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");
```

```
4       vector<double>* p = new vector<double>(10);
        cout << "\nsizeof(p)=" << sizeof(p);
6       cout << "\nsizeof(*p)=" << sizeof(*p);
        cout << "\np->size()=" << p->size();
8       cout << "\np␣=␣" << p;
        cout << "\n&p␣=␣" << &p;
10      cout << "\n*p␣=␣" << (*p)[0];
        int n = 0; // number of elements
12      for (double d; read>>d; )  {
            if (n==p->size())  {
14              vector<double>* q = new vector<double>(p->size()*2);
                copy(p->begin(), p->end(), q->begin());
16              delete p;
                p = q;
18              cout << "\np->size()=" << p->size();
            }
20          (*p)[n++] = d;
            //++n;
22      }
        for(auto d: *p) cout << d << "␣";
24  }
```

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`.