CMSC 202 Spring 2024 Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exam 2 Review Worksheet**

**General Concepts**

Exam 2 will be based on chapters 1, 2, 3, 4, 5, 6, 7, 8, 10, 14 and linked list part of 17. You can also be asked questions on any of the topics covered in the lab material including debugging and makefiles. There is an extra presentation on makefiles available on Blackboard under course materials.

You are responsible for any material in the book and available on the slides (meetings 1 – 18).

**Fill-in-the-Blanks**

* Child classes can **inherit, extend, override** parent class behaviors.
* Dynamically allocated memory that is no longer pointed to by any pointer variable causes a **memory leak**. (two words) [seg fault is when you access memory you never owned, int \* p = 0; \*p = 17;
* When a class object goes out of scope, the **destructor** deallocates the memory occupied by the nodes of a list .
* **Encapsulation** is a form of information hiding and abstraction.
* The name of the function to overload the operator <= is **operator <=()** .
* Each node of a linked list must store the data as well as the **address of / pointer to** the next node in the list.
* The ability to reuse objects already defined, perhaps for a different purpose, with modification appropriate to the new purpose, is referred to as **inheritance???**. [not really true]
* I have a pointer, **nodePtr** to a node that is a **struct** in a linked list. I want to access the member named **data**. I do this using the expression **nodePtr->data**.
* In C++, **\*** is called the *dereferencing* operator.
* The components of a class are called the **members** of the class.
* When a class inherits from another class, **private** properties are inherited from the parent class but are not directly accessible by the Child class.
* To **override** a public member function of a base class in the derived class, the corresponding function in the derived class must have the same name, number, and types of parameters.
* **enums** are a type of variable used to set up collections of named constants.
* To guarantee that the member variables of a class are initialized, you use **the constructor**.
* Aggregation is an example of a(n) **has a** relationship.
* The preprocessor directive(s) **too complex for a single blank** is used to prevent multiple inclusions of a header file in a program.
* In total it is called a header guard.
* #ifndef HEADERTHINGY
* # define HEADERTHINGY
* ... code...
* #endif

**Short Answer**

* Explain how "pass by reference" in C++ is similar and different than passing parameters with pointers.

They are similar in that you are able to modify the data when passed by reference, and also you can modify the data that the pointer points to as well.

When you pass by reference, to the programmer it looks like a regular variable, but when you pass a pointer, you'll have to access it and use it as a pointer, you will probably have to dereference.

Underlying all of this, they are both passing pointers, but references hide that fact a little.

function( int& x)

function(3) = 3 is a literal doesn't have an address so can't be passed by reference.

* What is the scope of a variable declared ***in*** a ‘while’ loop control statement?   
  (i.e. **while (counter < 10) { int x; })**

in a code block {...} the scope is that block, so that variable doesn't exist outside of the block. [applies to functions, for, while, all the things. A while loop with {} will create a bit of scope.

* What are virtual functions used for and how do they apply in **polymorphism**?

Polymorphism = a function can have the same name, but multiple implementations in various parent vs child/grandchild classes. Virtual function is a function in a class which allows you to determine which kind of object it is, and whether to run the parent or child version of the function at runtime. Hides a virtual table pointer into each version of the class.

It's not just overriding because if you have a child class that re-implements a function it will get called instead of the parent class. In Polymorphism/virtual functions they will get called even if you have a pointer that thinks it's the parent class.

* Why would we put the **const** keyword in front of a pass-by-reference parameter?
* void fun( const BlahThing & blah) // prevents us from modifying the underlying object but it also doesn't require us to call the copy constructor, if the object is big, making copies is expensive.
* void fun( BlahThing blah) // passed by value, invokes the copy constructor to make a new local version of the class.

Why would we use dynamically allocated arrays vs vectors?

dynamically allocated array has a fixed size, but determined at runtime, so the user can still tell us hwo many things they want, but then it's fixed after that.

vector also dynamic but it resizes itself.

**True or False**

* True or False? Why?

|  |  |
| --- | --- |
| (a) True | Enums create integer constants and can be named or unnamed. |
| (b) True [hopefully] | In a class, access modifiers affect the accessibility of the members of the class. |
| (c) True [super careful about this] | In C++, an array is passed by value. |
| (d) False | Inheritance results in a *has-a* relationship. [is-a] |
| (e) False [close, private-> protected then true] | The constructors of a derived class can (directly) initialize the (public and private data) members inherited from the base class of the derived class. |
| (f) False | A class that has a pure virtual member function can be instantiated by declaring an abstract object of that class. |
| (g) False | A class’s destructor is automatically called when an object of that class is declared. |
| (h) True | Each node in the linked list points to either another node in the linked list or to **NULL**. |
| (i) False | The last element in a singly-linked list takes the least time to access. [unless of course there's a tail them maybe true?] |
| (j) False | You can free all the memory in a linked list (that starts at node **m\_head**) with: **delete[] m\_head***;* |
| (k) False | The **->** operator is used to access member functions of a structure using its object [pointer]. |
| (l) False | The **?:** operator can be overloaded. [ternary cannot be overloaded]  x ? y : z if x then does y, else z |
| (m) False | You can directly access the nth node of a linked list. |
| (n) True | Vectors store all of their data contiguously in memory. |
| (o) False | In the following class, sour will be constructed before bar:  **#include "Candy.h"**  **class Candy {**  **public:**  **Candy() : bar(), sour() {}**  **Candy sour;**  **Candy bar;**  }; |

