Arab Academy For Science and Technology & Maritime Transport

College of Engineering & Technology

Computer Engineering Department

EXAMINATION PAPER – Week 7

Course Title: Data Structures

Course Code: CC215

Date: Mon. Nov, 10-2014 Lecturer: Dr. Manal Helal

Time allowed: 60 mins Start Time: 10:30 a.m.

Student's name:	Reg.# :

Question #	**Marks**	
	Available	**Actual**
Lists	4	
Stacks	8	
Queues	8	
Total	**20**	
Lecturer	Name : Dr. Manal Helal	
	Signature :	
	Date:	

MPC6/1-1

Lists: **[4 points]**

1) Explain one benefit that:

A) [0.5 points] arrays have over linked lists.

Ans. 1) O(1) access time, (using indices to specify a location in the contagious array memory blocks)

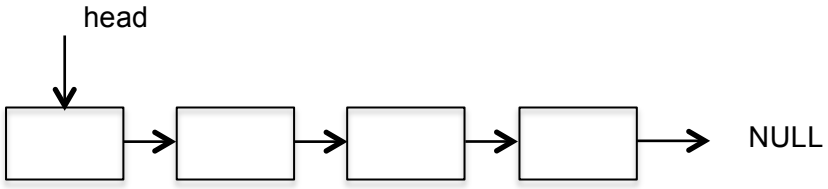
B) [0.5 points] linked lists have over arrays.

Ans. O(1) insertion and deletion (changing links only, not shifting backward or forward)

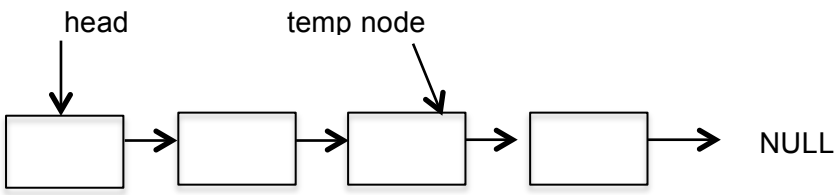
2) Swap two adjacent elements by adjusting only the links (and not the data) using:

A) [1.5 points] Singly linked lists

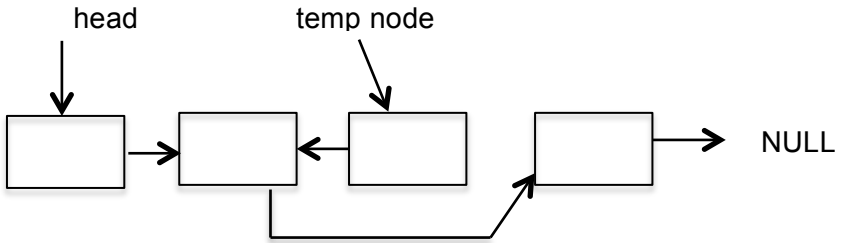
1)



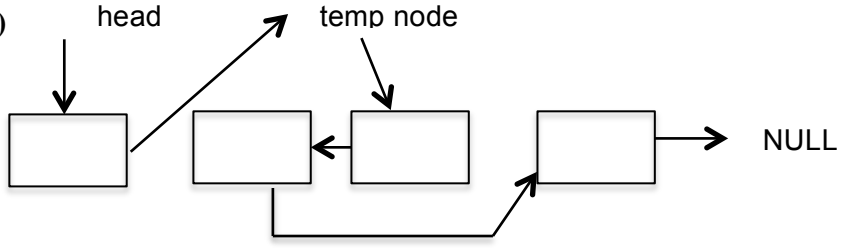
2)



3)

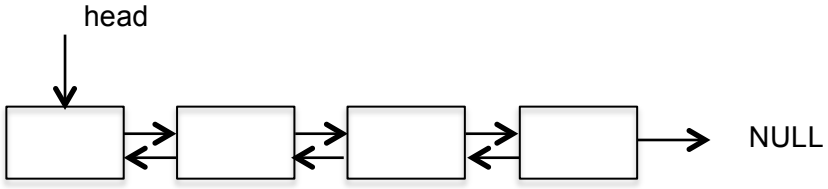


4)

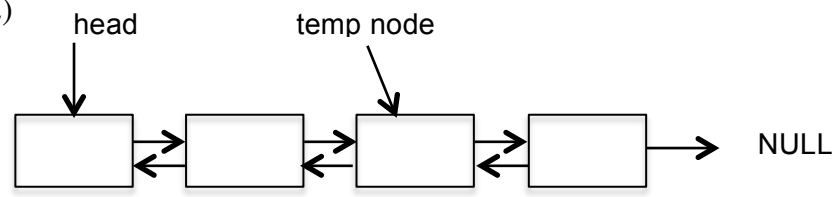


B) [1.5 points] Doubly linked lists

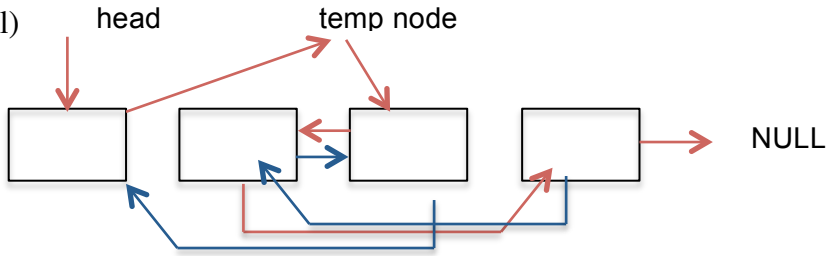
1)



2)



final)



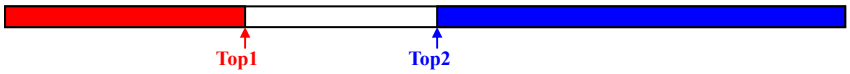
The final step illustration is approximating the reverse (previous pointers) in the temporary node from the bottom links, but using red for next pointers, and blue for previous pointers.

