

Write responses to all questions on separate paper. Submit code by email, as appropriate.

1. Write the number of the definition on the right next to the term it defines.
  - (a) **copy**\_\_\_\_\_ (1) (1) a value used to identify a typed object in memory; (2) a variable holding such a value.
  - (b) **overload**\_\_\_\_\_ (2) an operation that transfers a value from one object to another, leaving behind a value representing "empty."
  - (c) **container**\_\_\_\_\_ (3) An operation that makes two objects have values that compare equal..
  - (d) **pointer**\_\_\_\_\_ (4) Define two functions or operators with the same name but different argument (operand) types.
  - (e) **reference**\_\_\_\_\_ (5) A user-defined type that may contain data members, function members, and member types.
  - (f) **class**\_\_\_\_\_ (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).
  - (g) **invariant**\_\_\_\_\_ (7) The region of program text (source code) in which a name can be referred to.
  - (h) **type**\_\_\_\_\_ (8) (1) a value describing the location of a typed value in memory; (2) a variable holding such a value.
  - (i) **byte**\_\_\_\_\_ (9) Something that defines a set of possible values and a set of operations for an object.
  - (j) **constructor**\_\_\_\_\_ (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.
  - (k) **scope**\_\_\_\_\_ (11) The basic unit of addressing in most computers.  
(12) An object that holds elements (other objects).

2. Consider the following complete program:

```

1  #include<iostream>
   #include<vector>
3  using namespace std;
   int main() { // read elements into a vector without using push_back:
5     vector<double>* p = new vector<double>(10);
       cout << "\nsizeof(p)=" << sizeof(p);
7     cout << "\nsizeof(*p)=" << sizeof(*p);
       cout << "\np->size()" << p->size();
9     int n = 0; // number of elements
       for (double d; cin>>d; ) {
11        if (n >= p->size()) {
           vector<double>* q = new vector<double>(p->size()*2);
13           copy(p->begin(), p->end(), q->begin());
           delete p;
15           p = q;
           cout << "\np->size()" << p->size();
17        }
           (*p)[n++] = d;
19    }
   }
```

- (a) State and explain the output you get from lines 8, 9 and 10.
- (b) Why is it not necessary to initialize `d` in the `for`-loop on line 10?
- (c) Why is ok that the update field of the `for`-loop on line 10 is 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 18. 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  $\backslash$ D"  
State and explain the output you get.

3. Consider the following code:

```

#include<iostream>
2 using namespace std;
class vector {
4     int sz;           // number of elements
    double* elem; // address of first element
6     int space;      // number of elements plus "free space"/"slots"
public:
8     vector();
    void reserve(int newalloc);
10    int capacity() const { return space; }
    int size() const { return sz; }
12    void resize(int newsize);
};
14 vector::vector() :sz(0), elem(0), space(0) {}
void vector::reserve(int newalloc) {
16     if (newalloc<=space) return;           // never decrease allocation
    double* p = new double[newalloc];       // allocate new space
18     for (int i=0; i<sz; ++i) p[i] = elem[i]; // copy old elements
    delete[ ] elem;                          // deallocate old space
20     elem = p;
    space = newalloc;
22 }
void vector::resize(int newsize) {
24 // make the vector have newsize elements
// initialize each new elements with the default value 0.0
26     reserve(newsize);
    for (int i=sz; i<newsize; ++i) elem[i] = 0; // initialize new elements
28     sz = newsize;
}
30 int main() {
    vector v;
32     v.reserve(10);
    cout << "\nv.capacity()_=_ " << v.capacity();
34     cout << "\nv.size()_=_ " << v.size();
    v.resize(4);
36     cout << "\nv.size()_=_ " << v.size();
    return v.capacity();
38 }

```

- (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:

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

6. 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.
7. Consider the conditional function

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

- (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):

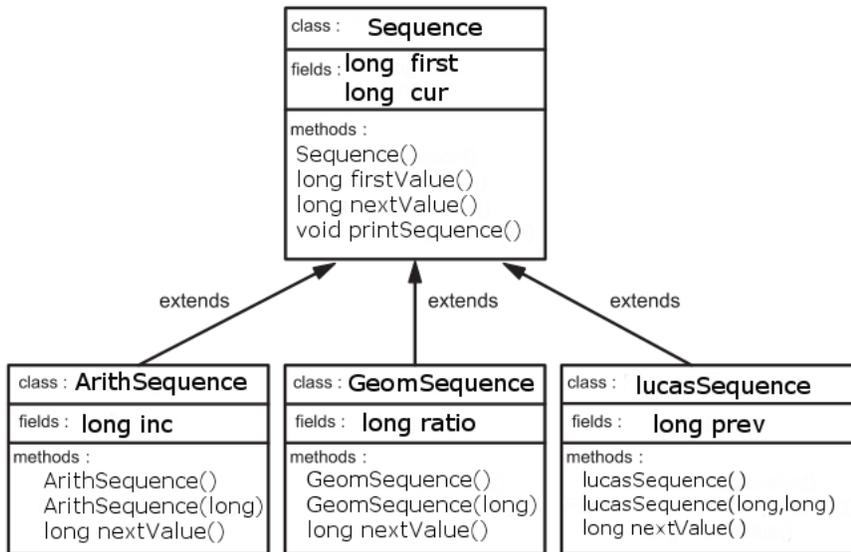
`find` value in range `[first,last]` returns an iterator to the first element in the range `[first,last)` that compares equal to `val`. If no such element is found, the function returns `last`.

```

1  template <class InputIterator, class T>
2  InputIterator find (InputIterator first, InputIterator last, const T& val);
3  //////////////////////////////////////
4  // standard usage
5  if (std::find(v.begin(), v.end(), x) != v.end())
6  {
7      /* v contains x */
8  } else {
9      /* v does not contain x */
10 }

