CMSC 202 Midterm October 19, 2006

Name: ________________________________ **Email ID:**______________

Section: (Circle your section)

101 – Tuesday 11:30 **105** – Wednesday 10:00 **102** – Thursday 11:30

103 – Tuesday 1:00 **104** – Thursday 1:00

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. If you have time at the end, come back to the more challenging ones.
- 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.
- **You will need your Picture ID to hand in your exam.**

Score

Page Number	Points Possible	Points Earned
2	10	
3	20	
4	15	
5	15	
6	15	
7	15	
8	10	
9 (EC)	**10**	
TOTAL	**100 (+10)**	

Roller Coaster Mania!

True/False (10 pts, 1 pts each)

Decide if the following are **true** or **false**; put the appropriate ***word*** in the blank.

____________ 1. The following code is a **valid** method for **opening** a file for **output** in C++:

```
string filename = "output.txt";
ofstream fout(filename);
```

____________ 2. To use **both `ofstream`** objects and **`ifstream`** objects, we can simply **include** the **`iofstream`** library.

____________ 3. The **compiler** uses function **name**, **parameter list** (both number and type), and **return type** to differentiate between functions when **deciding** which function to call.

____________ 4. The following is a **valid** function **prototype** using **default** parameters:

```
void foo(int x, int y = 1, int z = 0);
```

____________ 5. The following code will **print each** item in the **vector**:

```
vector<int> values(10, 5);
for (unsigned int i = 1; i <= values.size(); ++i)
      cout << values[i] << endl;
```

____________ 6. The following is perfectly **valid** in C++:

```
string message = "Hello";
cin >> message;
```

____________ 7. **Zombie** objects are **one** strategy for dealing with **unmet preconditions** in a Constructor.

____________ 8. **Functions** that are declared as "**friend**"s of a class have the ability to **directly access** the class's **private** data members.

____________ 9. **Static methods** may access **static** and **non-static** data while **non-static** methods may ***only*** access **non-static** data.

____________ 10. The **insertion** and **extraction** operators can be given **access** to the **private data** member of their **right**-hand operand by being defined as **member functions** of the **operand's** class.

Short Answer

The following questions are all related and deal with the same system. Assume that the proper header files have been included.

11. (2 pts) **Prompt** the user for the **name** of their **favorite** carnival or theme park **ride**. **Read** in the **name** (Ex: The Big Bad Wolf).

12. (4 pts) **Use** a **vector** to store a collection of **names** of carnival/theme park **rides**. Use a **second vector** to store the **maximum speed** of each ride (Ex: 120.7).

13. (6 pts) Assume there is a **file** (rides.txt) that stores each **ride name** followed by the **max speed** on the **next line**. **Open** this **file**. **Read** in all of the **rides** in the **file**, **storing** them in the above **vectors**.

Ex:

```
The Big Bad Wolf
67.9
Top Thrill Dragster
120.0
```

14. (10 pts) Use a **loop** to **print** all of the items in the **vector**, **aligning decimal** points **vertically**.

Ex:

```
The Big Bad Wolf               67.9
Top Thrill Dragster           120.0
```

Class Construction

The following questions all have to do with the same system. Make appropriate decisions about data types, return types, const, and parameter passing. Ignore header-file guarding and includes.

15. (15 pts) You are designing a theme park and want to include your favorite rides in the park (do not implement, yet). Each **ride** will have a **name**, a maximum **speed**, and a **number of passengers**. No ride at the park is allowed to have more than **50 passengers** on the ride at a time (for insurance reasons).

    Your **Ride** class must have:

    a. A **single constructor** that serves as both the **default** and **non-default** constructor. (default data: "Default Ride", speed of 0.0, no passengers)
    b. Appropriate **accessors** for each data member
    c. Appropriate **mutators** for each data member
    d. **<u>3 data members</u>** that represent the **name, max speed** and **number of passengers**
    e. All minimum and maximum values for data members should be **constant**, **shared** data that is **inaccessible** to outside classes/functions
    f. An overloaded **addition operator** that will add a **number of passengers** to the Ride, this operator should **not** have **direct** access to the data members. The **left** operand should be of type **Ride**, the **right** operand should be of type **int**.

16. (2 pts) **Create** a Ride object called the **"Millennium Force"**, with a max speed of **93 MPH**, and **23 people** on the ride. In **another line**, use the **addition operator** to add **7 more people** to the ride. Use an **accessor** to **print** the **number of people** currently on the ride.

17. (2 pts) **Initialize** your **shared data** member(s).

18. (3 pts) **Implement** the **mutator** for the **number of passengers** on the ride. **Include** appropriate **error** checking.

19. (4 pts) **Implement** the **constructor** for your Ride class. **Use** other class **methods** when appropriate.

20. (4 pts) **Implement** the **addition operator** for your Ride class. **Use** other class **methods** as appropriate.

Aggregation

21. (15 pts) Declare a **ThemePark** class (again, do not implement, yet). Obviously, your ThemePark holds a **collection** of **Rides**.

Your **ThemePark** class must have the following:

a. A **default** constructor
b. A method to **add** a Ride to the ThemePark.
c. A method to **remove** some of the passengers from one of the rides. This method should have its **first parameter** as the **ride name** and the **second** parameter as the **number of passengers** to remove.
d. A method to **find** the **index** of a Ride from the ThemePark given the **name** of the Ride. This method should **only** be **accessible** to the **ThemePark** class.
e. An overloaded **insertion operator<<** to **print** all of the **rides** in the ThemePark – this operator **should** have **direct** access to the data members.
f. A **dynamic data member** to store a **collection** of Rides.

22. (5 pts) **Implement** the method to **find** the index of a **ride** based on the **name** of the ride for your **ThemePark** class.

23. (5 pts) **Implement** the **remove method** for your ThemePark class. **Use other methods** as appropriate.

24. (5 pts) **Implement** the **add method** for your ThemePark class.

25. (5 pts) **Describe** the potential problem when using **both** **`getline( )`** and the **extraction operator >>** to read strings and integers. Be **specific** about **exactly** how the input is treated. **Use** an **example** to **support** your argument. **Describe** one **solution** to the problem and **provide** an **example** of this solution.

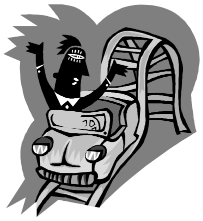
26. (5 pts) Define **abstraction**. Provide an **example** that demonstrates the **power** and necessity of abstraction.

Extra Credit

27. (4 pts) **Implement** a **Sort** method for your ThemePark. It should **sort** the Rides alphabetically by name.

28. (4 pts) Explain the **difference** between these **two declarations**. **Draw** a **picture** to illustrate your point.

```
vector< int > v1( 10 );
vector< int > v2[ 10 ];
```



29. (2 pts) What is your favorite carnival/theme park ride? Why? If you don't have a favorite ride, list your favorite trip you've ever taken and why it was your favorite.