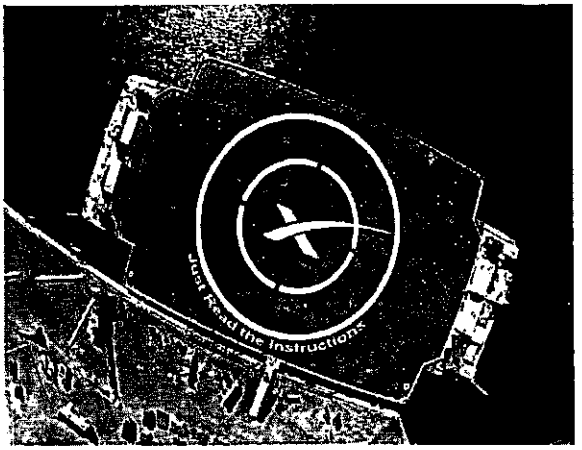
Name: _____ NetID_____

CS2124 Exam Two

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, 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
2-9	5
10	6
11	54

Answer questions 1–10 in the exam book. For multiple choice questions, **circle** the correct answer. There should be only one correct answer / question.

Answer questions 11 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.

Name: _____ NetID_____

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:

```
void foo(int x) {
    int* const p = &x;   // line A
    x = 28;              // line B
    cout << *p << ' ';   // line C
    *p = 42;             // line D
}

int main() {
    int y = 17;
    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 runtime error (or undefined behavior) at line D.
f) The program will print out: 17 17
g) The program will print out: 17 28
h) The program will print out: 17 42
i) The program will print out: 28 17
j) The program will print out: 28 42
k) The program will print out: 42 17
l) All of the above
m) None of the above.

Name: _____ NetID_____

3. Given:

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

class Child : public Parent {
public:
    virtual 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 marked virtual in Child**

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

4. Given:

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

Pick an expression that is equivalent to: `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`	

5. Given:

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

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

int main() {
    Cat* catP = new Cat();
    Pet* petP = catP;
    petP->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.

Name: _ NetID_ _____

6. 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& argA, Base& argB) {
    argA.method(argB);
}

int main() {
    Derived d;
    Base b;
    someFunc(b, 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

Name: NetID_

7. Given

```
class FederationStarship  {
public:
     FederationStarship() {}
     void attack() { cout << "FederationStarship firing phasers"; }
};

class Constitution : public FederationStarship {
public:
     virtual void transport() { cout << "Beam me up!"; }
     void attack() { cout << "Constitution firing phasers"; }
};
```

what would be the result of:

```
int main() {
     FederationStarship* NCC_1701 = new Constitution();
     NCC_1701->transport();
}
```

slicing

a. The program runs and prints:
   `Beam me up!`

b. The program compiles but has a runtime error

c. Compilation error because there is no `Constitution` constructor

d. Compilation error other than (c).

e. None of the above

Name: _ NetID__ ____

8. Given

```
class Base {
public:
    void display() { cout << "Base: " << n << endl; }
protected:
    int n = 42;
};

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

int main() {
    Base* der = new Derived();
    der->display();
}
```

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

a) Outputs:
`Base: 42`

b) Outputs:
`Derived: 42`

c) Fails to compile because the member variable `n` is being set in the protected section, instead of in the constructor.

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

e) Fails to compile because display is *not* marked virtual in `Base`.

f) Runtime error (or undefined behavior)

g) None of the above.

Name: _____ NetID_____

9. Given:

```
class Foo {
public:
    Foo(string s, int n) { 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();
}
```

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
```

c. The program runs and prints:
```
abc:17
```

d. The program runs and prints:
```
def:17
```

e. The program fails to compile

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

g. The program compiles but crashes with no output

h. None of the above.

Name: NetID_

10. Given:

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

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

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

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

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)
o. None of the above

Name: _____ NetID__ _____

Answer question 11 in your blue book.

11. For this question you will define two classes: `Food` and `Chocolate`.

On the back of every **food** product in a store is a list of the ingredients in that food. The list varies in size from one piece of food to another. We are working on a system with such limited memory that using the vector class is not feasible. Please create a `Food` class which stores the ingredient list as a dynamic array of strings. You should also store an integer for the number of ingredients. In our case, the capacity of the dynamic array will always be the same as the number of ingredients, so no need to store that.

