The University of Jordan	**KASIT/CS Department**
Data structures (Mid exam)	**Second Semester(2008)**
Name:	**Reg.no:**
Section: **Lecturer:**	**Seat no.:**

NOTE:
In all questions assume that the functions that appear in class definition as prototypes are fully defined, and you must use them to define your functions if needed.

Q1) Given the following definition of a list:

```
const int max=100;
class list
{
        int L[max];
        int length;
public:
        errorcode delete_element(int el);    // deletes particular element
        bool search (int el);   //returns true if the element exists in a list
                .
                .
                .
};
```

A) Wrie a member function **return_first** that will return the value of the first element in the list. **(2 marks)**

```
int list::return_first( )        è 1 Mark
{ return L[0];}          è 1 Mark
```

B) Write a member function **remove_all** that will delete all occurrences of particular element from the above defined list. **(3 marks)**

```
void list::remove_all(int el)      è 1 Mark
{ while (search(el))        è 1 Mark
delete _element(el);}       è 1 Mark
```

Q2) Given the following definition of a simply linked list:

```
template <class t>
class list
{
        node<t> * first;        //points to the first node
public:
        int size ( );           // returns number of nodes in a list
                .
                .
                .
};
```

A) Define a a member function **return_last** that will return the value of the last element of the above list. **(5 marks)**

```
template <class t>                                  è 1 Mark
t list<t>::return_last( )                           è 1 Mark
{ node<t>* tmp=first;                               è 1 Mark
for( ;tmp->link!=0;tmp=tmp->link);                  è 1 Mark
return tmp->info;}                                  è 1 Mark
```

B) Define a member function **count** that will return number of times a particular element appears in a list. **(5 marks)**

```
template <class t>
int  list<t>::count(t el )                          è 1 Mark
{ int c=0;                                          è 1 Mark
node<t>* tmp=first;
for( ;tmp->link!=0;tmp=tmp->link)                   è 1 Mark
        if(tmp->info==el)                           è 1 Mark
        c++;
        return c;                                   è 1 Mark
}
```

Q3) Given the following definition of a doubly linked list:

```
template <class t>
class list
{
        node<t> * last;        //points to the last node
public:
                .
                .
                .
};
```

A) Write a member function **print_for** that will print the above list forward **(3 marks)**

```
template <class t>
void  list<t>::print_for( )
{ node<t>* tmp=last;
for( ;tmp->back!=0;tmp=tmp->back);
for( ;tmp!=0;tmp=tmp->next)
        cout<<tmp->info<<" ";
}
```

è 1 Mark

è 1 Mark

è 1 Mark

B) Write a member function **min** that will return the pointer that will point to minimum element in the above list. **(5 marks)**

```
template <class t>
node<t>*   list<t>::min( )
{ node<t>* tmp=last,mp=last;
for( ;tmp->!=0;tmp=tmp->back);
if(tmp->info<mp->info)
        mp=tmp;
return mp;
}
```

è 1 Mark

è 1 Mark

è 1 Mark

è 1 Mark

è 1 Mark

Q4) Given the following definition of a simply circular linked list:

```
template <class t>
class list
{
        node<t> * last;        //points to the last node
public:
        bool is_empty( )       //returns true if the list is empty
                .
                .
                .
};
```

A) Define a proper **costructor** for a linked list defined above. **(2 marks)**

```
template<class t>
list<t>::list( )        è 1 Mark
{last=0;}       è 1 Mark
```

B) Define a **destructor** for the above class. **(5 marks)**

```
template <class t>
 list<t>::~list( )     è 1 Mark
{
while(!is_empty( ))        è 1 Mark
{
node<t>* tmp=last;        è 1 Mark
last=last->next;     è 1 Mark
delete  tmp; è 1 Mark
}
}
```