# CS2124 Exam One

## 2018 Spring

Note that I have omitted any #includes or "using namespace std;" statements in all questions, in order to save space and to save your time thinking about them. You may assume that all such statements that are needed are present. And you don't have to write them either!!!

Please, read all questions carefully!

Answering the short-answer questions, in particular, require that you read and *understand* the programs shown. You need to read them *carefully* if you are going to understand them.

If a question asks you to write a class or a function and shows you test code, be sure your class / function works with that test code.

If a question asks you to write a class or a function and shows you output, be sure your class / function generates that output, unless the spec states otherwise.

| Questions | Points |
|---|---|
| 1-6 | 4 |
| 7 | 10 |
| 8 | 10 |
| 9 | 20 |
| 10 | 40 |

Answer questions 1–8 in **this exam book**. For multiple choice questions, circle the correct answer. There should be only one correct answer / question.

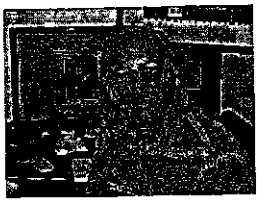
Answer questions 9 and 10 in your **blue book**.

Place you name and id on every page in this book before the end of the exam.

**For mulitple choice questions, circle one answer!**

1. **[Extra credit]** Who created C++?

   a) Gosling
   b) Hopper
   c) Ritchie
   d) Stroustrup



   e) Thompson
   f) van Rosum
   g) Wall
   h) None of the above

2. The expression `p->x` means the same thing as:

   a) `*p.x`
   b) `*(p.x)`
   c) `(*p).x`
   d) all of the above
   e) none of the above

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

```
Thing thingOne, thingTwo;
```

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

```
thingTwo = thingOne;
```

   a) `Thing& Thing::operator=(const Thing& rhs)`
   b) `operator=(thingTwo, thingOne)`
   c) `thingOne.operator=(thingTwo)`
   d) Either (b) or (c), depending on how the programmer chose to implement the operator.
   e) None of the above because it is using the `Thing` copy constructor.
   f) None of the above

4. Given:

```
void foo(int x) {
    const int* p = &x;    // line A
    x = 17;               // line B
    cout << *p << ' ';    // line C
    *p = 28;              // line D
}

int main() {
    int y = 42;
    foo(y);
    cout << y << endl;
}
```

What is the result of compiling and running the above code? (Circle only one answer)

a) The program will have a compilation error at line A

b) The program will have a compilation error at line B

c) The program will have a compilation error at line C

d) The program will have a compilation error at line D

e) The program will have a runtme error (or undefined behavior) at line D.

f) The program will print out: 17 17

g) The program will print out: 17 42

h) The program will print out: 42 17

i) The program will print out: 42 42

j) The program will print out: 17 28

k) The program will print out: 42 28

l) All of the above

m) None of the above.

5. Given the following code:

```
class Dragon {
public:
    Dragon(string val) : payload(val) {}
    void display() { cout << "Dragon payload: " << payload << endl; }
private:
    string payload = "Roadster";
};

class Falcon {
public:
    void display() {
        cout << "Falcon class ship\n";
        fly.display();
    }
private:
    Dragon fly;
};

int main() {
    Falcon heavy;
    heavy.display();
}
```

What is the result of compiling and running the above code?

a) Outputs:
```
Dragon payload:  Roadster
```

b) Outputs:
```
Falcon class ship
```

c) Outputs:
```
Dragon payload: Roadster
Falcon class ship
```

d) Outputs:
```
Falcon class ship
Dragon payload:  Roadster
```

e) Compile time error

f) Run time error (or undefined behavior)

g) None of the above.

