<u>Question 1)</u> Consider the following partial C++ program:

```
int count(char* &b)
{     char A[5]={'1', '2', '3', '4', '5'}, *c, t;
      int X, *p;
      c = A;
      t = *c; *c = *b; *b = t;
      b = c;
      p = &X;
      for (int i=0; i<=4; i++)
            if (A[i] = *b) {X = i; i = 4};
      return *p;
};

void main (void)
{     char X[3]={'M', 'T', 'A'}, *y;
      y = X + 2;
      cout << count(y);
...
```

Show the contents of all local and global data items. What will be printed out?

Initially in Main:

X: ['M', 'T', 'A']
y points to X[2], passed to count(y).

Initially in Count:

B is a pointer to char, passed by reference,
so it refers to y in Main and points to X[2]

A: ['1', '2', '3', '4', '5']
c points to A[0], t is a char variable
X is a local integer, p is a local ptr-to-integer

t gets value '1', A[0] gets 'A' when "to the location pointed by c, carry the contents of location pointed by b which refers to y" is executed...

X[2] gets value '1' when "to the location pointed by b which refers to y, carry the contents of t" is executed...

y points to A[0] when "copy c into b which refers to y" is executed

p points to the local integer X

upon the first turn of the for loop with i=0, A[0] is found to be equal to itself pointed by y, so the local integer X takes the value of i which is 0, and i is made 4. before the next turn, i is incremented, making it 5, so the loop is not executed any more.

The value of integer X pointed by local p, is returned from count.

So: Output: 0

At the point of output:

X: [‘M’, ‘T’, ‘1’]
Y points to A[0] in the local area of count, no longer active.

In the local area of count:
A: [‘A’, ‘2’, ‘3’, ‘4’, ‘5’] c points to A[0], t is a char and contains ‘1’
X is a local integer, contains 0, p is a local ptr-to-int and points to the local integer X

<u>Question 2)</u>Consider each of the following program structures. Determine their O( ), Ω( ) and θ( ) time complexities, explaining your reasoning briefly. For all cases, the following time complexities are known:

Func1(n)=θ(n); Func2(n)=θ(2^n); Func3(n)=θ(log n);
Func4(n)=O(n); Func4(n)=Ω(log n).

(a) for (int k=0; k < n; k++)
if (k%2==0) Func1(n);
else Func2(n);

O(2^n)	Ω(2^n)	θ (2^n)
Reason: *Func2( ) is dominant as n → ∞*		

(b) if (x < A) Func1(n);
else if (x < A + 1000) Func2(n);
else if (x < A + 5000) Func3(n);
else Func4(n);

O(2^n)	Ω(log n)	θ(not defined)
Reason: *Best case and worst case behaviours depend on x.*		

(c) int Func0(int n)
{ if n < 2 return 2;
if (n%3 == 0) Func4(n);
else Func3(n);
Func0(n/3);
}

O(n log n)	Ω($\log^2 n$)	θ(not defined)
Reason: *The number of recursive calls is θ($\log_3 n$). In the best case, in all calls, Func3 is called. In the worst case, in all calls, worst case behaviour of Func4 is encountered.*		

Question 3)

a. (5 pts)

<table>
<tr><td>Consider the following C++ class declaration:

<pre>class Z
{
private:
    int z1;
    int *z2;
public:
    void Z(int n=0)
    {
    z1=n;
    z2 = new int[z1];
    for (int i=0; i<z1; i++)
        *(z2+i) =i*i;
    };
    int Zread(int j)
    {
    return *(z2+j);
    }
}</pre></td><td>Draw the constructed data structures after the execution of the following code:
<pre>...
Z *A;
A = new Z(3);
Z B(A -> Zread(1));
...</pre><hr>Your answer:</td></tr>
</table>

b) (10 pts)

<table>
<tr><td>Consider the following C++ class declaration:

<pre>template <class T>
class K
{
private:
    T member1;
    T *member2
public:
    K (const T &m1, const T &m2)
    {
        member1 = m1;
        member2 = new T(m2);
    }
}</pre></td><td>i. Draw the constructed data structure after the execution of the following code:
....
<pre>K <int> A(20,10), B(700,900);
B = A;</pre>......<hr>Your answer:</td></tr>
</table>

ii. Is there a problem in the solution? What would you do to solve this problem?

