CS2124 Final Exam
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 **final_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, you will **lose five points**.
Do this before you answer any questions in the exam!

> 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 final_exam.txt at the appropriate places. You will have **120 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 [re]upload up to 10 times. Do not overwrite the previous submissions!.

CS2124

Spring 2020

Final Exam Late Edition

NOTE:

- There are LONGER problems at the end of the test that are worth more points than the shorter ones.
- A good strategy would be to do all the short questions that you can do quickly,
    - then get to the LONGER problems at the end of the test;
    - and finally go back through the shorter ones.

1) Put your name and ID number as indicated in the file **final_exam.txt.**

2) You are not required to write comments for any code in this test.

3) Note that I have omitted `#include` and `using namespace std;` statements in all questions. You may assume that all such statements that are needed are present.
And you don't have to write any either!!!

4) Do not use auto anywhere on this test (unless the question says to use it)

Questions	Points
1	Xtra credit
2-10	5
11	6
12	12
13	15
14	22

Multiple Choice

There is only one correct choice for each question.

1. **[Extra Credit] Who created Python?**

   a. Gallagher
   b. Gosling
   c. Katz
   d. Liskov
   e. Ritchie

   f. Stroustrup
   g. van Rossum
   h. Wirth
   i. Wall
   j. None of the above

2. Given
   - a class Thing with an output operator
   - and the following function definition:

```
void func(const Thing& something) {
    // definition for the variable p goes here...

    *p = 17;
}
```

   Which of the following definitions for the local variable `p` will allow the function `func` to successfully compile?

   a. `Thing* p = something;`
   b. `Thing* p = &something;`
   c. `Thing* p = *something;`
   d. `const Thing* p = something;`
   e. `const Thing* p = &something;`
   f. `const Thing* p = *something;`
   g. `Thing* const p = something;`
   h. `Thing* const p = &something;`
   i. `Thing* const p = *something;`
   j. (a) or (d)
   k. (b) or (e)
   l. (c) or (f)
   m. (a) or (g)
   n. (b) or (h)
   o. (c) or (i)
   p. None of the above

3. What is the output of the following program:

```
class Base {
public:
    virtual void foo(Base b) { cout << "Base::foo(Base)\n"; }
};

class Derived: public Base {
public:
    void foo(Derived d) { cout << "Derived::foo(Derived)\n"; }
};

int main() {
    Derived der;
    Base base = der;      // line A
    base.foo(der);        // line B
    der.foo(der);         // line C
}
```

a. `Base::foo(Base)`
   `Base::foo(Base)`

b. `Derived::foo(Derived)`
   `Derived::foo(Derived)`

c. `Base::foo(Base)`
   `Derived::foo(Derived)`

d. `Derived::foo(Derived)`
   `Base::foo(Base)`

e. The program fails to compile at line A

f. The program fails to compile at line B

g. The program fails to compile at line C

h. The program fails to compile for some other reason

i. The program compiles but does not generate any output when it runs.

j. The program crashes when run.

k. None of the above

4. Given the following code:

```
class B {
public:
    B(int val = 42) : n(val) {}
    virtual void foo() const { cout << "B"; }
private:
    int n;
};

class A {
public:
    void foo() { cout << "A"; }
private:
    B someB;
};

int main() {
    A a;
    a.foo();
}
```

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

a. Outputs:
   B
b. Outputs:
   A
c. Outputs:
   BA
d. Outputs:
   AB
f. Compile time error because method foo in class A does not override method foo in class B.
g. Other compile time error.
h. Runtime error (or undefined behavior)
i. None of the above

5. What is the result of compiling and running the following code?

```
int main() {
    vector<int> v { 6, 42, 28, 17, 16, 49 };
    cout << *find_if(v.begin() + 1, v.end(),
                     [] (int val) { return val % 2 == 0; })
         << endl;
}
```

a. 6
b. 42
c. 28
d. 16
e. 6, 42, 28
f. 6, 42, 28, 16
g. Fails to compile
h. Compiles but crashes when run or undefined behavior
i. None of the above

6. What is the output of the following program:

```
class Pet {
public:
    Pet(string name) : name(name) {}
    void display() { cout << name << ' '; }
protected:
    void setName(string name) { this->name = name; }
private:
    string name;
};

class Cat : public Pet {
public:
    Cat(string name) : Pet(name) {}
};

class Dog : public Pet {
public:
    Dog() : Pet("ruff") {}
    void setCatName(Cat& rhs) {
        rhs.setName("Mehitabel");    // A
    }
    void setDogName(Dog& rhs) {
        rhs.setName("Fido");         // B
    }
};

int main() {
    Cat felix("Felix");
    Dog fido;
    Dog rover;

    fido.setCatName(felix);
    felix.display();
    fido.setDogName(fido);
    fido.display();
}
```

a. The program will output: “Mehitabel Fido “.

b. The program will not compile due to an error on line A

c. The program will not compile due to an error on line B

d. The program will not compile due to errors on both lines A and B

e. The program will not compile even though there is no error on line A or B.

f. None of the above.