6. Using the ranged for (also known as the "foreach"), add 2 to each item in the vector of ints called `intVec`
(No, do not put this in a function.)

```
for(int &num : intVec) { num += 2; }
```

7. Given:

```
class ShuttleCraft {
public:
    ShuttleCraft(string s) : s(s) {}
    // ... possibly other methods
private:
    string s;
    // ... possibly other fields
};

class Orville {
public:
    Orville(string s) : p(new ShuttleCraft(s)) {}
    ~Orville() { delete p; }
private:
    ShuttleCraft* p;
};
```

Implement an appropriate assignment operator, i.e. a deep copy, for the class `Orville`. Write it below. (Oh, yes, the class `ShuttleCraft` supports copy control.)

```
Orville& operator=(const Orville& rhs) {
    string s = rhs->getName();
    delete p;
    p = new ShuttleCraft(s);
    return *this;
}
```

-4

8. Given:

```
vector<int*> vip;
```

In the following three parts modify the contents of the vector defined above.
(And no, you do not need to put your code in functions.)

a) First, fill `vip` with the addresses of 100 ints that you have allocated on the heap. Each one will have the value 28.

```
int x = 0;
while (x < 100) {
    vip.pushback(new int(28));
    x++;
}
```

b) Now, modify those values in the vector by adding the index of the entry to the *value* that was stored on the heap, e.g. add 17 to integer pointed to by vip[17].

```
for (size_t i=0; i < vip.size(); ++i) {
    *(vip[i]) += i;
}
```

c) Finally, free up all of the entries on the heap. Do not leave any dangling pointers.

```
for (int *ip : vip)
    delete ip;
    ip = nullptr;
}
```

-2

**Blue book**

**Answer questions 9 and 10 in your blue book.**

9. Given the type `Thing`:

```
struct Thing {
    vector<int> stuff;
};
```

write the following two functions,

**`fill`**: fills a vector with data from a file stream.

- The lines of the file each
  - start with a string "one", or "three".
  - And then have that many ints on the rest of the line.
  - Example file:
    three 2 4 6
    one 12
    one 18
- The vector holds Things, as defined above. Put the ints that show up on a single line into the vector in a Thing object. There should be one Thing object for each line in the file.
- Note the stream is already open, so you don't have to worry about that.

**`totalStuff`**:

- Passed the vector of Things. Computes and returns a single `int` which is the total of all the ints in all of Things in the vector.
- For the above sample input file the function would return the sum 2+4+6+12+18 (so I think the answer is 42).

Below is an example program in which `fill` and `totalStuff` are called from main.

- Note that neither `fill` nor `totalStuff` are *methods*.
- Do not modify the `Thing` struct.

```
int main() {
    ifstream ifs("stuff.txt");
    vector<Thing> things;       // Note, not Thing pointers
    fill(ifs, things);          // Implement this function
    ifs.close();
    cout << "Total stuff: "
         << totalStuff(things)  // Implement this function
         << endl;
}
```

Answer in the blue book!!!

10. These days the company that CS students all want to work for is *SnackChat*! This start-up knows that everyone likes to snack and to chat. They are not quite sure what their product will be, but that isn't going to stop them from seeking employees and investors!

Part of what makes working there so nice is the flexibility they provide in picking what group you get to work with. It's completely up to you! Not only that, but at the same time you are working as a member of one group, you can also be leading another group! Everyone can be a leader!

So, how does this all work?

- Well, if you want to *join* a group, all you do is say that you want to join the *leader* of that group. For example, `john.join(george)`, would result in john joining the team that george is in charge of. Remember, in this flexible company, george can also be a member of someone *else's* group. Work is never boring! (and there are plenty of snacks!)
- Naturally there are some issues to address. First, if you are already part of one group and then decide to instead join a different group your original will have to be notified and remove you. Again, if we remember john from our earlier example, who joined with george, if he now chooses to blend with mary, i.e. `john.join(mary)`, that call needs to **remove him** from george's group before adding him to mary's.
- By the way, we don't care about the order of the members in a group. After all, they're just all chatting together, anyway! (If that makes it easier for you to write your code, great. If not, don't worry about it.)
- A couple of cases we want to avoid:
  - Joining with yourself would be uninteresting, so we won't allow that to occur.
  - Also, it would be confusing if first george joined with mary and then mary joined with george. If they had meetings, how would they know who was in charge?!? Can't have that, so you can't join with someone who is currently already joined with you. (i.e. you can't join the group of someone who is in the group that you lead.)
  - And of course, we don't want people joining a group that they are already in. That would result in confusing duplication if allowed or annoying paperwork for the human resources department. So, trying to join with the leader of your current group doesn't work.
- Finally, we really should have *some* error checking code. Our `join` method will return `true` if we successfully joined with the leader and `false` otherwise. This way we won't "fail silently".

