `n` and `m` be two variables of type `Node`. What is the result of each of the following operations on a *singly linked list* where `n` (but not `m`) belongs to?

- `n.next = n.next.next`
- `m.next = n.next; n.next = m`
- `n.next = m; m.next = n.next`

8. ** Point out the syntax errors and/or the semantic error in the following code fragments.

- This function traverses a *singly linked list*.

```
void traverse(Node *head){
    while (head->next != nullptr){
        cout << head->key;
        head = head->next;
    }
}
```

- This code segment performs a *linked list* operation.

```
Node *nodePtr, *nextNode;
nodePtr = head;
while (nodePtr != nullptr){
    nextNode = nodePtr->next;
    nodePtr->next = nullptr;
    nodePtr = nextNode;
}
```

- This function deletes the last node of a *singly linked list,* in which the pointer `head` points to the first node of the linked list.

```
void delete_end_node(Node* head){
    Node *temp1, *temp2;
    if (head == nullptr)
        cout << "The list is empty!" << endl;
    else {
        temp1 = head;
        while (temp1->next != nullptr){
           temp2 = temp1;
           temp1 = temp1->next;
        }
        delete temp1;
        temp2->next = nullptr;
    }
}
```

9. ** What will be printed out for each of the following code segments. Assume that `head` points to the first node of the *singly linked list.*

- `34, 5, 2, 1, 9, 8, 0, 3, 6`

```
while (list->key > 1)
    list = list->next->next;
cout << list->key + list->next->key;
```

- 1, 2, 3, 4, 5, 6

```
sum = 0;
while (list->key < 5) {
    sum += list->key;
    list = list->next;
}
cout << sum;
```

- 4, 5, 21, 28, 29, 18, 60, 73, 86

```
list->next->next = list;
cout << list->next->next->next->next->key;
cout << list->next->next->next->next->next->next->next->next->key;
```

- 4, 5, 21, 28, 29, 18, 60, 73, 86

```
sum = 0;
while (list->key < list->next->key){
    list = list->next;
    sum += list->key;
}
cout << sum;
```

10.** What task does the following function perform?

```
Node* func(Node* &head){
    Node *p, *q;
    if ((head == nullptr) || (head->next == nullptr))
        return head;
    q = nullptr; p = head;
    while (p->next != nullptr){
        q = p;
        p = p->next;
    }
    q->next = nullptr;
    p->next = head;
    head = p;
    return head;
}
```

11.** Consider the following operations for a *singly linked list.*

(a) Create an empty list.

(b) Destroy a list.

(c) Determine whether a list is empty.

(d) Determine the number of items in a list.

(e) Insert an item at a given position in the list.

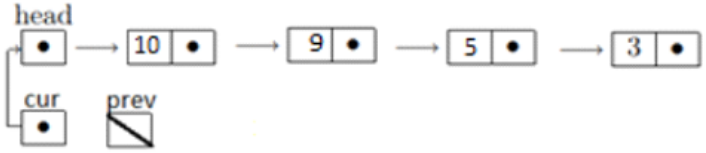
(f) Delete the item at a given position in the list.

(g) Look at (retrieve) the item at a given position in the list.

Using only the operations above, describe how to replace an item at a given position with a new item by picking appropriate operations and specifying necessary parameters.

For example, given the list of integers 0, 3, 8, 9, 4, 10, if we replace the fourth item with 7, the new list will be 0, 3, 8, 7, 4, 10.

12. ** Given the following *sorted singly linked list*,



Write the code to perform the following operations.

- Create a new node, `toBeInserted`, having value 4 and find the right position to insert it into the above sorted linked list.
- Delete the last node in the sorted linked list.
- Delete the first node in the sorted linked list.
- Traverse the linked list and count the number of elements that are greater than 5.

13. ** Is it possible to delete a node X from a *singly linked list* if only the pointer to X is available and the head pointer is not known. If yes, describe the delete operation. If no, briefly explain.

14. *** Explain clearly how to fulfill the following operations in constant time on a *singly linked list*.

- Remove the middle node, given only the pointer to that node (i.e., no head pointer).
- Given the pointer `current` to a node in the list, implement the `InsertBefore` operation.

15. *** There are many ways to link dynamically allocated data structures together. Two linked list variations are *doubly linked list* and *circular linked list*.

What are their advantages and disadvantages over the *singly linked list*? List some practical applications of these linked list variations.

Fill-in-the-Blank

1. * Fill in each of the following blanks with an appropriate terminology.

