

COSC 2P95 – Lab Exercise 6 – 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 the domain is always 0..255 (i.e. there are only 256 possible members that may or may not be present).

For visualization, *if* the domain had been 0..2, then possible selections it could have contained would be:

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

The set can be represented (internally) based on whether or not the members are present, so the *same* sets as above might be:

- $[000]$, $[100]$, $[010]$, $[001]$, $[110]$, $[101]$, $[111]$, $[011]$
 - Naturally, it wouldn't matter if one had $[100]$ or $[001]$, so long as it was consistent (as it's internal)

Of course, since the domain goes to 255, the number of possible sets increases a wee bit.

The defined operations are as such:

Expr	Name	Description	C++
$A \cup B$	Union	set of all elements contained within A and/or B	$A+B$, $A+b$
$A \cap B$	Intersection	set of all elements common to both A and B	$A^{\wedge}B$, $A^{\wedge}b$
$A \setminus B$	Difference	set of all elements contained within A, but not within B	$A-B$, $A-b$
$a \in B$	Element	test if a is a member of A	$B[a]$
$A \subseteq B$	Subset	test if all elements of A are included within B	$A \leq B$
$A \supseteq B$	Superset	test if all elements of B are included within A	$A \geq B$
$A \subsetneq B$	Strict subset	test if all elements of A are within B, but $A \neq B$	$A < B$
$A \supsetneq B$	Strict superset	test if all elements of B are within A, but $A \neq B$	$A > B$
$A = B$	Equality	test if both sets contain exactly the same elements	$A == B$
$A \neq B$	Inequality	test if there exists an element within A or B not found in the other	$A != B$
$U \setminus A$	Complement	set of all possible elements not found within A	$\sim A$
$ A $	Cardinality	number of elements within A	$A()$
U	Universe	set of all possible elements	$+A$
\emptyset	Empty set	set containing no elements	$-A$
$A = \emptyset$	Empty test	Test if set is empty	$!A$
	Output	Stream insertion	$\text{ostream} \ll A$
	Input	Stream extraction	$\text{istream} \gg A$

You've been provided a header file to match this specification. You may make slight modifications (e.g. to tweak `consts` versions), but probably don't need to.

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

You'll notice that there's both a `public` constructor and a `private` one. Should be obvious why, right?

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
- Union, difference, and intersection have two versions each: one that accepts two sets, **and one that accepts a set and an element**
 - That's why there's an extra to receive an `int`, and that's what the `A+b/A-b` stuff above is
- 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 include 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[256];
    char open;
    in>>open;
    if (in.fail() || open!='{') {
        in.setstate(std::ios::failbit);
        return in;
    }
    for (int i=0;i<256;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<256)
            arr[el]=true;
    }
    set=Set(arr);
    return in;
}
```

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, just make sure you demonstrate the operators, insertion, extraction, etc.

It isn't a 'real' test harness (it's not like we've covered exceptions yet), so don't knock yourself out here.

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.