CS 7B Midterm 2 Solutions - Fall 2017 - Midterm 2 11/29/17

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** 3
   (b) **overload** 4
   (c) **container** 12
   (d) **pointer** 1
   (e) **reference** 8
   (f) **class** 5
   (g) **invariant** 10
   (h) **type** 9
   (i) **byte** 3
   (j) **constructor** 6
   (k) **scope** 7
   (l) **move** 2

   (1) (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 making 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 true at given point(s) 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:

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

(a) State and explain the output you get from lines 6, 7 and 8.
ANS: The output is OS/Compiler dependent. I'm using gcc/mingw32/5.3.0 on a 64=bit Windows7.8 and an Intel Core i7-6700 with 16 Gigabytes of RAM (free stores stuff) and I get the following:

```
sizeof(p)=4
sizeof(*p)=12
p->size()=10
```

hmmm...so, even though I'm on a 64 bit OS, because I'm using MinGW32, I'm only getting a 4 byte pointer. However, if I print out `p` I get `p = 0x716f18` which is a 3-byte address. So what's going on? Leading zeros aren't printed. The actual address is XXXXXXXX00716F18, where the first 8 hex values are determined by the compiler and the last 8 form the 4 bytes of what the MinGW32 addressing system produced in this instance, with two leading zeros.
To elaborate further, here are some other commands you might try in this context:

```
cout << "\np = " << p;
cout << "\n&p = " << &p;
cout << "\n*p = " << (*p)[0];
```

On my system now I get

```
p = 0x3b6f18
&p = 0x28fe04
*p = 0
```

The two addresses are of the addresses of `*p` and `p`, respectively, ang the dereferenced `*p` is the default value of the first element of the default-initialized vector: all zeros.

(b) Why is it not necessary to initialize `d` in the `for`-loop on line 10?
It *is* necessary! At least, according to Stroustrup: never have unitialized variables. Ok, it's not necessary for compiling and running the program, but it's necessary for good program design: RAII! The reason it compiles is that the compiler doesn't check for initialization of a declared variable. The reason it runs as expected is that the first time the variable is encountered is when it's written to from the keyboard, which initializes it.

(c) Why is it ok that the update field of the `for`-loop on line 10 is blank?
ANS: The update is actually handled by the condition field, `cin>>d`, which is true if it successfully gets a `double` from the keyboard input.

(d) How can the condition of the `for`-loop be `false` (what keyboard entry would lead to that?)
ANS: There are a variety of ways that could happen: anything that causes the `cin` object to go into a `false` state. The user could enter something other than a double, or '`cntrl+D`' on the Windows OS or '`cntrl+Z`' on the Linux OS.

(e) Describe the conditional of the `if` statement in the `for`-loop. What circumstance will first trigger that as `true`?
ANS: On line 5, enough memory for 10 doubles is allocated on the free store. The value of `n` is incremented on each iteration of the `for`-loop so when `n` grows to 10, enough memory for 20 doubles is allocated on the free store and `q` is assigned to the memory address of the first of these. Then the memory from `p` is assigned to `q`, `p` is freed and `q` is assigned to `p` and the resizing is announced to the console.

(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?
ANS: Following the rules of precedence, first the pointer to a vector object is dereferenced, then the element with index `n` is accessed and finally the index is incremented with the post-increment operator.

(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.

```
int main() {
2    // read elements into a vector without using push_back:
     ifstream read("input.txt");
```

```
    vector<double>* p = new vector<double>(10);
    cout << "\nsizeof(p)=" << sizeof(p);
    cout << "\nsizeof(*p)=" << sizeof(*p);
    cout << "\np->size()=" << p->size();
    cout << "\np␣=␣" << p;
    cout << "\n&p␣=␣" << &p;
    cout << "\n*p␣=␣" << (*p)[0];
    int n = 0; // number of elements
    for (double d; read>>d; )  {
        if (n==p->size())  {
            vector<double>* q = new vector<double>(p->size()*2);
            copy(p->begin(), p->end(), q->begin());
            delete p;
            p = q;
            cout << "\np->size()=" << p->size();
        }
        (*p)[n++] = d;
        //++n;
    }
    for(auto d: *p) cout << d << "␣";
}
```

The output is `p->size()=201 2 3 4 5 6 7 8 9 10 11 0 0 0 0 0 0 0 0 0`.

