CMSC 202 Final

Name: ____________________________ UserID: _________________
(Circle your section)
Section: 101 – Tuesday 11:30
105 – Tuesday 1:30
102 – Thursday 11:30
104 – Thursday 12:30

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

<table>
<thead>
<tr>
<th>Page Number</th>
<th>Points Possible</th>
<th>Points Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>10 (EC)</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>11 (EC)</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>100 (+15 EC)</td>
<td></td>
</tr>
</tbody>
</table>

Have a Great Summer!
True/False (10 pts total, 1 pt each)
Read each statement carefully and write true or false on the blank to the left.

__________ 1. The following code does not create a memory leak
    int* ptr = new int(b);
    ptr = new int(a);
    delete ptr;

__________ 2. Like the assignment operator, we must protect an object from
self-assignment in the copy-constructor using the following:
    if (this != &rhs)

__________ 3. Copy constructors, assignment operators and destructors are not
inherited in polymorphism

__________ 4. An abstract class is defined as a class that has at least one
virtual method and cannot be instantiated.

__________ 5. Class methods (member functions) cannot be declared as
protected.

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

__________ 7. Functions cannot be templated, only classes

__________ 8. Given this templated prototype of the class Stack:
    template <class T> class Stack;

    The following is an appropriate way of defining a Stack object:
    Stack<T = int> myStack;

__________ 9. When polymorphism is used in C++, the base-class constructor
is called before the derived-class constructor.

__________ 10. When an exception is thrown in a constructor, the object
creation is completed, but the object is set as invalid, or a
Zombie object.

I pinch
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 Crab with derived classes named HermitCrab and BlueCrab.

11. (2 pt) Define a dynamic array of Crab pointers. Assume that the size of the array is in a variable named 'size'.

12. (2 pt) Assume there are already 2 Crabs (of various subtypes) in the array. Add a BlueCrab to the array. Assume size > 2.

13. (6 pts) Assume that the Clone() method is overloaded for all Crab types. Using the Clone() method, implement the code that will allocate new memory for the Crab array such that the old array information is copied into the new array of size = size * 2 (the new array is twice the size of the old).
14. (5 pts) Assume the **HermitCrab** has an overloaded **constructor** that accepts a **shell-size** (integer size > 0). Assume there are also a **related mutator** and an **accessor**. Assume the following lines are defined:

```cpp
HermitCrab a(1);
const HermitCrab b(3);
```

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

<table>
<thead>
<tr>
<th>Will Compile</th>
<th>Code…</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td><code>HermitCrab* const q = &amp;a; q-&gt;MoveIntoShell(8);</code></td>
</tr>
<tr>
<td>Yes</td>
<td><code>const HermitCrab* p = &amp;a; p-&gt;MoveIntoShell(8);</code></td>
</tr>
<tr>
<td>Yes</td>
<td><code>HermitCrab* const m = &amp;b; m-&gt;MoveIntoShell(2);</code></td>
</tr>
<tr>
<td>Yes</td>
<td><code>const HermitCrab* r = &amp;b; r-&gt;MoveIntoShell(8);</code></td>
</tr>
<tr>
<td>Yes</td>
<td><code>const HermitCrab* p = &amp;b; p = &amp;a;</code></td>
</tr>
</tbody>
</table>

15. (5 pts) **Prototype** the **accessor** of the **HermitCrab** class so that the following code **compiles**.

```cpp
const HermitCrab* t = &b;
b.GetShellSize();
```
16. (10 pts) Assume that the HermitCrab MoveIntoShell() used in the previous question throws a ShellTooSmall and some other exception. Assume there are 5 (five) Crabs in the dynamic array from page 3.
   a. Write a loop that will call MoveIntoShell() to move each Crab into a new shell. Use srand() and rand() to generate random shell sizes to pass as the parameter.
   b. Using a try/catch block, correctly catch the exceptions thrown by MoveIntoShell().
      i. If a ShellTooSmall exception is caught, use the GetShellSize() method and move the Crab into a shell one greater than its current size. Continue processing the next crab.
      ii. If some other exception is caught, the exception should be re-thrown.

17. (5 pts) Implement the HermitCrab MoveIntoShell that accepts a single integer parameter (shellSize). Assume there is a data member named 'm_currShell'. If the new shell size is less than or equal to m_currShell, throw a ShellTooSmall exception. Ignore the other exception described in the previous question.
Class Implementations

18. (10 pts) Write the class definition (header file) for the Crab class. Use static, constants, virtual and references whenever appropriate. The Crab class has the following members:

a. name – dynamic data member, string
b. Default constructor – sets name to empty string [may combine with non-default]
c. Non-default constructor – sets name to parameter [may combine with default]
d. Copy constructor – performs a deep copy of parameter
e. Destructor – destroys object
f. GetName – returns the Crab's name
g. NewShell – Crab obtains a new shell, this may be overridden by derived classes
h. Move – Crab moves "ahead", this must be overridden by derived classes
19. (4 pts) Discuss the difference between a shallow and deep copy for the copy-constructor of the Crab class. Draw a picture to illustrate your argument.

20. (3 pts) Implement the copy constructor of the Crab class using a deep copy.

21. (3 pts) Implement the destructor for the Crab class.
22. (2 pts) Assume that we would like to create a collection of Crabs without using polymorphism, called a Bushel. Prototype (i.e. forward-declare) the Bushel class as a class templated on a single type of Crab.

23. (2 pts) Define the collection data member of the Bushel class using a vector of pointers to the type of Crab. Ignore the rest of the class definition.

24. (2 pts) Create a Bushel of HermitCrabs.

25. (4 pts) Implement the AddItem method for the Bushel class. The method accepts a single object to add to the collection and then stores it in the collection item from #23.
Exposition
26. (5 pts) Describe the differences between method overwriting and method overloading. Provide an example to support your comparison.

27. (5 pts) Briefly discuss the pros and cons of using inline functions.

28. (5 pts) Why is it important to protect an object from self-assignment (i.e. assigning A to itself)? (Hint: think about dynamic memory)
Extra Credit

For Problems 29 and 30, assume that you want to implement a templated Stack (push, pop), 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.

[Hint: think of the Vector as the data member of the Stack class]

29. (3 pts) Implement the `push()` method for your Stack using the Vector.

30. (3 pts) Implement the `pop()` method for your Stack using the Vector.
31. (3 pts) If I had asked you to **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?

32. (4 pts) Use the STL algorithm 'for_each' to **print** all of the items in your *Stack*.

33. (2 pts) If you were a crab, what would you say if I told you that I had some tongs and butter in the back of my SUV?