c) (10 pts) Assume that class DClass is declared and implemented in file “d.h” with all necessary member functions to handle its dynamic data. Considering the code segment given below complete the following table so that for each executed code line write if constructor, copy constructor, destructor or assignment operator is used and write details as given in the first line as an example.

```
line# Code
1     # include <iostream.h>
2     # include "d.h"
3     template <class T>
4     DClass<T> MyFunc(DClass<T> A, DClass<T> &B, T m1)
5     {  DClass<T> C(m1);
6        DClass<T> D=C ;
7        C=B;
8        return C;
9     };
10    void main()
11    { DClass<int> E(100);
12      DClass<int> *g ;
13      g=new DClass<int>(200);
14      E=MyFunc(*g,E,300);
15      delete g ;
16    }
```

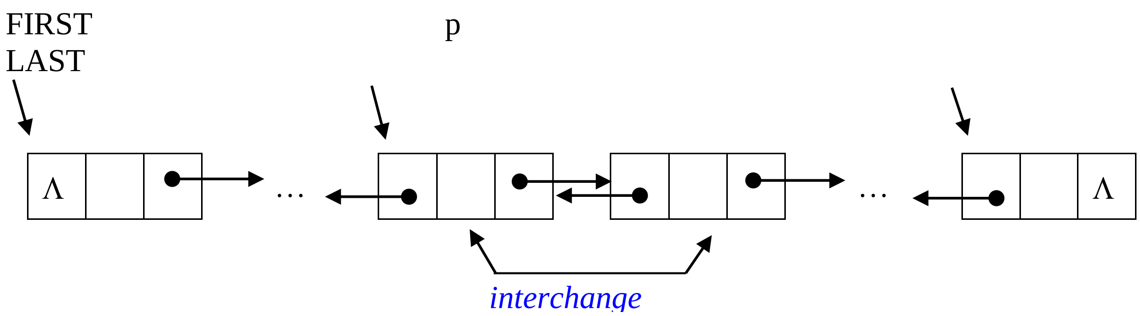
output produced	**on object**	**at line#**	**by calling DClass member function**
A static object E with member variable 100	E	11	Constructor
A dynamic object g with member variable 200	g	13	Constructor
Object A with member variable 200	A	4	Copyconstructor
A static object C with member variable 300	C	5	Constructor
A static object D with member variable 300	D	6	Copycontructor
Object C with member variable 100	C	7	Assignment operator
Return object with member variable 100	return	8	Copycontructor
Objects A,C,D are deleted	A,C,D	9	Destructor
Object E with member variable 100	E	14	Assignment operator
Return object is deleted	return	14	Destructor
Dynamic object g is deleted	g	15	Destructor
Static object E is deleted	E	16	Destructor

Question 4) (20 pts.) SOLUTION

Consider the doubly linked list node structure:

prev	data	next

in which **prev** is used to point to the previous node and **next** is used to point to the next node. Assume that FIRST and LAST are pointers to point the first and the last nodes of a doubly linked list, and p is a pointer to a node (not necessarily an intermediate node) in this list. The **prev** field of the node pointed by FIRST and the **next** field of the node pointed by LAST are NULL.



Write a function that will interchange the places of node pointed by p and the next node. Do not attempt to interchange data fields but change only link fields. Be careful handling the special cases mentioned below.

```
void    Interchange(doublynode<T>    &FIRST,    doublynode<T>    &LAST,
doublynode<T> p)
{if ((p == null) || (p->next == null))
{cout<<"node interchange impossible"; return;}

// adjust prev and next fields as necessary
{ doublynode<T> *q=p->next;

  q->prev=p->prev;
  p->next=q->next;
  if (q->next!=null){q->next->prev=p};
  if (p->prev!=null){p->prev->next=q};
  q->next=p;
  p->prev=q;

// adjust headers as necessary
if(p=FIRST)
// case: p originally pointed to the first node
{
FIRST=q;
}
if (q=LAST)
// case: the next node was originally the last node
{
LAST=p;
}
}}
```

<u>Question 5)</u>. Consider the following C++ class statement:

```
template <class T>
class X
{public: int n; // number of items in the dynamic array
      p *T;      // dynamic array
/* initializing constructor initializes all entries in the
dynamic array of size nn identically as item */
      X (const int nn=0, const T &item);
      X (const X<T> &xx);
      ~X(void);
      X<T> &operator =(const X<T> &r);}
```

(a) (20 pts.) Implement the initializing and copy constructors, destructor and overloaded assignment operator for this class.

