Name: Sandeep Khatri NYU Net Id: SK5437

CS1124

Fall 2014

Exam Two

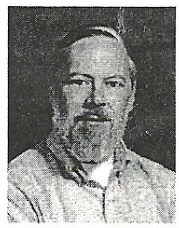
NOTE:

- There is one LONG problem 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 LONG problem at the end of the test;
  - and finally go back through the shorter ones.
- You will have **one hour forty minutes** for this exam.

1) **DO NOT CHEAT**. (They told me I have to say that.)
2) Write CLEARLY. If we can't read it, we can't give you credit for it.
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* 9 pages of this test, including this cover sheet.
5) I didn't put any #includes or using namespace statements in white book questions. Assume any that are necessary were there.
6) Place your answers for questions **1–10** in this document.
7) Place your answer for **the programming question** in your **Blue Book**.
8) Put your name and NYU Net ID number on the cover of your Blue Book. Please circle your last name. Thank you.
9) Put your name and NYU Net ID number as indicated on *each* page of this test. Please circle your last name. Thank you.
10) If you need "scratch" paper, use your Blue Book but cross out anything you do not want graded.
11) You are not required to write comments or include statements for any code in this test.
12) Do not begin until you are instructed to do so.
13) **Good Luck!**

Name: Sundeep Kalra NYU Net Id: SK5437

1. [extra credit] Who created C?



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

[Questions 2 – 11 are worth five points each]

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

```
Thing thingOne;
```

What function **call** is the following line equivalent to?

```
Thing thingTwo = thingOne;
```

a. `operator=(thingTwo, thingOne)`
b. `thingTwo.operator=(thingOne)`
c. `ostream& Thing::operator=(const Thing& rhs);`
d. Neither (a) nor (b) because it is using the `Thing` copy constructor.
e. Neither (a) nor (b) because the operator has to be overridden as a friend
f. Either (a) or (b), depending on how the programmer chose to implement the operator.
g. None of the above

3. Given that class Thing has a post-increment operator that is implemented as a member function, what is the function call equivalent to line A below:

```
Thing a;
a++;        // Line A
```

a.operator++(0)

Answer: 2. operator++(int)

Name: Sundeep Khatri NYU Net Id: SK5437

4. Given the following code, what is the output?

```
int main() {
    int x = 6;
    int* arr = new int[10];
    for (int i = 0; i < 10; ++i) {
        arr[i] = 2*i;
    }
    int* p = arr + 1;
    int* q = p + x;
    cout << "A: " << *q << endl;
    *p = x;
    p++;
    cout  << "B: " << *p << endl;
}
```

0 1
2 2
4 3
6 4
8 5
10 6
12 7
14
16
18

arr[] address of arr[0]

q = arr[0] + 6 address of arr[7]

Output:

A: 7 14

B: 2 4

5. Given:

```
const int x = 10;
```

Which of the following **will** compile?

a. `int* p = &x;`
b. `const int* q = &x;` ← only const int
c. `int* const r = &x;`
d. All of the above
a. None of the above

6. Given:

```
class Base {
public:
    Base() { cout << "1"; }
    Base(const Base& rhs) { cout << "2"; }
};

class Member {
public:
    Member() { cout << "3"; }
    Member(const Member& rhs) {cout << "4";}
};

class Derived : public Base {
public:
    Derived() {cout << "5";}
    Derived(const Derived& rhs) {cout << "6";}
private:
    Member member;
};

int main() {
    Derived der;
    cout << "\nLine A\n";  // Line A
    Derived der2(der);
    cout << "\nLine B\n";  // Line B
}
```

What is the output between lines A and B? (i.e. what numbers are printed?)

a. 135	g. 235	m. 351
b. 136	h. 236	n. 361
c. 146	i. 246	o. 362
d. 153	j. 253	p. 462
e. 163	k. 263	q. 632
f. 164	l. 264	r. None of the above

Name: Sundeep Khaler NYU Net Id: sk5437

7. Given the following class declaration (and assuming we have definitions for these functions):

```
class Orc {
public:
   Orc();
   Orc(int myStrength, int myBeauty);

   int getStrength() const;
   int getBeauty() const;
   void display();

   friend bool operator< (const Orc& lhs, const Orc& rhs);
private:
   int strength;
   int beauty;
};
```

Identify a function that we can add to the class in order for the following code to compile, **other than** another overload of operator<. Just provide the **prototype** of the function.

```
Orc dabu(2,-3);
if ( 3 < dabu ) {
      dabu.display();
}
```

Answer: Orc(int);

8. What is the result of compiling and running the following program?

```
class Base {
public:
    void method() { cout << "method1\n"; }
};
class Derived : public Base {
public:
    virtual void method() { cout << "method2\n"; }
};

int main() {
    Base* bp = new Derived();
    bp->method();
}
```

needs to be virtual

a. The program compiles and prints "method1"
b. The program compiles and printis"method2"
c. The program compiles and runs to completion without printing anything.
d. The program compiles and crashes when it runs.
e. The program does not compile.
f. None of the above.

