CPSC 2120/2121
Lab 10

Set theory functions implemented with a balanced binary search tree
Due date/time are on the CPSC 2121 Canvas website

Learning Objectives

- To improve our ability to implement algorithms in C++
- To implement an algorithm that uses a fundamental data structure (a balanced binary search tree)
- To become proficient in fundamental data structures used throughout computer science
- To improve our analytical skills while gaining familiarity with mathematical tools used in the analysis of algorithms

Problem / Exercise

For this lab, you will implement several set theory functions using the C++ STL set class. Before starting this lab, you should read about the set class, its iterators, and the pair class in the C++ STL at `www.cplusplus.com`. The main thing to note is that when you use a set from the C++ STL, you are using a balanced binary search tree that has the awesome property that inserting, finding, or erasing a single value in a set has a time complexity in $O(\log n)$, where n is the number of nodes in the tree.

Your main task for this lab is to implement the set theory functions in the provided file lab10.cpp (these functions should be implemented below the provided main function). The print functions are already implemented in that file, and they should not be changed. Also, do not change any of the function prototypes in lab10.cpp. You are required to use the set class, and you may use any functions in the set class; however, you are not allowed to use the algorithm library. The use of the algorithm library will result in a zero on this assignment.

Additional Requirements

- Your program must compile and run with an unmodified version of our provided Makefile to produce the same output as shown in the examples. If your program does not, then you'll need to fix your program. After your program works correctly with the test code in the provided main function, then you'll need to create additional tests to ensure your functions are working correctly.
- When you finish this assignment, on a piece of paper, analyze the worst case run time complexity of each function.

Examples

Your C++ source code should be in a file called `lab10.cpp`, and it should be compiled into an executable called `lab10.out` using our provided `Makefile`. The output of your program must look exactly like the examples below (except the word wrapping and spacing for the Cartesian product output may look different depending on what terminal you are using) when run on the command line on our Unix machines.

```
./lab10.out
A = 1 2 3 4 5 6
contains<int>(A, 3) = 1
contains<int>(A, 12) = 0
```

```
B = 2 4 6 8 10 12 14 16
A union B = 1 2 3 4 5 6 8 10 12 14 16
A intersect B = 2 4 6
A - B = 1 3 5
B - A = 8 10 12 14 16
A x B = (1, 2) (1, 4) (1, 6) (1, 8) (1, 10) (1, 12) (1, 14) (1, 16) (2, 2)
(2, 4) (2, 6) (2, 8) (2, 10) (2, 12) (2, 14) (2, 16) (3, 2) (3, 4) (3, 6)
(3, 8) (3, 10) (3, 12) (3, 14) (3, 16) (4, 2) (4, 4) (4, 6) (4, 8) (4, 10)
(4, 12) (4, 14) (4, 16) (5, 2) (5, 4) (5, 6) (5, 8) (5, 10) (5, 12) (5, 14)
(5, 16) (6, 2) (6, 4) (6, 6) (6, 8) (6, 10) (6, 12) (6, 14) (6, 16)
|A x B| = 48
C = 1 2 3
A subset A = 1
A proper subset A = 0
A subset B = 0
A subset C = 0
C subset A = 1
C proper subset A = 1

D = Bolton Frey Lannister Stark Targaryen Tully
E =
F = Banner Odinson Rogers Romanova Stark
House in Game of Thrones that is also an Avenger's last name = Stark
D union E = Bolton Frey Lannister Stark Targaryen Tully
D intersect E =
E subset D = 1
E proper subset D = 1
E subset E = 1
E proper subset E = 0
E x F =
F x E =
D x F = (Bolton, Banner) (Bolton, Odinson) (Bolton, Rogers) (Bolton, Romanova)
(Bolton, Stark) (Frey, Banner) (Frey, Odinson) (Frey, Rogers) (Frey, Romanova)
(Frey, Stark) (Lannister, Banner) (Lannister, Odinson) (Lannister, Rogers)
(Lannister, Romanova) (Lannister, Stark) (Stark, Banner) (Stark, Odinson)
(Stark, Rogers) (Stark, Romanova) (Stark, Stark) (Targaryen, Banner) (Targaryen, Odinson)
(Targaryen, Rogers) (Targaryen, Romanova) (Targaryen, Stark) (Tully, Banner)
(Tully, Odinson) (Tully, Rogers) (Tully, Romanova) (Tully, Stark)
```

Source Code Requirements

- Put a comment at the top of your source code file(s) with your name (first and last), the date of your submission, your lab section, and the assignment's name.
- All functions should be commented. Use inline documentation, as needed, to explain ambiguous, tricky parts, or important portions of your code.
- All source code must follow good programming style standards such as properly indenting source code and all variables, functions, and classes should be well-named.
- Your program must use dynamic memory allocation and deallocation properly, which means your program cannot contain a memory leak. You may use valgrind to check for memory leaks. Refer to the Unix manual and valgrind's online documentation for more information on valgrind.

Submission

Before the date/time stated on the CPSC 2121 Canvas webpage, you need to submit your code to our CPSC 2121 Canvas webpage under the correct lab assignment. Make sure to submit all of the following.

1. All source files required for this lab (lab10.cpp)

After you submit, always double check that the file(s) you submitted were the correct version. To double check, download the submitted file(s), put them on one of our Unix machines, and make sure they compile and run correctly.

Grading: 10 points

If your program does not compile on our Unix machines, your assignment was not submitted on time, your program did not use a set, or your program used the algorithm library, then you'll receive a grade of 0 on this assignment. Otherwise, your program will be graded using the criteria below.

Your program has correct output on various test cases	10 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly and efficiently on our Unix machines with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. We will only test your program with valid sets of inputs.