COSC 2P95 – Lab Exercise 6 – Section 01 – Object Orientation

Lab Exercises are uploaded via the Sakai page.
Since you need to submit both a sample execution and your source files, package it all up into a .zip to submit.

This week, we'll be using object orientation to create a *finite set* data type.

Background:

The concept of a finite set is actually pretty simple: taken from some domain, a set is simply a selection of some (or all, or none) elements to include. Each element may appear either one or zero times, and there's no concept of sequence.
For the sake of this exercise, you can assume that a set has a capacity, *N*, the maximum possible N is 255, and the choosable elements that could be members of the set are the numbers *0* .. *N-1*.

That is, if you create a set of potential size 3, then possible selections it could contain are:

- {} (the empty set)
- {0}
- {1}
- {2}
- {0,1}
- {0,2}
- {0,1,2}
- {1,2}

The defined operations are as such:

Expr	Name	Desc	C++
A∪B	Union	set of all elements contained within A and/or B	`A+B, A+b`
A∩B	Intersection	set of all elements common to both A and B	`A^B, A^b`
A\B	Difference	set of all elements contained within A, but not within B	`A-B, A-b`
a∈B	Element	test if a is a member of A	`B[a]`
A⊆B	Subset	test if all elements of A are included within B	`A<=B`
A⊇B	Superset	test if all elements of B are included within A	`A=>B`
A⊊B	Strict subset	test if all elements of A are within B, but A≠B	`A<B`
A⊋B	Strict superset	test if all elements of B are within A, but A≠B	`A>B`
A=B	Equality	test if both sets contain exactly the same elements	`A==B`
A≠B	Inequality	test if there exists an element within A or B not found in the other	`A!=B`
U\A	Complement	set of all possible elements not found within A	`~A`
|A|	Cardinality	number of elements within A (excluding λ)	`A()`
U	Universe	set of all possible elements	`+A`
Ø	Empty set	set containing no elements	`-A`
A=Ø	Empty test	Test if set is empty	`!A`
	Output	Stream insertion	`ostream<<A`
	Input	Stream extraction	`istream>>A`

You've been provided a header file to match this specification. You may make slight modifications (e.g. to add additional const versions), but should not need to.

Your primary task is to write the corresponding implementation, and then write a *very* simple test harness to demonstrate it.

The maximum possible number of states is 255, but that doesn't mean a given set could have that many. If you use the constructor that accepts an integer, then it defines a different N (which, as mentioned above, allows for possible elements of 0..N-1).

Because of this, you shouldn't need any dynamic allocation, and similarly shouldn't need to worry about assignment operator overloading, copy constructors, or destructors.

Tips:

- Sets are treated as *immutable*. That means all operations modifying sets actually return a new set containing the result of the operation
  - This is why we don't have operators like +=, and why [] doesn't return a reference
  - The *stream extraction* is the only exception here. Initialization is far easier if it can bend the rules
- Remember that operators taking two sets could use sets of different capacities
  - e.g. if A has a capacity of 5, and B has a capacity of 10, then A∪B has a capacity of 10
  - Similarly, +A yields {0,1,2,3,4}, while +B yields {0,1,2,3,4,5,6,7,8,9}
- Union, difference, and intersection have two versions each: one that accepts two sets, and one that accepts a set and an element
- We aren't including the bool() operator (test if not empty), but we can simulate it: `!!A`
- {1,2,3} is the same as {3,2,1}, so it's perfectly fine for us to store each set in ascending sequence

For your submission, in addition to including your source files, also including a sample execution or two.

Because it's somewhat of a nuisance, one possible version of stream extraction is:

```
std::istream& operator>>(std::istream &in, Set &set) {
	bool arr[255];
	int cap=set.capacity;
	char open;
	in>>open;
	if (in.fail() || open!='{')  {
		in.setstate(std::ios::failbit);
		return in;
	}
	for (int i=0;i<cap;i++)
		arr[i]=false;
	std::string buff;
	std::getline(in,buff,'}');
	std::stringstream ss(buff);
	std::string field;
	while (true) {
		std::getline(ss,field,',');
	if (ss.fail()) break;
		int el;
		std::stringstream se(field);
		se>>el;
		if (el>=0&&el<cap)
			arr[el]=true;
	}

	set=Set(arr,cap);
}
```

You're certainly not required to use this; you're welcome to write something better, if you prefer.

As stated above, write your own test harness, but to understand the syntax, consider the following:

```
Set s1(10),s2(6),s3(3),s4;
cout<<"First set ({x,y,z}): ";
cin>>s1;
cout<<"A: "<<s1<<endl;
cout<<"Second set: ";
cin>>s2;
cout<<"B: "<<s2<<endl;
cout<<"Third set: ";
cin>>s3;
cout<<"C: "<<s3<<endl;
cout<<"Fourth set: ";
cin>>s4;
cout<<"D: "<<s4<<endl;
cout<<"(B\\(U\\A+D))^C: "<<((s2-((~s1)+s4))^s3)<<endl;
```

with a possible sample execution of:

```
First set ({x,y,z}): {0,1,2,3,4,88}
A: {0,1,2,3,4}
Second set: {0,2,1,4,5,3,3,3}
B: {0,1,2,3,4,5}
Third set: {1,2}
C: {1,2}
Fourth set: {}
D: {}
(B\(U\A+D))^C: {1,2}
```

Requirements for Submission:

For your submission, in addition to including your source files, also include a sample execution or two. Pack it all up into a .zip to submit.