- After creating the head pointer for a *linked list*, you should make sure it points to the _____ node in the list, before using it in any operations.
- The address of the last node in a *linked list* is stored in a separate location, called the _____ pointer.
- Consider a *singly linked list* that keeps track of a head pointer and a tail pointer. Which of these pointers will change during an insertion into an empty queue (i.e., insert to the end of the list)? _____

2. ** The following function is supposed to reverse a *singly linked list*. There is one line missing at the end of the function. Fill in the blank with an appropriate line of code.

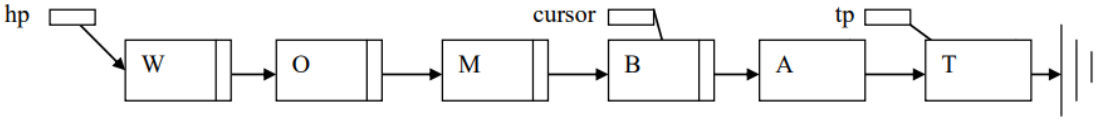
```
void reverse(Node* &head){
    Node* prev = nullptr;
    Node* current = head;
    Node* next;
    while (current != nullptr){
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    ____________________________________________ /*ADD A STATEMENT HERE*/
}
```

3. ** The following function takes a *singly linked list* of integers, `{1, 2, 3, 4, 5, 6, 7}` in given order, as a parameter and rearranges the elements of the list.

```
void rearrange(Node* &node){
    if ((!node) || !node->next)
        return;
    Node *p = node;
    Node *q = node->next;
    while (q){
        int temp = p->key;
        p->key = q->key;
        q->key = temp;
        p = q->next;
        q = p ? p->next : 0;
    }
}
```

After the function completes execution, the content of the list is _____.

4. ** Using the *singly linked list* depicted below, in which each node contains a single character as data and a next pointer, answer the following questions.



- What letter is stored in `hp->key`? _____
- What letter is stored in `hp->next->key`? _____
- What letter is stored in `cursor->next->key`? _____
- Identify the link that points to NULL by giving its reference
  from `tp` _____
  and from `cursor` _____
- Show what would happen if the following code is executed

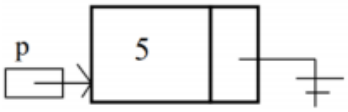
```
for (cursor; cursor != tp; cursor = cursor->next);
```

_____

5. ** In each part below, you are given a code segment and you are asked to draw a picture that shows the result of the execution of the code segment. Your picture should indicate the value of every declared variable and the value of every field in every node.

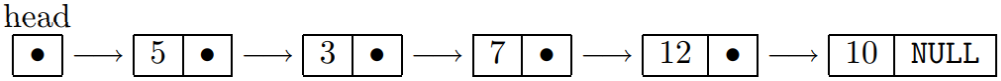
For example, after the following code segment executes, the picture should look like this.

```
Node *p = new Node();
p->key = 5;
p->next = nullptr;
```



`Node *r, *p, *q;`
`p = new Node();`
`p->key = 10;`
`q = new Node();`
`p->key = 10;`
`q->next = nullptr;`
`p->next = q;`
`r = p;`
	`Node *p, *q, *r;`
	`p = new Node();`
	`p->key = 5;`
	`p->next = nullptr;`
	`q = new Node();`
	`q->key = 10;`
	`q->next = nullptr;`
	`p->next = q;`
	`r = p;`
	`r->next = nullptr;`
		`Node *p, *q, *r;`
		`r = new Node();`
		`r->key = 5;`
		`r->next = nullptr;`
		`p = new Node();`
		`p->key = 6;`
		`p->next = r;`
		`q = new Node();`
		`q->key = 10;`
		`q->next = p->next;`

6. ** Consider the *singly linked list* of integers represented by the following diagram.



Draw a diagram of the list after the following code segment is executed.

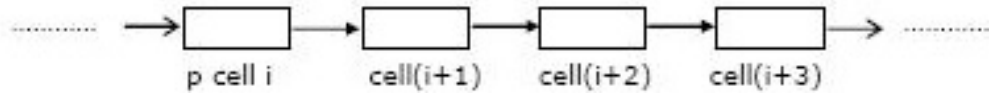
```
Node* prev = head->next;
Node* nodeToInsert = new Node();
nodeToInsert->key = 4;
nodeToInsert->next = prev->next;
prev->next = nodeToInsert;
```

What is the value of `prev->key` after the above code segment is executed? _____

In addition to the code above, assume the following code is also executed. Draw a diagram of the list after code execution.

```
prev = prev->next;
prev = prev->next;
Node* curr = prev->next;
prev->next = curr->next;
delete curr;
curr = nullptr;
```

7. ** Let `p` be a pointer in a *singly linked list*, as shown in the below figure.