7. What is the result of the following?

```
class Base {
public:
    virtual void foo() { cout << " - Base::foo()\n"; }
};

class Derived : public Base {
public:
    virtual void foo() { cout << " - Derived::foo()\n"; }
};

void func(Base& arg) {
    cout << "func(Base)";
     arg.foo();
}

void func(Derived& arg) {
    cout << "func(Derived)";
    arg.foo();
}

void otherFunc(Base& arg) {
    func(arg);
}

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

a. The program runs and prints:
`func(Base) - Base::foo()`

b. The program runs and prints:
`func(Base) - Derived::foo()`

c. The program runs and prints:
`func(Derived) -  Derived::foo()`

d. The program runs and prints:
`func(Derived) - Base::foo()`

e. The program fails to compile

f. A runtime error (or undefined behavior)

g. None of the above

8. What is the result of the following?

```
class P1 {
public:
      void whoamI() { cout << "P1"; }
};

class P2 {
public:
      void whoamI() { cout << "P2"; }
};

class Child : public P2, public P1 {
    // Inherits from both P2 and P1
};

int main() {
      Child c;
      c.whoamI();
}
```

a. The program will display just `"P1"`

b. The program will display just `"P2"`

c. The program will display `"P1P2"`

d. The program will display `"P2P1"`

e. The program will not compile.

f. The program will compile but will crash when run.

g. None of the above

Short Answer

9. Write a **function template** that will *act like* the **generic** algorithm **`count_if`**, which returns how *many* times an item satisfying the predicate occurs in a **half-open range**.

   For **example**, given the definition:

   ```
   int arr[] = { 2, 5, 2, 4, 5, 7, 2 };
   ```

   The following code

   ```
   count(arr, arr+8, isEven)  // isEven returns true if and only if the argument is even
   ```

   would return the value 4 because there are four even numbers in the range. Because you are writing a function ***template***, your code will also work for an STL list of integers or vector of integers...

10. Given an **STL** vector of doubles called **`myList`**,
    - Using iterators write a loop to compute the sum of the elements in the list
    - display the sum

      You **do not** have to write a function or even define the list, just *use* **`myList`**.
      Do not use `auto`.
      Do not use a ranged for.

```
9.
template <typename T, typename U>
int count(T start, T stop, U predicate)
{
   int result = 0;
   for (T p = start; p != stop; ++p)
   {
      if (predicate(*p))
      {
         ++result;
      }
   }
   return result;
}
```

```
10.
int main()
{
   vector<double> myList = {1, 2, 3, 4, 5};
   vector<double>::iterator ptr;
   double sum = 0;
   for (ptr = myList.begin(); ptr !=
myList.end(); ++ptr) {
      sum += *ptr;
   }
   cout << sum;
}
```

You do not need to comment your code.

NOTE: **<u>For all programming questions</u>**, don't waste time doing things we won't grade.

- **<u>Do not</u>** write *any* test code, main function, includes, include guards or using namespace statements.
- **<u>Do not</u>** use separate compilation, i.e. you do not have to provide separate header and implementation files nor do you have to define any methods outside of their classes.

11. ## <u>Using Recursion!!!</u>

write a *recursive* function to do **<u>one</u>** of the following.
If you implement both, **<u>only the first</u>** will count.
Do <u>not</u> implement any additional functions.
Do not pass any parameters other than those specified.

a. Write a *recursive* function **`baseThree`** that is passed a single non-negative integer, **`val`**. Your function will display the number in base three.

<u>Or</u>

b. Given:

```
struct TernNode { int data;  TernNode *left, *mid, *right; }; // a ternary tree
```

Write a *recursive* function **`treeMin`**, that is passed a pointer to the root TernNode and will return the *minimum* of all the values in the tree.

If your function is passed an empty tree, then it should throw the exception **`invalid_argument`**.

```
int convert(int number)
{
   if (number == 0)
      return number;
   return (number % 3) + 10 *
convert(number / 3);
}

void baseThree(int val) {
if (val > 0) {
  baseThree(val / 3);
  cout << val % 3;}
else {
return;
}
}
```

```
int minTree(node *root)
{
   if (!root)
   {
      throw std::invalid_argument("received negative value")
   }
   int rootData = root->data;

}
```

12. Task: Given a class for singly linked lists, called **SLL**, you will add a nested Iterator class.

```
class SLL {  // SLL is short for Singly Linked List
    struct Node {
        Node(int data = 0, Node* next = nullptr) : data(data), next(next) {}
        int data;
        Node* next;
    };
public:
    SLL() { headPtr = nullptr; }

    // adds the data to the head of the Singly Linked List
    void addFront(int data);

    // begin and end methods correctly defined. You don't define them

private:
    // This is the only field in SLL. In an empty list headPtr is null.
    // The last Node in an SLL has a next pointer that holds null.
    Node* headPtr;
};
```

```
class Iterator
  {
    public:
      //define constructors for this class
      Iterator():
  current (headPtr) { }
    Iterator(const Node* node) :
    current (node) { }
      Iterator& operator=(Node* node)
      {
        this->current = node;
        return *this;
      }
      Iterator& operator++()
      {
        if (current)
          current = current->next;
        return *this;
      }
      Iterator operator++(int)
      {
        Iterator iterator = *this;
        ++*this;
        return iterator;
      }
      bool operator!=(const Iterator& iterator)
      {
        return current != iterator.current;
      }
      int operator*()
      {
        return current->data;
      }
    private:
      const Node* current;
  };
```

Our goal is for the following function to work properly, i.e. to double every `int` in the `SLL v`.

