This is part 2 of the Final exam. This part is worth 40 points (part 1 is implemented as a blackboard exam and is worth 60 points). If you wish to type your answers or provide them in a separate document, please clearly indicate what part (labeling each part as “A.”, “B.”, “C.”, and “D.” at the right margin. Please provide answers in order (A, B, C, and D).

Please read the instructions carefully. The last two pages are included for your reference.

A. (10 points) In the space below, write the implementation (as a non-inline member function) of an overloaded version of operator== that takes a string value representing a street address as its RHS parameter, and compares the parameter to the street address of the UMLPerson calling object.

For example:

```
UMLPerson jd("John", "Doe", "10 Main Street", 12345678);
if ( jd == "10 Main Street" ) { } // should be true
```

Note: You should not add any additional data members to the `UMLPerson` class.

B. (10 points) In the space below, write the implementation of a ”UnaryPredicate” function (i.e., a function that takes a single parameter and returns a bool value) based on the following function declaration:

```
bool IDNumberIsInRange(const UMLPerson&);
```

Your function should return `true` if the `IDNumber` of its actual parameter is within the range from 10000000 to 19999999 inclusive.

C. (10 points) In the space below, write a call to the `any_of` STL generic algorithm that returns `true` if any of the `UMLPerson` objects in the database contained in the variable `vector<UMLPerson> upDB` has an `IDNumber` in the range 10000000 to 19999999 inclusive. (Hints: Use your function `IDNumberIsInRange` from part (B), and see the example of `any_of` on the last page.)

D. (10 points) In the space below, write the implementation of a Compare function based on the following function declaration:

```
bool compare_Addresses(const UMLPerson& lhs, const UMLPerson& rhs);
```

Your function should return `true` if the street address of lhs is less than the street address of rhs according to lexicographic ordering. Also write a call to the `stable_sort` STL generic algorithm that sorts the database `upDB` according to street addresses by using your `compare_Addresses` function (just a single line to call `stable_sort`).

The `UMLPerson` class (also on blackboard as `UMLPerson.cpp`):

```
class UMLPerson { // base class
public:
    // constructors and destructor
    UMLPerson() : _firstName("NoName"), _lastName("NoName"),
                  _address("NoAddress"), _IDNumber_(0) {}

    UMLPerson(string firstName, string lastName, string address,
              unsigned _IDNumber) :
        _firstName(firstName), _lastName(lastName), _address(address),
        _IDNumber_(_IDNumber) {}

    UMLPerson(const UMLPerson&); // copy constructor

    ~UMLPerson();


    // Implements lexicographic comparison of UMLPerson objects based on
    // last name
    bool operator< (const UMLPerson& rhs) const
    { return this->_lastName < rhs._lastName; }

    // Implements equality based on IDNumber
    bool operator== (const UMLPerson& rhs) const
    { return this->_IDNumber == rhs._IDNumber; }

    bool operator== (const unsigned rhs) const
    { return this->_IDNumber == rhs; }


    // accessor functions
    string getFirstName() const { return _firstName; }
    string getLastName() const { return _lastName; }
    string getAddress() const { return _address; }
    unsigned get_IDNumber() const { return _IDNumber_; }

    void output(ostream& out) const { out << *this; }

    friend ostream& operator<< (ostream&, const UMLPerson&);
    friend ofstream& operator<< (ofstream&, const UMLPerson&);
    friend ifstream& operator>> (ifstream&, UMLPerson&);

private:
    string _firstName;
    string _lastName;
    string _address;
    unsigned _IDNumber_;
};
```

```
// any_of example
// The generic algorithm std::any_of returns true if the predicate function
// used for its third parameter returns true for any item in the range
// specified by its first and second parameters; otherwise, it returns false.

#include <iostream>      // std::cout
#include <algorithm>     // std::any_of
#include <array>         // std::array

bool isNegative(int i)
{
    return (i < 0);
}

int main ()
{
  std::array<int,7> foo = {0,1,-1,3,-3,5,-5};
  if ( std::any_of(foo.begin(), foo.end(), isNegative) )
    std::cout << "There are negative elements in the range.\n";
  return 0;
}

// stable_sort example
#include <iostream>     // std::cout
#include <algorithm>    // std::stable_sort
#include <vector>       // std::vector

bool compare_as_ints (double i,double j)
{
  return (int(i)<int(j));
}

int main () {
  double mydoubles[] = {3.14, 1.41, 2.72, 4.67, 1.73, 1.32, 1.62, 2.58};

  std::vector<double> myvector;

  myvector.assign(mydoubles,mydoubles+8);

  std::cout << "using default comparison:";
  std::stable_sort (myvector.begin(), myvector.end());
  for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  myvector.assign(mydoubles,mydoubles+8);

  std::cout << "using 'compare_as_ints' :";
  std::stable_sort (myvector.begin(), myvector.end(), compare_as_ints);
  for (std::vector<double>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    std::cout << ' ' << *it;
  std::cout << '\n';

  return 0;
}
```