What does each of the following assignment statements achieve?

- `q = p->next` _____
- `p->next = q->next` _____
- `q->next =(q->next)->next` _____
- `p->next)->next = q` _____

8. ** The intent of the method below is to delete the last node of the list.

```
void removeLast(Node first) {
    Node p = first;
    Node q = p.next;
    while (q.next != null) {
        p = q;
        q = q.next;
    }
    p.next = null;
}
```

Which of the following linked lists for which this method works correctly? Justify your selection.

- ○ No linked lists
- ○ All nonempty linked lists
- ○ All linked lists with more than one node
- ○ The empty list and all linked lists with more than one node
- ○ All linked lists

9. *** Given the following statements.

A. An empty linked list

B. A linked list with only one node

C. A linked list with more than one node

D. No appropriate statement

Fill in each of the following table entries with an appropriate statement when the corresponding conditions are TRUE. *Note that* `head` *and* `tail` *are pointers that point to the first and last nodes of the singly linked list, respectively, and the function* `getNext()` *returns the next node in the list.*

Conditions	Statements
`head == nullptr`	
`head != tail`	
`head == tail`	
`head->getNext() == tail`	
`tail == nullptr`	

10. *** Given the following statements.

A. An empty node

B. The next-to-last node

C. The last node

D. No appropriate statement

Let `curNode` be the pointer of type `Node`, which is initialized to the head of a *singly linked list*. Choose an appropriate statement about `curNode` after each of the following loops is executed. *Note that the function* `getNext()` *returns the next node in the list.*

Loops	Statements
`while (curNode->getNext()->getNext() != nullptr)` `  curNode = curNode->getNext();`	
`while (curNode != nullptr)` `  curNode = curNode->getNext();`	
`while (curNode->getNext() != nullptr)` `  curNode = curNode->getNext();`	

True or False

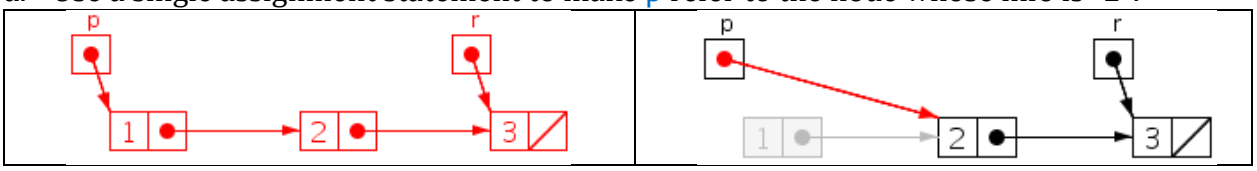
Choose T (True) or F (False) for each of the following statements and then briefly explain in one or two sentences.

1. T F The *linked list* supports both random and sequential access to its elements.
2. T F The programmer must know in advance how many nodes in a *linked list*.
3. T F Arrays have to fix their sizes in advance, while *linked lists* can change their sizes any time.
4. T F To store a fixed number of integers, an array always uses more memory than a *singly linked list* does
5. T F When the head pointer points to `nullptr`, it signifies an empty *linked list*.
6. T F Insertion/deletion in a *linked list* incurs less data moves than those in an array.
7. T F In a *singly linked list*, every node (except the last) contains the address of the next node.
8. T F You can use the pointer to the head of a *linked list* to traverse the list.
9. T F In a *singly linked list*, it is easier to insert a new node after a specific node rather than before the specific node.
10. T F To insert a new node into a *singly linked list*, the addresses of every other nodes in the list must change.

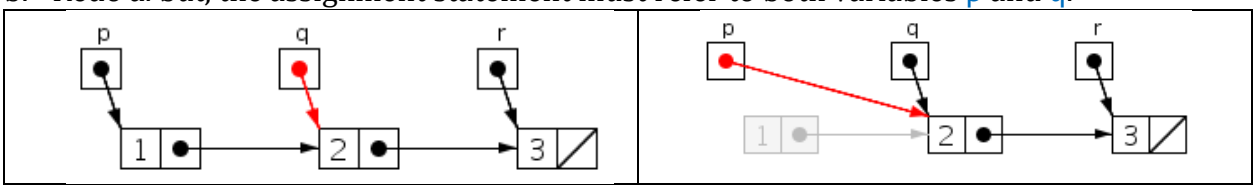
Algorithm Workbench

1. * Let S be a *singly linked list* that has at least three elements. Write code to complete each of the following requirements.
   - Return the last-but-two node of S
   - Invert S in place (i.e., no use of additional data structures)
   - Remove a given node n from S
2. * Define a singly linked list that holds double values. Write code to store the following values, 12.7, 9.65, 8.72, and 4.69, to the list and then reverse the order of the items.
3. * Assume that a *linked list* has been created with the head pointer head. Write code to traverse the linked list and display the content of each node. Then write code to destroy this linked list.
4. ** In this question, the tables are shown with left columns indicating the initial configurations and right columns indicating the corresponding final configurations, which will be achieved by solving the problems raised in the questions.