3. Consider the following code:

```
#include<iostream>
using namespace std;
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);
    cout << "\nv.capacity()␣=␣" << v.capacity();
    cout <<"\nv.size()␣=␣" << v.size();
    v.resize(4);
    cout <<"\nv.size()␣=␣" << v.size();
    return v.capacity();
}
```

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

```
v. capacity () = 10
v. size () = 0
v. size () = 4
```

(b) Give a detailed description of what `resize()` does and when it is used.
ANS: It's nicely commented! As said, it makes the vector have `newsize` elements and initializes each new elements with the default value 0.0.

(c) Write code for a function to overload the assignment operator for the above `vector` class.
ANS: This is straight out of the first part of chapter 19:

```
vector& vector::operator=(const vector& a)
{
    if (this==&a) return *this;
    // self-assignment, no work needed
    if (a.sz<=space)
    {
        // enough space, no need for new allocation
        for (int i = 0; i<a.sz; ++i) elem[i] = a.elem[i];
        // copy elements
        sz = a.sz;
        return *this;
    }
    double* p = new double[a.sz];
    // allocate new space
    for (int i = 0; i<a.sz; ++i) p[i] = a.elem[i];
    // copy elements
    delete[] elem;
    // deallocate old space
    space = sz = a.sz;
    // set new size
    elem = p;
    // set new elements
    return *this;
    // return a self-reference
}
```

(d) Write code for a `push_back()` function to add a `double` to the vector.
ANS: Again, Stourstrup says it best:

```
void vector::push_back(double d)
// increase vector size by one; initialize the new element with d
```

```
{
    if (space==0)
    reserve(8);
    // start with space for 8 elements
    else if (sz==space)
    reserve(2*space); // get more space
    elem[sz] = d;
    // add d at end
    ++sz;
    // increase the size (sz is the number of elements)
}
```

(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`?
ANS: Write "`template<typename T>`" before the definition of the class and change all references to type `double` to type `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.
ANS: Explain to someone you know why this works:

```
#include<iostream>
using namespace std;

int multiply(int m, int n) {
    if(n == 0) {
        return 0;
    }
    return m + multiply(m,n-1);
}

int main() {
    int x{0}, y{0};
    while(cin >> x >> y) {
        cout << "\nThe␣product␣of␣" << x << "␣and␣" << y
             << "␣is␣" << multiply(x,y) << endl;
    }
}
```

Here is a typical run:

```
3 4

The product of 3 and 4 is 12
```

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

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

ANS: I'm sure you can improve on this. Why does it bomb out ager only 40000 terms?

```
#include<iostream>
using namespace std;
const double ln2{0.693147180559945309417232121458 18};
```

```
5  double altHarmonic(double m) {
       if(int(m) == 1) {
7          return 1;
       }
9      if(int(m)%2)
           return altHarmonic(m-1)+1/m;
11     return altHarmonic(m-1)-1/m;
   }
13
   int main() {
15     double n{0.};
       cout << "\nEnter␣n␣to␣compute␣the␣partial␣sum␣of␣the␣alternating"
17          << "\nseries,␣1␣-␣1/2␣+␣1/3␣-␣1/4␣+␣...␣1/n:␣";
       while(cin >> n) {
19         cout << "\nThe␣partial␣sum␣of␣" << n << "␣terms␣is␣"
                << altHarmonic(n) << ",␣an␣error␣of␣"
21              << altHarmonic(n)-ln2 << "\nEnter␣another␣n:␣";
       }
23 }
```

```
Enter n to compute the partial sum of the alternating series, 1 - 1/2 + 1/3 - 1/
4 + ... 1/n: 100

The partial sum of 100 terms is 0.688172, an error of -0.004975
Enter another n: 1000

The partial sum of 1000 terms is 0.692647, an error of -0.00049975
Enter another n: 10000

The partial sum of 10000 terms is 0.693097, an error of -4.99975e-005
Enter another n: 20000

The partial sum of 20000 terms is 0.693122, an error of -2.49994e-005
Enter another n: 30000

The partial sum of 30000 terms is 0.693131, an error of -1.66664e-005
Enter another n: 40000

The partial sum of 40000 terms is 0.693135, an error of -1.24998e-005
Enter another n: 50000

Process returned -1073741571 (0xC00000FD)   execution time : 47.072 s
```

