CS 7B - Spring 2014 - Midterm 2. 5/8/14

Write responses on separate paper.

1. Write a recursive method that uses only addition, subtraction, and comparison to multiply two numbers. The basic engine for this recursion is `multiply(n-1,m)+m;` where the base case returns m when n-1 = 1. Be sure to handle the case where one or more factors is negative.

2. Write a recursive function to add the first n terms of the alternating harmonic series:

$$1 + \frac{1}{2} - \frac{1}{3} + \frac{1}{4} - \frac{1}{5} \cdots$$

3. Write a recursive method to print a sequence that begins with the number n_0 and each element n_i of the sequence is $n_{i-1}/2$ if n_{i-1} is even and $3 \cdot n_{i-1} + 1$ otherwise. That is, it's the conditional function

$$f(n) = \begin{cases} n/2, & \text{if } n \text{ is even} \\ 3n+1, & \text{if } n \text{ is odd} \end{cases}$$

   (a) Find the sequence of values if $n_0 = 7$

   (b) It is conjectured that this sequence ends with a 1. Take 1 as the base case and write a recursive function that will print the sequence.

4. Consider the conditional function

$$\texttt{Next}(n) = \begin{cases} 3n/2 & \texttt{if } n(\texttt{mod})2 = 0 \\ (3n+1)/4 & \texttt{if } n(\texttt{mod})4 = 1 \\ (3n-1)/4 & \texttt{if } n(\texttt{mod})4 = 3 \end{cases}$$

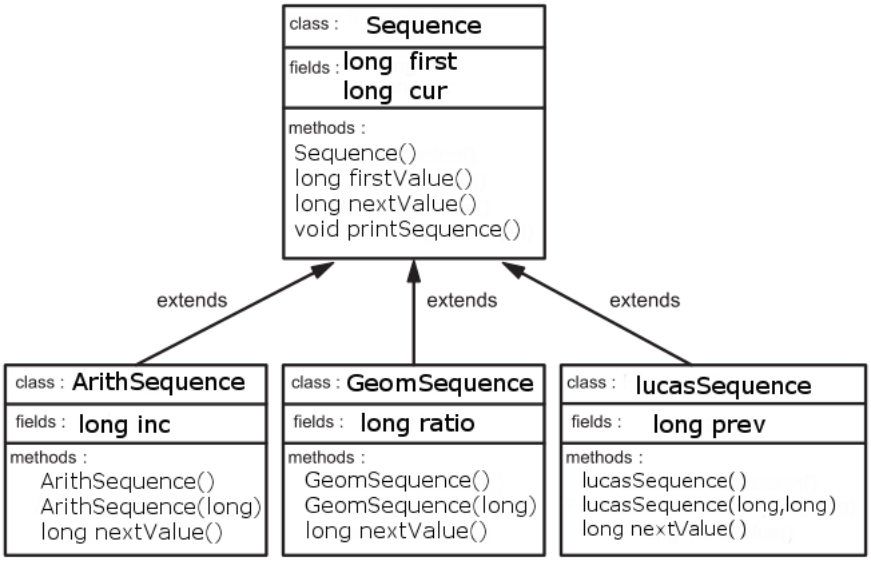
   (a) What happens when you iterate this function with an initial value of $n = 4$?

   (b) The base case for recursion with this function is that a previously iterated value is repeated. For example, you should have found that the fifth iterate of the part (a) sequence is a repeat of the first value. Write a recursive function to produce the iterates of this function until a value repeats. For the base case, use the `find` algorithm (specified below):

   `find` value in range [`first`, `last`] returns an iterator to the first element in the range [first,last) that compares equal to val. If no such element is found, the function returns last.

```
template <class InputIterator, class T>
   InputIterator find (InputIterator first, InputIterator last, const T& val);
//////////////////////
// standard usage
   if(std::find(v.begin(), v.end(), x) != v.end())
   {
       /* v contains x */
   } else {
       /* v does not contain x */
   }
```

5. Consider the inheritance diagram for the class `Sequence` and its derived classes shown below:




```
class Sequence {
public:
    Sequence(long f = 0) : first(f), cur(f) {}
    virtual ~Sequence() {};
    void printSequence(int n);
protected:
    virtual long firstValue();
    virtual long nextValue();
    long first;
    long cur;
};
void Sequence::printSequence(int n) {
    std::cout << firstValue();
    for(int i = 2; i <= n; i++)
        std::cout << ' ' << nextValue();
}
long Sequence::firstValue() {
    cur = first;
    return cur;
}
long Sequence::nextValue() {
    return ++cur;
}
class ArithSequence : public Sequence {
public:
    ArithSequence(long i = 1);
protected:
    virtual long nextValue();
    long inc;
};
ArithSequence::ArithSequence(long i) : Sequence(), inc(i) { }
long ArithSequence::nextValue() {
    cur += inc;
    return cur;
}
```

Given the partial definition above,

(a) Given that a geometric sequence has each term as a constant (ratio) multiple of the previous term, define the derived class `GeomSequence` and its methods.