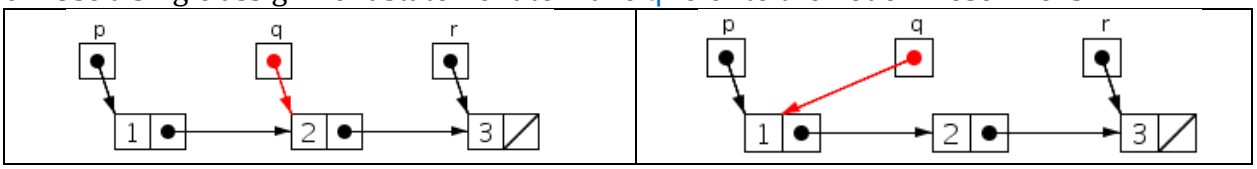
   a. Use a single assignment statement to make p refer to the node whose info is "2".



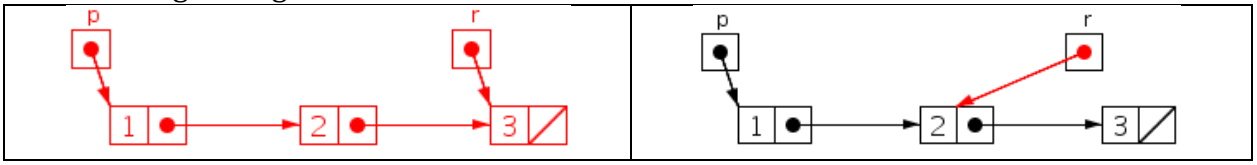
   b. Redo a. but, the assignment statement must refer to both variables p and q.



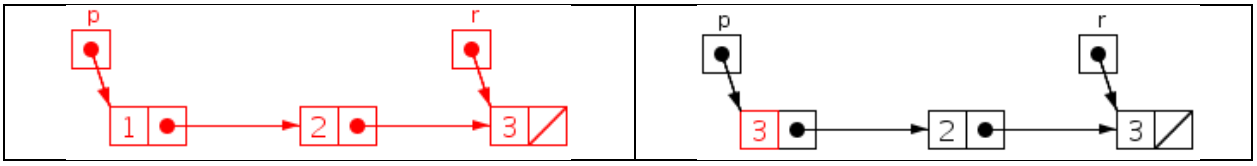
   c. Use a single assignment statement to make q refer to the node whose info is "1".



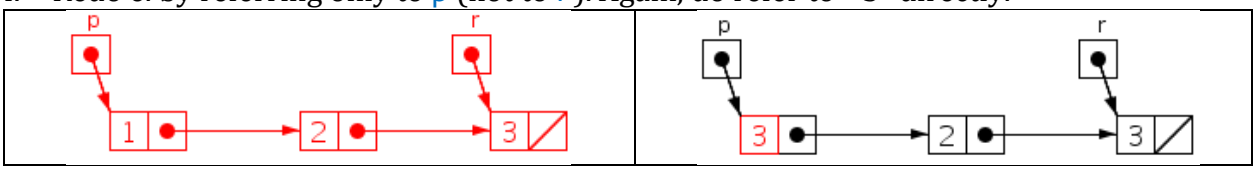
   d. Use a single assignment statement to make r refer to the node whose info is "2".



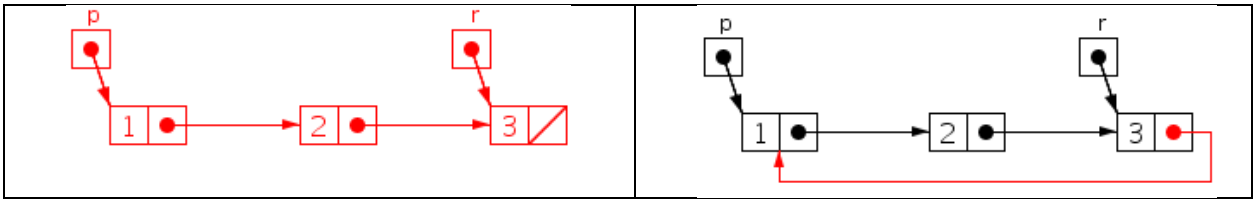
   e. Use a single assignment statement to set the info of the node referred to by p equal to that referred to by r. You must access this info through r; do not refer to "3" directly.



