Practice test for midterm 3

November 15, 2018

1 Classes

► Here is a pair of class definitions, and a pair of variable declarations:

```
class A {
  public:
    int x;
  private:
    float y;
};

class B {
  public:
    int y;
    A a;
  private:
    float x;
    A b;
};

A foo;
B bar;
```

Label each of the following expressions with OK if it is OK, or Error if it would cause an error:

a) `foo.x`

b) `bar.x`

c) `bar.a`

d) `foo.y`

e) `bar.b.x`

f) `bar.a.y`

► Add whatever *constructors* are necessary to the following class so that the code after it will be correct:

```
class A {
  public:

    // Constructors here...
  private:
    int x;
    string y;
    float z;
};

A a1 = 12;
A a2{1.2, 10};
A a3{"Hello", 15};
```

▶ Here is a class with two member functions defined inside of it. Move the definitions of these member functions out of the class.

```
class dog {
  public:

    void feed() {
        if(!fed) {
            cout << "Dog is now fed.";
            fed = true;
        }
        else
            cout << "Dog is not tired.";
    }

    void walk() {
        if(!tired) {
            cout << "Walkies";
            tired = true;
        }
        else
            cout << "Too tired to walk.";
    }

  private:
    bool tired, fed;
};
```

▶ Complete the following class definition for a class that stores information about teachers by filling in the definitions of the member functions:

```
class teacher {
  public:

    void give_tenure() {

    void assign_class(string c) {

    string get_name() {


  private:
    bool has_tenure = false;
    vector<string> classes;
    string name;
};
```

▶ Think about a class designed to represent a *color*. How would you represent a color? Would your representation support mixing colors together to get new colors? Sketch a class (data members and function declarations only) `color` and explain why you think it would work for this purpose (or explain what its limitations are).

2 Multi-file projects

▶ Suppose we want to split the following program into three files: `main.cpp`, `triangle.hpp` (containing declarations) and `triangle.cpp` (containing implementations). Circle the parts of the code that code into each file, and add anything else that would be needed to make the resulting project work.

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

class triangle {
  public:
    void set_size(int s);
    void draw();

  private:
    int size;
};

int main() {
    triangle t;
    t.set_size(10);
    t.draw();
    return 0;
}

void triangle::set_set(int s) {
    size = s;
}

void triangle::draw() {
    string t = "*";
    string s{size, ' '};

    for(int i = 1; i < size; ++i) {
        cout << s << t << s << endl;
        t += "**";
        s.pop_back();
    }
}
```

▶ What are the commands you would use to manually compile the project in the previous problem?

▶ Explain what the rules are for the order of object (`.o`) files in the final *link step*. If `A.cpp` uses definitions from `B.cpp`, where should `A.o` appear in the list of object files, relative to `B.o`?

▶ For each of the following, state whether it can/should appear in *source* files, *header* files, or both:

a) Function definitions

b) Function declarations

c) `using namespace std;`

d) `#include<...>`

e) `#pragma once`

f) `int main()`

▶ Explain what problem header files are intended to solve; why do we need `.hpp` files at all?

3 Exceptions

▶ What is wrong with the following code? How would you fix it?

```
try {
    f();
}
catch(logic_error& e) {
    cout << "LE";
}
catch(length_error& e) {
    cout << "LenE";
}
catch(runtime_error& e) {
    cout << "RE";
}
catch(range_error& e) {

}
```

► The following function takes a vector of pairs of ints and divides the first element of each pair by the second. E.g., if the input vector was {4, 2, 9, 3, 12, 3} then the returned vector would be {2, 3, 4}. What kinds of errors could occur in this function? Add assertions to check for them.

```
#include <cassert>
vector<int> divide_by(vector<int> v) {



    vector<int> vout;



    for(int i = 0; i < v.size(); i += 2) {


        vout.push_back(v.at(i) / v.at(i+1));


    }


    return vout;
}
```

▶ For each of the standard expression types to the right, indicate what the following code would print if it were thrown from the function h

```
void h() {
    throw   // exception thrown here
}

void g() {
    try {
        h();
    }
    catch(domain_error& e) {
        cout << "DE in g";
    }
    catch(runtime_error& e) {
        cout << "RE in g";
    }
}

void main() {
    try {
        g();
        h();
    }
    catch(range_error& e) {
        cout << "RE in main";
    }
    catch(out_of_range& e) {
        cout << "OOR in main";
    }
    catch(logic_error& e) {
        cout << "LE in main";
    }
    catch(...) {
        cout << "Other in main";
    }
}
```

a) `domain_error`

b) `range_error`

c) `out_of_range`

d) `length_error`

e) `system_error`

f) `exception`

▶ Explain the difference between assertions and expressions. When would you use each?

► The following code uses *assertions* to check for problems. Convert it to using standard exceptions (and choose exception types that seem appropriate to you).

```
// Uses remainder hashing to compute the hash value of a string s.
// Take CSci 133 if you want to know more!
int hash(string s, int m) {
    assert(!s.empty());  // Input string cannot be empty
    assert(m > 0);       // Size must be positive

    int h = 0;
    for(char c : s) {
        assert(c > 0);           // No non-ASCII characters
        assert(256 * h + c > h); // No numeric overflow

        h = (256 * h + c) % m;
    }

    return h;
}
```