Name:__ Poly-Id: __

CS1124

Spring 2011

Exam Two

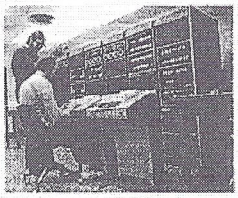
NOTE:

- There are two LONG problems at the end of the test.
- A good strategy would be to do all the short questions that you can do *quickly*,
  - then get to the LONGER problems at the end of the test;
  - and finally go back through the shorter ones.

1) **DO NOT CHEAT**. (They told me I have to say that.)
2) Write NEATLY.
3) *Do not* tear any pages out of your Blue Book.
4) *Do not* tear any pages from this document. Be sure that you hand in *all* 6 pages of this test, including this cover sheet.
5) Place your answers for questions **1–11** in this document.
6) Place your answers for questions **12 and 13** in your Blue Book.
7) Put your name and ID number on the cover of your Blue Book.
8) Put your name and ID number as indicated on *each* page of this test. Please circle your last name. Thank you.
9) If you need "scratch" paper, use your Blue Book but cross out anything you do not want graded.
10) You are not required to write comments for any code in this test.
11) Do not begin until you are instructed to do so.
12) **Good Luck!**

Name:_____ Poly-Id:_____

1. [extra credit] Who created C, (**not** C++)



a. Gosling
b. Kildall
c. Ritchie
d. Stroustrup
e. Thompson
f. van Rossum
g. Wirth
h. Wall

2. [2 pts] The data members in the initialization list are initialized ***after*** the body of a constructor begins executing.

True False

3. [2 pts] You cannot change the *associativity* of the less than operator when you overload it.

True False

Questions 4—10 are four points each.

4. Given a class called `Thing` and the code

```
Thing thingOne;
```

What function call is the following expression equivalent to?

```
Thing thingTwo = thingOne;
```

a. `operator=(thingTwo, thingOne)`
b. `thingTwo.operator=(thingOne)`
c. Neither a. nor b. because the operator can't be overridden
d. Neither a. nor b. because the operator has to be overridden as a friend
e. Neither a. nor b. because the operator needs to be defined in the class.
f. Either (a) or (b), depending on how the programmer chose to implement the operator.
g. None of the above

Name: _____ Poly-Id: _____

Note: Questions 5 – 7 refer to the classes defined below.

Read carefully.

```
class Base {
public:
   Base() {}
   virtual void bar() {cout << "In Base bar()"; }
private:
};

class Derived : public Base {
public:
   void foo() {cout << "In Derived foo()"; }
   void bar() {cout << "In Derived bar()"; }
   ~Derived() {cout << "In Derived destructor"; }
private:
};
```

E 5. What would be the result of:

```
int main() {
   Base* p = new Derived();
   p->foo();
}
```

a. The program runs and prints: `In Derived foo()`

b. The program runs and prints: `In Base foo()`

c. Runtime error

d. Compilation error because there is no Derived constructor

e. None of the above

Compile time error because Base can't find()

A 6. What would be the result of:

```
int main() {
   Derived derived;
   Base base;
   base = derived;
   derived.bar();
}
```

a. The program runs and prints: `In Derived bar()`

b. The program runs and prints: `In Base bar()`

c. Runtime error

d. Compilation error because there is no Derived constructor

e. Compilation error because Base cannot be assigned to Derived.

f. None of the above

I

7. Using the classes on the previous page, what would be the result of:

```
int main() {
  Base* p = new Derived();
  delete p;
}
```

Not virtual so destructor is not overridden.

- a. The program runs and prints: `In Derived destructor`
- b. The program runs and prints: `In Base destructor`
- i. The program runs and prints nothing.
- j. Runtime error
- k. Compilation error because there is no Base destructor
- l. Compilation error because Derived object cannot be assigned to Base pointer.
- m. None of the above

C

8. Given:

const * to an int

```
int theAnswer = 17;
int* const p = &theAnswer;  // Line A
```

Which of the following is true?