SOLUTION:

```
template <class T> //Initializing constructor
X<T>::X (const int nn=0, const T &item)
{n=nn;
 if(n) {p=new T [n]; for (int i=0; i<n; i++) *(p+i)=item;}};


template <class T> //Copy constructor
X<T>::X (const X<T> &xx)
{n=xx.n;
 if(n) {p=new T [n]; for (int i=0; i<n; i++) *(p+i)=xx.p+i;}};


template <class T> //destructor
X<T>::~X(void)
{if(n) delete [ ] p;};


template <class T> //Overloaded assignment
X<T> &X<T>:: operator =(const X<T> &r)
{if(n) delete [ ] p;
 n=r.n;
 if(n) {p=new T [n]; for (int i=0; i<n; i++) *(p+i)=r.p+i;}
 return *this;
};
```

Continued at the back of the sheet... →

(b) (5 pts.) Assuming that linked list based implementations of the `Stack<T>` and `Queue<T>` classes are available with the following public methods:

```
void Push(T item); T Pop(void); int StackEmpty(void);
void QInsert(T item); T QRemove(void); int QueueEmpty(void);
Stack<T> &operator =(const Stack<T> &r);
Queue<T> &operator =(const Queue<T> &r);
```

Draw a diagram that shows the data structures created when the following code is executed:

```
void main (void)
{ Queue<int> A; for (int i=0; i<5; i++) A.QInsert(i);
  X < Queue <int> > B (2, A);
  X < int > C (3, 5);
  Stack <X <int> > D; for (int i=5; i<6; i++) D.Push (C);}
```

SOLUTION:

<u>Question 6)</u> (25 pts) Show all your work. You are given the `Node` Class and the `GetNode` function as defined in the lectures. **Use `GetNode` if you need to create a new node rather than the constructor.**

`LStack` class uses a linked list for storing the items in the stack. **After a member function that modifies the stack content returns, the number of nodes which consume memory is the same as the number of items stored in the stack.**

```
template <class T>
class Node
{
private:
Node <T> *next;
T data;
Node (const T &item, Node<T>*
ptrNext=0);
void InsertAfter(Node<T> *p);
Node <T> *DeleteAfter(void);
Node<T> *NextNode(void) const;}
template <class T>
Node<T> *GetNode(const T& item,
Node<T> *nextPtr=NULL)
```

```
#include "node.h"
template <class T>
class LStack
{
private :
Node<T> *top;
public :
LStack(void); //constructor
to initialize top to NULL for
empty stack
void Push(const T& item);
T Pop(void); };
```

a) Implement the member function `Push` for the `LStack` class above in the space provided below.

```
template <class T>
void LStack<T>::Push(const T& item)
{
top=GetNode(item, top);
}
```

b) Implement the member function `Pop` for the `LStack` class above by completing the blanks. `Pop` returns the popped item.

```
template <class T>
T LStack<T>::Pop()
{//Handle the case when the stack is empty
if(top==NULL)
{cerr<<"Stack Empty\n";
exit(1);
}
T popped=top->data;
if(top->NextNode()==NULL)//Handle the case when the stack becomes empty after Pop
{delete top;
top=NULL;}
else
{
//create a new top that is a copy of the item that is below the original top
Node<T>*second=top->NextNode();
top=GetNode(second->data, top);
//Delete the items that are extra copies and get the desired stack content
delete top->DeleteAfter();
delete top->DeleteAfter();
}
return popped;
}
```

`LQueue` class given below uses a linked list for storing the items in the queue. **After a member function that modifies the queue content returns, the number of nodes which consume memory is the same as the number of items stored in the queue.**

```
#include "node.h"
template <class T>
class LQueue
{
private:
Node<T> *qfirst;//Pointer to the first item in the
queue
public:
LQueue(void); //constructor to initialize qfirst
to NULL for empty queue
void QInsert(const T& item);
QDelete(void);
void QMovetoFront(int pos); //constructor to
initialize qfirst to NULL for empty queue
};
```

c) Implement the member function `QMovetoFront` which moves the node at position `pos` to the front of the Queue. The first item in the Queue is at position 0. Assume `pos` is always a valid number with respect to the size of the queue and the queue is never empty.

```
template <class T>
void LQueue<T>::QMovetoFront(int pos)
{
//Handle the case when there is only one item in the queue
if(qfirst->NextNode()==NULL)
return;
Node<T> *current=qfirst;
Node<T> *prev=qfirst;
//current and prev pointers move in pair until current is at pos
for(int i=0;i<pos; i++)
{prev=current;
current++;}
//create a new first node with the data content of the current
node and delete the extra copy at pos
qfirst=GetNode(current->data, qfirst);
prev->DeleteAfter();
delete current;
}
```

d) Implement the global template function `void QDeletePos (LQueue<T>___Q, int pos)` in the space provided below which deletes the item at position `pos`. Assume `pos` is always a valid number with respect to the size of the queue and the queue is always non-empty.

```
template <class T>
void QDeletePos(LQueue<T> &Q,int pos)
{
Q.MoveFront(pos);
Q.QDelete();
}
```

