Full Name: Utsav Shah

Student ID#: 1002158824

CSE 1325 OBJECT-ORIENTED PROGRAMMING

Exam #3 «---» 9 001 1 «---» Exam #3

Instructions

1. Students are allowed pencils, erasers, and beverage only.
2. All books, bags, backpacks, phones, **smart watches**, and other electronics, etc. must be placed along the walls. **Silence all notifications.**
3. PRINT your name and student ID at the top of this page **and every coding sheet**, and verify that you have all pages.
4. **Read every question completely before you start to answer it.** If you have a question, please raise your hand. You may or may not get an answer, but it won't hurt to ask.
5. If you leave the room, you may not return.
6. You are required to SIGN and ABIDE BY the following Honor Pledge for each exam this semester.

NOTE: The number of questions in each section, and the topic of Free Response questions, may vary on the actual Final Exam.

Honor Pledge

On my honor, I pledge that I will not attempt to communicate with another student, view another student's work, or view any unauthorized notes or electronic devices during this exam. I understand that the professor and the CSE 1325 Course Curriculum Committee have zero tolerance for cheating of any kind, and that any violation of this pledge or the University honor code will result in an automatic grade of zero for the semester and referral to the Office of Student Conduct for scholastic dishonesty.

Student Signature: Utsav

WARNING: Questions are on the BACK of this page!

Vocabulary

Write the word or phrase from the Words list below to the left of the definition that it best matches. Each word or phrase is used at most once, but some will not be used. {10 at 2 points each}

Vocabulary

Word	Definition
1 method	A function that manipulates data in a class
2 declaration	A statement that introduces a name with an associated type into a scope
3 field	A class member variable
4 shadowing	A variable declared in a narrower scope than that of a variable of the same name declared in a broader scope
5 operator overloading	Providing a user-defined meaning to a pre-defined operatorfor a user-defined type
6 container	An object that stores and manages other objects
7 override	A subclass replacing its superclass' implementation of a virtual method
8 constructor	A special class member that creates and initializes an object from the class
9 abstract method	A method declared with no implementation
10 encapsulation	Bundling data and code into a restricted container

Word List

Abstract Class	Abstract Method	Abstraction	Algorithm	Class
Constructor	Container	Declaration	Definition	Destructor
Encapsulation	Exception	Field	Friend	Inheritance
Invariant	Iterator	Method	Multiple Inheritance	Namespace
Object	Operator	Operator Overloading	Override	Polymorphism
Shadowing	Standard Template Library	Subclass	Superclass	

20

Multiple Choice

Read the full question and every possible answer. Choose the one best answer for each question and write the corresponding letter in the blank next to the number. {15 at 2 points each}

1. ___ **When no more data can be read from a C++ input stream, its state becomes**

