Name: Urs Evora Poly-Id: 0499140

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

0

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;
```

5

a. `operator=(thingTwo, thingOne)`

[✓ 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]

3. Given:

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

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

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

✓ 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);
}
```

0

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.

0

Prototype: Cat(const string& catName);

Name: Urs Erora Poly-Id: 0499140

✓ 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;
}
```

1

What is the output?

a. 1234

b. 1243

c. 1324

d. 1342

e. 1423

f. 1432

g. 2134

h. 2143

[✓ i. 2314]

j. 2341

k. 2413

l. 2431

m. 3124

n. 3142

o. 3214

p. 3241

q. 3412

r. 3421

s. 4123

t. 4132

u. 4213

v. 4231

w. 4312

x. None of the above

Name: Clrs Evora Poly-Id: 0499140

7. Given:

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

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:

3

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

const

Answer: bool Dog::operator bool() { return (age > 10); }

✓ 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();
    Derived2* d2p = bp;
    d2p->method();
}
```

0

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: Urs Erona Poly-Id: 0499140

✓ 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);
}
```

0

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)
```

j. None of the above

Name: Urs Evora    Poly-Id: 4/15/13

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 implement** 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 no arguments. It is called on the Employee when he wishes to quit.
    - Any other methods necessary.

**[Continued on next page]**

Name: Urs Evora Poly-Id: 0499140

**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.
```

```
class Employee;
class Company {
        public:
        void removeEmp(Employee*& empPtr);
};
```

?

Doesn't work

-1

```
class Employee {
    public:
  Employee(const string& theName):name(theName), myCompany(nullptr){}
    void quit(){
        myCompany -> removeEmp(this);
        myCompany = nullptr;
    }
    bool getHired()const{// returns whether hired or not
          return(myCompany != nullptr);
    }
    void setCompany(Company*~~&~~ theCompany) ~~const~~ {
         myCompany = theCompany;
    }
    string getName() const { return name; }
  private:
    string name;
    Company* myCompany;
};
class Company{
      public:
       Company(const string& theName): name(theName){}
       Company& operator=(const Company& rhs){
         for(Employee*&e : employees){
            delete e;
         }
```

scA?,

-3

```
            employees.clear();
            for(size_t i=0; i< rhs.employees.size(); ++i){
               employees.push_back(new Employee(*(rhs.employees[i]))
            }      myCompany?  -2
            name = rhs.name;
        };
    void  hire( Employee*~~&~~ theEmp){
            if(!(theEmp->getHired)){
                theEmp->set Company(this);
                employees. push_back(theEmp);
            }
          }
                                                            const
    Employee* operator[](const string& theName){
              for(Employee* e : employees){
                  if(e->getName() == theName){
-4                    return e;
                  }
            } return?   }
  private:
    friend ostream& operator<<(ostream& os, const Comp
    string name;
    vector< Employee* > employees;
 };
```

ny rhsD

```
ostream& operator<<(ostream& os, const Company&
         os << "Company: " << rhs.name << ""; Em
         for(Employee* e : rhs.employees){
           os<< e->getName();
         }
         return os;
}
```

format?

-2