<u>Question 7)</u> (a) (8 pts) Assuming that the `TreeNode` structure is defined as in class, complete the missing parts of the following C++ function that searches a given binary search tree for a given key value and returns either a pointer to the node where it is found, or a null pointer value if the key value is not found in the tree:

```
template <class T>
TreeNode<T>* sbt (TreeNode<T>* t, T key)
{ if (t == NULL) return NULL;//nothing to process

  else {if (t -> data == key) return t;//key found

        else {if (t -> data < key) //must search left subtree

                 {if (t -> left == NULL) return NULL;//not
              there
                    else
                   return sbt (t -> left, key)};//search left

              else {if (t -> right == NULL) return NULL;
                 else

                      return sbt (t -> right, key)};//search right
              } //end key not found at this node
        } //end (sub)tree not null
} //end sbt
```

(b) (5 pts) What is the O(.) complexity of the function in part (a) in terms of n, the number of items in the tree? Justify your answer with one sentence.

ANSWER: O(n) as each node is visited at most once.

(c) (6 pts) Given the following execution time complexities:

`Func1(n) = θ(n), Func2(n) = θ(2ⁿ), Func3(n) = θ(log n)`

What are the `Ω(.)`, `O(.)` and `θ(.)` complexities of the following function in terms of `n`? Explain your reasoning briefly:

```
int Funcm (int n)
{ if (n <= 1)   return 1;
  if (n <= 100) return Func2(Funcm(n/10));
  if (n <= 1000)return Func3(Funcm(n/10));
  return Func1(Funcm(n/10))};
```

SOLUTION:

As $\Omega(.)$, $O(.)$ and $\theta(.)$ are all defined for $n \rightarrow \infty$, **only** the call to Func1 (Funcm (n/10))has to be considered.

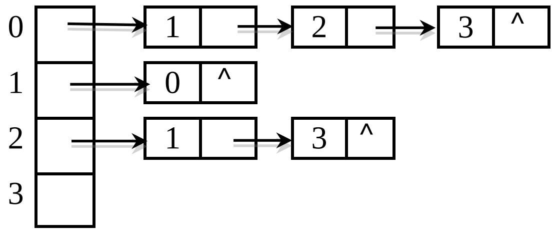
For large n, $\theta(\log_{10} n)$ calls to Funcm are necessary to reduce n to $n <= 1000$.

Since each call to Func1(n) executes in $\theta(n)$ time , Funcm(n) = $\theta(n.\log n)$

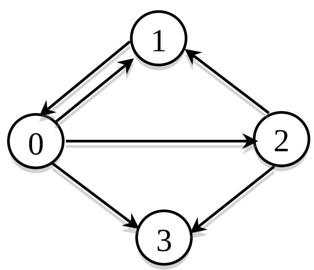
(d) (6 pts) Insert, in the given sequence, the following key values into a B-Tree of order three, based on alphabetical ordering. Draw the resulting B-Tree after the underlined keys are inserted:

ROSE, <u>VIOLET</u>, HYACINTH, <u>BEGONIA</u>, DAFFODIL, ORCHID, NARCISSUS, <u>MAGNOLIA</u>, EDELWEISS, AZALEA, PANSY, <u>MIMOSA</u>.

Question 8) (6 points) Here is an adjacency list representation of a directed graph:



(a) Draw a picture of the directed graph that has the above adjacency list representation.



(b) Another way to represent a graph is an adjacency matrix. Draw the adjacency matrix for this graph.