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 $n_0 = 7$
   Let's start with the recursive method. Here it's couched in terms of a function that is looped to prompt the user for the initial value and then print out all the values to the console.

```
/** 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    n_( i  1 ) + 1 otherwise. */

#include<iostream>
using namespace std;

double collatz(int64_t m) {
    if(m == 1) {
        return 1;
    }
    if(int(m)%2) {
        cout << 3*m+1 << " " << (3*m+1)/2 << " ";
        return collatz((3*m+1)/2);
    }
    cout << m/2 << " ";
    return collatz(m/2);
}

int main() {
    int64_t n{0};
    cout << "\nEnter n to see the collatz sequence from there: ";
    while(cin >> n) {
        cout << "\nThose are the terms starting from " << n
             << ", as always, ending in " << collatz(n)
             << "\nEnter another n: ";
    }
}
```

Here is some typical output, starting the sequence from 7:

```
Enter n to see the collatz sequence from there: 7
22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Those are the terms starting from 7, as always, ending in 1
Enter another n: 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 2
51 754 377 1132 566 283 850 425 1276 638 319 958 479 1438 719 2158 1079 3238 161
9 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
Those are the terms starting from 31, as always, ending in 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.
ANS: Consider it done!

7. 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 */
   }
```

Here's a listing of a program that produces an array of sequence values and then sorts them:

```
/// G. Hagopian
/**  Consider the conditional function
Next(n) =
3n/2 if n(mod)2 = 0
(3n + 1)/4 if n(mod)4 = 1
(3n     1)/4 if n(mod)4 = 3*/

#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;

void quadlatz(vector<int64_t>& vi, int64_t m) {
    if(find(vi.begin(),vi.end(),m)!=vi.end() || m<0) {
        return;
    }
    else if(m%2==0) {
        vi.push_back(m);
        cout << m/2*3 << "␣";
        quadlatz(vi,m/2*3);
    }
    else if(m%4==1) {
        vi.push_back(m);
        cout << (3*m+1)/4 << "␣";
        quadlatz(vi,(3*m+1)/4);
    }
    else if(m%4==3) {
        vi.push_back(m);
        cout << (3*m-1)/4 << "␣";
        quadlatz(vi,(3*m-1)/4);
    }
}

int main() {
    vector<int64_t> vi;
```

```
    int64_t n{0};
    cout << "\nEnter␣n␣to␣see␣the␣sequence␣from␣there:␣";
    while(cin >> n) {
        vi.clear();
        cout << "\nHere␣are␣the␣terms␣starting␣from␣" << n << endl;
        quadlatz(vi,n);
        sort(vi.begin(),vi.end());
        cout << "\nAnd␣here␣they␣are␣in␣order:␣";
        for(v:vi) cout << v << "␣";
    }
}
```

Starting with 4, we get this output:

```
Enter n to see the sequence from there: 4
Here are the terms starting from 4
6 9 7 5 4
And here they are in order: 4 5 6 7 9
```

A fairly short sequence...which raises interesting questions!

8. 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.
ANS: Here it is!

```
/**(a) Given that a geometric sequence has each term as a constant (ratio) mul
the derived class GeomSequence and its methods.*/
template<typename T>
class GeomSequence : public Sequence<T> {
public:
    GeomSequence<T>(T g = 1, T rtio = 1.);
protected:
    virtual T nextValue();
    T r;
};
template<typename T>
GeomSequence<T>::GeomSequence(T f, T rtio) : Sequence<T>(f), r(rtio) {}

template<typename T>
T GeomSequence<T>::nextValue() {
    this->curr *= r;
    return this->curr;
}
```

(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.
ANS: Here's a go at that:

```
template<class T>
class lucasSequence {
    T first, second, curr;
public:
    lucasSequence(T f, T s) : first(f), second(s) {
        ///curr = first+second+third;
    }
    T nextTerm() {
        curr = first + second;
        first = second;
        second = curr;
        return curr;
    }
    void printSeq(int n) {
        std::cout << first << ",␣" << second << ",␣";
        for(int i = 0; i < n-4; ++i)
            std::cout << nextTerm() << ",␣";
        std::cout << nextTerm();
    }
};
```

