11.
```
void fillRoles(ifstream& ifs, vector<Role>& roles){
    Role tempRole;
    string word1;
    string word2;
    while(ifs >> word1 >> word2){
        tempRole.actor = word1;
        tempRole.character = word2;
        roles.push_back(tempRole);
    }
}
void displayRoles(vector<Role>& roles) const {
    for(size_t i=0; i<roles.size(); i++){
        cout << "Name: " << roles[i].actor <<
            ", Character: " << roles[i].character << endl;
    }
}
```

const &

function cannot be const

-2

17

12. class Employee {
public:

```
Employee(const string& newName) : name(newName), boss(nullptr),
                                  staff(NULL) {}
Employee* getBoss() const {
    return boss;
}
void removeEmployee(Employee* removed) {
    for(size_t i=0; i< staff.size(); i++){
        if(staff[i] == removed){
            staff[i] = staff[staff.size()-1];
        } staff.pop_back()
          return
    }
    staff.pop_back();
}
void setBoss(Employee* newBoss){
    boss = newBoss;
}
bool hire(Employee& newStaff) {
    if(&newStaff == this || &newStaff == boss ||
       &newStaff == boss->getBoss()) {
        newStaff.setBoss(this);
        staff.push_back(&newStaff);
        return true;
    }
    else return false;
}
```

−1

−1

You don't need getters & setters for your own class! −1

not in the specs.

Check that new Staff is not already hired

−2

−5

```
void quit(){
    if(boss != nullptr)
        boss->removeEmployee(this);
        boss= nullptr;
}
string getName() const{
    return name;
}
void display() const{
    cout << "Name:" << name << ";Boss:";
    if(boss != nullptr)
        cout << boss->getName();
    else
        cout << "none";
    cout << "; Staff:";
    if(staff.size() > 0)
        for(size_t i=0; i< staff.size(); i++)
            cout << staff[i]->getName();
    else
        cout << "none";
    cout << ".\n";
}
private:
    String name;
    Employee* boss;
    vector<Employee*> staff;
};
```

(no need for setter)

no need for getter

Name: Sundeep Khaler Poly-Id: N17360020 skk5437

Professor: Kletenik

CS1124

Spring 2015

Exam One

NOTE:

1. **There are long programming problem(s) at the end of the test.**
2. A good strategy would be to do all the short questions that you can do *quickly*,
   - then get to the programming problems at the end of the test;
   - and finally go back through the shorter ones if you have time.
3. **DO NOT CHEAT**. (They told me I have to say that.)
4. Write NEATLY. (please)
5. *Do not* tear any pages out of your Blue Book.
6. *Do not* tear any pages from this document. Be sure that you hand in *all 8 pages* of this test, including this cover sheet.
7. Place your answers for questions **1–10** in this document. For multiple choice questions please **circle** the correct answer.
8. Place your answer for **the programming questions 11-12** in your Blue Book.
9. Put your name and ID number on the cover of your Blue Book.
10. Put your name and ID number as indicated on *each* page of this test.
    Please circle your last name. Thank you.
11. If you need "scratch" paper, use your Blue Book but cross out anything you do not want graded.
12. You are not required to write comments for any code in this test.
13. Do not begin until you are instructed to do so.
14. **Good Luck!**

Name: Sundeep Kalur Poly-Id: sk5437

Short Answer

Note: #includes have been omitted to save space. Assume that if they are needed, they are there.

For multiple choice questions, circle only one answer.
Questions 1-10 are all worth 4 pts.

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

0

a. Gallagher
b. Gosling
c. Kildall
d. McCarthy
e. Ritchie

(f.) Stroustrup ✓
g. Thompson
h. van Rossum
i. Wirth
j. Wall

2. Given:

-4

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

works

Which of the following is true? (Choose one!)

p = constant pointer
Cannot change location

(a.) Line A does not compile.
b. *p = 42; // Does not compile
c. int another = 42;
   p = &another; // Does not compile
d. (b) and (c)
   because the Line A above does not compile.
(e.) (b) and (c)
   but line A above does compile.
f. None of the above

3. Given a vector of ints called intVec, use a "ranged for" loop, also sometimes known as a "foreach" loop, to compute the sum of all the elements in the vector.

