CS2124 Exam One

2019 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! They may *look* familiar and yet be completely different.

Answering the short-answer questions, in particular, requires 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 provides you with test code, be sure your class / function works with that test code. If the question provides you with sample output, then your answer should match that output.

|  |  |
| --- | --- |
| Questions | Points |
| 1 | xtra |
| 2-8 | 5 |
| 9 | 10 |
| 10 | 16 |
| 11 | 39 |

Answer questions 1–9 in **this exam book**. For multiple choice questions, circle the correct answer. There should be only one correct answer / question.

Answer questions 10 and 11 in your **blue book**.

Place you name and Net 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++?
   1. Gosling
   2. Hopper
   3. Ritchie
   4. Stroustrup
   5. Thompson
   6. van Rosum
   7. Wall
   8. None of the above
2. The expression p->x means the same thing as:
   1. \*p.x
   2. \*(p.x)
   3. all of the above
   4. none of the above
3. Given a class called Thing and the code

Thing thingOne;

Thing thingTwo;

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

thingTwo = thingOne;

* 1. Thing& Thing::operator=(const Thing& rhs)
  2. operator=(thingTwo, thingOne)
  3. thingOne.operator=(thingTwo)
  4. Either (b) or (c), depending on how the programmer chose to implement the operator.
  5. None of the above because it is using the Thing copy constructor.
  6. None of the above

1. 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)

* 1. The program will have a compilation error at line A
  2. The program will have a compilation error at line B
  3. The program will have a compilation error at line C
  4. The program will have a compilation error at line D
  5. The program will have a runtme error (or undefined behavior) at line D.
  6. The program will print out: 17 17
  7. The program will print out: 17 42
  8. The program will print out: 42 17
  9. The program will print out: 42 42
  10. The program will print out: 17 28
  11. The program will print out: 42 28
  12. All of the above
  13. None of the above.

1. Given:

struct CandyStruct {

   CandyStruct () { cout << "sweet\n"; }

};

class CandyClass {

   CandyClass () { cout << "sweet\n"; }

};

int main() {

   CandyStruct candy1;      // line A

   CandyClass candy2;       // line B

}

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

* 1. Runs and prints:  
     sweet  
     sweet
  2. Runs with no output because calling the default constructor
  3. compilation error on line A
  4. compilation error on line B
  5. compilation error on both line A and line B
  6. Other compilation error
  7. Runtime error
  8. None of the above

1. Given a vector of ints, called intVec, use a ranged for (also known as the “foreach”), to increment each item in the vector   
   (No, do not put this in a function.)
2. Given:

int\* foo() {  
 int x = 10;  
 return &x;  
}  
  
int main(){  
 int\* y = foo();  
 cout << \*y << endl;  
}

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

* 1. 10
  2. an address
  3. undefined
  4. compilation error
  5. compiles but crashes
  6. none of the above

1. Given:

class Dragon {

public:

Dragon(string s) : s(s) {}

// ... possibly other methods that you don’t need to know about

private:

string s;

// ... possibly other fields that you don’t need to know about

};

class Falcon {

public:

Falcon(string s) : name(s), p(new Dragon(s)) {}

~Falcon() { delete p; }

private:

Dragon\* p;

string name;

};

Implement an appropriate assignment operator, i.e. a deep copy, for the class Falcon. Write it below.

Yes, the class Dragon supports copy control.

**Your answer here:**

1. Given:

**struct Thing{ int val; };**

* 1. Define a variable datp that points to a dynamic array of 100 Thing pointers.   
     No the pointers will not be assigned / initialized here.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Fill the array with addresses of 100 Things that you allocate on the heap. Each Thing will have its val field hold a value from 1 to 100. i.e. the first Thing will hold 1, the second 2, ...

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Now, modify those values by adding the index of the entry to the *value* that was stored on the heap, e.g. add 17 to val field of the Thing pointed to by dai[17].