Driving that with
`BiSequence<int64_t> bs(2,1); bs.printSeq(100);` produces the following:

```
2, 1, 3, 4, 7, 11, 18, 29, 47, 76, 123, 199, 322, 521, 843, 1364, 2207, 3571, 5778, 9349,
15127, 24476, 39603, 64079, 103682, 167761, 271443, 439204, 710647, 1149851, 1860498,
3010349, 4870847, 7881196, 12752043, 20633239, 33385282, 54018521, 87403803, 141422324,
228826127, 370248451, 599074578, 969323029, 1568397607, 2537720636, 4106118243, 6643838879,
10749957122, 17393796001, 28143753123, 45537549124, 73681302247, 119218851371, 192900153618,
312119004989, 505019158607, 817138163596, 1322157322203, 2139295485799, 3461452808002,
5600748293801, 9062201101803, 14662949395604, 23725150497407, 38388099893011,
62113250390418, 100501350283429, 162614600673847, 263115950957276, 425730551631123,
688846502588399, 1114577054219522, 1803423556807921, 2918000611027443, 4721424167835364,
7639424778862807, 12360848946698171, 20000273725560978, 32361122672259149,
52361396397820127, 84722519070079276, 137083915467899403, 221806434537978679,
358890350005878082, 580696784543856761, 939587134549734843, 1520283919093591604,
2459871053643326447, 3980154972736918051, 6440026026380244498, -8026563074592389067,
-1586537048212144569, 8833643950905017980, 7247106902692873411, -2365993220111660225,
4881113682581213186, 2515120462469552961, 7396234145050766147
```

where, evidently, we run into wrap-around errors toward the end.

(c) How would you go about making the `Sequence` class an `abstract base class`? Explain.
ANS: Here's what we wrote in class:

```
#include<cstdint>
#include<iostream>
template<typename T>
class Sequence {
public:
    Sequence(T f = 0) : first(f), curr(f) {}
    virtual ~Sequence() {};
    void printSequence(int n);
protected:
    Sequence() { }
    virtual T firstValue();
    virtual T nextValue();
    T first;
    T curr;
};

template<typename T>
void Sequence<T>::printSequence(int n) {
    std::cout << firstValue();
    for(int i = 2; i <=n; i++)
        std::cout << ' ' << nextValue();
}

template<typename T>
T Sequence<T>::firstValue() {
    curr = first;
    return curr;
}

template<typename T>
T Sequence<T>::nextValue() {
    return ++curr;
```

```
}

//-----------------------------------------------------------------------
template<typename T>
class ArithSequence : public Sequence<T> {
public:
    ArithSequence<T>(T f = 0, T i = 1);
protected:
    virtual T nextValue();
    T inc;
};

template<typename T>
ArithSequence<T>::ArithSequence(T f, T i) : Sequence<T>(f), inc(i) {}

template<typename T>
T ArithSequence<T>::nextValue() {
    this->curr += inc;
    return this->curr;
}
```

The constructor `Sequence() { }` is protected. That means that it can only be used directly from classes derived from `Sequence` (using the `:Sequence` notation). In other words, `Sequence` can only be used as a base for classes, such as `ArithSequence` and `geomSequence`. The purpose of that protected: is to ensure that we don't make `Sequence` objects directly.

Sequence's `nextValue()` must somehow invoke one of `Sequence's` functions if the `Sequence` is an `ArithSequence` and one of `geomSequence's` functions if the `Sequence` is a `geomSequence`. That's what the word `virtual` in the `nextValue()` declaration ensures: if a class derived from `Sequence` has defined its own `nextValue()` (with the same type as `Sequence's nextValue()`), that `nextValue()` will be called rather than `Sequence's nextValue()`. Defining a function in a derived class so that it can be used through the interfaces provided by a base is called **overriding**.

Note that despite its central role in `Sequence`, `nextValue()` is protected; it is not meant to be called by "the general user" that's what `nextValue()` is for but simply as an "implementation detail" used by `nextValue()` and the classes derived from `Sequence`.

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 `Point2d` class, a `Triangle` class and a `Polygon` class and associated methods that will allow you to construct a polygon with $n \geq 3$ sides.

   ANS: Here is a `Point2d` class that may be sufficient for our needs:

```
/// Point2d.h
#include<iostream>
#pragma once

class Point2d {
    double x;
    double y;
public:
    Point2d(double x1=0, double y1=0) : x(x1), y(y1) {}
    double getX() { return x; }
    double getY() { return y; }
    void incX(double incx) { x += incx; }
    void incY(double incy) { y += incy; }
    void setX(double xin)  { x = xin; }
```

```
    void setY(double yin)  { y = yin; }
    bool operator==(Point2d p) {
        return p.getX()==x && p.getY()==y;
    }
};

std::ostream& operator<<(std::ostream& os, Point2d p) {
    os << '(' << p.getX() << ',' << p.getY() << ')';
    return os;
}
```