	0	1	2	3
0	0	1	1	1
1	1	0	0	0
2	0	1	0	1
3	0	0	0	0

<u>Question 9)</u> (14 points) You are given the Graph class definition as follows

```
class Graph
{public:
        Graph(int n) {};                            // Constructor;  n: number of vertices
        Graph(const Graph&) {};                     // Copy constructor
       ~Graph(void) {};                             // Destructor
        int n();                                    // Returns the number of vertices
        int e();                                    // Returns the number of edges
        int degree(int v);                          // Returns the degree of vertex v
        void setEdge(int v1, int v2, int weight);   // Set the weight for an edge
        void delEdge(int v1, int v2);               // Delete an edge  (v1, v2)
        bool isEdge(int v1, int v2);                // Determine if the edge (v1, v2) is in
                                                    //        the graph
        int weight(int v1, int v2);                 // Return weight of the edge (v1, v2):
        int first(int v);                           // Returns v's first neighbor;  Returns
                                                    //        //"-1" if there is no neighbor
        int next(int v, int w) ;                    // Returns v's next neighbor after w;
                                                    //        //Returns "-1" if there is no
                                                    //        // more neighbor after w
        ….
        int getMark(int v);                         // Get mark value (visited=1,
                                                    //        //unvisited=0) for vertex v
        void setMark(int v, int val);               // Set mark value for vertex v
        void clearMark(void);                       // Set all mark values as unvisited
};
```

A **tree** is an directed connected graph without cycles. In order to check if a graph G is a tree having node v as root, a modified depth first or breadthfirst traversal algorithm starting with node v may be used. You are requested to complete the following modified breadthfirst graph traversal algorithm to check whether the given undirected graph G corresponds to a tree with root v (returns True) or not (returns False). Assume that G is not empty.

```
int isTree_BreadthFirst(Graph G,  int v)
{// Traverses graph  G beginning at vertex v iteratively by using breath first strategy
        const int visited=1, unvisited=0;
        const int True=1, False=0;
        Queue q;
        q.insert(v);                  // add v to the queue
        G.setMark(v, visited)         //Mark v as visited;
        int nodes_visited=1;          // number of nodes visited initialised to 1
        while (!q.isEmpty())
        {
          w=q.delete()
          for (u=G.first(w); u!= -1;  u=G.next(w, u))     // for each vertex u adjacent to  w
             {
                if   (                                              )     // if cycle detected
```

```
                    return False;

                else  //  no cyle detected yet
                    {



                      q.insert(u);
                    } //end of else
              } //end of  for
        } // end of while; traversal completed, therefore no cyle occurred

        if  (                                    )                 )    //if connected

              return  True
        else
              return  False ;

}// end of function
```

Solution:

```
int isTree_BreathFirst(Graph G,  int v)
{// Traverses graph  G beginning at vertex v iteratively by using breath first strategy
        const int visited=1, unvisited=0;
        const int True=1, False=0;
        Queue q;
        q.insert(v); // add v to the queue
        G.setMark(v, visited) //Mark v as visited;
        int nodes_visited=1; // number of nodes visited initialised to 1
        while (!q.isEmpty())
        {
          w=q.delete()
          for (u=G.first(w)      ; u!= -1                    ;  u=G.next(w, u)          ) // for each
vertex u adjacent to  w
              {
                if   (          G.getMark(u)= =visited                    )     // if cycle detected

                    return False;

                else  //  no cyle detected yet
                    {

                      G.setMark(u,visited);
                      nodes_visited ++;
```

```
                    q.insert(u);
                  } //end of else
           } //end of  for
    } // end of while; traversal completed, therefore no cyle occurred

     if  (           nodes_visited== G.n()                    )    //if connected

         return  True
    else
         return  False ;

}// end of function
```

Question 10) The following numbers are stored in an array are to be sorted:

1	709
2	015
3	702
4	660
5	221
6	162
7	603

a) Show the contents of the array for the first three passes (a "pass" refers to a full turn of the inner loop for a single value of the outer loop index) if Selection Sort is used.

	initial		Step1		Step2		Step3
1	709	1		1		1	
2	015	2		2		2	
3	702	3		3		3	
4	660	4		4		4	
5	221	5		5		5	
6	162	6		6		6	
7	603	7		7		7	

b) Repeat (a) if Bubble Sort is used

	initial		Step1		Step2		Step3
1	709	1		1		1	
2	015	2		2		2	
3	702	3		3		3	
4	660	4		4		4	
5	221	5		5		5	
6	162	6		6		6	
7	603	7		7		7	

c) Repeat (a) if Quicksort is implemented, taking the item with the lowest index as pivot in each pass:

	initial		Step1		Step2		Step3
1	709	1		1		1	
2	015	2		2		2	
3	702	3		3		3	
4	660	4		4		4	
5	221	5		5		5	
6	162	6		6		6	
7	603	7		7		7	

<u>Question 11)</u> (10 pts)

```
void GnomeSort(int A[], int N)
{   int pos = 1;
    int pass = 0;
    while (pos < N)
        if (A[pos] >= A[pos-1])
            pos = pos + 1;
        else {
            swap(A[pos],A[pos-1]);
            if (pos > 1)
                pos = pos - 1;
            else
                pos = pos + 1;
            pass=pass+1;
            // check point: output pass, A, pos
            }//end if
        //end if
    //end while
 }//end function
```

a) Given the initial value for array A (note that array size N=7), show the content of A and pos as the GnomeSort fuction passes through the check point (show only upto 6 passes).

