CS 7B Midterm - Fall 2019 - Midterm 11/20/19

Write answers to problem 1 on this paper. For other problems, write responses on separate paper and/or submit code by email, as appropriate.

1. Write the number of the definition on the right next to the term it defines.
   (a) **copy**_____
   (b) **overload**_______
   (c) **container**_______
   (d) **pointer**_______
   (e) **reference**_______
   (f) **class**______
   (g) **invariant**_______
   (h) **type**_______
   (i) **byte**_______
   (j) **constructor**_______
   (k) **scope**_______
   (l) **move**_______

   (1) a value used to identify a typed object in memory; (2) a variable holding such a value.
   (2) an operation that transfers a value from one object to another, leaving behind a value representing "empty."
   (3) An operation that makes two objects have values that compare equal..
   (4) Define two functions or operators with the same name but different argument (operand) types.
   (5) A user-defined type that may contain data members, function members, and member types.
   (6) An operation that initializes an object. Typically establishes an invariant and often acquires resources needed for an object to be used (which are then typically released by a destructor).
   (7) The region of program text (source code) in which a name can be referred to.
   (8) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.
   (9) Something that defines a set of possible values and a set of operations for an object.
   (10) Something that must be always true at a given point (or points) of a program; typically used to describe the state (set of values) of an object or the state of a loop before entry into the repeated statement.
   (11) The basic unit of addressing in most computers.
   (12) An object that holds elements (other objects).

2. Consider the following complete program:

```cpp
#include<iostream>
#include<vector>
int main() { // read elements into a vector without using push_back:
    std::vector<double>* p = new std::vector<double>(10);
    std::cout << "\nsizeof(p)=" << sizeof(p);
    std::cout << "\nsizeof(*p)=" << sizeof(*p);
    std::cout << "\np->size()=" << p->size();
    int n = 0; // number of elements
    for (double d; std::cin>>d; )  {
      if (n >= p->size())  {
        std::vector<double>* q=new std::vector<double>(p->size()*2);
            copy(p->begin(), p->end(), q->begin());
            delete p;
            p = q;
            std::cout << "\np->size()=" << p->size();
      } (*p)[n++] = d;
   }
}
```

(a) State and explain the output you get from lines 5, 6, and 7.

(b) The loop variable `d` in the `for`-loop on line 10 is not initialized. Does it need to be initialized? Why or why not? Consider the RAII principle in your answer.

(c) Why is the update field of the `for`-loop on line 10 blank?

(d) How can the condition of the `for`-loop be `false` (what keyboard entry would lead to that?)

(e) Describe the conditional of the `if` statement in the `for`-loop. What circumstance will first trigger that as `true`?

(f) Explain carefully exactly all that happens on line 16. What kind of object is dereferenced? What is indexed by what? What is incremented? In what order do these operations occur?

(g) Modify the program to read from a text file that contains the text "`1 2 3 4 5 6 7 8 9 10 11 ∧D`" State and explain the output you get.

3. Consider the following code:

```cpp
#include<iostream>
class vector {
    int sz;        // number of elements
    double* elem; // address of first element
    int space;     // number of elements plus "free space"/"slots"
public:
    vector();
    void reserve(int newalloc);
    int  capacity() const { return space; }
    int size() const { return sz; }
    void resize(int newsize);
};
vector::vector() :sz(0), elem(0), space(0) {}
void vector::reserve(int newalloc) {
    if (newalloc<=space) return;      // never decrease allocation
    double* p = new double[newalloc];         // allocate new space
    for (int i=0; i<sz; ++i) p[i] = elem[i]; // copy old elements
    delete[] elem;                           // deallocate old space
    elem = p;
    space = newalloc;
}
void vector::resize(int newsize) {
// make the vector have newsize elements
// initialize each new elements with the default value 0.0
    reserve(newsize);
    for (int i=sz; i<newsize; ++i) elem[i] = 0;    // initialize new elements
    sz = newsize;
}
int main() {
    vector v;
    v.reserve(10);
    std::cout << "\nv.capacity()␣=␣" << v.capacity() << '\n';
    std::cout <<"v.size()␣=␣" << v.size() << '\n';
    v.resize(4);
    std::cout <<"v.size()␣=␣" << v.size() << '\n';
    return v.capacity();
}
```

(a) What is the output of `main()`?

(b) Give a detailed description of what `resize()` does and when it is used.

(c) Write code for a function to overload the assignment operator for the above `vector` class.

(d) Write code for a `push_back()` function to add a `double` to the vector.

(e) How would you modify this code to make `vector` a template class which will allow you to have a vector of an abstract datatype, `T`?

4. 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.

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

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

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

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

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;
    }
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);
    out.write(reinterpret_cast<const char*>(&year),sizeof(int));
    out.write(reinterpret_cast<const char*>(&salary),sizeof(long));
}
void Person::readFromFile(fstream& in) {
    in.read(ID,9);
    in.read(name,nameLen);
    in.read(city,cityLen);
    in.read(reinterpret_cast<char*>(&year),sizeof(int));
    in.read(reinterpret_cast<char*>(&salary),sizeof(long));
}
void Person::readKey() {
    char s[80];
    cout << "Enter ID: ";
    cin.getline(s,80);
    strncpy(ID,s,9);
}
ostream& Person::writeLegibly(ostream& out) {
    ID[9] = name[nameLen] = city[cityLen] = '\0';
    out << "ID = " << ID << ", name = " << name
        << ", city = " << city << ", year = " << year
        << ", salary = " << salary;
    return out;
}
istream& Person::readFromConsole(istream& in) {
    ID[9] = name[nameLen] = city[cityLen] = '\0';
    char s[80];
    cout << "ID: ";
    in.getline(s,80);
    strncpy(ID,s,9);
    cout << "Name: ";
    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 *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)) {
        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() {
    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 two `Student` 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.