CS1124 Exam Two

2017 Fall

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, requires that you read and *understand* the programs shown.

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	xtra
2-3	4
4-11	5
12	6
13	46

Answer questions 1–12 in the exam book. For multiple choice questions, **circle** the correct answer. There should be only one correct answer / question.

Answer questions 13 in your blue book.

Place your name and id on every page in this book before the end of the exam. You will lose points if you have not done so when we call "time to put your pens / pencils down." Sorry to be punitive, but some students seem to want to get that small advantage over their classmates.

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. Given:

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

pointer

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

3. Using the ranged for (also known as the "foreach"), modify the vector of ints called `intVec` so that each entry becomes twice what it was. i.e. double each int in the vector.
(No, you do not need to put this in a function.)

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

✓

4. Given: **[Not the same question as on exam one!]**

```
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;
}
```

pointer to const int

can't change int

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 runtime 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:

```
class Parent {
public:
    virtual void foo() = 0;
};

class Child : public Parent {
public:
    void foo() { cout << "Parent\n"; }      // Line A
};

class GrandChild : public Child { };

int main() {
    GrandChild gc;                          // Line B
    gc.foo();                               // Line C
}
```

What will happen when we build and run the program?

a) It will fail to compile at line A, because foo is not marked virtual

b) It will fail to compile at line B because GrandChild is an abstract class.

c) It will fail to compile at line C because foo is an abstract method.

d) It will fail to compile for some other reason

[✓ e) It will print out: Parent]

f) It will print out: Child

g) It will compile, but will crash when run.

h) None of the above

Name: ______

6. Given:

```
class Pet {
public:
    virtual void eat() { cout << "Pet::eat\n"; }
};

class Cat : public [redacted] Pet {
public:
    void eat() { cout << "[redacted] Cat::eat\n"; }
};

int main() {
    Pet* petP = new Cat(); ✓
    Cat* catP = petP;
    catP->eat();
}
```

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

a. The program compiles and runs, printing "`Pet::eat`"

~~b.~~ The program compiles and runs, printing "`Cat::eat`"

~~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.

7. What is the result of the following?

```
class Derived;   // Yes, we need this.

class Base {
public:
    virtual void method(Base& arg) {
        cout << "Base::method(Base)\n";
    }
    virtual void method(Derived& arg) {
        cout << "Base::method(Derived)\n";
    }
};

class Derived : public Base {
public:
    void method(Base& arg) {
        cout << "Derived::method(Base)\n";
    }
    void method(Derived& arg) {
        cout << "Derived::method(Derived)\n";
    }
};

void someFunc(Base& arg) {
    arg.method(arg);
}

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

a. The program runs and prints: `Base::method(Base)`

~~b.~~ The program runs and prints: `Base::method(Derived)`

✓ [✓ c.] The program runs and prints: `Derived::method(Base)`

~~d.~~ The program runs and prints: `Derived::method(Derived)`

e. The program fails to compile

f. A runtime error (or undefined behavior)

g. None of the above

Note: Questions 8 – 9 refer to the classes defined <u>below</u>.

```
class FlyingMachine {
public:
     FlyingMachine() {}
     void fly() {cout << ´In FlyingMachine fly()´; }   No virtual
};

class HangGlider : public FlyingMachine {
public:                                         ← no constructor
     virtual void crash() {cout << ´HangGlider crashing´; }
     void fly() { cout << ´In HangGlider fly()´; }  ✗ will not get called
};
```

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

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

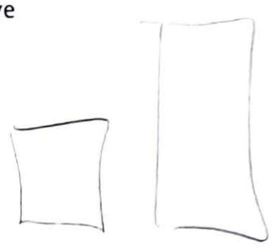
[✓ e.] Compilation error other than (d).

f. None of the above

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

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

base der ✓



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

[✓ b.] The program runs and prints: `In FlyingMachine fly()`

~~c.~~ Runtime error.

~~d.~~ Compilation error because hanger cannot be assigned to flier.

~~e.~~ Compilation error because fly is not virtual.

f. Other compilation error.

g. None of the above

10. Given

```
class Base {
public:
    virtual void display() { cout << "Base: " << n << endl; }
protected:
    int n = 5;
};

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

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

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

a) Outputs:
```
Base: 5
```
[✓ b) Outputs:]
```
Derived: 5
```
c) Fails to compile because the member variable `n` is being set in the class, instead of in the constructor.

d) Fails to compile because the member variable `n` is protected.

e) Runtime error (or undefined behavior)

f) None of the above.

11. Given:

```
class Foo {
public:
    Foo(string s, int n = 0) { str = s; num = n; }
    void display() { cout << str << ':' << num << endl; }
private:
    string str;
    int num;
};

int main() {
    Foo thingOne("abc", 17);
    string s = "def";
    thingOne = s;
    thingOne.display();
}
```

abc : 17

What will be the result of compiling and running the program?

~~a.~~ The program runs and prints:
`abc:0`

✓ [✓ b.] The program runs and prints:
`def:0` ~~The program runs and prints:~~
~~`abc:17`~~

~~c.~~ The program runs and prints:
`def:17`

d. The program fails to compile

e. The program compiles and runs but doesn't print anything

~~f.~~ The program compiles but crashes with no output

g. None of the above.