|  |  |
| --- | --- |
|  |  |

**Code Evaluation**

* What is the output of the following code?

**#include <iostream>**

**using namespace std;**

**int main () {**

**string \*p1, \*p2;**

**p1 = new string;**

**p2 = new string;**

**\*p1 = "UMBC";**

**\*p2 = "CHALLENGE";**

**cout << \*p1 << " " << \*p2 << endl;**

**p1 = p2;**

**cout << \*p1 << " " << \*p2 << endl;**

**\*p1 = "HACKATHON"; /// p1 = p2 then they are both the same address \*p1 sets the actual thing stored at p2 as "HACKATHON"**

**cout << \*p1 << " " << \*p2 << endl;**

**return 0;**

**}**

* What would be the corresponding integer value for D in:  
  **enum foo{A=2, B=5, C=6, D, E=10, }?**

D = 7

* What is the output of the following code?

**int a = 4;**

**int c = a++ == 3 ? 123 : 456;**

**int c = (a++ == 3) ? 123 : 456;**

**int c = a++ == (3 ? 123 : 456);**

**cout << "c = " << c << endl;**

x = 5 + (3 < 5);

= 5 + 1 = 6;

**Debugging**:

The code below has five errors. The errors may be syntax errors or logic errors so examine the code carefully to find them. **Assume that include <string>, <iostream> are loaded as well as using namespace std.**

You must find and correct all five of the errors.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | **Class Toy {**  **public:**  **void SetType(string type){**  **m\_type = type;**  **}**  **void SetMaterial(string material){**  **m\_material = material;**  **}**  **private:**  **string m\_type;**  **string m\_material;**  **};**  **class Phone : private Toy{**  **public:**  **void MakeCall() {**  **cout << "You made a call!" << endl;**  **}**  **};**  **class Truck : public Toy{**  **public:**  **void Zoom() {**  **cout << "Your truck zoomed all around." << endl;**  **}**  **}**  **int main () {**  **Toy myTruck;**  **myTruck.SetType("Tonka");**  **myTruck.SetMaterial("Steel");**  **Phone myPhone;**  **SetType("Play Phone");**  **myPhone.SetMaterial("Plastic");**  **myTruck.Zoom();**  **myPhone.MakeCall();**  **return 0;**  } |

|  |  |
| --- | --- |
| Line Number | Correction |
| 1 | **Class Toy { => class Toy** |
| 12 | **class Phone : private Toy** |
| 23-25 | **};** |
| 26 | **Toy myTruck; ==> Truck myTruck;** |
| 29 | **myPhone.SetType("Play Phone");** |

**Coding Problems**

* Write the C++ code that builds a new **struct** called light that is the node for a linked list. The node should hold a string which is the type of the light and a Boolean that says if it was “on” or “off”.

struct Light

{

string m\_type;

bool m\_on\_off;

Light \* m\_next;

};

* Build a class named Instrument and the subclasses Guitar, Violin, and Bass using inheritance. The name of the instrument and the price are included in the Instrument class. A Guitar has a whammy bar(Boolean). Violins should have a pointer to an aggregated Bow() class. Bass should be made up of both a Body() and Neck();

class Instrument

{

public:

private:

};

class Guitar : public Instrument

{

public:

private:

};

class Violin : public Instrument

{

public:

private:

};

class Bass: public Instrument

{

public:

private:

};

* Build a set of parallel vectors of integers that are designed to manage your scores on your labs. One vector holds the score you earned. The second vector holds the maximum possible score. Write the code to populate them with your scores 10/10, 8/10 and 9/10.

vector<int> totals;

vector<int> scores;

for (int i = 0; i < MAX\_SCORES; i++)

{

cin >> new\_score >> new\_total;

totals.push\_back(new\_total);

scores.push\_back(new\_score);

}

* Write an overloaded function called product. The first function should accept two doubles that multiples them together. The second function should accept three doubles that multiples them together.

double product(double x, double y)

{ return x \* y;}

double product(double x, double y, double z)

{ return x \* y \* z;}

**Completion**

* Fill in the blanks from the implementation of a Student class:

**\_\_\_\_\_\_\_\_\_\_\_\_\_ Student {**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**void setName(string name);**

**void setGPA(\_\_\_\_\_\_ gpa);**

**\_\_\_\_\_\_ getName();**

**double getGPA();**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**string m\_name; //student’s name**

**double m\_gpa; //student’s GPA**

**};**

In my main function, I create a Student object named **bob** and use the "set"

methods to initialize bob's variables. Complete the following code to print

bob's name and, if he has a GPA of at least 3.5, print the message "I have good grades!":

**cout << \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ << endl;**

**if ( \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ )**

**cout << "I have good grades!" << endl;**

* Write the C++ code that builds a new **struct** called Tooth that is the node for a linked list. The node should hold a string which is the name of the tooth and a Boolean that says if it was “extracted” or “natural”.

