Name: Mohan Dhar Poly-Id: 0620191

CS1124

Spring 2014

Exam Two

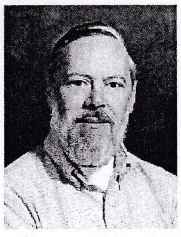
NOTE:

- 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.

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 in white book questions.
   Assume any necessary includes were there.
6) Place your answers for questions **1–11** 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: Mohan Dhal Poly-Id: 0520191

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 – 11 are worth five points each]

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

```
Thing thingOne, thingTwo;
```

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

```
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 a vector of ints called intVec, write a "ranged for" loop, also sometimes known as a "foreach" loop, to double the values of all the elements in the vector.

```
for(int& x : intVec){
    x *= 2;
}
```

4. Given:

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

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

&(*(data+5))

a) `data*5`	d) `*data+5`	g) `&(data+5)`	j) `data+5&`
b) `&(data*5)`	e) `*(data+5)`	h) `(data+5)&`	k) `data&+5`
c) `&data+5`	f) `(data+5)*`	i) `data+5`	

data[5]

&*(data+5)

5. Given:

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

Which of the following will compile?

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

6. 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

Name: Mohan Dhal Poly-Id: 0520191

7. Given:

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

class Base {
public:
  Base( ) {cout << 2;}
  Member member;
};

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

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

123

What is the output?

a. 123
b. 132
c. 213
d. 231
e. 312
f. 321
g. Fails to compile
h. Runtime error (or undefined behavior)

8. Given:

```
class Integer {
public:
    Integer(int n) { val = n; }
private:
    int val;
};
```

What has to be added to the Integer class, so that the following will correctly display "myInt is positive" when the value in myInt is positive:

```
int main() {
    int n;
    cin >> n;
    Integer myInt(n);
    if(myInt) cout << "myInt is positive\n";
}
```

```
operator bool() const {
    return val > 0;
}
```

Name: Mohan Dhar Poly-Id: 0520191

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

```
class Base {};
class Derived : public Base {
public:
    void method() { cout << "method1\n"; }
};
class Derived2 : public Base {
public:
    void method() { cout << "method2\n"; }
};

int main() {
    Base* bp = new Derived();
    Derived2* d2p = bp;
    d2p->method();
}
```

can't assign derived ptr to base ptr.

0

a. The program compiles and runs, printing "method1"

b. The program compiles and runs, printing "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: Mohan Dhar Poly-Id: 0520191

10. 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);
}

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

func(Base) - Derived::foo()

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: Mohan Dhar Poly-Id: 0520191

11. Given:

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

class Derived : public Base {

};

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

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: Mohan Dhar Poly-Id: 052091

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!*

12. [50 pts] One of the most important jobs in our country is the baker. He makes all those treats that we crave and that provide us with the energy to study for (and write) exams! **You will implement a class** to represent this important national resource. (Note, you are only implementing the Baker class.)

    What do bakers do?

    - Make treats! The baker has to create treats on demand, e.g.
      `theBaker.bakes("Twinkie");`
      - Each treat will be created **on the heap** so it can have a long shelf-life.
      - The Treat class has
        - a constructor that takes a string which is that name of the Treat
        - an output operator that displays the name.
        - Anything else Treats have is a trade secret. (Ok, they do support copy control.)
    - Deliver the treats to a company, who will in turn package them and sell them (to us!).
      - This requires that he pass the collection off to the bakery company. After handing the collection over, he is back to having nothing. All those treats, and in fact the container itself that held them now belongs to the company. E.g.
        `aCompany.receives(theBaker.delivers());`

    Just to keep life entertaining, we will want to support copy control for our baker.

    - Deep copy, of course.
    - To keep this exam to a reasonable length, you only have to implement the assignment operator. For the other copy control functions, just provide their prototypes in the class and assume someone else will write them for you.

    And naturally you should provide a reasonable output operator displaying him and his products. Se the sample test code and output on the next page.

    And finally, let's have an equality operator. We will consider two bakers to be "equal" if they currently have the same number of treats.

    So, what do *you* have to implement? **Just the Baker class.**

    - You don't have to worry about the bakery company or even defining the Treat class..
      (Well, worry all you like, but you are not writing those classes.)
    - You can also assume that Treat provides any needed operators or constructors.
    - The baker has:
      - a constructor ✓
      - bakes method ✓
      - delivers method ✓
      - output operator ✓
      - equals operator ✓
      - copy control ✓

Ok, so **how will you represent** all of this? You should be able to work out a good design, but let me "help" you.

- Since the baker hands over the collection of treats that he has baked, he needs to have a *pointer* to the collection. If the baker does not currently have any treats, then he also should not yet have a collection.
- No, don't ask me what sort of collection to use. Use whatever you like. But the baker better be free to bake and store as many treats in your container as needed. We don't know how many that will be.
- And the **collection itself needs to be on the heap**.
- When the baker bakes a treat, if he has no place to put it, either because he just came on the job or because he just delivered his collection to a company, he will first need to get / create another collection. Where? Again, obviously from the heap.
- The treats will later be repackaged and sold and eaten, finally being destroyed in the process. No you don't have to represent all of those steps, but clearly the treats must each exist on the heap so they can have an arbitrarily long shelf-life and can be moved about as needed by all the code that will use your class.

Sample test code:

```
int main() {
    Baker fred("fred");
    cout << fred << endl;
    fred.bakes("Twinkie");
    fred.bakes("Cupcake");
    fred.bakes("Twinkie");
    fred.bakes("Twinkie");
    fred.bakes("Cupcake");
    fred.bakes("Wonderbread");
    cout << fred << endl;

    Baker joe("joe");
    cout << joe << endl;
    joe = fred;
    cout << fred << endl;
    cout << joe << endl;

    // Don't implement Bakery
    Bakery hostess("Hostess");
    hostess.receives(fred.delivers());
    cout << fred << endl;
    cout << joe << endl;
}
```

Resultant output:

```
Baker: fred; No treats :-(
Baker: fred; Twinkie Cupcake Twinkie Twinkie Cupcake Wonderbread.
Baker: joe; No treats :-(
Baker: fred; Twinkie Cupcake Twinkie Twinkie Cupcake Wonderbread.
Baker: fred; Twinkie Cupcake Twinkie Twinkie Cupcake Wonderbread.
Baker: fred; No treats :-(
Baker: fred; Twinkie Cupcake Twinkie Twinkie Cupcake Wonderbread.
```

```
class Baker{
    static const int INITIAL_SIZE = 20;
    friend ostream& operator<<(ostream& os, const Baker& baker);

private:
    string name;
    Treat** treats;
    size_t size;
    size_t capacity;
public:
    Baker(const string& name): name(name), treats(nullptr), size(0), capacity(0)
    {}
    ~Baker()
    Baker(const Baker& rhs);
    void bakes(const string& treatName){
        if(treats == nullptr){
            treats = new Treat*[INITIAL_SIZE];
            capacity = INITIAL_SIZE;
        } else if (size == capacity){
            Treat** newCollection = new Treat*[capacity*2]
            for (size_t i=0; i<size; ++i){
                newCollection[i] = new Treat(*(treats[i]));
                delete treats[i];
            }
            delete [] treats;
            capacity *= 2;
            treats = newCollection;
        }else
        treats[size] = new Treat(treatName);
        ++size;
    }
```

Why?

```
size_t getSize() const { return size; }


Baker& operator=(const Baker& rhs){
    if (this != &rhs){
        for (size_t i=0; i<size; ++i){
            delete treats[i];
        }
        delete [] treats;
        treats = new Treat*[rhs.capacity];
        for (size_t i=0; i<rhs.size; ++i){
            treats[i] = new Treat(*(rhs.treats[i]));
        }
        name = rhs.name;
        size = rhs.size;
        capacity = rhs.capacity;
    }
    return *this
}


Treat** delivers(){
    Treat** newCollection = new Treat*[capacity];
    for(size_t i=0; i<size; ++i){
        newCollection[i] = new Treat(*(treats[i]));
        delete treats[i];
    }
    delete [] treats;
    treats = nullptr;
```

if treats is new!? (-2)

```
    capacity = 0;
    size = 0;
    return newCollection;
  }

};

ostream& operator<<(ostream& os, const Baker& baker){
    os << "Baker: " << baker.name << ": ";
    if (baker.size == 0){
        os << "No treats :-( ";
    } else {
        for (size_t i=0; i< baker.size; ++i){
            if (i = baker.size-1){
                os << *(baker.treats[i]) << ".";
            } else {
                os << *(baker.treats[i]) << " ";
            }
        }
    }
    return os;
}
```

can do that more easily

```
bool operator==(const Baked& lhs, const Baked& rhs){
    return lhs.getSize() == rhs.getSize();
}
```