Someone else, *not you*, will provide a method `addIngredient` that takes a string and adds it to the array, doing all the necessary work of expanding the array, so *you do not have to worry* about it. You don't even have to write its prototype! Isn't that nice?

Of course, for a class like this, copy control (aka "the Big 3") is very important. Inside your class you should

- provide the implementation for the assignment operator. [Yes, write the code for it.]
- prototypes for the destructor and the copy constructor. You do not have to write the implementations for these two functions.

Be sure to write a default constructor for the class `Food`. A default food contains no ingredients.

Given some `Food` instance `aFood`, we wish to be able to write the line of code below and have it print the truth, depending on whether or not there are any ingredients in `aFood`:

```
if (aFood) { cout << "aFood actually has some ingredients!\n"; }
```

Finally, for the `Food` class, provide: an **output operator**, that prints the ingredients separated by spaces. There can be an extra space at the end if this makes it easier for you. Note that the output operator does not print its own newline. That is left to the code that uses it.

Chocolate is a *type of food* which contains cocoa, along with other ingredients. An important aspect of chocolate is the percentage of cocoa in the chocolate. We will represent chocolate with its own class, `Chocolate` as described below.

Provide a constructor for `Chocolate` that will accept an integer value for the percent of cocoa. The default value will be 50, meaning 50%. Be sure that your constructor adds the cocoa as an ingredient! Oh, for reasons beyond our control, you must store the percent of cocoa as a pointer to an int, not just an int. Your `Chocolate` constructor will allocate the integer value on the heap.

Implement *all* of the Big 3 for `Chocolate`.

You do not need to re-implement the output operator for the `Chocolate` class. You're right we won't see the amount of cocoa, but we'd rather save you time on this exam.

So, if you missed any of the points above, this question involves inheritance, copy control and operator overloading. And yet it is still only about three dozen lines which is about the same as the two questions together from the last exam's blue book.

Name: _____ NetID_____

Some Sample Code

```
int main() {
    Food quiche;
    if (!quiche) {
        cout << "quiche has no ingredients!\n";
    }
    quiche.addIngredient("cheese");
    quiche.addIngredient("milk");
    quiche.addIngredient("flour");
    cout << quiche << endl;
    Food shroomQuiche(quiche);
    shroomQuiche.addIngredient("mushrooms");
    cout << shroomQuiche << endl;
    cout << quiche << endl;
    Chocolate choc;
    Chocolate choc2(30);
    cout << choc << endl;
    cout << choc2 << endl;
}
```

The Output

```
quiche has no ingredients!
cheese milk flour
mushrooms cheese milk flour
cheese milk flour
cocoa
cocoa
```

```
class Food {
  friend ostream& operator<<(ostream& os, const Food& aFood);
public:
  Food(int num=0): num(num) {}
  Food(const Food& rhs);
  virtual ~Food();
  Food& operator=(const Food& rhs){
    if (this != &rhs){
      this->num = rhs.num;
      for(size_t i=0; i < vIngredients; ++i){
        vIngredients[i] = rhs.vIngredients[i];
      }
    }return *this;
  }
  bool operator()(const Food& aFood)const{
    if(aFood.vIngredients.size() >= 1){
      return true;
    }return false;
  }
};
ostream& operator<<(ostream& os, const Food& aFood){
  for(string* ingre : vIngredients){
    os << *ingre << ' ';
  }return os;
}
private:
  int num;
  vector<string*> vIngredients;
```

```
class Chocolate : public Food {
public:
  Chocolate(int amount=50) : percent(&amount),
    Food(1) { addIngredient("cocoa"); }
  Chocolate(const Chocolate& rhs) {
    Food(rhs);
    this->percent = rhs.percent;
  }
  Chocolate& operator=(const Chocolate& rhs) {
    if (this != &rhs) {
      Food(rhs);
      this->percent = rhs.percent;
    } return *this
  }
  ~Chocolate() {
    delete percent;
    percent = nullptr;
  }
private:
  int* percent;
};
```