* In the class Shoe defined below what would an overloaded > operator look like that compares specifically **m\_size**?

**#include <iostream>**

**#include <string>**

**using namespace std;**

**class Shoe{**

**public:**

**Shoe(int size, string brand){**

**m\_size = size;**

**m\_brand = brand;**

**}**

**//Write overloaded > operator here that compares m\_size**

**private:**

**int m\_size;**

**string m\_brand;**

**};**

**int main () {**

**Shoe myShoe(10, "Nike");**

**Shoe theirShoe(15, "Adidas");**

**cout << (myShoe>theirShoe?"MY SHOE BIGGER":"THEIR SHOE BIGGER") << endl;**

**return 0;**

**}**

**//Output of this run is THEIR SHOE BIGGER**

**Linked Lists**

Consider the List class with the following private members:

**class List{**

**/\* public members here ... \*/**

**private:**

**struct Node{**

**int m\_item; // the data of the node**

**Node\* m\_next; // points to the next node of the list**

**};**

**Node\* m\_head; // point to first node in the list**

**};**



* Draw a diagram of the above list after the following lines of code have been executed:

**Node\* prev = m\_head;**

**Node\* curr = m\_head;**

**for(int i = 0;i < 2; i++){**

**prev = curr;**

**curr = curr->m\_next;**

**}**

**Node\* nodeToInsert = new Node;**

**nodeToInsert->m\_next = curr;**

**nodeToInsert->m\_item = 25;**

**prev->m\_next = nodeToInsert;**

**10 -> 10 -> 25 -> 15 -> 35 -> 19 -> null**

* After this code executes, what is the value of **prev->item?**
* Given the class declaration above, write a function called **PrintMiddle()** that finds the middle of the linked list and outputs the data at that location.

**Makefiles**

I am writing a program to manage a new ice cream shop. So far, I've written three classes — Customer, Shop, and Cashier — and a main program called **icecream.cpp**. Each of the classes consists of a header (.h) file and an implementation (.cpp) file.

Consider the following makefile:

|  |  |
| --- | --- |
| **1** | **CPP = g++** |
| **2** | **CPPFLAGS = -ansi -Wall** |
| **3** |  |
| **4** | **icecream: Shop.o Customer.o Cashier.o** |
| **5** | **$(CXX) $(CXXFLAGS) Gym.o Customer.o Cashier.o -o icecream** |
| **6** |  |
| **7** | **Shop.o: Shop.cpp Customer.h Cashier.h** |
| **8** | **$(CXX) -c Shop.cpp** |
| **9** |  |
| **10** | **Customer.o: Customer.cpp Customer.h** |
| **11** | **$(CXX) -c Customer.cpp** |
| **12** |  |
| **13** | **Cashier.o: Cashier.cpp Cashier.h** |
| **14** | **$(CXX) -c Cashier.cpp** |

* If I make a change to **Cashier.h**, which files will be recompiled?
* Shop.o Cashier.o then icecream
* For the rule on lines 7 and 8, what are the dependencies?
* For the rule on lines 13 and 14, what is the target?
* What is the meaning of the "-c" flag on lines 8, 11, and 14?

* This is a super challenging problem. If is far more related to a puzzle than something you would see on a CMSC 202 exam, but it is interesting and somewhat related. What does this output?

**#include <iostream>**

**using namespace std;**

**class Container {**

**public:**

**int value;**

**Container( int amount ) { value = amount;**

**cout << "Value " << value << endl; };**

**~Container() { cout << " You just killed: " << value << endl; };**

**};**

**class ExamQuestion {**

**public :**

**Container data;**

**ExamQuestion(int A) : data(A) { cout << "New Object\n";};**

**ExamQuestion( const ExamQuestion& X ) : data(X.data.value + 10)**

**{ cout << "Special\n"; };**

**};**

**void TrickyPart(ExamQuestion why){**

**ExamQuestion PartB = why;**

**cout << "After PartB\n";**

**}**

**int main(){**

**ExamQuestion Answer(1);**

**cout << "Call TrickyPart\n";**

**TrickyPart(Answer);**

**cout << "end" << endl;**

**}**