CMSC 202 Final **December 20, 2005**

Name: _____ **UserID:** _____

(Circle your section)

Section: **101** – Tuesday 11:30 **102** – Thursday 11:30

103 – Tuesday 12:30 **104** – Thursday 12:30

105 – Tuesday 1:30 **106** – Thursday 1:30

Directions

- This is a closed-book, closed-note, closed-neighbor exam.
- Read through the entire test before you begin.
- Start with the questions that are easiest for you, come back to the rest.
- Write CLEARLY, if I cannot read your writing, you will receive a zero for the problem in question.
- Feel free to continue your answer on the backs of the pages, but make sure that you indicate where your answer continues.
- When you are done, read over your answers and then bring your exam to the front of the room.
- **Show your Picture ID AND Exam paper to a TA/Instructor, place in correct pile.**

Score

Page Number	Points Possible	Points Earned
2	10	
3	12	
4	7	
5	15	
6	10	
7	12	
8	8	
9	10	
10	16	
11 (EC)	6	
12 (EC)	9	
TOTAL	100 (+15 EC)	

True/False (10 pts total, 1 pt each)

Read each statement ***carefully*** and write **true** or **false** on the blank to the left.

_____ 1. It is legal to instantiate an object of an abstract class.

_____ 2. The following lines of code correctly modify the value of 'a' to be 7.

```
int a = 2;
int* ptr = new int(a);
ptr = 7;
```

_____ 3. Dynamic objects are allocated on the stack while static object are allocated on the heap.

_____ 4. The following code correctly creates and deletes an array.

```
int* array = new int[ 10 ];
delete array [ ];
```

_____ 5. If try/catch blocks are nested, the exception is always thrown to the outermost catch block.

_____ 6. The default overloaded operator= (provided by the compiler) results in a shallow copy of memory.

_____ 7. When using dynamic memory, one should always overload the copy-constructor and should be sure to protect from self-assignment (i.e. assigning A to itself).

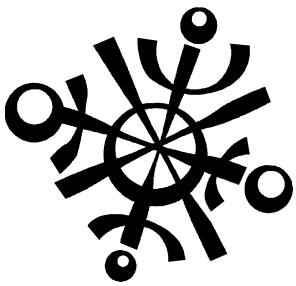
_____ 8. Derived classes can use, modify or extend methods from their parent class(es).

_____ 9. When polymorphism is used in C++, the base-class destructor is called ***before*** the derived-class destructor.

_____ 10. Assume that myVector is a vector of integers, myVector.end() returns an iterator that points to the last item in myVector.

_____ pts

Short Answer



Complete each of the short-answer coding questions. You may assume that the questions build on each other and that previously implemented lines can be used in later questions.

Assume there is a class named Rider with derived classes named Skier and Snowboarder.

11. (1 pt) Define a **dynamic array** of **Rider pointers**. Assume that the size of the array is in a variable named 'size'.

12. (1 pt) Assume there are already 4 Riders (of various subtypes) in the vector. **Add** a **Snowboarder** to the array.

13. (3 pts) Assume that the **insertion** operator is **overloaded** for all **Rider** types. Using a **for-loop**, iterate through the array, **printing** each rider to the screen.

14. (7 pts) Assume that the **> (greater than) operator** is defined for all Rider types and returns a **boolean** (a > b == true if a is a better Rider than b). Define a **templated** function (the function should now know what a Rider is) that finds the **Best** item in the array and **returns the object**.

_____ pts

15. (6 pts) Assume the **Skier** has an overloaded **constructor** that accepts a **skill-level** (1 -> 9, beginner -> advanced). Assume there are also a **related mutator and an accessor**. Assume the following lines are defined:

```
Skier a(6);
const Skier b(1);
```

Identify whether the following lines are **compilable.** If not, describe why. Assume each chunk of code is examined in isolation of the others.

Will Compile (Yes/No)?	**Code…**
_____	`const Skier* p = &a;` `p->SetSkillLevel(8);`
_____	`Skier* const q = &a;` `q->SetSkillLevel(8);`
_____	`const Skier* r = &b;` `r->SetSkillLevel(8);`
_____	`Skier* const m = &b;` `m->SetSkillLevel(2);`
_____	`const Skier* p = &a;` `p = &b;`
_____	`Skier* const q = &a;` `q = &b;`

16. (1 pts) What limitation must be placed on the skill-level accessor of the Skier classes to have the following code compile?

```
const Skier * t = &b;
b.GetSkillLevel();
```


_____ pts

17. (10 pts) Assume that the Skier **constructor** used in the previous question **throws** an **OutOfRange** and **some other exception**. Write a **loop** that will create **10 Skiers** and put them into the original dynamic **array** (assume it is empty), using **5 to 15** (consecutively) as the **skill-level** parameter. Using a **try/catch** block, correctly **catch** the exceptions. If an **OutOfRange** exception is **caught**, the **default constructor** for the Skier should be used and **processing should continue** with the **next skier**. If some other **exception** is caught, the exception should be **re-thrown**.

18. (5 pts) **Implement** the Skier **constructor** that accepts a single **integer** parameter (skillLevel). Assume there is a **data member** named **'m_skillLevel'**. If the skillLevel parameter is **less than 1** or **greater than 9**, **throw** an **OutOfRange** exception. Ignore the other exception described in the previous question.

