CMP-245 Object Oriented Programming Lab
BS Fall 2018
Lab 02
Configuration-C: PF >= 65

Issue Date: 27-Sep-2019
Marks: 35

Objective:

- This task will check your syncing with practice files and labs.
- It will heavily jolt your concepts of array of objects, dynamic memory allocation and most importantly the logic ☺.
- It will also help you understand that how to distribute code in different modules.

Challenge: *Relation* (35)

Prerequisite:

You must have code of 'Set' structure, which you implemented in Practice File-2 and 4. As a reminder: the following operations were implemented for Set structure.

```
//Set.h
struct Set
{
    int * data;
    int noOfElements;
    int capacity;
};
void createSet ( Set &, int n );
bool addElement ( Set &, int element );
bool removeElement ( Set &, int element );
bool searchElement ( Set, int element );
int searchElementPosition ( Set, int element );
bool isEmpty( Set );
bool isFull( Set );
void displaySet ( Set );
Set intersection ( Set, Set );
Set calcUnion ( Set, Set );
Set difference ( Set, Set );
int isSubset ( Set, Set );
void reSize( Set &, int newSize );
void displayPowerSet ( Set );
Set createClone ( Set & source );
void deallocateSet ( Set & );
```

~~deallocate set~~ 0.5

```
//Set.cpp
//definitions of set.h function
comes here
```

Before I told you what to do, review following:

For your work, you need to recall/review the following:

Set: Set is a collection of distinct elements.

$A = \{1,2,3,4,5\}$
$B = \{3,2,5,6\}$

Relation: Set of ordered pairs

Let $R1$ be a binary relation from A to B is a subset of $A * B$

$\mathbf{R1} = \{(1,3),(2,2),(2,5),(1,6),(5,3)\}$

Let relation $R2$ is a subset of $A * A$.

$\mathbf{R2} = \{(1,3),(1,2),(3,3),(1,1),(2,1),(5,3)\}$

There are some basic properties of Relation on a set, which are as follows:

Let R be a binary relation on set A.

- **Property-1: Reflexive**
  $R\ is\ reflexive, iff\ for\ all\ x \in A, (x,x) \in R.$
- **Property-2: Symmetric**
  $R\ is\ symmetric, iff\ for\ all\ x,y \in A, if\ (x,y) \in R, then\ (y,x) \in R$
- **Property-3: Transitive**
  $R\ is\ transitive\ iff\ for\ all\ x,y,z \in A, if\ (x,y) \in R\ and\ (y,z) \in R, then\ (x,z) \in R$

- **Property-4: anti-Symmetric**
  $R \text{ is anti} - \text{symmetric if for all } x, y \in A, \text{if } (x, y) \in R \text{ then } (y, x) \notin R \text{ except } x = y$

- **Property-5: Irreflexive**
  $R \text{ is irreflexive iff for all } x \in A, (x, x) \notin R$

Following table shows examples of different relations defined on set A with certain properties.

Relations	Reflexive	Symmetric	Transitive	Anti-symmetric	Irreflexive
{(1, 3), (3, 2), (3, 1), (1, 1), (2, 3)}	X	✓	X	X	X
{(1, 3), (1, 2), (3, 3), (1, 1), (2, 1), (3, 1), (2, 2), (3, 2), (2, 3)}	X	✓	✓	X	X
{(1, 2), (2, 3), (1, 3)}	X	X	✓	✓	✓
{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (1,5)}	✓	X	✓	✓	X
{(1, 1), (2, 2), (3, 3), (4, 4), (5, 5)}	✓	✓	✓	✓	X

Now what to do? You need to create a Relation structure and related operations which implemented above discussed properties.

So, considering the above example: a relation can be considered as a set of ordered pairs.

'OrderedPair' struct is used to hold an ordered pair.

```
struct OrderedPair
{
    int a;
    int b;
};
```

'Relation' struct will be used to hold a set of Ordered Pairs elements of any size given by the user.

```
struct Relation
{
    Set setA;
    OrderedPair *orderedPairList;
    int capcity;
    int noOfOrderedPair;
};
```

- 'orderedPairList' attribute will point to an array of OrderedPair objects of any size given by the user.
- 'capacity' attribute holds the size of array pointed by 'orderedPairList'.
- 'setA' attribute is associated with 'Relation' object. And in this task the relations are defined on A*A.