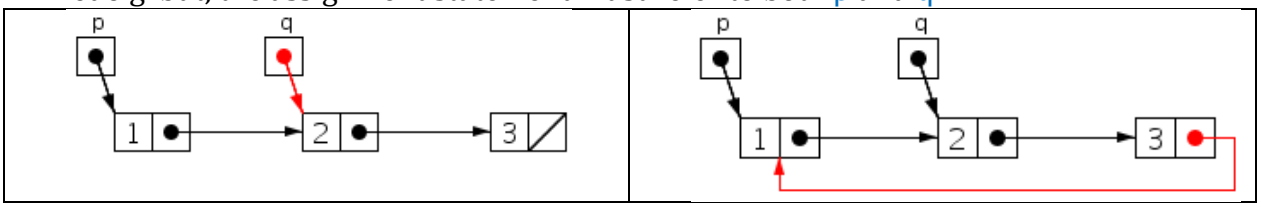
f. Redo e. by referring only to p (not to r). Again, do refer to "'3" directly.



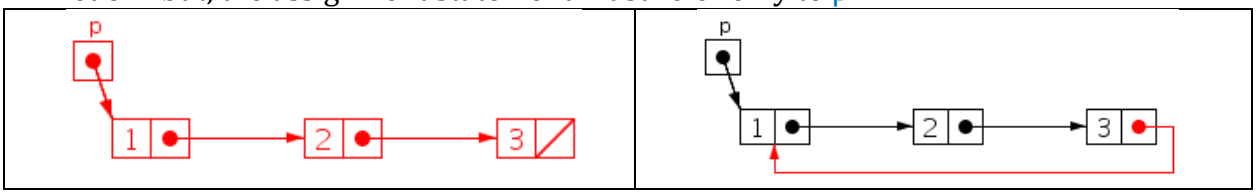
g. Write a single assignment statement to transform the *linked list* into a *circular linked list*. The assignment statement must refer to both p and r.



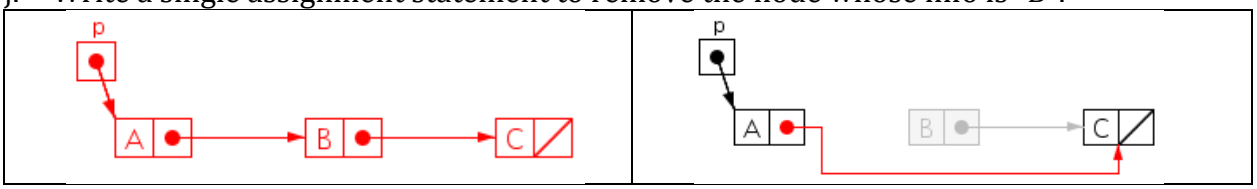
h. Redo g. but, the assignment statement must refer to both p and q.



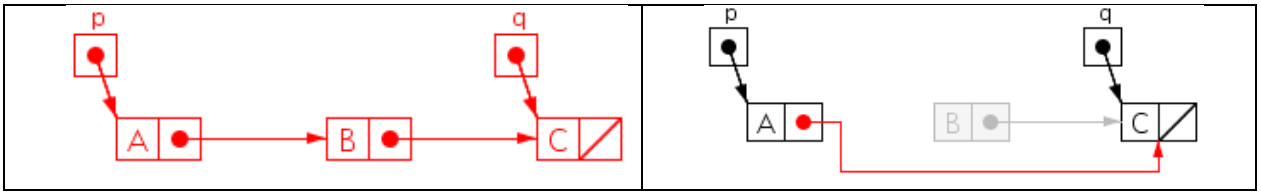
i. Redo h. but, the assignment statement must refer only to p.



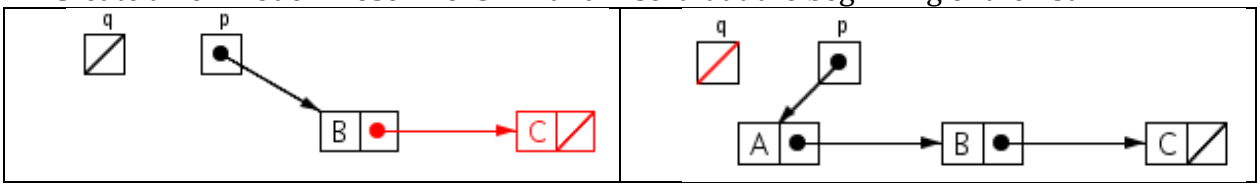
j. Write a single assignment statement to remove the node whose info is "B".