Anything resembling these steps will take the marks for the count of correct links.

th

Stacks: **[8 points]**

2) [6 points] Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used.




```
#ifndef _Stack_h
#define _Stack_h
struct StackRecord;

#define MaxElements 100
struct StackRecord {
        int Capacity;
        int Top1;
        int Top2;
        ElementType Array[MaxElements];
}
typedef struct StackRecord Stack;

int IsEmpty( Stack S, int Stacknum );
int IsFull( Stack S );
Stack CreateStack( int MaxElements );
void Push( ElementType X, Stack S, int Stacknum );
ElementType Top( Stack S, int Stacknum );
void Pop( Stack S, int Stacknum );

#endif /* _Stack_h */

/* The main idea is that the bottom of the first stack is at Array[0]  */
/* The bottom of the second stack is at Array[ MaxElement ] */
/* Top1 and Top2 point to the top elements of two stacks respectively */
Stack CreateStack ( int MaxElements ) {
                Stack S;
                /* create an array for stack elements */
                S.Capacity = MaxElements;
                S.Top1 = -1;
                S.Top2 = MaxElements;
                /* initialize 2 empty stacks */
                return ( S );
        }
}
int IsEmpty ( Stack S, int Stacknum ) {
        if ( Stacknum ==1) /* check stack 1 */
                return S.Top1 == -1;
        else {
                if ( Stacknum == 2 ) /* check stack 2 */
                        return S.Top2 == S.Capacity;
                else
                        return Error("stack number error");
        }
}
int IsFull( Stack S ) {
        return S.Top1+1 == S.Top2;
}

void Push( ElementType X, Stack S, int Stacknum ) {
        if ( IsFull( S ) ) FatalError("Stack is full");
        if ( Stacknum == 1 ) /* push onto stack 1 */
                S.Array[++S.Top1] = X;
        else {
                if ( Stacknum == 2 ) /* push onto stack 2 */
                        S.Array[- -S.Top2] = X;
                else
                        Error("stack number error");
        }
}
void Pop( Stack S, int Stacknum ) {
        if ( IsEmpty( S, Stacknum ) )
                FatalError("Stack is empty");
        if(Stacknum==1) /*popstack1*/
                S.Top1 --;
        else {
                if(Stacknum==2) /*popstack2*/
                        S.Top2 ++;
                else
                        Error("stack number error");
        }
}
```

3) [2 points] Show how to implement a program to store new actions by users to be able to undo and redo them when required, taking into considerations actions bounds, such as no history of actions, or reaching the limit of actions to store.

Ans:

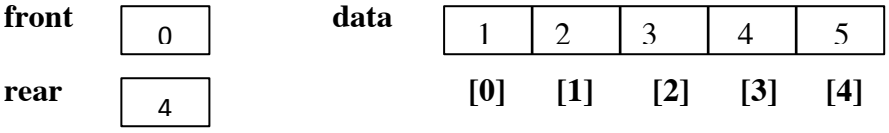
```
bool insert_Action(ActionType action) {
        return undoStack.push(action);
}

bool undo_Action(ActionType action) {
        ActionType = action;
        If (undoStack.pop(&action) != true)
                return false;
        If (redoStack.push(action) ) != true)
                return false;
        execute(reverse(action));              /* model the reverse of the execution of the action*/

        return true;
}

bool redo_Action(ActionType action) {
        ActionType = action;
        If (redoStack.pop(&action) != true)
                return false;
        execute(action);                       /* model the execution of the action*/
        return true;
}
```

Queues: **[8 points]**

4) [6 points] After executing the following code:

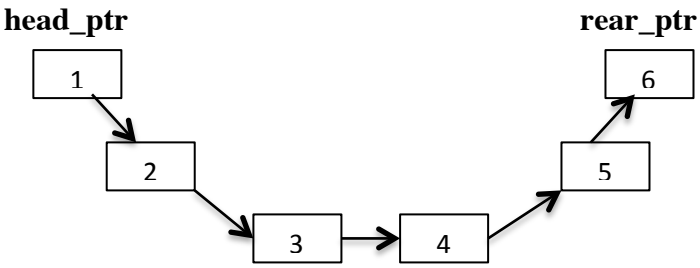
```
Queue<int> q;
q.enqueue (1);
q.enqueue (2);
q.enqueue (3);
q.enqueue (4);
q.enqueue (5);
cout << q.get_front( );
cout << q.get_front( );
q.enqueue (6);
```

A. Suppose that q is represented by a partially filled array with a CAPACITY of 5. Draw the state of the private member variables of q after the above code:



1.5 marks for the right front and rear and 1.5 points for the right contents of the array

B. Suppose that q is represented by a linked list. Draw the linked list after the above code:



1.5 marks for the right front and rear pointers and 1.5 points for the right contents of the linked list

5) [2 points] Describe two "real life" applications of a queue.

Anything that will be processed later (will take time) and reqyure First in First Out ordering, such as print jobs spooling to the printer, task priority queues to be processed by the processor, restaurants' orders,