CSI 2372 – Lab Task 2

Abdorrahim Bahrami

Constructors, Destructors, and Dynamic Memory in C++

Your task in this lab is to get yourself familiarized with constructors, destructors, and dynamic memory in C++. Make sure you have C++ installed, and you are familiar with the header files, and coding files. If you need help, ask your TA to help you with this.

Then, you should do the following programming task. Each programming task in the lab is a design based on the subjects you learned during lectures. There is a test code that you can use to test your design. If you have questions, ask your TAs.

Design a class for the concept of relations on a set of integers. Relation is a set of pairs on another set. For example, we say relation R is defined on a set A.

$A = \{1, 2, 3\}$

$R = \{(1, 1), (1, 2), (2, 1), (2, 2), (3, 3)\}$

It is clear that to represent a relation you need to store both A and R. You can use the set class we had in the first lab for A, but for R, you need to allocate dynamic memory. You can assume that A does not have more than 1000 elements but R can have any number of elements. You can define a struct as follows for elements of R in your header file.

```
struct pair

{

    int first;

    int second;

};
```

You need to allocate 100 elements for the relation at the beginning. This is just the capacity not the number of elements. You increase the number of elements in the relation, when it is needed by 50. Remember you should allocate memory again.

Your class must have the following methods. Use the name as they are in the table to be able to use the test file for testing your class design.

Class Relation	
Method	Description
Relation	The default constructor that initializes an empty relation with capacity of 100
Relation	The copy constructor
~Relation	The destructor
cardinality	Returns the number of pairs in the relation
add_element	Adds a pair into the relation (Remember you need to allocate memory here)
remove_element	Removes an element from the relation
is_member	Checks if an element is in the relation
equal	Checks if two relations are equal
reflexive	Checks if the relation is reflexive
irreflexive	Checks if the relation is irreflexive
symmetric	Checks if the relation is symmetric
asymmetric	Checks if the relation is asymmetric
transitive	Checks if the relation is transitive
is_function	Checks if the relation is a function
inverse	Returns the inverse of a relation (R^{-1})
combination	Returns the combination of two relations (Returns an empty relation if relations are not defined on the same set)

You can add any method you need. To make your task easy, you can add the following method to the Relation class to add elements to its set. In the following **root** is the set that the Relation is defined on.

```
bool Relation::add_to_set(int x)
{
	if (root.add_element(x))
		return true;
	return false;
}
```

Note:

Note that symmetric and asymmetric are not the exact opposite of each other.

Also, note that reflexive and irreflexive are not the exact opposite of each other.

Recall that

Reflexive

$\forall x \in A, (x, x) \in R$

Irreflexive

$\forall x \in A, (x,x) \notin R$

Symmetric

$\forall x, y \in A, (x,y) \in R \Longrightarrow (y,x) \in R$

Asymmetric

$\forall x, y \in A, (x,y) \in R \ and \ x \neq y \Longrightarrow (y,x) \notin R$

Transitive

$\forall x, y \in A, (x,y) \in R \ and \ (y,z) \in R \Longrightarrow (x,z) \in R$

The relation is a function if

$\forall x_1, y_1, x_2, y_2 \in A, (x_1, y_1) \in R \ and \ (x_2, y_2) \in R \ and \ x_1 = x_2 \Longrightarrow y_1 = y_2$

In other words, no two pairs have the same first element and different second elements.

Inverse

$R^{-1} = \{ (y,x) \mid (x,y) \in R\}$

Combination

Let’s say we relations R and S that are defined on the same set.

$RoS = \{(x,z) \mid (x,y) \in R \ \ and \ (y,z) \in S \}$