```

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



```

1 class Sequence {
2 public:
3     Sequence(long f = 0) : first(f), cur(f) {}
4     virtual ~Sequence() {};
5     void printSequence(int n);
6 protected:
7     virtual long firstValue();
8     virtual long nextValue();
9     long first;
10    long cur;
11 };
12 void Sequence::printSequence(int n) {
13     std::cout << firstValue();
14     for(int i = 2; i <= n; i++)
15         std::cout << '□' << nextValue();
16 }
17 long Sequence::firstValue() {
18     cur = first;
19     return cur;
20 }
21 long Sequence::nextValue() {
22     return ++cur;
23 }
24 class ArithSequence : public Sequence {
25 public:
26     ArithSequence(long i = 1);
27 protected:
28     virtual long nextValue();
29     long inc;
30 };
31 ArithSequence::ArithSequence(long i) : Sequence(), inc(i) { }
32 long ArithSequence::nextValue() {
33     cur += inc;
34     return cur;
35 }
  
```

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.
9. 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:

```
1 #include <iostream>
2 #include <fstream>
3 #include <cstring>
4
5 using namespace std;
6
7 class Person {
8 public:
9     Person();
10    Person(char*, char*, char*, int, long);
11    void writeToFile(fstream&) const;
12    void readFromFile(fstream&);
13    void readKey();
14    int size() const {
15        return 9 + nameLen + cityLen + sizeof(year) + sizeof(salary);
16    }
17    bool operator==(const Person& pr) const {
18        return strcmp(pr.ID, ID, 9) == 0;
19    }
20 protected:
21    const int nameLen, cityLen;
22    char ID[10], *name, *city;
23    int year;
24    long salary;
25    ostream& writeLegibly(ostream&);
26    friend ostream& operator<<(ostream& out, Person& pr) {
27        return pr.writeLegibly(out);
28    }
29    istream& readFromConsole(istream&);
30    friend istream& operator>>(istream& in, Person& pr) {
31        return pr.readFromConsole(in);
32    }
33 };
34
35 Person::Person() : nameLen(10), cityLen(10) {
36     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);
85     cout << "City: ";
    in.getline(s, 80);
87     strncpy(city, s, cityLen);
    cout << "Birthyear: ";
89     in >> year;
    cout << "Salary: ";
```

```
91     in >> salary;
92     in.getline(s,80); // get '\n'
93     return in;
94 }
```

```
#include "Person.h"
2
3 class Student : public Person {
4 public:
5     Student();
6     Student(char*,char*,char*,int,long,char*);
7     void writeToFile(fstream&) const;
8     void readFromFile(fstream&);
9     int size() const {
10         return Person::size() + majorLen;
11     }
12 protected:
13     char *major;
14     const int majorLen;
15     ostream& writeLegibly(ostream&);
16     friend ostream& operator<<(ostream& out, Student& sr) {
17         return sr.writeLegibly(out);
18     }
19     istream& readFromConsole(istream&);
20     friend istream& operator>>(istream& in, Student& sr) {
21         return sr.readFromConsole(in);
22     }
23 };
24
25 Student::Student() : majorLen(10) {
26     Person();
27     major = new char[majorLen+1];
28 }
29 Student::Student(char *ssn, char *n, char *c, int y, long s, char *m) :
30     majorLen(11) {
31     Person(ID,n,c,y,s);
32     major = new char[majorLen+1];
33     strcpy(major,m);
34 }
35 void Student::writeToFile(fstream& out) const {
36     Personal::writeToFile(out);
37     out.write(major,majorLen);
38 }
39 void Student::readFromFile(fstream& in) {
40     Personal::readFromFile(in);
41     in.read(major,majorLen);
42 }
43 ostream& Student::writeLegibly(ostream& out) {
44     Personal::writeLegibly(out);
45     major[majorLen] = '\0';
46     out << " ,major=" << major;
47     return out;
48 }
```

```
istream& Student::readFromConsole(istream& in) {
50     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 }

24 template<class T>
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 }

32 template<class T>
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();
58         return true;
        }
60   }
    database.close();
62   return false;
}
64 template<class T>
std::ostream& Database<T>::print(std::ostream& out) {
66   T tmp;
    database.open(fName, std::ios::in|std::ios::binary);
68   database.clear();
    while (true) {
70     tmp.readFromFile(database);
        if (database.eof())
72         break;
        out << tmp << std::endl; // overloaded <<
74   }
    database.close();
76   return out;
}
78 template<class T>
void Database<T>::run() {
80   std::cout << "File name: ";
    std::cin >> fName;
82   std::cin.ignore(); // skip '\n';
    database.open(fName, std::ios::in);
84   if (database.fail())
        database.open(fName, std::ios::out);
86   database.close();
    char option[5];
88   T rec;
    std::cout << "1. Add 2. Find 3. Modify a record; 4. Exit\n";
90   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 ";
100         if (find(rec) == false)
            std::cout << "not ";
102         std::cout << "in the database\n";
104     }
106     else if (*option == '3') {
108         rec.readKey();
110         modify(rec);
112     }
114     else if (*option != '4')
116         std::cout << "Wrong option\n";
118     else return;
120     std::cout << *this; // overloaded <<
122     std::cout << "Enter an option: ";
124 }
126 }
```

```
1 #include <iostream>
2 #include "Person.h"
3 #include "Database.h"
4 using namespace std;
5
6 int main() {
7     Database<Person>().run();
8     // Database<Student>().run();
9     return 0;
10 }
```

Based on this code

- 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?
- Describe how the two `Student` constructors work.
- 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.