CS2124 Exam One
2020 Spring

Read this page **thoroughly**.

You are about to take an exam. The purpose of the exam is to fairly and accurately measure your knowledge and skill in this course's material.

Exam rules You must abide by these rules:

- The exam must represent your work alone. You may not misrepresent anyone else's work as your own. You may not submit work of which you are not the sole author.
- The exam is to be completed in isolation. During the exam, you may not communicate with any other person for any reason. This prohibition includes the use of electronic communication, such as email, texting and phone.
- The exam is to be completed without additional resources. You must formulate your answers based only on the contents of your brain. Specifically:
    - Do not use the internet.
    - Do not use other software on your computer, such as IDE, compiler, interpreter or debugger.
    - Do not consult written notes.
    - Do not consult other files on your computer, including other work you've completed for this course.
    - Do not consult books.
- Do not copy or distribute the exam document or your answers at any time. After you submit your answers delete the exam and your answers from your own computer.

How to take the exam Before starting, make sure that you are in a quiet place where you can work without interruption. To avoid distraction, please turn off your phone before starting the exam. You will not need any materials beside your computer and a reliable internet connection. Make sure your computer is sufficiently charged or plugged in to a power source. In addition you may want to have some scrap paper and a pen or pencil.

Download the file **exam.txt** and open it in your text editor. Enter your name and NYU NetID at the beginning of the file. At the section "**Affirmation**" enter the following text exactly as it appears, substituting your name. Without a typed affirmation, your exam will not be graded.

> I, *[your name]*, affirm that I have completed the exam completely on my own without consulting outside resources. I have followed the required rules. I understand that violating any of these rules would represent academic dishonesty.

Enter all of your exam answers into exam.txt at the appropriate places. You will have **80 minutes** to complete the exam. At the end of that period, you will have a **five minute** window to upload your completed exam to NYU Classes. After that time no work will be accepted.

You may upload as many times as you like. The last one before the deadline is what will be graded.

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!!!

You also do not need to write any comments in any of your code.

Please, read all questions carefully! They may *look* familiar and yet be completely different.

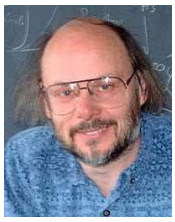
Answering the short-answer questions, in particular, requires that you read and *understand* the programs shown. You need to read them *carefully* if you are going to understand them.

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
2-6	5
7-8	6
9	8
10	16
11	39

For mulitple choice questions, choose just one answer!

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



a. Callahan
b. Gosling
c. Hopper
d. Ritchie
e. Sterling
f. Stroustrup
g. Thompson
h. van Rossum
i. Wall
j. None of the above

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

```
Thing thingOne;
```

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

```
Thing thingTwo = thingOne;
```

a. `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

3. Given:

```
void foo(int& x) {
    int* const 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 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.

4. Given the following code

```
class Thing {
public:
    Thing(int val = 17) : n(val) {}
    void display() { cout << "Thing: " << n << endl; }
private:
    int n;
};

class ThingHolder {
public:
    void display() { cout << "ThingHolder\n"; }
private:
    Thing something;
};

int main() {
    ThingHolder th;
    th.display();
}
```

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

a. Outputs:
```
Thing: 17
```

b. Outputs:
```
ThingHolder
```

c. Outputs:
```
Thing: 17
ThingHolder
```

d. Outputs:
```
ThingHolder
Thing: 17
```

e. Compiles and runs but output is undefined

f. Compile time error because display method is not const

g. Other compile time error

h. Runtime error (or undefined behavior)

i. None of the above

5. Given:

```
int* foo() {
    int x = 17;
    return &x;
}

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

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

a. 17 is displayed
b. an address is displayed
c. compilation error
d. compiles but crashes
e. undefined behavior
f. none of the above

6. Given a vector that was defined as:

```
vector<int> things;
```

Later in the program you want to use a ranged for (also known as the “foreach”), to double the value of each item in the vector things. Write that loop:

7. What is the output of the following program?

```
int main() {
    int x = 6;
    int* arr = new int[8];
    for (int i = 0; i < 8; ++i) {
        arr[i] = i*i;
    }
    int* p = arr + 1;
    int* q = p + x;
    cout << "A: " << *q << endl;
    *p = x;
    ++p;
    cout << "B: " << *p << endl;
}
```