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* 1. Finally, free up all of the space you allocated on the heap.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Blue book**

**Answer questions 10 and 11 in your blue book.**

1. Given the type Thing defined as:

struct Thing {

vector<int> stuff;

};

write the following two functions,

**fill**: fills a vector of Things 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
* 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. And we are closing it for you, so you don’t have to worry about that either.

**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;

}

**Write your answer in the Blue Book!**

1. Widg. Inc. makes widgets. Fascinating devices widgets. Widgets are made of other widgets! (Ok, obviously some widgets don’t have “sub-widgets” or it would be widgets all the way down! Somewhere it has to stop.)

Your job will be to model widgets, so you will write a **Widget** class.

**Assembling**

As we assemble our final widget for sale, we have to add on new parts (which are of course other widgets). So the Widget class will have a method **add** which adds in a new part. Of course the widget we are adding may have its own parts, but those belong to *it*.

* To ensure quality assembly, some controls have to be provided when attempting to add a new subpart.
  + Obviously you cannot be a sub-widget of yourself.
  + A widget can’t be a sub-widget of two widgets
  + And you can’t be a sub-widget of something that is one of your own direct sub-widgets.
* Happily there are no concerns about the *order* of the sub-widgets.

**Maintenance**

Unfortunately widgets do break. Luckily all of out designs are resilient in the face of losing a part. The part just has to remove itself. Our designs are amazing in that the order of the widgets does not matter.

Note that if a widget breaks, its sub-widgets are no longer part of it.

You might be concerned about whether a broken widget can be reused. No worries! They quickly regenerate and after breaking can again be reused, so you won’t need to track or update any “broken” state.

**Displaying**

Of course it would be quite some output if we tried to output a widget with all its sub-widgets and all of their sub-widegets and so on. But we do want the ability to display a single widget and know what widget it is part of and what widgets are directly a part of it. So that’s what our output operator will display. (See the sample output for the format.)

Note the output. Your **output operator** should generate the output *as shown*, except possibly for the order of the widgets 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, note that widget’s names are ***not*** unique!!! Comparing widgets’ names won’t be very useful.

**[Example test program and output on the next page]**

**Test Code:**

int main() {int main() {

Widget w1("A");

Widget w2("B");

Widget w3("C");

Widget w4("C");

Widget w5("D");

w1.add(w2); // Returns true

cout << w1 << endl

<< w2 << endl;

cout << "---------\n";

w1.add(w3); // Returns true

w1.add(w4); // Returns true. Now w1 has two parts named C

w4.add(w5);

cout << w1 << endl // w5 does NOT show in w1's list, it is part of w4.

<< w4 << endl;

w1.add(w5); // Returns false, w5 is already part of another widget

w2.add(w1); // Returns false. Can't add own super

w2.add(w2); // Returns false. Can't add yourself

cout << "---------\n";

w4.breaks(); // Oh no! w4 is broken! Its parts are now free.

cout << w1 << endl

<< w4 << endl

<< w5 << endl;

cout << "---------\n";

w1.add(w5); // Returns true, w5 was no longer part of w4

cout << w1 << endl

<< w2 << endl

<< w3 << endl

<< w4 << endl

<< w5 << endl;

}

**Output:**

Name: A; Super: none; Sub-Widgets: B

Name: B; Super: A; Sub-Widgets: none

---------

Name: A; Super: none; Sub-Widgets: B C C

Name: C; Super: A; Sub-Widgets: D

---------

Name: A; Super: none; Sub-Widgets: B C

Name: C; Super: none; Sub-Widgets: none

Name: D; Super: none; Sub-Widgets: none

---------

Name: A; Super: none; Sub-Widgets: B C D

Name: B; Super: A; Sub-Widgets: none

Name: C; Super: A; Sub-Widgets: none

Name: C; Super: none; Sub-Widgets: none

Name: D; Super: A; Sub-Widgets: none

**Write your answer in the Blue Book!**