(b) Given that a Lucas sequence is a Fibonacci type sequence with the initial values 2 and 1, so that the first values in the sequence are 2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843 ..., define the derived class `lucasSequence` and its methods.

(c) How would you go about making the `Sequence` class an `abstract base class`? Explain.

6. A triangle "is a" polygon, while a polygon "has" triangles, in the sense that every polygon can be decomposed into triangles. With this in mind, define a 2dPoint class, a Triangle class and a Polygon class and associated methods that will allow you to construct a polygon with $n \geq 3$ sides.

Consider the code below, which is a complete program for creating a school consisting of students and student records:

```
#include <iostream>
#include <fstream>
#include <cstring>

using namespace std;

class Person {
public:
    Person();
    Person(char*,char*,char*,int,long);
    void writeToFile(fstream&) const;
    void readFromFile(fstream&);
    void readKey();
    int size() const {
        return 9 + nameLen + cityLen + sizeof(year) + sizeof(salary);
    }
    bool operator==(const Person& pr) const {
        return strncmp(pr.ID,ID,9) == 0;
    }
protected:
    const int nameLen, cityLen;
    char ID[10], *name, *city;
    int year;
    long salary;
    ostream& writeLegibly(ostream&);
    friend ostream& operator<<(ostream& out, Person& pr) {
        return pr.writeLegibly(out);
    }
    istream& readFromConsole(istream&);
    friend istream& operator>>(istream& in, Person& pr) {
        return pr.readFromConsole(in);
    }
};

Person::Person() : nameLen(10), cityLen(10) {
    name = new char[nameLen+1];
    city = new char[cityLen+1];
}
Person::Person(char *ID, char *n, char *c, int y, long s) :
        nameLen(10), cityLen(10) {
    name = new char[nameLen+1];
    city = new char[cityLen+1];
    strcpy(ID,ID);
    strcpy(name,n);
    strcpy(city,c);
    year = y;
    salary = s;
}
void Person::writeToFile(fstream& out) const {
    out.write(ID,9);
    out.write(name,nameLen);
    out.write(city,cityLen);
```

```
53      out.write(reinterpret_cast<const char*>(&year),sizeof(int));
        out.write(reinterpret_cast<const char*>(&salary),sizeof(long));
55  }
    void Person::readFromFile(fstream& in) {
57      in.read(ID,9);
        in.read(name,nameLen);
59      in.read(city,cityLen);
        in.read(reinterpret_cast<char*>(&year),sizeof(int));
61      in.read(reinterpret_cast<char*>(&salary),sizeof(long));
    }
63  void Person::readKey() {
        char s[80];
65      cout << "Enter ID: ";
        cin.getline(s,80);
67      strncpy(ID,s,9);
    }
69  ostream& Person::writeLegibly(ostream& out) {
        ID[9] = name[nameLen] = city[cityLen] = '\0';
71      out << "ID = " << ID << ", name = " << name
            << ", city = " << city << ", year = " << year
73          << ", salary = " << salary;
        return out;
75  }
    istream& Person::readFromConsole(istream& in) {
77      ID[9] = name[nameLen] = city[cityLen] = '\0';
        char s[80];
79      cout << "ID: ";
        in.getline(s,80);
81      strncpy(ID,s,9);
        cout << "Name: ";
83      in.getline(s,80);
        strncpy(name,s,nameLen);
85      cout << "City: ";
        in.getline(s,80);
87      strncpy(city,s,cityLen);
        cout << "Birthyear: ";
89      in >> year;
        cout << "Salary: ";
91      in >> salary;
        in.getline(s,80); // get '\n'
93      return in;
    }
```

```
    #include "Person.h"
2
    class Student : public Person {
4   public:
        Student();
6       Student(char*,char*,char*,int,long,char*);
        void writeToFile(fstream&) const;
8       void readFromFile(fstream&);
        int size() const {
10          return Person::size() + majorLen;
        }
12  protected:
        char *major;
14      const int majorLen;
        ostream& writeLegibly(ostream&);
16      friend ostream& operator<<(ostream& out, Student& sr) {
            return sr.writeLegibly(out);
18      }
        istream& readFromConsole(istream&);
20      friend istream& operator>>(istream& in, Student& sr) {
```

```
        return sr.readFromConsole(in);
    }
};

Student::Student() : majorLen(10) {
    Person();
    major = new char[majorLen+1];
}
Student::Student(char *ssn, char *n, char *c, int y, long s, char *m) :
        majorLen(11) {
    Person(ID,n,c,y,s);
    major = new char[majorLen+1];