- a. Line A does not compile
- b. `*p = 42;  // Does not compile`

dereferences const p not const int

- c. `int another = 42;`
  `p = &another;  // Does not compile` — exactly
- d. None of the above

9. A student implements a vector class. Testing his class, he writes some code:

```
Vector v;
for (int i = 0; i < 42; ++i) v.push_back(i);
v = 17;   // Huh?
```

Looking at his test code, he sees that the last line was a mistake, and is surprised that it compiled without an error.

What is the best way he can fix his Vector class so that the last line will not compile?

declare his constructor "explicit".

Constructor that takes int.

Name: _____ Poly-Id: _____

10. Given:

```
class BaseClass {
public:
   void foo() const;
};
```

We want to be sure that all classes that inherit from BaseClass to override the method `foo()` Modify the above code to guarantee that.

```
class BaseClass {
public:
virtual void foo() const = 0;
};
// Note: This does not guarantee that all classes will implement foo() method
// that you can only create instances of subclasses that do.
```

(-2)

11. [5 pts] Given
    - a class `Base`
    - a class `Derived`
        - Derived publicly inherits from `Base`
        - Derived has an `int` member variable named `foo`

Write a copy constructor for `Derived`.

we want to call Base copy constructor, too.

```
Derived (const Derived& rhs) : Base(rhs) {
    foo = rhs.foo;
}
```

(3)

[Yes, you're right, it would normally be unnecessary to write this for the Derived, but we want you to it anyway.]

Name:_____ Poly-Id: ___

Programming – Blue Book

- Place the answers to the following questions in your Blue Book.
- **Comments** are **not** required in the blue book!
  However, if you think they will help us understand your code,
  feel free to add them.
- **Separate compilation** is **not** required.
  It will just take more of your time to define the methods outside of the class and will *not* earn you any additional points.
- **Includes** and **using namespace** directives are **not** required.
- Read the questions *carefully!*

12. [30 points] **Define the class `Person`.**

- People in the land of Woz like to get married.
- Unfortunately, they are not terribly concerned about the sanctity of marriage, which means that when marrying,
  - a Person may dump (i.e. divorce) their current spouse
  - or steal someone else's spouse.
- Your job is to define the class Person so that the code below works correctly. (Note the use of the output operator.)
- Yes, this involves pointers. Yes, people have to be told if they have been dumped, stolen, etc. **Do not store your spouse's name** in the Person class!
- Note: the marries method **does not display** anything

Test code:

```
int main() {
    Person john("John"), fred("Fred"), mary("Mary"), sue("Sue");
    john.marries(mary);    // Fine.
    mary.marries(fred);    // Mary dumps John for Fred.
                           // John is not married anymore.
    john.marries(sue);     // Fine.
    fred.marries(sue);     // Fred dumps Mary and steals Sue from John.
                           // Poor John is alone again.  So is Mary
    cout << john << endl;  // Prints: John: married to: No one
    cout << mary << endl;  // Prints: Mary: married to: No one
    cout << fred << endl;  // Prints: Fred: married to: Sue
    cout << sue << endl;   // Prints: Sue: married to: Fred
}
```

13. [31 points] **Define the class `Foo`:**

- `Foo` has just one field, a vector of `Glub` pointers.
  The class `Glub` has any necessary methods and operators.
- **Define the assignment operator for `Foo`.**
  (Don't bother with the destructor or copy constructor)
  Naturally, copying should involve making a *deep* copy. Don't just copy pointers!
- **Define an equality operator for `Foo`.**
  Two `Foo`s are considered equal if all of their `Glub`s are equal. (I.e. same number of `Glub`s and each `Glub` in one `Foo` matches the `Glub` in the same position in the other Foo.)
  The Glubs do not have to have the same address to be equal.
  And yes, Glub does have an equality operator.
- Do **NOT** define Glub. Do **NOT** write a main. Just define the class `Foo` as stated above.
- **Note:** Foo does have a method to add Glubs, **but you don't have to write it or use it**.
  I just mention it, in case you were wondering how the Glubs got there.
  Also, Glubs are never removed from their Foo. This makes your life easier.