	Initial	pass	1	2	3	4	5	6
A 0	6		4	4	2	2	2	2
1	4		6	2	4	4	3	3
2	2		2	6	6	3	4	4
3	3		3	3	3	6	6	6
4	1		1	7	7	7	7	5
5	7		7	5	5	5	5	7
6	8		8	8	8	8	8	8
pos	1		2	1	2	2	1	4

b) What are O(.) and Ω(.) complexities for Gnome Sort

$O(n^2)$, $\Omega(n)$

Question 12) (10pts)

```
void ShakerSort(int *myArray, int first, int last){
      int exchange=1;
   while(exchange){
     exchange = 0;
     for(int i=last; i > first; i--) {
       if(myArray[i-1] > myArray[i]) {
         Swap(myArray[i],myArray[i-1]);
         exchange = 1;
       }/* end if */
     }/* end for */
      /* cpt1 */
      first++;
     for(i=first+1; i <= last; i++) {
       if(myArray[i-1] > myArray[i]) {
         Swap(myArray[i],myArray[i-1]);
         exchange = 1;
       }/* end if */
     }/* end for */
     /* cpt2 */
     last--;
   } /* end while */
 } /* end function */
```

Given the initial values of `first`, `last`, `myArray` and `exchange`, write their values as `ShakerSort` passes through control points `cpt1` and `cpt2`

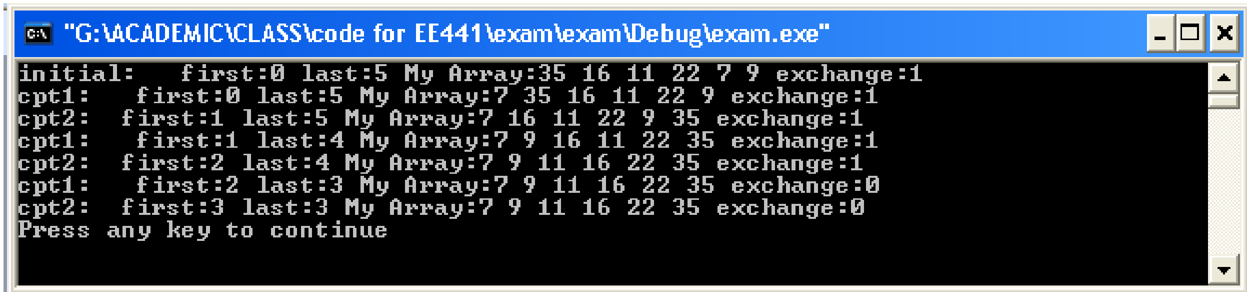
Control point	First	Last	myArray						exchange
			0	1	2	3	4	5	
initial	**0**	**5**	**35**	**16**	**11**	**22**	**7**	**9**	**1**
Cpt1	0	5	7	35	16	11	22	9	1
Cpt2	1	5	7	16	11	22	9	35	1
Cpt1	1	4	7	9	16	11	22	35	1
Cpt2	2	4	7	9	11	16	22	35	1
Cpt1	2	3	7	9	11	16	22	35	0
Cpt2	3	3	7	9	11	16	22	35	0

Question 13) **(15 pts)** Find out what will be the output if the following code is executed. For each output line, if a part of the line is produced by PrintMembers function, mention the name of the object on which this member function is applied. For example if the constructor is called for object A(1,100), then the output line together with your comment will be as: "Constructor: 1/100 //A"

```
template <class T>
class DC
{
public:
T  member1;
T *member2;
DC(const T  &m1, const  T &m2)
   {member1=m1;
    member2=new T(m2);
    cout<<"Constructor:"; PrintMembers( ); };
DC(const DC<T> & obj)
  {member1=obj.member1;
  member2=new T(*obj.member2);
  cout<<"Copy Constructor:"; PrintMembers( ); };
DC<T>& operator=(const DC<T> rhs)
  {member1=rhs.member1;
   *member2=*rhs.member2;
     cout<<"Assignment:"; PrintMembers( );
    return *this;};

void MemberOne(T m1)
    {member1=m1;
      cout <<"MemberOne: ";PrintMembers( );};
void MemberTwo(T m2)
  {*member2=m2;
   cout <<"MemberTwo: "; PrintMembers( );};
void PrintMembers (void)
     {cout<<member1<<"/"<<*member2<<endl; };
~DC(void)
  {cout<<"Destructor:"; PrintMembers();
   delete member2;};
};

template <class T>
DC<T> MyFunc(DC<T>& X, DC<T>  Y)
{ X.MemberOne(3);
  Y.MemberTwo(4);
  DC<T> Z(X.member1,*Y.member2);
   return Z;
};

void main( )
{ DC<int> A(1,100), B=A;
  B.MemberOne(2);
  cout << "call MyFunc"<<endl;
  B=MyFunc(A,B);
  cout << "return MyFunc"<<endl;
}
```