Here is a `Triangle` class the uses this:

```
#include "Point2d.h"

class Triangle {
    Point2d p1;
    Point2d p2;
    Point2d p3;
public:
    Triangle(Point2d q1, Point2d q2, Point2d q3) :
        p1(q1), p2(q2), p3(q3) {}
    Point2d getP1() { return p1; }
    Point2d getP2() { return p2; }
    Point2d getP3() { return p3; }
};


std::ostream& operator<<(std::ostream& os, Triangle t) {
    os << '{' << t.getP1() << ',' << t.getP2()
       << ',' << t.getP3() << '}';
    return os;
}
```

...and here is the independent, and necessarily more robust `Polygon` class. It tests that the line segment from (x_1, y_1) to (x_2, y_2) does not intersect the line segment from (x_3, y_3) to (x_4, y_4). That is, The lines $y = \frac{y_1 - y_2}{x_1 - x_2}x + \frac{x_1y_2 - x_2y_1}{x_1 - x_2}$ and $y = \frac{y_3 - y_4}{x_3 - x_4}x + \frac{x_3y_4 - x_4y_3}{x_3 - x_4}$ don't intersect in common space. Assuming the lines are not parallel, the extended lines will intersect where $x^* = \frac{x_1(x_3(y_2 - y_4) + x_4(y_3 - y_2)) + x_2(x_3(y_4 - y_1) + x_4(y_1 - y_3))}{(x_1 - x_2)(y_3 - y_4) + (x_4 - x_3)(y_1 - y_2)}$ must be in the intersection of $[x_1, x_2] \cap [x_3, x_4]$.

```
//Polyton.h adapted from Stroustrup's Graph.h by G. Hagopian
#include<vector>
#include<utility>
#include<stdexcept>
#include "Point2d.h"

class Polygon  { // Holds a sequence of points that form a polygon
public:
    void draw() const;  // list the sequence of points in text form
    virtual void move(int dx, int dy); // move the shape +=dx and +=dy
    Point2d point(int i) const { return points[i]; } //get ith point
    int number_of_points() const { return int(points.size()); }
```

```
    ~Polygon() { }
    Polygon()=default;
    Polygon(std::vector<Point2d> vin) : points(vin) {}
    void add(Point2d p) ;                // add p to points
    void set_point(int i,Point2d p);   // points[i]=p;
    void clear() { points.clear(); }
private:
    std::vector<Point2d> points;
    Polygon& operator=(const Polygon&);
};
//-----------------------------------------------------------------------
void Polygon::move(int dx, int dy)  {  // move the shape +=dx and +=dy
    for (int i = 0; i<points.size(); ++i) {
        points[i].incX(dx);
        points[i].incY(dy);
    }
}
//-----------------------------------------------------------------------
// do two lines (p1,p2) and (p3,p4) intersect?
// if so return the distance of the intersect point as distances from p1
inline std::pair<double,double> line_intersect(Point2d p1, Point2d p2,
                                               Point2d p3, Point2d p4,
                                               bool& parallel) {
    double x1 = p1.getX();
    double x2 = p2.getX();
    double x3 = p3.getX();
    double x4 = p4.getX();
    double y1 = p1.getY();
    double y2 = p2.getY();
    double y3 = p3.getY();
    double y4 = p4.getY();

    double denom = ((y4-y3)*(x2-x1) - (x4-x3)*(y2-y1));
    if (denom == 0){
        parallel= true;
        return std::pair<double,double>(0,0);
    }
    parallel = false;
    return std::pair<double,double>(((x4-x3)*(y1-y3)-(y4-y3)*(x1-x3))/denom,
                                    ((x2-x1)*(y1-y3)-(y2-y1)*(x1-x3))/denom);
}
//-----------------------------------------------------------------------
//intersection between two line segments
//Returns true if the two segments intersect,
//in which case intersection is set to the point of intersection
bool line_segment_intersect(Point2d p1, Point2d p2,
                            Point2d p3, Point2d p4,
                            Point2d& intersection){
   bool parallel;
   std::pair<double,double> u = line_intersect(p1,p2,p3,p4,parallel);
   if (parallel || u.first < 0 || u.first > 1
                || u.second < 0 || u.second > 1) return false;
   intersection.setX(p1.getX() + u.first*(p2.getX() - p1.getX()));
```

```
   intersection.setY(p1.getY() + u.first*(p2.getY() - p1.getY()));
   return true;
}
//-----------------------------------------------------------------------
void Polygon::add(Point2d p)
try {
    int np = number_of_points();
    std::cout << "\nnp = " << np;
    if (1<np) { // check that the new line isn't parallel to the previous one
        if (p==point(np-1))
            std::runtime_error("polygon point equal to previous point");
        bool parallel;
        line_intersect(point(np-1),p,point(np-2),point(np-1),parallel);
        std::cout << "\nparallel = " << parallel;
        if (parallel)
            throw std::runtime_error("adjacent vertices in a line");
    }
    Point2d ignore(0,0);
    for (int i = 1; i<np-1; ++i) { // does new segment interset old?
        if (line_segment_intersect(point(np-1),p,point(i-1),point(i),ignore)) {
            std::cout << "At " << ignore;
            throw std::runtime_error(", the sides intersect.");
        }
    }
    points.push_back(p);
}
catch (std::exception &e) {
                std::cout<<"Caught exception: "<<e.what()<<"\n";
}
//-----------------------------------------------------------------------
void Polygon::draw() const
try {
    if (number_of_points() < 3)
        std::runtime_error("less than 3 points in a Polygon");
    for(int i = 0; i < number_of_points()-1; ++i)
        std::cout<<point(i)<<"->"<<point(i+1)<<'\n';
    std::cout<<point(number_of_points()-1)<<"->"<<point(0);
}
catch (std::exception &e) {
                std::cout<<"Caught exception: "<<e.what()<<"\n";
}
```