```
void doubleValues(SLL& v) {
    for (int& val : v) {
        val *= 2;
    }
}
```

The SLL class will support the methods `begin()` and `end()`, returning appropriate instances of your `Iterator` class. The `begin()` method will initialize its `Iterator` with a pointer to the first Node of the singly linked list and the `end()` method will initialize its `Iterator` with a null pointer.

But this is not your job!

It is up to *you* to just define your `Iterator` class appropriately with *any* functions or operators that are needed. Note that you do *not* need to know anything more about the `SLL` class, nor do you need access to any of its methods or fields.

```
bool removeLastValue(Node* root, int val)
{
   Node* holder;
   while (root && root->next) {
      if (root->next->value == val) {
         holder = root->next;
      }
      root = root->next;
   }
   if (holder)
   {
      root->next = holder->next;
      delete root;
   }
}
```

13. Write a function **`removeLastValue`** that is passed a singly linked list and a target value. It will not return true if anything was removed and false otherwise.

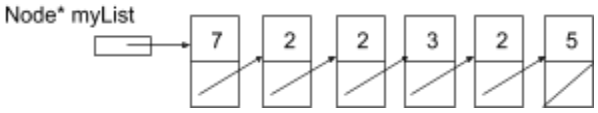
The function will *remove* only the last node in the list that contains that value, leaving all other nodes in order.

By *linked list* we mean a Node pointer, where the type Node is defined as usual by:

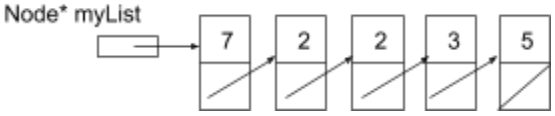
```
struct Node {
    Node(int data = 0, Node* next = nullptr) : data(data), next(next) {}
    int data;
    Node* next;
};
```

[This is ***not*** an instance of the class used in question 12.]

For example, if we start with the following list (note that we are showing a Node* myList):



And then make the call **`removeLastValue(myList, 2)`**, our list will be modified to be:



Note:

- **No Nodes are being created.**
- **The return type for `removeValue` is bool**
- **Feel free to use recursion *or not*, as you like.**

14. You are to write classes to model a theatre company (using inheritance)

The company hires performers and puts on shows.

This theatre currently hires only two kinds of performers: actors and mimes, but this may change in the future – they might start hiring other kinds of performers.

We will model the company's hiring o a performer by writing a method in the class **Company** that adds the performer to the company's group of performers. Once a performer has been hired they never leave the company (so you don't have to write a method to model them leaving). Needless to say, you should **not** be hiring someone who is already employed, by you or anyone else.

We will model the theatre company's putting on a show by writing a method called **showtime** in the class **Company**. The way the **showtime** method does this is by calling the **perform** method in each performer that the company has hired. The performers will perform in the order that they were hired.

All performers have names, implemented as a string.

All performers perform, of course, but actors and mimes (and others) perform differently:

Actors are given speaking roles.
An actor performs by stating their name and then reading their script.
We model this by having their **perform** method write their name and the script on the screen:
"I am Jane. Woe is me!"
where Jane is the actor's name and Woe is me! is Jane's script.

Store the actor's script in a string.
Actors are given a script *when they are created*.

Mimes do not speak. At all. Not even to identify themselves.
They do **not have scripts**. A mime's perform method simply displays "(silence)" on the screen.
.
You must provide the following classes:

- **`Company`**
- **`Performer`**
- **`Actor`**
- **`Mime`**

In the future the company might decide to hire other kinds of performers. Your code should allow any future kinds of performers to be hired and to be part of putting on a show. Be sure to guarantee that any future performer class that might be created ***must*** provide its own implementation of the perform method.

And no, this has nothing to do with templates or copy control.

[This question continued on next page.]

Below is an example of using your classes. Be sure that your class definitions will allow this test code to work.

```
// Your class definitions would go here but of course
// they are in the final_exam.txt file

int main() {
    Company troupe;
    Company otherTroupe;
    Actor frog("Froggie", "Hiya, kids!");
    Mime red("Red Skelton");
    Actor bogie("Humphrey Bogart", "Play it again.");
    troupe.hire(frog);
    otherTroupe.hire(frog); // Nothing happens. Froggie was already hired.
    troupe.hire(red);
    troupe.hire(bogie);
    troupe.showtime();
}
```

NOTE: At `showtime()`, the user sees:

```
I am Froggie. Hiya, Kids!
(silence)
I am Humphrey Bogart. Play it again.
```