Your main task is to implement the properties of relations. Some of the function definitions are given below and some are left empty. You need to implement the incomplete functions and functions with no definitions. In order to achieve the task, you may add extra functions if you have a solid rational in your mind.
Remember: Place the declaration in header file and definitions in code files.

The function names are quite obvious and reveal their purpose but if you have any queries about their objective then you may ask it from T.As.

//OrderedPair.h
```
struct OrderedPair
{
    int a;
    int b;
};
bool isEqual(OrderedPair, OrderedPair);
```

//OrderedPair.cpp
```
bool isEqual(OrderedPair a,
OrderedPair b)
{
    return a.a==b.a && a.b==b.b;
}
```

//Relation.h
```
struct Relation
{
    Set setA;
    OrderedPair *orderedPairList;
    int capcity;
    int noOfOrderedPair;
};
void createRelation(Relation & r, Set a, int size);
void deAllocateRelation(Relation &);//------------------------------------------(2)
Relation createClone(Relation & r); //------------------------------------------(3)
2 bool isUniqueOrderedPair (Relation r, OrderedPair op ); //---------------------(2)
bool isFull(Relation r);
bool isValidOrderedPair(Relation r,OrderedPair op);
bool insertOrderedPair(Relation & r, OrderedPair op);
3 bool removeOrderedPair(Relation &, OrderedPair); //---------------------------(3)
void displayRelation(Relation r);
bool isReflexive (Relation); //-------------------------------------------------(5)
bool isSymmetric (Relation); //-------------------------------------------------(5)
bool isAntiSymmetric (Relation); //---------------------------------------------(5)
bool isTransitive (Relation); //------------------------------------------------(5)
bool isIrreflexive (Relation); //-----------------------------------------------(5)
```

3 creat clone
2 deallocate
deallocate set 0.5

Relation.cpp
```
void createRelation(Relation & r, Set a, int size)
{
    r.setA = a;
    r.capcity = size;
    r.orderedPairList = new OrderedPair[r.capcity];
    r.noOfOrderedPair = 0;
}
bool isUniqueOrderedPair (Relation r, OrderedPair op )
{
    return true;
}

bool isFull(Relation r)
{
    if (r.noOfOrderedPair==r.capcity)
    {
       deAllocateRelation(r);
       return true;
    }
    deAllocateRelation(r);
    return false;
}

bool isValidOrderedPair(Relation r,OrderedPair op)
{
    if (searchElementPosition(createClone(r.setA), op.a)==-1 ||
        searchElementPosition(createClone(r.setA), op.b)==-1)
    {
        deAllocateRelation(r);
        return false;
    }
    deAllocateRelation(r);
    return true;
}
```

$(x,y) \to (y,x)$

$(x,y) \to ($

```
bool insertOrderedPair(Relation & r, OrderedPair op)
{
    if (isFull(createClone(r)))
        return false;
    if (!isValidOrderedPair(createClone(r), op))
        return false;
    if (!isUniqueOrderedPair(createClone(r), op))
        return false;
    r.orderedPairList[r.noOfOrderedPair] = op;
    r.noOfOrderedPair = r.noOfOrderedPair + 1;
    return true;
}
void displayRelation(Relation r)
{
    cout<<"{";
    for ( int i=0; i<r.noOfOrderedPair; i++)
    {
        cout<<"("<<r.orderedPairList[i].a<<","<<
            r.orderedPairList[i].b<<"), ";
    }
    cout<<"\b\b}";
    deAllocateRelation(r);
}
bool isReflexive (Relation);
bool isSymmetric (Relation);
bool isAntiSymmetric (Relation);
bool isTransitive (Relation);
bool isIrreflexive (Relation);
```

Driver.cpp

```
int main()
{
    Set a;
    createSet(a, 8);
    addElement(a, 10);
    addElement(a, 1);
    addElement(a, 30);
    addElement(a, 5);
    addElement(a, 3);

    displaySet(createClone(a));
    cout<<"\n\n";
    Relation r;
    createRelation(r,createClone(a),10);
    OrderedPair op = {8,8};
    insertOrderedPair(r, op);
    op.a = 5;
    op.b = 1;
    insertOrderedPair(r, op);
    op.a = 1;
    op.b = 1;
    insertOrderedPair(r, op);
    op.a = 5;
    op.b = 1;
    insertOrderedPair(r, op);
    op.a = 45;
    op.b = 3;
    insertOrderedPair(r, op);
    displayRelation(createClone(r));

    return 0;
}
```

A diamond is just a piece of charcoal that handled stress exceptionally well.