   A. empty

   B. end

   C. bad

   D. eof

2. ___ **To call a method polymorphically in C++,**

   A. The superclass method must be marked `static` and the subclass method must be marked `override`

   B. The class must inherit from at least two superclasses that are both marked `virtual`

   C. The superclass method must be marked `virtual` and the call must be via a pointer or reference

   D. The superclass must implement `virtual` inheritance and the call must be via a `const`

3. ___ **A C++ stream will evaluate as FALSE unless the stream is in state**

   A. good

   B. eof

   C. bad

   D. All of the above

4. ___ **In which instances would a copy constructor be called by C++?**

   A. Pass-by-value method parameters and returns

   B. Dereferencing a pointer to an object

   C. Pass-by-reference method parameters and returns

   D. Only when explicitly invoked by the programmer

5. ___ **To override the `+` operator in C++ class Complex,**

   A. Override method `+(const Complex& c1, const Complex& c2)`

   B. Write function `operator_add(const Complex& c)`

   C. Override method `operator+(const Complex& c)`

   D. C++, like Java, does not support operator overloading

8

6. A **C++ inheritance is different from Java inheritance in that**

A. C++ destructors may have parameters, but Java destructors never have parameters

B. C++ constructors inherit, but Java constructors do not inherit

C. Java interfaces support multiple inheritance, but C++ interfaces only support single inheritance

D. C++ supports multiple inheritance of classes, but Java supports only single inheritance of classes

7. A **The two types of iterators in C++ are the basic iterator and the**

A. Const interator

B. Virtual iterator

C. List iterator

D. Reverse iterator

8. C **Which is TRUE about enum classes in C++?**

A. An enum class may include constructors

B. An enum class may include methods

C. An enum class cannot be compared to an integer

D. An enum class is identical to a C enum

9. B **In C++, the `std::sort` function accepts two iterators rather than a container (like `std::vector`) because**

A. Collections cannot be passed as parameters to a function

B. Most containers are only accessible indirectly through iterators

C. Collections already have a sort *method*, so the function would be redundant

D. The iterators allow directly sorting any subset of the container

10. C **In C++, a `std::map` by default is**

A. Sorted by value

B. Sorted by hash code

C. Sorted by key

D. Unsorted

11. A **Most STL containers use `v[index] = value;` to overwrite a value, but `std::set` has no index. How would you overwrite a value in a `std::set`?**

A. `v.replace(old_value, value);`

B. `v.overwrite(old_value, value);`

C. `v[v.find(old_value)] = value;`

D. `v.insert(value);`

12. C **Given superclass `Truck` and subclass `F150`, what is the surprising result of the C++ upcast `Truck t = F150{};`?**

A. It will not compile without an explicit upcast operator

B. The `F150` object will be allocated on the heap, not the stack

C. It will segfault

D. Variable `t` will contain an instance of `Truck`, not `F150`

13. A **In C++, a package-private method**

A. may be called only by methods in the same class

B. may be called only by methods in the same class or its friends

C. may be called by any object in the system

D. does not exist

14. A **The "Rule of 3" states**

A. Each method in a "version 1.0" program contains an average of 3 bugs

B. Each feature of your program should consist of at least 3 substantial commits to your git repository

C. If 3 or more developers work on a program, they should use version control to coordinate their work

D. If any of the destructor, copy constructor, or copy assignment operator are needed, all 3 are needed

15. A **A difference between C++ `std::string` and Java `String` is**

A. std::string is mutable (can change value), but String is immutable (cannot change value once instanced)

B. std::string is just an alias for char*, while String is a true class

C. std::string supports Unicode characters, but String only supports 8-bit ASCII characters

D. std::string must be zero-terminated (end with a null char), while String does not

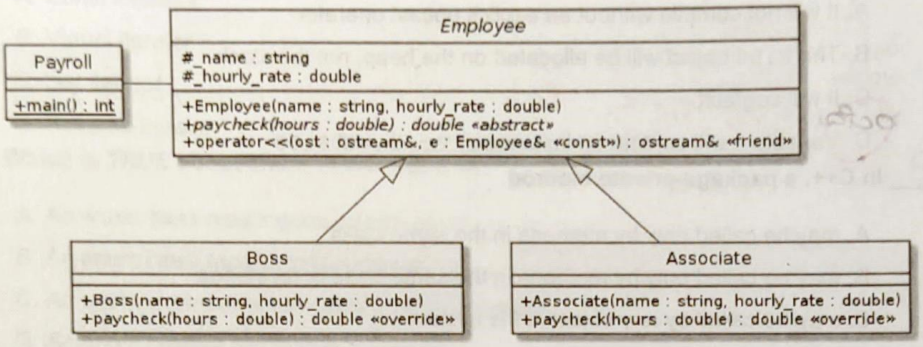
10

Free Response

Provide clear, concise answers to each question. Write only the code that is requested. You will NOT write an entire application! You need NOT copy any code provided to you - just write the additional code specified. You need NOT write `#include` statements - assume you have what you need. You will write a .h guard only once (question 2.b) - skip them on all other .h files to save time.

While multiple questions may relate to a given application or class diagram, **each question is fully independent and may be solved as a stand-alone problem.** Thus, if you aren't able to solve a question, skip it until the end and move on to the next.

1. (polymorphism, abstract, operator overloading, file I/O) Consider the following class diagram for a C++ application. (You will NOT write the entire application!)