We can test the program with

```
#include "Point2d.h"
#include "Triangle.h"
#include "Polygon.h"
using namespace std;

int main() {
    int n{0}; ///number of vertices
    double x{0}, y{0};
    Triangle T1(Point2d(0,0),Point2d(1,0), Point2d(1,1));
    cout << T1;
```

```
    cout << "\nHow␣many␣vertices␣does␣your␣polygon␣have?:␣";
    Polygon p;
    while(cin>>n) {
        for(int i = 0; i < n; ++i) {
            cout << "\nEnter␣x␣for␣point␣"<<i<<":␣";
            cin >> x;
            cout << "\nEnter␣y␣for␣point␣"<<i<<":␣";
            cin >> y;
            p.add(Point2d(x,y));
        }
        cout << "\nHere's␣your␣polygon:\n";
        p.draw();
        cout << "\nHow␣many␣vertices␣does␣your␣polygon␣have?:␣";
        p.clear();
    }
}
```

A typical run, showing the behavior when the fourth point of a quadrilateral causes the intersection error:

```
{(0,0),(1,0),(1,1)}
How many vertices does your polygon have?: 4

Enter x for point 0: 0
Enter y for point 0: 0

Enter x for point 1: 2
Enter y for point 1: 2

Enter x for point 2: 2
Enter y for point 2: 1

Enter x for point 3: 1
Enter y for point 3: 1
At (1,1), the sides intersect

Here's your polygon:
(0,0)->(2,2)
(2,2)->(2,1)
(2,1)->(0,0)
How many vertices does your polygon have?:
```

10. 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);
    }
```

```
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();
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 ";
            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 }
}
```

```
1 #include <iostream>
  #include "Person.h"
3 #include "Database.h"
  using namespace std;
5
  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?
ANS: Virtual methods allow a derived class to override methods inherited from a base class. When is it appropriate/inappropriate to use virtual methods? It's not always known whether or not a class will be sub classed. Should everything be made virtual, just "in case?" Or will that cause significant overhead?
When you design a class you should have a pretty good idea as to whether it represents an interface (in which case you mark the appropriate overrideable methods and destructor virtual) **OR** if it's intended to be used as-is, possibly composing or composed with other objects.
Here a `Student` "is a" `Person`, but you could have a stand-alone `Person` who isn't necessarily pigeonholed as this kind or that kind.
Your intent for the class should be your guide. Making everything virtual is often overkill and sometimes misleading regarding which methods are intended to support runtime polymorphism.
Here are some heuristics to follow:

- As long as you do not need to derive from a class, then don't write any virtual method, once you need to derive, only make virtual those methods you need to customize in the child class.
- If a class has a virtual method, then the destructor shall be virtual (end of discussion).
- Try to follow NVI (Non-Virtual Interface) idiom, make virtual method non-public and provide public wrappers in charge of assessing pre and post conditions, so that derived classes cannot accidentally break them.

(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.