SOLUTION:
Constructor: 1/100   // A
CopyConstructor: 1/100   // B
MemberOne: 2/100   // B
Call MyFunc
CopyConstructor:2/100 //Y
MemberOne: 3/100 //X
MemberTwo: 2/4 //Y
Constructor:3/4 //Z
CopyConstructor:3/4 //return obj
Destructor: 3/4 //Z
Destructor:2/4 //Y
Assignment:3/4 //B
Destructor:3/4 // return obj
returnMyFunc
Destructor:3/4 //B
Destructor:3/100 //A

Output


"G:\ACADEMIC\CLASS\code for EE441\exam\exam\Debug\exam.exe"
initial: first:0 last:5 My Array:35 16 11 22 7 9 exchange:1
cpt1:   first:0 last:5 My Array:7 35 16 11 22 9 exchange:1
cpt2: first:1 last:5 My Array:7 16 11 22 9 35 exchange:1
cpt1:   first:1 last:4 My Array:7 9 16 11 22 35 exchange:1
cpt2: first:2 last:4 My Array:7 9 11 16 22 35 exchange:1
cpt1:   first:2 last:3 My Array:7 9 11 16 22 35 exchange:0
cpt2: first:3 last:3 My Array:7 9 11 16 22 35 exchange:0
Press any key to continue


"G:\ACADEMIC\CLASS\code for EE441\exam\exam\Debug\exam.exe"
Constructor:1/100
Copy Constructor:1/100
MemberOne: 2/100
call MyFunc
Copy Constructor:2/100
MemberOne: 3/100
MemberTwo: 2/4
Constructor:3/4
Copy Constructor:3/4
Destructor:3/4
Destructor:2/4
Assignment:3/4
Destructor:3/4
return MyFunc
Destructor:3/4
Destructor:3/100
Press any key to continue

Question 14) (15 pts) The following overflow handling method uses a collision resolution function in addition to the hash function. If hash(key) results in no collision key is stored at hash(key). If collision occurs, next probes are performed by considering the formula:

$$h_i(key) = (hash(key) + g(i)) \% N$$

where:

hash(key) is the hash function
g(i) is the collision resolution function
i=1, 2 … is the number of the current attempt (probe) to insert an element
N is the hash table size

a)
Let N=8
hash(key)=key % N
g(i) =i*i

Insert the following keys into the hash table in the given order:

10, 3, 2, 14, 6

(not possible to insert 6)

	keys
0	
1	
2	10
3	3
4	
5	
6	2
7	14

b) What is the problem with the resulting h_i**(key)** when the collision resolution function g(i) given in part (a) is used

The resulting h_i(key) function may repeat itself before an empty position is reached, which happens when 6 is to be inserted.

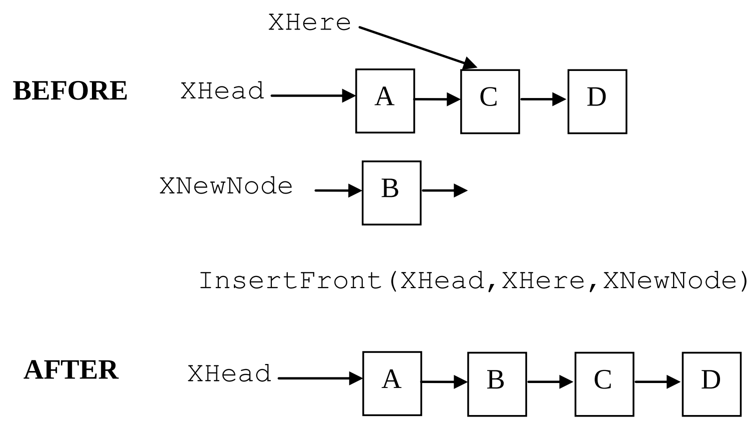
c) propose a g(i) function such that the resulting method is equivalent to linear probing