Class Employee represents someone who works for your company. Associate employees are paid their hourly rate for each of the first 40 hours worked each week and 1½ x their hourly rate for each additional hour. Boss employees are paid 40 x their hourly rate regardless of the number of hours worked. `operator<<` is a friend function that overloads the << operator for these classes, printing their name and hourly rate.

a. {3 points} In file Employee.h, write the guard, class declaration, and the protected section with its two fields. Do NOT write the rest of the class, except where requested below.

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
using namespace std;
class Employee {
    protected:
        std::string _name;
        double _hourly_rate; };
#endif
```

-1 use std::

2

b. {2 points} In file Employee.h, write ONLY the declaration for abstract method `paycheck`.

```
public:
virtual double paycheck(double hours) =0;
```

2

c. {2 points} In file employee.cpp, write the implementation of the constructor for class Employee. Use an init list to specify construction of each field to copy the corresponding parameter. If the `hourly_rate` parameter is less than 0, throw a runtime error with your choice of message.

3

```
Employee::Employee(std::string name, double hourly_rate)
    : _name{name}, _hourly_rate{hourly_rate}
{
    if (_hourly_rate < 0) throw std::runtime_error
        ("Hourly rate can't be negative"); }
```

3

d. {~~2~~ 3 points} In file employee.cpp, write the implementation of `operator<<`. For Employee "Prof Rice" with an hourly rate of "8.25", for example, stream out "Prof Rice ($8.25)". Use I/O manipulators (but NOT `printf`) to ensure 2 digits follow the decimal point on the salary.

```
std::ostream& operator<<(std::ostream& ost, const Employee& e) {
    std::cout<<std::setprecision(2) << std::fixed;
    ost << e._name << " ($" << e._hourly_rate << ")" <<
                std::endl;
    return ost;
}
```

3

e. {~~2~~ 3 point} In file Associate.cpp, implement ONLY the constructor. Use an init list to construct the superclass parameters using chaining.

```
Associate::Associate(std::string name, double hourly_rate)
    : Employee{_name}, Employee{_hourly_rate} {}
```

3

f. {~~1~~ 2 point} In file Associate.cpp, implement ONLY the `paycheck` method. The first 40 hours are paid at the superclass field's `_hourly_rate`, and the rest are paid at 1.5 times that amount. If `hours` are negative, pay $0.00.

"override" is checked in .h file!

```
double Associate::paycheck(double hours) { int res1, res2;
    if (_hourly_rate < 0) return 0.00;
                                    res1 = 40 * _hourly_rate;
else if (hours > 40.00) {
    res2 = (hours - 40) * 1.5; return res1 + res2; }
    else { return hours * _hourly_rate }
}
```

2

In file payroll.cpp, begin writing the main function which was begun for you starting on the next page.

This program reads file "payroll.dat" and prints the payroll to standard out for both `Boss` and `Associate` employees. The fields on each line are employee type ("A" for Associate, "B" for Boss), hours worked, wage, and name, each whitespace separated. For example,

A 55.0 15.0 Lee Chen

B 70.3 31.5 Ursula Garcia

When complete, the program should output something like this from the above data:

Lee Chen ($15.00) is paid $937.50

Ursula Garcia ($31.50) is paid $1260.00

You will write the main function a few lines at a time beginning on the next page. If you can't solve one of the steps, simply skip it and continue with the next step. If you prefer to write the remainder of Free Response #1 as a single program rather than in code snippets, please simply ask for an additional sheet of paper and write it on that in its entirity.

```
#include "Associate.h"
#include "Boss.h"

#include <vector>
#include <fstream>
using namespace std;
int main() {
    std::string type, name;
    double wage, hours;
    Employee* e;
```

g. {2 points} Continuing to write the payroll.cpp main program, open file "payroll.dat" for input.

```
std::ifstream ifs{"payroll.dat"};
```

2

h. {2 points} Continuing to write the payroll.cpp main program, if the file failed to open in part (g), write "Open failed" to standard error and return an error code of -1 to the operating system.

```
if (!ifs) {
    std::cerr << "Open failed" << std::endl;
    return -1; }
```

2

i. {1 point} Continuing to write the payroll.cpp main program, loop while data is available from the file.

```
std::string s;
while (std::getline(ifs, s)) {
```

1

j. {3 points} Continuing to write the payroll.cpp main program, while in the loop begun in part (i), read `type`, `hours`, and `wage` (whitespace separated) and then `name` (the rest of the line to the n) into variables defined above.