Below is a sample scenario. **Your job is just to define the `Chip` class**, together with its constructor, a `display` method and the `join` method. (Yes people who work there call themselves Chips. They tried very hard to find a word that connecting snacking and chatting and this was the best they could do. I hope they are more successful with their product.)

Note the output. Your output operator should generate the output *as shown*, except possibly for the order of the Chips in a group. (Remember, we don't care about order.)

And finally, no, this problem does **not involve copy control or the heap**.

And even more finally, do note that people's names are *not* unique!!! Just knowing someone's name won't be very useful.

**Test Code:**

```
int main() {
    Chip moe("Moe");
    Chip larry("Larry");
    Chip curly("Curly");
    Chip curly2("Curly");  // See, names really are not unique.

    moe.join(larry);    // Returns true
    cout << larry << endl;
    curly.join(larry);  // Returns true
    curly2.join(larry); // Returns true. Now we have two Chips named Curly in the group
    cout << larry << endl;
    larry.join(moe);    // Returns false
    moe.join(moe);      // Returns false
    moe.join(larry);    // Returns false
    curly2.join(moe);   // Returns true.  Curly2 has switched allegiance.
    cout << moe << endl;
    cout << larry << endl;
    cout << curly2 << endl;
}
```

**Output:**

```
Name: Larry; Leader: none; Chips: Moe.
Name: Larry; Leader: none; Chips: Moe Curly Curly.
Name: Moe; Leader: Larry; Chips: Curly.
Name: Larry; Leader: none; Chips: Moe Curly.
Name: Curly; Leader: Moe; Chips: none.
```

Write your answer in the Blue Book!

```
void fill(const fstream& ifs, vector<Thing>& things){
    string num;
    ifs >> num;
    if (num == "three"){
        int a, b, c;
        ifs >> a >> b >> c;
        Thing thing;
        thing.stuff.push_back(a);
        thing.stuff.push_back(b);
        thing.stuff.push_back(c);
        things.push_back(thing);
    }else if(num == "one"){
        int a;
        ifs >> a;
        Thing thing;
        thing.stuff.push_back(a);
        things.push_back(thing);
    }
}
int totalStuff(const vector<Thing>& things){
    int count = 0;
    for (const Thing t : things){
        for (int n : t.stuff){
            num += n;
        }
    } return num;}
```

```
class Chip {
public:
  Chip(const string& name) : name(name),
    leader(nullptr), inGroup(false) {}
  bool join(Chip& person) {
    bool check = false;
    for(Chip* chip : group) {
      if(chip == &person.) {
        check = true; // checks if the address of leader in group
      }
    }
  if(this != &person && inGroup == false && check == false)
  { leader = &person;
    person.group.push_back(this);
    inGroup = true;
    return true;
  } else if(this != &person && inGroup == true && check == false){
    int index;
    for(size_t i = 0; i < leader.group.size(); ++i) {
      if(leader.group[i] == this) {
        index = i;
      } break
    }
    leader.group[index] = nullptr;
    leader = &person;
    person.group.push_back(this);
    return true;
  } return false; }
```

Are you in person's group? (-2)

Hole? (-3)

```
void display() const {
    cout << "Name: " << name << ";
private:
    friend ostream& operator<<(ostream& os, const Chip& chip);
    string name;
    Chip* leader;
    bool inGroup;
    vector<Chip*> group;
};

ostream& operator<<(ostream& os, const Chip& chip) {
    os << "Name: " << chip.name << "; Leader: "
    if (chip.leader == nullptr) { os << "none"; }
    else { os << chip.leader->name; }
    os << "; Chips: ";
    if (chip.group.size() == 0) { os << "none"; }
    else { for (Chip* chipP : chip.group) {
Crash on Mac  os << chipP->name << " ";
            }
        }
    os << '\n';
    return os;
}
```

-format (-1)

(-1)