Overview

This homework contains a single programming problem worth 75 points toward your cumulative homework grade.

Practice Problems

1. Copy the contents of a vector into a list.
2. Write a function that removes every other value from a list.

Project Warm-Up

The project described below will require using a vector of structs, where each struct contains a list. You could also use a list of structs. This is something that we haven’t discussed in much detail yet, but most of the complexity is in the syntax. Here’s an example that should help quite a bit.

Suppose we declare the following struct:

```cpp
struct foo {
    list<int> x;
    double y;
};
```

Next, suppose we have a vector of foos:

```cpp
vector<foo> v;
```

We can create a new foo, put values in it and push it on the back of the vector:

```cpp
foo ex1;
ex1.y = 1.5;
ex1.x.push_back( 13 );
ex1.x.push_back( 15 );
ex1.x.push_front( 22 ); // ex1.x now has 22, 13, 15 in that order
v.push_back( ex1 );
```

Suppose that several foos have been added to the vector, v. Here’s code to print the contents of the entire vector, using subscripting on the vector:
for ( unsigned int i = 0; i < v.size(); ++i )
{
    cout << "Next foo: \n" << "Contents of x: \n";
    for ( list<int>::iterator xi = v[i].x.begin(); xi != v[i].x.end(); ++xi )
    {
        cout << "  " << *xi << "\n";
    }
    cout << "y = " << v[i].y << "\n" << endl;
}

Here's code that does exactly the same thing, using iterators only.

for ( vector<foo>::iterator vi = v.begin(); vi != v.end(); ++ vi )
{
    cout << "Next foo: \n" << "Contents of x: \n";
    for ( list<int>::iterator xi = vi->x.begin(); xi != vi->x.end(); ++xi )
    {
        cout << "  " << *xi << "\n";
    }
    cout << "y = " << vi->y << "\n" << endl;
}

Project

Now here's the actual assignment.

Write a program to build and maintain a class list for a series of courses,
not just a single course as in the example discussed in lecture. To simplify
things, however, your program does not need to maintain a waiting list for
each course. A further simplification is that your program does not need to
"prompt" for input. Instead, the input will come from a series of input lines.
Each line will indicate a change to the courses: adding a course, adding a
student to a course, removing a student from a course, printing a class list,
finding all the classes that a student is enrolled in.

Each line will start with a digit from the range 1...6. The rest of the
input line will depend on the digit. Here's a summary of these input lines
and what your program must do:

1 course_id n : Add a course with the given course id and maximum
   number of students. Do not allow this to occur if the course already
   exists.
2 course_id student_id : Add the student to the course, but not if the student is already in the course or the course is full.

3 course_id student_id : Remove the student from the course. Output an error message if the course does not exist or if the student is not in the course.

4 course_id : Output the class list for the given course.

5 student_id : Output the course ids for all courses the student is in. (Do this by searching through the course information rather than trying to maintain separate information for each student.)

6 : This signals the end of execution. Before terminating, your program must output the courses in order of increasing id. For each course output the id, the actual enrollment and the maximum allowed enrollment.

The course ids and student ids are just strings. All input will be correctly structured. Therefore you can ignore the fact that the input comes one line at a time.

For each operation on the course information, output a line of text indicating what your program has done. Indicate whether a student was added to a course, removed from a course, etc. Be sure this includes the student id and course id. If there is an error, such as if the course is full or the student is already in the course, indicate this in the output.

Example input and output will be posted on the course web page to guide you. Your program may be tested on other data as well.

Your program should be well-structured, using functions as appropriate with correct parameter passing. You must use a struct to represent each course and a std::list for the student id’s in a given course. To keep things simple, you are welcome to place your entire program in a single .cpp file. For other programming guidelines see the handout from the start of the semester.