```
std::istringstream iss{type, hours, wage};
std::cout << type << hours << wage<<std::endl;
std::cout << name << std::endl;
ifs>> type>> hours >> wage;
std::getline(ifs, name);
```

1/2

k. {2 points} Continuing to write the payroll.cpp main program, while in the loop begun in part (i), if `type` is "A", instance `Associate` **on the heap** pointed to by variable `e`, otherwise, instance `Boss` **on the heap** pointed to by variable `e`.

```
if (type == "A")e=new Associate a;
else {e = new Boss b;}
```

1 1/2

l. {3 points} Continuing to write the payroll.cpp main program, while in the loop begun in part (i), using the variable `e` created in part (k), print the employee pointed to by `e` and then *polymorphically* their paycheck amount using its method `paycheck`.

1 1/2

```
std:: cout << e.name << "$(" << e.wage << ")" << " is
paid" << "$" << Employee.paycheck(e.hours) << std::endl;
```

m. {1 points} Continuing to write the payroll.cpp main program, after the loop, if the program did NOT reach the end of the payroll file, print an error message "Incomplete payroll" to standard error.

2

```
if (!ifs.eof()) {
    std::cerr << "Incomplete payroll" << std::endl;
    return -3 }
return 0; }
```

2. (vector, map, enum, shuffle, iterators) If you prefer to write all of Free Response #2 as a single program rather than in code snippets, please ask for an additional sheet of paper.

a. {1 points} In file Component.h, write enum class `Component` with the values `RED`, `GREEN`, and `BLUE`. You may omit the guard to save time.

2

```
enum class Component {
    RED, GREEN, BLUE;
}
```

b. {2 points} In file Color.cpp, typedef `Color` as a `std::map` with the key a `Component` and the value an `int`. Assume you have all of the `#include` statements that you need; don't write them.

2

```
# typedef std::map<Component, int> Color;
```

X

c. {1 point} Begin writing the main function. Accept command line parameters. Assume you have all of the `#include` statements that you need; don't write them.

2

```
int main(int argc, char* argv[]) {
```

d. {1 point} As part of the main function, declare a vector named `colors` that will contain objects of type `Color`.

1

```
std:: vector <Color *> colors;
```

e. {1 point} As part of the main function, declare a variable of type `Color`.

```
Color matching;
```

1

f. {3 points} As part of the main function, loop over the command line arguments *by threes*, adding each to the `Color` map in the order `RED`, `GREEN`, and then `BLUE`. Once the `Color` map is updated, add it to the `colors` vector. So if the command line was "main 32 64 96 128 192 255", the `colors` vector would contain 2 maps, a map with `RED` = 32, `GREEN` = 64, and `BLUE` == 96, and a second map with `RED` = 128, `GREEN` = 192, and `BLUE` == 255.

```
for(int i; i<=argc; i+=3) {
    matching.push_back(Component[i], i);
matching[Component:: RED] = atoi(argv[i]);
                 GREEN          i+1
                 BLUE           i+2
colors.push_back(matching);
}
```

1

g. {2 points} As part of the main function, *after* the loop, shuffle the `colors` vector.

```
std::shuffle(colors.begin(). colors.end());
```

1 1/2

random_

h. {3 points} As part of the main function, **use iterators only** to print each of the `colors` vector's elements in order. You may assume types `Component` and `Color` have defined the `<<` operator already; you need not code them yourself.

```
auto it = colors.begin();
while (it != colors.end()) {
    std:: cout << *it++ << std::endl;
} return 0;
}
```

3

Bonus

Bonus 1: {+3 points} List the 3 types of C++ casting that we discussed, when they should be used, and give an example of each.

downcast → static_cast & dynamic_cast

upcast

+2

Bonus 2: {+3 points} `std::find` returns an iterator pointing to the first matching element in its container (collection). How would you find ALL of the matching elements in the container? You may write C++ code (only correct enough to demonstrate the algorithm or give a *brief* and *clear* text description.

```
std:: int it = std:: distance {cont_name.begin(),
    std::find(cont_name.begin(), cont_name.end(), target)};
```

+1 1/2