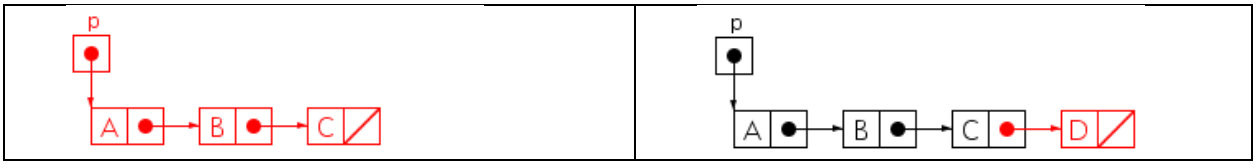
k. Write a single assignment statement to remove the node whose info is "B". The statement must refer to both p and q.



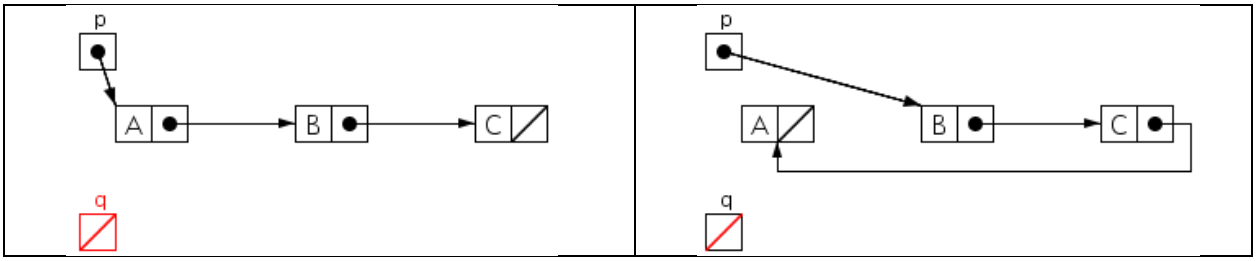
l. Create a new node whose info is "A" and insert it at the beginning of the list.



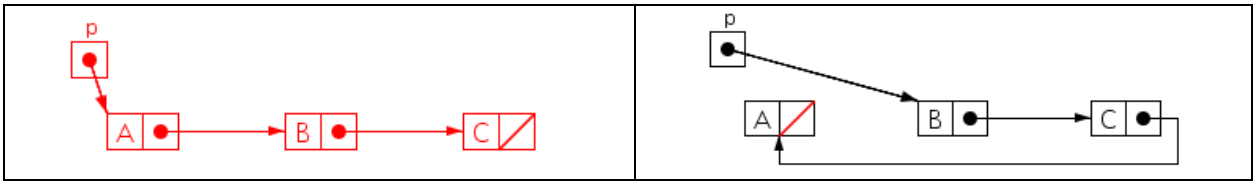
m. Create a new node whose info is "D" and insert it at the end of the list.



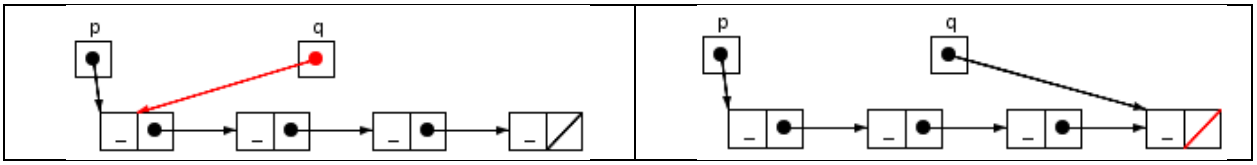
n. Remove the node at the beginning of the list and insert it at the end of the same list. The removal must refer to both p and q.



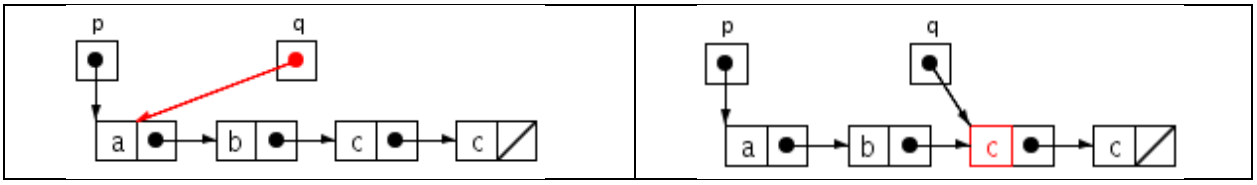
o. Redo e. but, this time, the removal must only refer to p.



