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.
   Solution: There are a number of special cases to consider. To minimize the number of function calls, and to standardize the procedure, we can assume that the smaller factor is first, for, if it's not, we can just call `multiply(m,n)` with the parameters reversed. Then, if one of the factors is negative, the product will be negated and we can deal with that by calling `-multiply(-n,m)`. If both factors are negative, then this will be called twice, with at least one reversal of factors in between, thus giving a double negative and making the product positive, as it should be. Finally, if one of the factors is 0, then the product is obviously 0, and we're done. Here's the code:

```
int multiply(int n, int m) {
    if (n == 0)
        return 0;
    else if (n < 0)
        return -multiply(-n,m); //prod is negative
    else f (n == 1)
        return m;
    else if (n <= m)
        return multiply(n-1,m)+m; //basic engine
    else return multiply(m,n); //if n>m reverse n&m
}
```

   Then the driver code

```
while(1) {
   cout << "\nEnter 2 nos to multiply,
              or 0*0 to quit:" << endl;
        cin >> j >> k;
        if(j == 0 && k == 0)
            break;
        else cout << multiply(j,k);
    }
```

```
Enter 2 nos to multiply, or 0 0 to quit:
12 13
156
```

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

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

   Solution:

```
double series(unsigned int n) {
  if (n == 1)
     return 1.0;
  else if (n % 2 == 0)
  // or: if ((n & 1) == 0)
     return series(n-1) + 1.0/n;
  else return series(n-1) - 1.0/n;
}
```

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 $n_0 = 7$
   Solution: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.

   (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.
   Solution: The code is relatively simple. A curious wrinkle here is unpredictable zigging and zagging through evens and odds as the elements of the sequence increase approximately 50% when odd (say, 13, 40, 20 is an increase from 13 to 20, a little more than 50% up) or decrease by 50% when even.

```
void f(int n) {
  cout << n << ' ';
  if (n == 1)
    return;
  else if (n % 2 == 0)
    f(n/2);
  else f(3*n+1);
}
```

   This is pretty slick! The driver code:

```
while(1) {
   cout<<"Enter sequence seed, 0 to quit:"
       << endl;
   cin >> j;
     if(j == 0)
        break;
   f(j);
}
```

   produces this output:

```
Enter sequence seed, or 0 to quit:
108
108 54 27 82 41 124 62 31 94 47 142 71 214 107 322 161 484
242 121 364 182 91 274 137 412 206 103 310 155 466 233 700
350 175 526 263 790 395 1186 593 1780 890 445 1336 668 334
167 502 251 754 377 1132 566 283 850 425 1276 638 319 958
479 1438 719 2158 1079 3238 1619 4858 2429 7288 3644 1822
911 2734 1367 4102 2051 6154 3077 9232 4616 2308 1154 577
1732 866 433 1300 650 325 976 488 244 122 61 184 92 46 23
70 35 106 53 160 80 40 20 10 5 16 8 4 2 1
```

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 $n = 4$?
   Solution: 4 6 9 7 5

   (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 */
   }
```

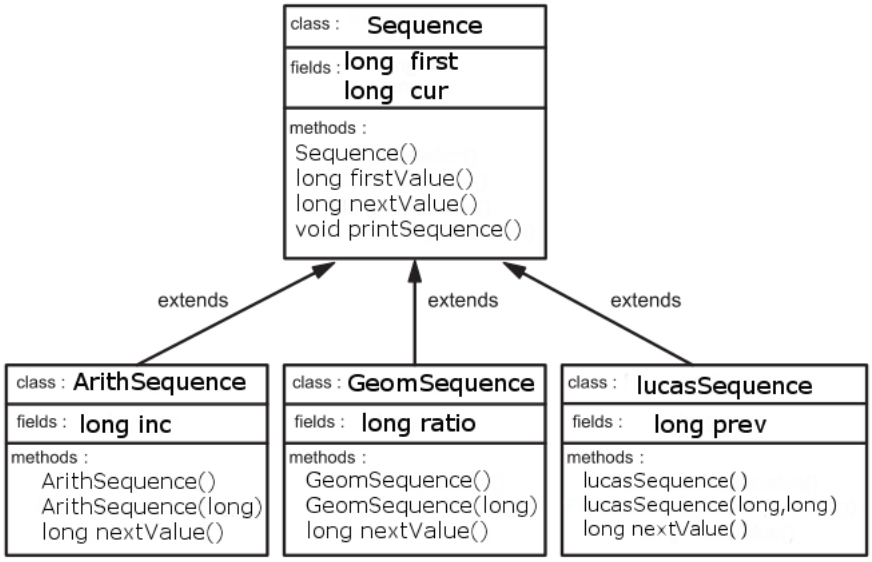
[language=C++, directivestyle=] **Solution:**

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

void Guy(vector<int>& v, int n) {
    if(std::find(v.begin(), v.end(), n) != v.end()) return;
    v.push_back(n);
    if(n%2==0) Guy(v, 3*n/2);
    else if(n%4==1) Guy(v, (3*n+1)/4);
    else Guy(v, (3*n-1)/4);
}
```