-2

```
int count = 0;
for (int& x : intVec){
    count += x;
}
```

int x or const int& x

Name: Sundeep Kalra Poly-Id: SK5437

Questions 4 and 5 use the following classes:

```
class FlyingMachine {
public:
    FlyingMachine() {}
    virtual void fly() {cout << "In FlyingMachine fly()"; }
};

class HangGlider : public FlyingMachine {
public:
    virtual void crash() {cout << "HangGlider crashing"; }
    virtual void fly() {cout << "In HangGlider fly()"; }
};
```

-0-

4. Given the above classes, what would be the result of:

```
int main() {
    FlyingMachine* flyingMachinePtr = new HangGlider();
    flyingMachinePtr->crash();
}
```

Base = derived slicing
Base has no crash

a. The program runs and prints: HangGlider crashing

b. The program runs and prints: FlyingMachine crashing

c. The program compiles but has a runtime error

d. Compilation error because there is no HangGlider constructor

✓ (e) Compilation error other than (d).

f. None of the above

5. Given the above classes, what would be the result of:

```
int main() {
    HangGlider hanger;
    FlyingMachine flier;
    flier = hanger;
    flier.fly();
}
```

Base = derived slicing
← virtual method, but not a pointer
only slices off information

-0-

a. The program runs and prints: In HangGlider fly()

✓ (b) The program runs and prints: In FlyingMachine fly()

c. Runtime error.

d. Compilation error.

e. None of the above

Name: Sundeep Kalar Poly-Id: sk5437

6. What would be the result of:

```
class Embedded {
public:
        Embedded() { cout << "1"; }
};
class Base {
public:
        Base() { cout << "2"; }
};
class Derived : public Base {
public:
        Derived() { cout << "3"; }
        Embedded e;
};

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

Base member constructor 2
Der member 1
constructor 3
213

-0

a. 23
b. 32
c. 123
d. 132
e. 213 ✓
f. 231
g. 312
h. 321
i. Does not compile
j. None of the above.

7. Given

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

class Derived : public Base {
public:
        Derived() { }
        void display() { cout << "Derived"; }
};

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

-4

For the classes Base and Derived defined above write the code in main() that will call the Base display method on the object der.

will call Base for der

```
int main() {
        Derived der;
        // Put your code here
        der.Base::display();

}
```

der.Base::display()

Name: Sundeep Kaler Poly-Id: sk5437

8. Given:

```
struct Foo { int mem; };
```

and

```
Foo* p;
```

Then

```
*p.mem
```

is the same as:

a. `(*p).mem`

b. `*(p.mem)`

c. `p->mem`

d. `none of the above`

dereference of mem, not p

9. Given a struct `Thing`:

Write a line of code that creates a `Thing` on the heap and stores its address in an appriately defined pointer variable `ptr`. (Yes you have to define the varible `ptr`. No you don't need to know anything else about `Thing`.)

```
Thing* ptr = new Thing;
```

Write a line of code that frees up the heap space allocated in the above line of code.

```
delete ptr;
```

Name: Sundeep Kaler Poly-Id: sk5437

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

```
class Base {
protected:
    void sayBoo() const { cout << "Boo!"; }
};

class Derived : public Base { };

class DerivedTwo : public Base {
public:

    int booHoo(const Derived& der) { der.sayBoo(); }      // A
    int booHoo(const DerivedTwo& der2) { der2.sayBoo(); } // B

};

int main() {

    Derived der;
    DerivedTwo derTwo, derTwoB;

    derTwo.booHoo(der);      // C
    derTwo.booHoo(derTwoB);  // D

}
```

Cannot call protected (private) members & methods of derived

Reminder: Circle one!

a) Outputs: Boo!Boo!

b) Outputs: Boo!
And then crashes

c) Compiles and runs but there is no output

(d) Fails to compile due to line A

e) Fails to compile due to line B

f) Fails to compile due to line C

g) Fails to compile due to line D

h) None of the above

Name: Sundeep Kaler Poly-Id: sk5437

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.
- **Do not use iterators!!!** (Sorry for the shouting) If you don't what they are, then I expect you won't be using them.

11. [24 pts] Given the type Role:

```
struct Role {
    string actor;
    string character;
};
```

write the two functions,

- **fillRoles**: fills a vector of Roleswith data from a stream. Note the stream has already been opened. The lines of the file each have an actor's name and the name of a role he played. Each name is a single token.
- **displayRoles**: display the information from the vector. Follow the output example below.

Below is an example program in which `fill` and `release` are called from main. Do not modify the definition of Thing.

```
int main() {
    ifstream ifs("roles.txt");
    vector<Thing*> roles;          vector<Role> roles
    fillRoles(ifs, roles);      // Implement this function
    displayRoles(roles);        // Implement this function
}
```

Sample input file:

```
Nimoy   Spock
Shatner Kirk
Kelley  Bones
```

Corresponding output:

```
Name: Nimoy, Character: Spock
Name: Shatner, Character: Kirk
Name: Kelley, Character: Bones
```

Name: Sundeep Kaler Poly-Id: SK5437

12. [40 pts]

In our company anyone is allowed to have a staff. Or not. Having a staff means that you can have people working for you. Lots of people! Yes, even the lowliest intern in the mail room has the possibility of acquiring a staff.

Note that just as you can have a staff, you can also be a member of someone else's staff.

There are some limitations to who can **hire** to be on your staff. You can't hire yourself, of course. You are also not allowed to hire your immediate boss. (While it may seem silly to be allow it, there is no rule against hiring your boss's boss, etc., the company decided that implementing such a rule would be too expensive.) Oh, you also can't hire someone who already is on someone's staff.

Just as you can hire other employees, they are free to **quit** your group. They do this by calling their quit method. Note, they are not quitting the company! They are just not working for *you*.

If someone tries to quit who is not currently a member of anyone's staff, nothing happens. (We might need to send them to the company's shrink, but that's not your program's responsibility.)

BTW, lots of people in the company have the same names. It's a strange place. Clearly you cannot compare two people based on their names.

Below is a test program and the resulting output. Note that if we use your class definition, the output should match, except as regarding the order of an employee's staff, which can be printed in any order.

Implement just the Employee class. No includes required.

Test Program

```
int main() {
   Employee groucho("Groucho");
   Employee harpo("Harpo");
   Employee chico("Chico");
   Employee zeppo("Zeppo");

   //Status
   groucho.display();
   harpo.display();
   chico.display();
   zeppo.display();
   cout << "=====\n";

   groucho.hire(harpo);     // return true
   harpo.hire(groucho);     // return false
   groucho.hire(zeppo);     // return true
   groucho.hire(chico);     // return true

   //Status
   groucho.display();
   harpo.display();
   chico.display();
   zeppo.display();
   cout << "=====\n";

   harpo.quit();
   harpo.hire(groucho);

   //Status
   groucho.display();
   harpo.display();
   chico.display();
   zeppo.display();
   cout << "=====\n";
}
```

Sample Output

```
Name: Groucho; Boss: none; Staff: none.
Name: Harpo; Boss: none; Staff: none.
Name: Chico; Boss: none; Staff: none.
Name: Zeppo; Boss: none; Staff: none.
=====
Name: Groucho; Boss: none; Staff: Harpo Zeppo Chico.
Name: Harpo; Boss: Groucho; Staff: none.
Name: Chico; Boss: Groucho; Staff: none.
Name: Zeppo; Boss: Groucho; Staff: none.
=====
Name: Groucho; Boss: Harpo; Staff: Chico Zeppo.
Name: Harpo; Boss: none; Staff: Groucho.
Name: Chico; Boss: Groucho; Staff: none.
Name: Zeppo; Boss: Groucho; Staff: none.
=====
```

class Employee
Employee(name)
display()
bool hire(Employee)
quit()
vector<Employee*>staff
Employee* boss
remove(Employee)
name
Employee* getBoss()
bool setBoss
getName()