Output:

A:

B:

8. Given:

```
class Dragon {
public:
    Dragon(string s) : s(s) {}
    // ... possibly other methods

private:
    string s;
    // ... possibly other fields
};

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. And yes, the class `Dragon` supports copy control.

9. Given:

```
struct Thing{ int val; };
```

a) Define a variable **`data`** that points to a dynamic **array** of 100 `Thing` pointers.
You allocate the array here. The pointers will be assigned values in the next part.

---

b) Fill the array from part (a) with addresses of 100 `Thing`s 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, ...

---

c) Now, modify those values by adding the index of the entry to the *value* that was stored in the Thing, e.g. add 17 to the `val` field of the Thing pointed to by `data[17]`.

---

d) Finally, free up all of the space that you allocated on the heap. Do not leave any dangling pointers.

---

10. 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.

Each Thing is represented by a single line in the file.
The first item in the line is the number of ints that the Thing will hold.
The rest of the line has that many ints.
Example file:

```
3 2 6 4
2 18 12
```

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 a 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+6+4+18+12 (so I think the answer is 42).

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

Note that neither of the functions `fill` or `totalStuff` are *methods*.

Do not modify the `Thing` struct.

```
int main() {
    ifstream ifs("things.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;
}
```

11. The latest big startup is Itsy. They connect people who are either looking to hire other people or else to get hired. It's important for Itsy's business model to keep track of all the bits of business they get, so for every client (known as a **Bit**) they track who the Bit is currently working for, i.e his boss, and which Bits he has hired, his team. *Everyone* who hires and everyone who is hired is a Bit, so they all form a bit of a community.

    What do you need to do?

    - Define a class **Bit** to represent a member of the community.
        - This is the only class you are defining.
        - Bits do have names, by the way, but of course names are **not** unique.
    - Track who each Bit is currently employing. A Bit may have many Bits working for them.
    - Track who each Bit is currently working for. There can be only one (at a time)!
    - Allow a Bit to hire another Bit with a **hire** method.
    - Allow a Bit to quit with a **quit** method.
    - Enforce a few rules:
        - When hiring, you cannot hire someone who:
            - already has a job
            - you work for
            - and of course you can't hire yourself.
        - If an attempt to hire fails, don't fail silently! (You know what that means.)
        - Bits who employ other Bits don't particularly care what order the Bits in their team are kept in, so when one quits, it should only be an O(1), i.e. constant time, operation.
        - Again, do not fail silently if it is not possible for the Bit to quit. How would that happen? If he doesn't have a job?

    Note the output. Your **output operator** should generate the output *as shown*, except possibly for the order of the Bits 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, again note that people's names are ***not*** unique!!! Just knowing someone's name won't be very useful.

[Example test program and output on the next page]

Test Code:

```
int main() {
    Bit moe("Moe");
    Bit larry("Larry");
    Bit curly("Curly");
    Bit curly2("Curly");

    larry.hire(moe);        // Returns true
    cout << larry << endl;
    larry.hire(curly);      // Returns true
    larry.hire(curly2);     // Returns true. Now we have two chips named
                            // Curly in the team
    cout << larry << endl;
    moe.hire(larry);        // Returns false.  Can't hire own boss
    moe.hire(moe);          // Returns false.  Can't hire self
    moe.quit();             // Returns true.
    cout << moe << endl;
    moe.hire(larry);        // Returns true.
    cout << moe << endl;
    curly2.hire(moe);       // Returns true.  We are allowed to have a cycle
    cout << moe << endl;
    cout << larry << endl;
    cout << curly2 << endl;
}
```

Output:

```
Name: Larry; Boss: none; Bits: Moe.
Name: Larry; Boss: none; Bits: Moe Curly Curly.
Name: Moe; Boss: none; Bits: none.
Name: Moe; Boss: none; Bits: Larry.
Name: Moe; Boss: Curly; Bits: Larry.
Name: Larry; Boss: Moe; Bits: Curly Curly.
Name: Curly; Boss: Larry; Bits: Moe.
```