

OBJECT ORIENTED PROGRAMMING

KEYS to 1st Midterm Exam

(There are 2 Questions. 2-Hour Exam, Give your answer inside the corresponding box)

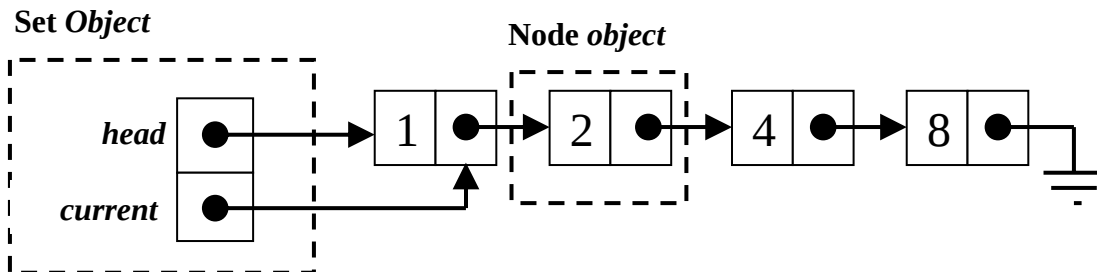
Student ID: 490606

Name: Binnur Kurt

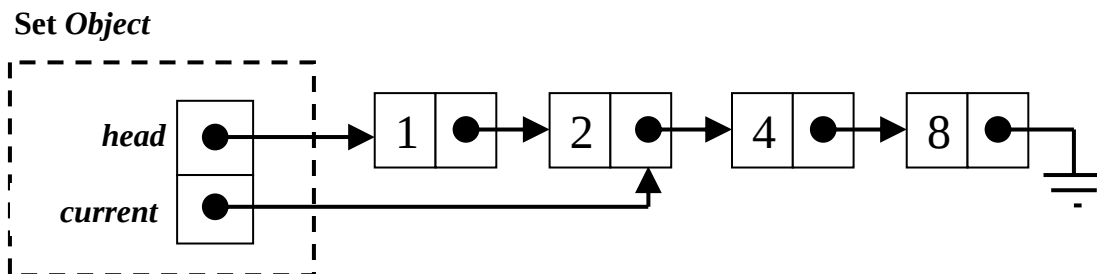
Signature _____

Q.1) (a) (20) You will design a **Set** class in order to model *set* data structure using **object oriented programming** approach. A set contains **unique** elements and keeps its elements **sorted** after insertions and deletions. The following figure will help you to decide on class data members. A typical application code and its output for the sample set (1,2,4,8) is given below.

Initially head and current points to the first node:



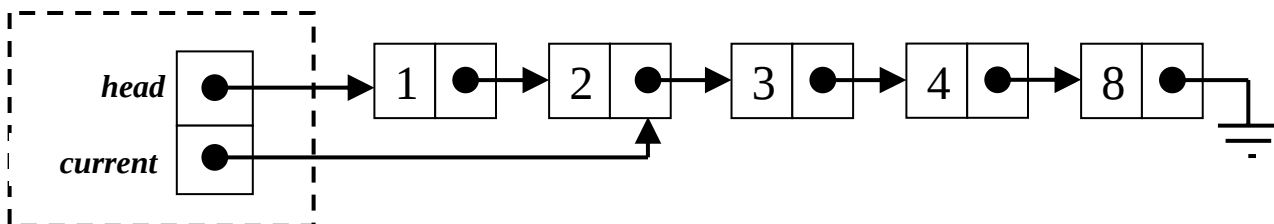
After `set->next()` call, `current` points to the next node:



After `set->add(4)` call the set remains the same since the set already contains "4".

The call `set->add(3)` causes the set to rearrange its elements so that it remains sorted:

Set Object



Typical usage of the Set class

```
Set *set = new Set();
...
while (set->hasMoreElements())
    cout << endl << "Set element: " << set->next();
```

The program output:

```
Set element: 1
Set element: 2
Set element: 3
Set element: 4
Set element: 8
```

```

class Set {
private:
    Node *head;
    Node *current;
    void DeepCopy(const Set&);
    void Destroy(Node*);
public:
    Set() ; //default constructor
    ~Set(); //destructor
    Set(const Set&); // copy constructor
    void operator=(const Set&);
    bool add(int);
    bool hasMoreElements()const;
    int next() ;
} ;

```



Give your answer inside the box

```

class Node {
private:
    Node * next;
    int hold;
public:
    Node(int);
    Node* getNextNode()const;
    void setNextNode(Node*);
    int getValue()const;
    void setValue(int);
};

```

Notes and Guidelines:

Consider the **only required** methods.

Do you need default constructor?

Do you need copy constructor?

Do you need destructor?

Do you need assignment operator?

(b) (40) Give the implementations of the methods.

Give your answer inside the box

```

Set::Set():head(0L),current(0L){}
void Set::Destroy(Node *node){
    if (node){
        Destroy(node->getNextNode());
        delete node;
    }
}
Set::~~Set(){
    if (head) Destroy(head);
}
void Set::DeepCopy(const Set& set){
    Node *p=set.head;
    while (p){
        add(p->getValue());
        p= p->getNextNode();
    }
}
Set::Set(const Set& set):
    head(0L),current(0L){
    if (!set.head) return;
    DeepCopy(set);
    current=head;
}
void Set::operator=(const Set& set){
    if (head){
        Destroy(head);
        head=current=0L;
    }
    DeepCopy(set);
    current=head;
}
bool Set::add(int value){
    if (!head){ // add to empty set
        head=current= new Node(value);
        head->setNextNode(0L);
        return true;
    }
    Node *p= head,*q=head;

```

```

        while (p){
            if (value==p->getValue()) return false;
            if (value<p->getValue()) break;
            q=p;
            p= p->getNextNode();
        }
        Node *n= new Node(value);
        if (p==q) // if we add to the front
            current=head=n;
        else
            q->setNextNode(n);
        n->setNextNode(p);
        return true;
    }
    bool Set::hasMoreElements()const{
        return (current!=0L) ;
    }
    int Set::next() {
        int ret= current->getValue();
        if (current) current= current->getNextNode();
        return ret;
    }
}

```

You may use the back of the page, but not elsewhere

Q.2) (40) What is the output of the following C++ code?

```

class B {
public:
    B(){ cout << "\nB's constructor"; }
    ~B(){ cout << "\nB's destructor"; }
};
class C {
private:
    static int n;
public:
    C(){ cout << "\nC's constructor # " << ++n ; }
    ~C(){ cout << "\nC's destructor # " << n-- ; }
};
int C::n=0;
class A {
protected:
    B *b;
    C c,d;
public:
    A(){ b= new B();
        cout << "\nA's constructor"; }
    ~A(){ delete b;
        cout << "\nA's destructor"; }
};
class D : public A {
public:
    D(){ cout << "\nD's constructor"; }
    ~D(){ cout << "\nD's destructor"; }
};
int main(){
    D a[2];
    return 0;
}

```

Line Number	Output
1	C's constructor # 1
2	C's constructor # 2
3	B's constructor
4	A's constructor
5	D's constructor
6	C's constructor # 3
7	C's constructor # 4
8	B's constructor
9	A's constructor
10	D's constructor
11	D's destructor
12	B's destructor
13	A's destructor
14	C's destructor # 4
15	C's destructor # 3
16	D's destructor
17	B's destructor
18	A's destructor
19	C's destructor # 2
20	C's destructor # 1