Name: Glen Newlk Poly-Id: 0485721

CS1124

Spring 2013

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.

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* **8** pages of this test, including this cover sheet.
5) I didn't put any includes in white book questions.
   Assume any necessary includes were there.
6) Place your answers for questions **1–9** in this document.
7) Place your answer for **the programming question** in your Blue Book.
8) Put your name and ID number on the cover of your Blue Book.
9) Put your name and 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: Glen Neville Poly-Id: 0485721

1. [extra credit] Who created **C?**

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

[Questions 2 – 8 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)` > Both possibly
   b. `thingTwo.operator=(thingOne)`
   c. Either (a) or (b), depending on how the programmer chose to implement the operator.
   d. Neither (a) nor (b) because the operator has to be overridden as a friend
   e. `ostream& Thing::operator=(const Thing& rhs);`
   f. None of the above

this ✓

3. Given:

```
int* data = new int[12];
```

Pick an expression that is equivalent to: `data[5]`

   a. `data*5`
   b. `*data+5`
   c. `&(data+5)`
   d. `data+5&`
   e. `&(data*5)`
   f. `*(data+5)`
   g. `(data+5)&`
   h. `data&+5`
   i. `&data+5`
   j. `(data+5)*`
   k. `data+5`
   l. None of the above

Name: Glen Neville Poly-Id: 0485721

4. What is the output of the following program:

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

int main() {
    Derived der;
    der.foo(42);
}
```

a. `Base::foo(int)`

b. `Derived::foo(double)`

c. The program does not compile

d. The program does not generate any output.

e. None of the above

5. Assume that the class Cat has been defined and that the < operator has been overloaded as a non-member function to compare two Cats. *What other function is* needed in order to allow the lines below to compile and use that overloaded operator.

```
Cat heathcliffe;
if ("Fred" < heathcliffe) { }
```

- *Do not* overload the < operator again.
- *Do not implement* this new function. Just give its prototype.

A cat ^ Need Constructor to [redacted] that takes a string

Prototype: [redacted] Cat(String Name);

Name: Glen Neville Poly-Id: 0485721

6. Given:

```
class MemberA {
public:
  MemberA() {cout << 1;}
};

class MemberB {
public:
  MemberB() {cout << 2;}
};

class Base {
public:
  Base( ) {cout << 3;}
  MemberB member;
};

class Derived : public Base {
public:
  Derived() {cout << 4;}
  MemberA member;
};

int main() {
  Derived der;
}
```

What is the output?

a. 1234	g. 2134	m. 3124	s. 4123
b. 1243	h. 2143	n. 3142	t. 4132
c. 1324	i. 2314	o. 3214	u. 4213
d. 1342	j. 2341	p. 3241	v. 4231
e. 1423	k. 2413	q. 3412	w. 4312
f. 1432	l. 2431	r. 3421	x. None of the above

Name: Glen Neville Poly-Id: 0485721

7. Given:

```
class Dog {
public:
    Dog(int n) { age = n; }
    int age;
};
```

operator bool ( ) const { return age > 10; } would looks like this

What has to be added to the Dog class, so that the following will correctly display "Barker is old" when barker's age is more than 10:

```
int main() {
    int n;
    cin >> n;
    Dog barker(n);
    if(barker) {
        cout << "Barker is old\n";
    }
}
```

Answer: an overload of the Bool operator so that it returns true when age is more than ten

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

```
class Base {
public:
    void method() { }
};
class DerivedA : public Base {
public:
    void method() { cout << "method: A\n"; }
};
class DerivedB : public Base {
public:
    void method() { cout << "method: B\n"; }
};

int main() {
    Base* bp = new DerivedA();
    DerivedB* d2p = bp;
    d2p->method();
}
```

Can't make Dr Base = Base

a. The program compiles and runs, printing "method: A"

b. The program compiles and runs, printing "method: B"

c. The program compiles and runs, printing nothing

d. The program fails to compile because method is not defined in Base.

e. The program fails to compile.

f. None of the above.

Name: Glen Neville Poly-Id: 0485721

9. [10 pts] What is the result of the following?

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

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

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

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

- Derived::Foo()

Should print

- ~~Base::Foo()~~ Derived::Foo()
  Derived::Foo()

- DF FD

~~Base::Foo()~~

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

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

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

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

e. The program fails to compile

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

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

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

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

None of the above

If runs
Derived::Foo()
and prints — Derived::Foo()
func(Base)

Name: Glen Neville Poly-Id: 0485721

Programming – Blue Book

- Place the answers to the following question in your Blue Book.
- **Comments, includes and using namespace** are **not** required in the blue book! However, if you think any comments will help us understand your code, feel free to add them.
- Read the question *carefully!*

10. [55 pts] You will define two classes: `Company` and `Employee`. You do not need separate header and implementation files. You may assume all of your code is in the same file with main. Where appropriate, you may define your methods / functions within the class definition.

- Overview:
  - A Company
    - has a name and a collection of Employees.
    - can hire Employees
  - An Employee
    - has a name
    - can quit his job
  - All employees exist on the heap.
  - When an employee is hired, the company becomes "responsible" for him.
  - Yes, the two classes do refer to each other. You must handle that.
- The `Company` class will have the following:
  - √ **Big 3**.
    - As stated, Employees when hired become part of the Company and so their fortunes live and die with the Company. If the Company goes under the Employee does, too... If the Company is cloned, so are the Employees.
    - **Of the Big 3**, you **only have to <u>implement</u>** the **assignment operator**.
  - √ **Output operator.** Follow the example output below.
  - √ **Index operator** that takes a name and returns the address of the first Employee in the Company with that name. (Yes, there might be other employees with the same name.) We will only use the operator to access an Employee, not to replace him.
  - √ **hire** method. It is passed the address of an Employee.
    - You may safely assume that the Employee is on the heap. There isn't any way for you to check.
    - An Employee may not be hired away from another company. i.e. Your company can only hired unemployed employees.
  - √ **removeEmp** method.
    - To save you time, you **do not have to implement this method**. You can use it in your code without implementing it.
    - It is passed the address of an Employee to be removed.
    - It only removes the Employee from the Company's vector. It does not modify the Employee or call any functions to do so.
  - **Any other functions needed by the program**.
- The `Employee` class will have the following:
  - √ a constructor that takes the Employee's name
  - √ a quit method. It takes <u>no arguments</u>. It is called on the Employee when he wishes to quit.
  - √ Any other methods necessary.

[Continued on next page]

Sample test function:

```
int main() {
  Company comp("hal");
  Employee* fred = new Employee("fred");
  comp.hire(fred);  // The company is now responsible for fred.
  comp.hire(new Employee("mary"));
  Employee* maryPtr = comp["mary"];
  cout << comp << endl;
  maryPtr->quit();
  cout << comp << endl;
}
```

Output for sample:

```
Company: hal; Employees: fred mary.
Company: hal; Employees: fred.
```