_____ pts

Class Implementations

19. (10 pts) Write the **class definition** (header file) for the Rider class. Use static, constants, virtual and references whenever appropriate. The Rider class has the following members:
    a. **skillLevel** data member, integer, min of 1, max of 9
    b. **MaxSkillLevel** data member, integer, represents the maximum skill level
    c. **MinSkillLevel** data member, integer, represents the minimum skill level

    d. **Default** constructor, sets skillLevel to minimum
    e. **Non-default** constructor, sets skillLevel to parameter value if valid
    f. **Copy** constructor – copies parameter
    g. **Destructor** – destroys object
    h. **GetSkillLevel** – returns the Rider's skill level
    i. **SetSkillLevel** – sets the Rider's skill level
    j. **ReplaceBindings** – method to be overridden by derived classes

_____ pts

20. (10 pts) Write the **class definition** (header file) for the **Skier** class. Use static, constants, virtual and references whenever appropriate. Assume there is a **Ski** class that represents a **single ski**. The **Skier** class has the following members:
    a. **Skier**, inherits from **Rider**
        i. **leftSki** dynamic data member that is the left ski
        ii. **rightSki** dynamic data member that is the right ski
        iii. **Default** constructor, a skier initially has no skis
        iv. **Copy** constructor
        v. **Destructor** – destroys any dynamic memory
        vi. **ReplaceBindings** – replaces the bindings on both skis

21. (2 pts) Discuss the **difference** between a **shallow** and **deep** copy for the **copy**-constructor of the Skier class. Use an **example** to **illustrate** (no code).

_____ pts

22. (3 pts) **Implement** the **copy** constructor of the **Skier** class using a **deep** copy.

23. (3 pts) Assume that a skier can have an **entire collection** of Skis, but that he must **choose only 2** from that collection to **use that day** (his "**Current Skis**"). Assuming that we represent the **set of skis** as a **vector** of **pointers** to Skis, briefly describe **two ways** to represent his **chosen pair**. **Compare** and **contrast** these two strategies, discussing the **time, space, and access tradeoffs** between them.

24. (2 pts) **Implement** the **destructor** for the **Skier** class.

_____ pts

(3 pts) Assume that we would like to **add** this **collection** of **foot-equipment** (i.e. Skis or Snowboards) to the base class, **Rider**. **Prototype** (i.e. forward-declare) the **Rider** class as a class **templated** on a **single type** of equipment.

25. (2 pts) Define the **collection** data member of the **Rider** class using a **vector** of **pointers** to the **equipment-type**.

26. (2 pt) Should the **collection** be **private**, **protected**, or **public**? **Why**?

27. (3 pts) Since the only **difference** between a Skier and a Snowboarder is that the **Skier** has **2 pieces of equipment** and the **Snowboarder** has **1**, how could you **combine** these **two classes** using **templates** into only the **Rider** class? **How** would you **eliminate** the **need** to store **two data members** (a left and right ski) for the **Skier**?

_____ pts

Exposition

28. (4 pts) **Describe** the **differences** between method **overriding** and method **overloading**. Provide an **example** to **support** your comparison.

29. (4 pts) What is the **purpose** of including a **Clone**() method in **inheritance**? Why should this **method** use **deep copies**?

30. (4 pts) Briefly **discuss** the **pros** and **cons** of using **inline** functions.

31. (4 pts) **Why** is it **important** to **protect** an object from **self-assignment** (i.e. assigning A to itself)? (Hint: think about **dynamic memory**)

_____ pts

Extra Credit

32. (3 pts) Assume that you want to implement a templated **List** (push_front, push_back, pop_front, pop_back), but ***only*** have access to a **Vector** with the following methods:
    - `insert(iter)`, inserts an item before the position pointed to by the iterator parameter
    - `erase(iter)`, removes the object pointed to by the iterator from the vector
    - Assume that the methods `begin`(), `end`(), and `size`() work exactly as in the STL vector class, you may also assume that the ++ and -- operators work with these iterators.

    Describe briefly how you would **use** the above **Vector** to **implement** a **List**. You may assume that your List class must support the following:
    - private data member: `Vector<T> list;`
    - `pop_back` – removes last item in the List
    - `push_back` – insert the parameter after the last item in the List
    - `pop_front` – removes first item in the List
    - `push_front` – insert the parameter before the first item in the List

33. (3 pts) **Implement** the **pop_back**() method for your **List** using the **Vector** described above. You may **allocate** any **additional memory** necessary.

_____ pts

Extra Credit – Part Deux

34. (3 pts) If I had asked **Extra Credit #1** in the **exact opposite** way (i.e. **build** a **Vector** on a **Linked-List**), what would be the **greatest difficulty** with implementing an at(i) method that returns the object in the ith position?

35. (4 pts) **Write** the **pseudocode** (or code) to **implement** the **at(i)** method of the **Vector** class on a **private data member** that is a **Linked-List named `'list'`**. You may **only use** the following methods: **push_front, push_back, pop_front, pop_back**. You may assume that the pop_* methods **return** the **object** they have **removed**.

36. (2 pts) If you knew the **world** was going to **end** tomorrow, **whom** (if anyone) **would you tell** and **why**?

_____ pts