g(i)=i

Question 15)

a) Using the `Node` **class below** complete the following **global** function `InsertFront` which inserts a given node pointed by `NewNode` in front of a node in a given Linked List. The Linked List is identified by a pointer to its head that is called `Head`. The insertion is infront of the node that is denoted by pointer `Here`. **After** **`InsertFront`** **is called, it should be possible to conduct further operations on the Linked List. Assume that the LinkedList is not empty.** The call of `InsertFront` on an example LinkedList ponted by `XHead` is shown in the figure below.

```
template <class T>
class Node
{
private:
Node <T> *next;
public :
T data;
Node (const T &item, Node<T>* ptrNext=0);
void InsertAfter(Node<T> *p);
Node <T> *DeleteAfter(void);
Node<T> *NextNode(void) const;
}
```



Implement in the given space below. **Also complete the blank in the function header.**

```
void InsertFront(Node<T>___________Head, Node<T>* NewNode, Node<T>*
Here)
{ Node<T>* current=Head, * prev =Head;
//Traverse the list until Here is found

while(_____________________________)
{


}
//Special case: Insert infront of the Head Node
if(Here==Head)
{


}

//No special case, insert simply in front of Here
else
{

}

}
```

b) UTEM university has 8 students. The student records are stored in a hash table of size 8 (bucket size=1). The student ID expressed in binary form is the key used for hashing. The student IDs are 5 bits. The Student list and their IDs are as follows:

ID (in binary)	**Name Last Name**
11000	Ellen Ripley
11010	Luke Skywalker
11100	Han Solo
11110	Vincent Vega
10000	Tyler Durden
10010	Lester Burnham
10100	Sarah Connor
10110	Marty McFly

i) What is the key density of this hash table

ii) Find a perfect hash function for this set of student IDs such that no two keys are synonyms. There will be no collision in the resulting hash table. State your hash function and show your resulting key placement in the hash table below:

Hash address	**ID (key)**
000	
001	
010	
011	
100	
101	
110	
111	

Hash Function:

Solution:

```
void InsertFront(Node<T>* & Head, Node<T>* NewNode, Node<T>* Here)
{ Node<T>* current=Head, * prev =Head;


//Traverse the list until Here is found
while(current!=Here)
{
prev=current;
current=current->NextNode();
}

//Special case: Insert infront of the Head Node

if(Here==Head)
{
NewNode->InsertAfter(Head);
Head=NewNode;
}


//No special case, insert simply in front of Here
else
{
prev->InsertAfter(NewNode);
}
}
```

b)

i) What is the key density of this hash table=8/32=0.25

Hash address	**ID (key)**
000	10000
001	10010
010	10100
011	10110
100	11000
101	11010
110	11100
111	11110

Hash Function: Take the middle 3 bits and use for addressing

Question 16. (x pts) Explain your answers with a single verbal sentence or a short proof.

I. Problem P1 is shown to be NP-complete. Problem P2 can be reduced to P1 using an algorithm A. A is $O(N^2)$

a) Can you find a polynomial time solution for P1?

NO (YES ONLY IF NP=P)
$P1 \in$ NP_Complete, no known algorithm in P for P1
otherwise it would be NP=P

b) Can you find a polynomial certifier for P1?

YES
$P1 \in$ NP_Complete $\Rightarrow P1 \in NP \Rightarrow$ Certifier(P1) $\in$ P1 exist by definition of NP

c) Can you find a polynomial certifier for P2?

$P2 \leq_p P1$ is given $\Rightarrow$ also $P2 \in NP$ (notice that $P \subseteq NP$)
$\Rightarrow$ Certifier (P2) $\in$ P exist by definition of NP

d) What are the possible problem complexity classes for P2?

certainly $P2 \in NP$, possibly $P2 \in P$ also

II. Problem Q1 is known to be NP-complete. Problem Q2 is known to be NP. Problem Q1 can be polynomially reduced to Q2. Problem Q3 is known to be NP. Can you find a polynomial time algorithm to reduce Q3 to Q2?

YES:

$Q1 \in$ NP_Complete and $Q1 \leq_p Q2 \Rightarrow Q2 \in$ NP_Complete by theorem in lecture notes

$Q3 \in NP$, $Q2 \in$ NP_Complete $\Rightarrow Q1 \leq_p Q2$ by definition of NP _completeness

So a polynomial time algorithmto reduce Q3 to Q2 exist.