    strcpy(major,m);
}
void Student::writeToFile(fstream& out) const {
    Personal::writeToFile(out);
    out.write(major,majorLen);
}
void Student::readFromFile(fstream& in) {
    Personal::readFromFile(in);
    in.read(major,majorLen);
}
ostream& Student::writeLegibly(ostream& out) {
    Personal::writeLegibly(out);
    major[majorLen] = '\0';
    out << ", major = " << major;
    return out;
}
istream& Student::readFromConsole(istream& in) {
    Personal::readFromConsole(in);
    char s[80];
    cout << "Major: ";
    in.getline(s,80);
    strncpy(major,s,9);
    return in;
}
```

```
#include <fstream>

template<class T>
class Database {
public:
    Database();
    void run();
private:
    std::fstream database;
    char fName[20];
    std::ostream& print(std::ostream&);
    void add(T&);
    bool find(const T&);
    void modify(const T&);
    friend std::ostream& operator<<(std::ostream& out, Database& db) {
        return db.print(out);
    }
};

template<class T>
Database<T>::Database() {
}
template<class T>
void Database<T>::add(T& d) {
    database.open(fName,std::ios::in|std::ios::out|std::ios::binary);
    database.clear();
```

```
    database.seekp(0,std::ios::end);
    d.writeToFile(database);
    database.close();
}
template<class T>
void Database<T>::modify(const T& d) {
    T tmp;
    database.open(fName,std::ios::in|std::ios::out|std::ios::binary);
    database.clear();
    while (!database.eof()) {
        tmp.readFromFile(database);
        if (tmp == d) {  // overloaded ==
            std::cin >> tmp; // overloaded >>
            database.seekp(-d.size(),std::ios::cur);
            tmp.writeToFile(database);
            database.close();
            return;
        }
    }
    database.close();
    std::cout << "The record to be modified is not in the database\n";
}
template<class T>
bool Database<T>::find(const T& d) {
    T tmp;
    database.open(fName,std::ios::in|std::ios::binary);
    database.clear();
    while (!database.eof()) {
        tmp.readFromFile(database);
        if (tmp == d) { // overloaded ==
            database.close();
            return true;
        }
    }
    database.close();
    return false;
}
template<class T>
std::ostream& Database<T>::print(std::ostream& out) {
    T tmp;
    database.open(fName,std::ios::in|std::ios::binary);
    database.clear();
    while (true) {
        tmp.readFromFile(database);
        if (database.eof())
            break;
        out << tmp << std::endl; // overloaded <<
    }
    database.close();
    return out;
}
template<class T>
void Database<T>::run() {
    std::cout << "File name: ";
    std::cin >> fName;
    std::cin.ignore();  // skip '\n'
    database.open(fName,std::ios::in);
    if (database.fail())
        database.open(fName,std::ios::out);
    database.close();
    char option[5];
    T rec;
    std::cout << "1. Add 2. Find 3. Modify a record; 4. Exit\n";
    std::cout << "Enter an option: ";
    while (std::cin.getline(option,5)) {
```

```
92          if (*option == '1') {
                std::cin >> rec;    // overloaded >>
94              add(rec);
            }
96          else if (*option == '2') {
                rec.readKey();
98              std::cout << "The record is ";
                if (find(rec) == false)
100                 std::cout << "not ";
                std::cout << "in the database\n";
102         }
            else if (*option == '3') {
104             rec.readKey();
                modify(rec);
106         }
            else if (*option != '4')
108             std::cout << "Wrong option\n";
            else return;
110         std::cout << *this;    // overloaded <<
            std::cout << "Enter an option: ";
112     }
}
```

```
#include <iostream>
#include "Person.h"
#include "Database.h"
using namespace std;

int main() {
    Database<Person>().run();
//  Database<Student>().run();
    return 0;
}
```

Based on this code

(a) As it is, none of the methods of `Person` are virtual. Would it be appropriate to make some of `Person` methods virtual? Why or why not? Which ones?

(b) Describe how the `Student` two constructors work.

(c) Write a definition for the derived class `tutor` which has all the attributes of a student but also has a list of tutees (other students that the student tutors and an hourly rate (how much the tutor is paid per hour. Define the constructor and destructor for this derived class.