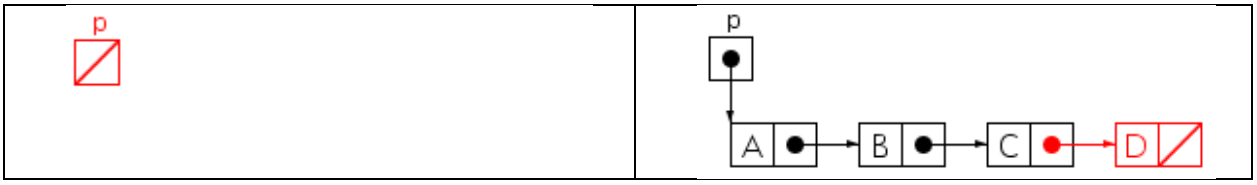
p. Write a while loop to make q refer successively to each node in the list and q must end up referring to the last node in the list.



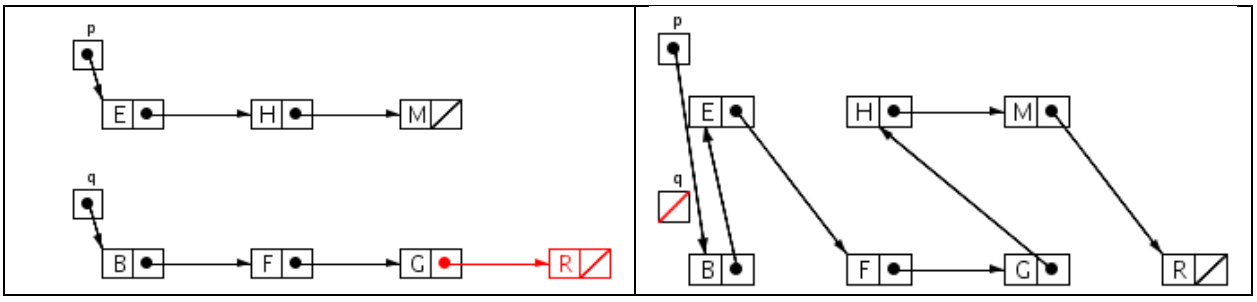
q. Write a while loop to make q refer successively to each node in the list until q refers to the first node with info (lowercase) "c".



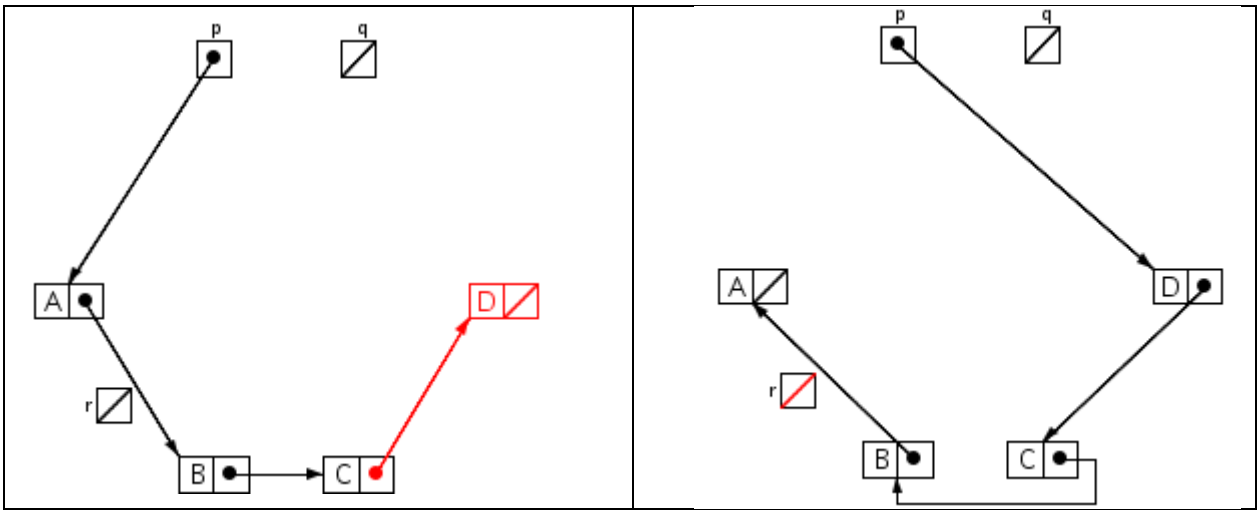
r. Use four assignment statements, each referring to p, to create a *linked list* headed by p and containing 4 nodes with info 'A', 'B', 'C', and 'D', in this order.



s. Merge the two lists headed by p and q into a single list headed by p such that the nodes are sorted in alphabetical order.



t. Use only the three variables, p, q, and r, to reverse the order of the nodes in the list.