12. Given:

```
class Member {
public:
    Member()  {cout << 1;}
    ~Member() {cout << 2;}
};

class Base {
public:
    Base()  {cout << 3;}
    ~Base() {cout << 4;}
};

class Derived : public Base {
    Member member;
public:
    Derived()  {cout << 5;}
    ~Derived() {cout << 6;}
};

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

315624

What is the output?

a. 135246
b. 135642
c. 153264
d. 153462
e. 315426
[✓ f. 315624] ✓
g. 351462
h. 351264
i. 513624
j. 513426
k. 531642
l. 531246
m. Fails to compile
n. Runtime error (or undefined behavior)

Blue book

Answer question 13 in your blue book.

private:
string name;
vector<Dragon*> dPV;

13. Define a class `Skyrim` (just that class and nothing else)

- The Skyrim class will inherit from the class `Elder`.
  - `Elder`
    - has a constructor that takes a string representing your registration code.
    - It also has any necessary operators and supports copy control. You should not need to know anything more about the class.
    - NB: you are **not responsible** for defining the Elder class.
- Skyrim has two fields, the player's name and a collection of Dragon pointers. There may be lots of different types of Dragons, but we won't be responsible for defining those derived classes.
- The Dragons will all be on the heap. In fact there is a method that you are **not responsible** for, called add, that creates the Dragons on the heap and inserts their addresses into the collection.
  - NB: you are **not responsible** for defining the Dragon class.
  - Dragons support copy control, along with all necessary operators

You are responsible for defining the `Skyrim` class and providing the following functiononality:

✓ A constructor taking in the player's name and registration code.

✓ Copy control.
  - Naturally, copying should involve making a deep copy. Don't just copy pointers!

✓ An output operator.
  - You may choose the format. Obviously all of the information you have about your Skyrim instance should be included.
  - Don't worry about printing information contained in the Elder class.

✓ An equality operator.
  - Two Skyrim instancess are considered equal if all of the *corresponding* Dragons are equal.
    - I.e. there are the same number of Dragons
    - and each Dragon in one Skyrim matches the Dragon in the same position of the other Skyrim.
    - NB, the Dragons do not have to have the same address to be "equal".

ostream& operator<<(ostream& os, const Skyrim& sky)

Q9 TOTAL = 12pts

Q10 TOTAL = 16pts

& (-3)

9)
```
void fill (ifstream ifs, vector<Thing>& things){
    string numE;
    int num1;
    while (ifs >> numE >> num1){
        Thing temp;
        if( numE == "one"){
            temp.stuff.pushback(num1);
        }
        else if (numE == "two"){
            int num2;
            ifs >> num2;

            temp.stuff.pushback(num1);
            temp.stuff.pushback(num2);
        }
        else{ // if (numE == "three")
            int num2, num3;
            ifs >> num2 >> num3;
            temp.stuff.pushback(num1);
            temp.stuff.pushback (num2);
            temp.stuff.pushback (num3);
        }
        things.pushback (temp);
    }
}
```

9

CONST (-4)

3

```
int totalStuff(vector<Thing>& things){  <- int sum=0;
  for(size_t i=0; i<things.size(); ++i){  (-1)
    for(size_t q=0; q<things[i].stuff.size(); ++q){
        sum += things[i].stuff[q];
    }
  }
  return sum;
}
```

(16)

```
10) class Chip{
public:                  CONST REF
(3) ~~1/2~~  Chip(string aName): name(aName){}

(10)  bool join(Chip& newEmp){   4 & +2
4       if(newEmp != this && ~~*~~myLeader != newEmp &&   (-2)
           *newEmp.myLeader != *~~self~~ THIS){
(-2)      if(~~newEmp~~.myLeader != nullptr){
             for(size_t i=0; i< ~~newEmp~~.myLeader->workers.size(); ++i){
                if(~~newEmp~~.myLeader->*workers[i] == ~~newEmp~~ THIS){
(-8) (X)           newEmp.myLeader->workers[i] = newEmp.myLeader->
                   workers[newEmp.myLeader->workers.size()];
                   newEmp.myLeader->workers.popback();
                   //takes worker in last spot in array and
                   // moves it to removed workers spot
                }
             }
          }
(-3)    NEWEMP.workers.pushback(~~newEmp*~~ THIS);
(-3)    ~~newEmp.~~myLeader = ~~self~~; &newEmp;
        return true;
        }
        if(isLeader == false){ isLeader = true;}

        return false;
      }
```

CONST (-5)

```
2   void display(){
        cout << "Name: " << name << "; Leader: ";
        if (myLeader == nullptr){
            cout << "none";
        }
        else{
            cout << *myLeader; -> name;
        }
        cout << "; Chips: ";
2       if( workers.size() != 0){
            for(size_t i=0; i<workers.size(); ++i){
                cout<< workers[i]->name << ' ';
            }
        }
        else{
            cout << "none";
        }
        cout << '.' << endl;
    }
private:
    string name;
1   bool isLeader = false;
    vector<Chip*> workers;
    Chip* myLeader = nullptr;
}
```