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;
}
```

Reference the partial definition above for these questions.

(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.

Solution:

```
//=================
// geometric
class GeomSequence : public Sequence {
public:
    GeomSequence(long r = 2);
protected:
    virtual long nextValue();
    long ratio;
};

GeomSequence::GeomSequence(long r)
: Sequence(1), ratio(r) { }

long GeomSequence::nextValue() {
    cur *= ratio;
    return cur;
}
```

To be sure, here's some driver code

```
for(long a=1;a<6;a++) {
    cout<<"s("<<a<<") = ";
    Sequence s(a);
    s.printSequence(a);
    cout << endl;
```

```
}
for(long a=1;a<6;a++) {
    cout<<"as("<<a<<") = ";
    ArithSequence as(a);
    as.printSequence(a);
    cout << endl;
}
for(long a=1;a<6;a++) {
    cout<<"gs("<<a<<") = ";
    GeomSequence gs(a);
    gs.printSequence(a);
    cout << endl;
}
```

That produces this output:

```
s(1) = 1
s(2) = 2 3
s(3) = 3 4 5
s(4) = 4 5 6 7
s(5) = 5 6 7 8 9
as(1) = 0
as(2) = 0 2
as(3) = 0 3 6
as(4) = 0 4 8 12
as(5) = 0 5 10 15 20
gs(1) = 1
gs(2) = 1 2
gs(3) = 1 3 9
gs(4) = 1 4 16 64
gs(5) = 1 5 25 125 625
```

(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.

Solution: Defining the derived class `FibonacciSequence`

```
//==============
// Fibonacci
class FibonacciSequence : public Sequence { // Fibonacci Sequence
public:
    FibonacciSequence(long f = 0, long s = 1); // constructor
protected:
    virtual long firstValue(); // reset
    virtual long nextValue(); // advance
protected:
    long second; // second value
    long prev; // previous value
};


FibonacciSequence::FibonacciSequence(long f, long s)
  : Sequence(f), second(s), prev(second - first) { }
long FibonacciSequence::firstValue() { // reset
    cur = first;
    prev = second - first; // create fictitious prev
    return cur;
}
long FibonacciSequence::nextValue() { // advance
    long temp = prev;
    prev = cur;
```

```
    cur += temp;
    return cur;
}

//driver code:
cout << "\nThe Lucas sequence is Fibonacci like in that the same recursive formula\n"
     << "governs its behavior.  The first two terms are L0 = 2 and L1 = 1:\n";
FibonacciSequence fs(2,1);
   fs.printSequence(20);
```

This produces the Lucas sequence:

```
The Lucas sequence is Fibonacci like in that the same recursive formula
governs its behavior.  The first two terms are L0 = 2 and L1 = 1:
2 1 3 4 7 11 18 29 47 76 123 199 322 521 843 1364 2207 3571 5778 9349
```

(c) How would you go about making the `Sequence` class an `abstract base class`? Explain.
Solution: At least one of the methods in the abstract base class, Sequence, will need to be a *pure virtual function*. The member function `nextValue()`, which computes the next value in the `Sequence` seems like a good candidate since the meaning of this function is clear for each of the derived classes: ArithSequence, GeomSequence, and FibonacciSequence. However, in the base class `Sequence` we invented a rather arbitrary default for the `nextValue()` function. (Go back and check it. What Sequence does it compute?) It may be better to leave this function undefined. Make it a *pure virtual* member function like so:

```
class Progression { // abstract base class
// ...
virtual long nextValue() = 0; // pure virtual function
// ...
};
```

Since the function `nextValue` is *pure virtual*, the compiler will not allow the creation of objects of type `Sequence`, However, its derived classes, `ArithSequence` for example, can be defined because they provide a definition for this member function.

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. **Solution:**

7. 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;
```

```
19      }
    protected:
21      const int nameLen, cityLen;
        char ID[10], *name, *city;
23      int year;
        long salary;
25      ostream& writeLegibly(ostream&);
        friend ostream& operator<<(ostream& out, Person& pr) {
27          return pr.writeLegibly(out);
        }
29      istream& readFromConsole(istream&);
        friend istream& operator>>(istream& in, Person& pr) {
31          return pr.readFromConsole(in);
        }
33  };

35  Person::Person() : nameLen(10), cityLen(10) {
        name = new char[nameLen+1];
37      city = new char[cityLen+1];
    }
39  Person::Person(char *ID, char *n, char *c, int y, long s) :
            nameLen(10), cityLen(10) {
41      name = new char[nameLen+1];
        city = new char[cityLen+1];
43      strcpy(ID,ID);
        strcpy(name,n);
45      strcpy(city,c);
        year = y;
47      salary = s;
    }
49  void Person::writeToFile(fstream& out) const {
        out.write(ID,9);
51      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);
    cout << "City: ";
    in.getline(s,80);
    strncpy(city,s,cityLen);
    cout << "Birthyear: ";
    in >> year;
    cout << "Salary: ";
    in >> salary;
    in.getline(s,80); // get '\n'
    return in;
}
```

```
#include "Person.h"

class Student : public Person {
public:
    Student();
    Student(char*,char*,char*,int,long,char*);
    void writeToFile(fstream&) const;
    void readFromFile(fstream&);
    int size() const {
        return Person::size() + majorLen;
    }
protected:
    char *major;
    const int majorLen;
    ostream& writeLegibly(ostream&);
    friend ostream& operator<<(ostream& out, Student& sr) {
        return sr.writeLegibly(out);
    }
    istream& readFromConsole(istream&);
    friend istream& operator>>(istream& in, Student& sr) {
        return sr.readFromConsole(in);
    }
};

Student::Student() : majorLen(10) {
    Person();
    major = new char[majorLen+1];
}
Student::Student(char* ID, 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];
```

```
52      cout << "Major: ";
        in.getline(s,80);
54      strncpy(major,s,9);
        return in;
56  }
```

```
    #include <fstream>
2
    template<class T>
4   class Database {
    public:
6       Database();
        void run();
8   private:
        std::fstream database;
10      char fName[20];
        std::ostream& print(std::ostream&);
12      void add(T&);
        bool find(const T&);
14      void modify(const T&);
        friend std::ostream& operator<<(std::ostream& out, Database& db) {
16          return db.print(out);
        }
18  };

20  template<class T>
    Database<T>::Database() {
22  }
    template<class T>
24  void Database<T>::add(T& d) {
        database.open(fName,std::ios::in|std::ios::out|std::ios::binary);
26      database.clear();
        database.seekp(0,std::ios::end);
28      d.writeToFile(database);
        database.close();
30  }
    template<class T>
32  void Database<T>::modify(const T& d) {
        T tmp;
34      database.open(fName,std::ios::in|std::ios::out|std::ios::binary);
        database.clear();
36      while (!database.eof()) {
            tmp.readFromFile(database);
38          if (tmp == d) {  // overloaded ==
                std::cin >> tmp; // overloaded >>
40              database.seekp(-d.size(),std::ios::cur);
                tmp.writeToFile(database);
42              database.close();
                return;
44          }
        }
46      database.close();
        std::cout << "The record to be modified is not in the database\n";
48  }
    template<class T>
50  bool Database<T>::find(const T& d) {
        T tmp;
52      database.open(fName,std::ios::in|std::ios::binary);
        database.clear();
54      while (!database.eof()) {
            tmp.readFromFile(database);
56          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)) {
        if (*option == '1') {
            std::cin >> rec;   // overloaded >>
            add(rec);
        }
        else if (*option == '2') {
            rec.readKey();
            std::cout << "The record is ";
            if (find(rec) == false)
                std::cout << "not ";
            std::cout << "in the database\n";
        }
        else if (*option == '3') {
            rec.readKey();
            modify(rec);
        }
        else if (*option != '4')
            std::cout << "Wrong option\n";
        else return;
        std::cout << *this;   // overloaded <<
        std::cout << "Enter an option: ";
    }
}
```

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

int main() {
```

```
7       Database<Person>().run();
//      Database<Student>().run();
9       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.