5. ** Given a *singly linked list* of integers. Implement bubble sort on the given list. Data moves should be performed by modifying the links rather than swapping the key values only.

   Repeat the question for selection sort, insertion sort, and interchange sort.

6. ** A decimal number is represented by a *linked list* in reverse order where each node contains a single digit. For example, the number 1234 is stored as $4 \rightarrow 3 \rightarrow 2 \rightarrow 1$.

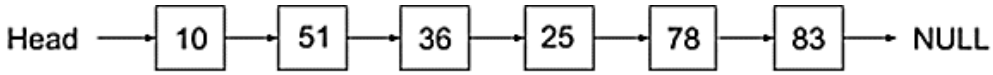
   Implement a function to add two decimal numbers and return the resulting sum, whose prototype is as follows: `Node* addDecimalNumbers(Node* list1, Node* list2)`

   Create your own recursive function (call it once from the function `addDecimalNumbers`) so that you can pass any additional information from call to call. You can assume that the size of the resulting linked list will be at least as large as the size of the larger of the two linked lists.

7. ** Given a *singly linked list* with the head pointer `head` and the tail pointer `tail`. Write code to split the linked list at the middle into two halves and store each half in a new linked list.

   For example, split $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$ into $1 \rightarrow 2 \rightarrow 3$ and $4 \rightarrow 5 \rightarrow 6$.

8. ** Assume you have the following *singly linked list* and you have direct access to the head of the list.



   Write a code to split the given linked list. The split point should be decided when the user inputs an integer S. Every value less than S should be in one list and every value greater than S should be in the second list. For example, if the user inputs 60, then the values, 10, 51, 36 and 25, should be in one list and the others should be in another list.

9. *** The polynomial $a_n x^n + a_{n-1} x^{n-1} + \cdots a_0$ can be represented by a *singly linked list*, in which each node contains an integer for the power of x and another integer for the corresponding coefficient.

   The user inputs from console a polynomial as follows, a_nx^n + a_{n-1}x^(n-1) + ... + a_0.

   A term a_ix^i can be omitted if a_i is 0, e.g., the polynomial $3x^4 + 7x^2 + 5$ can be inputted as either 3x^4 + 7x^2 + 5 or 3x^4 + 0x^3 + 7x^2 + 0x^1 + 5. If a coefficient is negative, a minus sign is used in place of a plus sign, e.g., 3x^5 - 7x^3 + 2x^1 – 8 or -7x^4 + 5x^2 + 9.

Define necessary structures for the polynomial and its terms.

The user inputs a polynomial and a value of x. The program evaluates the polynomial with the given value.

The user inputs two polynomials. The program performs additions, subtraction, and multiplication on the two given polynomials.

10. *** In an ancient land, the beautiful princess Eve had many suitors. She decided on the following procedure to determine which suitor she would marry. First, all the suitors would be lined up one after the other and assigned numbers. The first suitor would be number 1, the second number 2, and so on up to the last suitor, number n. Starting at the first suitor, she would then count three suitors down the line (because of the three letters in her name) and the third suitor would be eliminated from winning her hand and removed from the line. Eve would then continue, counting three more suitors, and eliminating every third suitor. When she reached the end of the line, she would continue counting from the beginning. For example, if there were six suitors, then the elimination process would proceed as follows:

123456	Initial list of suitors, start counting from 1
12456	Suitor 3 eliminated, continue counting from 4
1245	Suitor 6 eliminated, continue counting from 1
125	Suitor 4 eliminated, continue counting from 5
15	Suitor 2 eliminated, continue counting from 5
1	Suitor 5 eliminated, 1 is the lucky winner

Write code to create a *singly linked list* and determine which position you should stand in to marry the princess if there are n suitors. Your program should simulate the elimination process by deleting the node that corresponds to the suitor to be eliminated in each step of the process. Be careful that you do not leave any memory leaks.

Although singly linked list works for this problem, it is not the best choice. Think of another linked list that better suits this problem.