Name: Sundeep Khaler NYU Net Id: SK5437

9. What is the result of the following?

```
class Base {
public:
    virtual void foo() { cout << " - Base::foo()\n"; }
};
class Derived : public Base {
public:
    void foo() { cout << " - Derived::foo()\n"; }
};

void func(Base& arg) {
    cout << "func(Base)";
    arg.foo();
}
void func(Derived& arg) {
    cout << "func(Derived)";
    arg.foo();
}

void otherFunc(Base& arg) { func(arg); }

void otherFunc(Derived& arg) { func(arg); }

int main() {
    Derived d;
    otherFunc(d);
}
```

a. The program runs and prints:
`func(Base) - Base::foo()`

b. The program runs and prints:
`func(Derived) - Derived::foo()`

c. The program runs and prints:
`func(Base) - Derived::foo()`

d. The program runs and prints:
`func(Derived) - Base::foo()`

e. The program fails to compile

f. A runtime error (or undefined behavior)

g. None of the above

Name: Sundeep Khaler NYU Net Id: skk437

10. Given:

```
class Base {
protected:
    void protectedMethod();
};

class Derived : public Base {
public:
    void foo(Derived a) {
        a.protectedMethod();  // line A
    }
};

int main() {
    Derived d;
    d.protectedMethod();      // line B
}
```

protected, even if foo isn't protected

Which of the following is true?:

a. line A will compile
   line B will compile

b. line A will compile
   line B will not compile

c. line A will not compile
   line B will compile

d. line A will not compile
   line B will not compile

Name: Sundeep Kalra NYU Net Id: sk5437

Programming – Blue Book

- Place the answer to the following question 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.
- **#includes** are not required in the blue book.
- Read the question *carefully!*

11. [55 pts] You will define two classes: `School` and `Instructor`. You may assume all of your code is in the the same file with main, i.e. do not create separate header and implementation files.

- Overview:
  - A School
    - has a name and a collection of Instructors. (Don't ask what "kind" of collection. You choose.)
    - can hire Instructors
  - An Instructor
    - has a name
  - All Instructors exist on the **heap**. (Don't forget your instructors. You wouldn't want them to become garbage.)
  - When an Instructor is hired, the School becomes "responsible" for him. (Amusing thought.)
  - Yes, the two classes do refer to each other. You must handle that.
- The `School` class will have the following:
  - **Big 3 / Copy control**
    - As stated, Instructors when hired become part of the School and so their fortunes live and die with the School. If the School goes under (e.g. ceases to exist) the Instructor does, too... If the School is cloned / copied / duplicated, so are the Instructors.
    - **Of the Big 3, only implement the assignment operator.**
      Be sure to provide prototypes for the other two.
  - **Output operator.** Follow the example output below.
  - **Index operator** that takes a name and returns the **address** of the first Instructor in the School with that name. (Yes, there might be other Instructors with the same name.) We will only use the operator to access an Instructor, not to replace him.
  - **hire** method. It is passed the **address** of an Instructor.
    - You may safely assume that the Instructor is on the heap. After all, there isn't any way for you to check.
    - You can hire an Instructor away from another School. Of course the other School better be informed of this (your responsibility).
  - **removeEmp** method.
    - To save time, you **will not implement this method**. You can use it in your code without implementing it.
    - It is passed the address of an Instructor to be removed.
    - It only removes the Instructor from the School's vector.
      It does not modify the Instructor or call any functions to do so.
  - **Any other functions needed by the program.**
- The `Instructor` class will have the following:
  - a constructor that takes the Instructor's name
  - Any other fields or methods necessary.

[Test code and output are on next page]

Name: Simleep Kaler NYU Net Id: sk5437

Sample test function:

```
int main() {
    School nyu("NYU");
    School poly("Poly");
    cout << poly << endl;

    poly.hire(new Instructor("Gallagher"));
    Instructor* sterling = new Instructor("Sterling");
    poly.hire(sterling);  // The School is now responsible for Sterling!
    cout << poly << endl;

    // Poly is still responsible for Gallagher after this.
    Instructor* gallagherPtr = poly["Gallagher"];

    nyu.hire(gallagherPtr);  // Gallagher was hired away from Poly!

    cout << poly << endl;
    cout << nyu << endl;
}
```

Output for sample (your output should match):

```
School: Poly; Instructors: none.
School: Poly; Instructors: Gallagher Sterling.
School: Poly; Instructors: Sterling.
School: NYU; Instructors: Gallagher.
```