EECS 268 Home-Work 6 (Advanced C++) - Fall 2012

Do not write you name on this answer sheet (only your KU-ID). Total: 65 points

Name: **Prasad Kulkarni**

1. State true or false: **(5 points)**
   - A derived class inherits the constructors of the base class. False
   - A derived class can access all private, protected, and public members of the base class. False
   - You can use an instance of a derived class anywhere you can use an instance of the base class. True
   - Both the constructor and destructor functions can be virtual. False
   - Order of magnitude analysis accurately compares the relative efficiency of algorithms for problems of all sizes. False

2. *Illustrate* the changes a bubble sort algorithm will make at each step to sort the following array of integers. (Do not write any code). **(6 points)**

   `A = {2, 20, 8, 4, 10}`

3. In what cases is it appropriate to use (a) public, and (b) private inheritance? Explain with one example each. **(4 points)**

   Public Inheritance: Appropriate when the 'base' and 'derived' classes are related by an *Is-A* relationship. Example: Ball (derived) is-a Sphere (Base). Cat is-an Animal.

   Private Inheritance: *May be* appropriate when the 'base' and 'derived' classes are related by an *As-A* relationship. Example: (Implement) Stack (derived) as-a List (Base).

4. Is there an error in the below code fragment? If so, show how you would first correct the error. List the statements output by the corrected program. **(6 points)**

```
class Base {
public:
virtual void methodA(){
  cout << ``Base::methodA()\n'';}
void methodB(){
  cout << ``Base::methodB()\n'';}
};
```

```
class Derived : public Base {
public:
void methodA(){
   cout << ``Derived::methodA()\n'';}
virtual void methodB(){
   cout << ``Derived::methodB()\n'';}
};
```

```
int main(){
  Base b;
  Derived d;
  Base *bptr;

  bptr = &b;
  bptr->methodA();
  bptr->methodB();

  bptr = &d;
  bptr->methodA();
  bptr->methodB();

  return 0;
}
```

   There is no error in the above program.
   The following statements will be printed:

```
Base::methodA()
Base::methodB()
Derived::methodA()
Base::methodB()
```

5. List one advantage and disadvantage of late name binding. **(4 points)**

   Advantage: With late name binding (or dynamic binding) the appropriate version of a polymorphic method is decided at execution time. Thus, a polymorphic method can have multiple meanings based on their class-type context. The outcome of an operation depends on the type of the object on which it acts. Thus, the main advabtage is that late binding provides flexibility and more intuitive behavior.

   Disadvantage: Late binding prevent compile-time determination of the method to call at a give call-site. This decision now needs to be taken at run-time, which causes higher overhead and can slow down program execution.

6. Using templates, convert the following code fragment so that it can operate over any type. **(5 points)**

```
class A{
public:
   void set(int n);
private:
    int data;
};

void A :: set(int n)
{
    data = n;
}
```

   The solution is:

```
template <typename T>
class A{
public:
   void set(T n);
private:
    T data;
};

template <typename T>
void A<T> :: set(T n)
{
    data = n;
}
```

7. Describe why converting a base class pointer to a derived class pointer is not allowed in C++. **(4 points)**

   A derived class contains everything from the base class (except constructors and destructors), but can also add new fields and methods. Thus, code operating with a derived class instance may use these added class members. Since these new derived class methods are not present in the base class, this code may no longer work. Therefore, it is prudent to not allow a base class pointer to convert to a derived class.

8. In the (non-template) program in Question 6, overload the operator "<<" such that the following code will print the string "Value in A is 10". **(6 points)**

```
int main()
{
    A instA;
    instA.set(10);

    cout << instA;
}
```

   Possible solutions:
   1. Make 'data' *public* in class A.
   2. Declare the operator overloaded function as a *friend* function of `class A`.
   3. Define a new public `get()` method in class A to return the value of the private member 'data'.

   Using solution #2:

```
class A{
public:
  void set(int n);
private:
  int data;
  friend ostream& operator <<(ostream &os, const A& rhs);
};

ostream& operator <<(ostream &os, const A& rhs)
{
  os << "Value in A is " << rhs.data << endl;
  return os;
}
```

9. The Big-O notation does which of the following? (more than one option may apply)
(a) provide a precise prediction of program execution time
(b) provide a mechanism to compare one algorithm's efficiency with another
(c) provide an order of magnitude estimation of the algorithm's performance
(d) allow classification of algorithms by how their run-time changes with changes in input size
(4 points)

Does (b), (c), and (d).

10. What is the order (Big-O) of the following functions $f(n)$?
(a) $f(n) = n^3 + 4(log_2 n)^3 = \mathrm{O}(n^3)$
(b) $f(n) = 2^n + 3^n = \mathrm{O}(3^n)$
(4 points)

11. Explain with an *appropriate* example when (and why) should a destructor be virtual. **(6 points)**

Virtual destructors are useful when you can delete an instance of a derived class through a pointer to base class, and there is cleanup (such as de-allocating dynamic memory, closing files, etc.) that needs to happen at the end of using an instance of the derived class.

```
class Base
{
    // some virtual methods
};

class Derived : public Base
{
    ~Derived()
    {
        // Do some important cleanup
    }
}
```

Here, you'll notice that I didn't declare Base's destructor to be virtual. Now, let's have a look at the following snippet:

```
Base *b = new Derived();
// use b
delete b;
```

Here's the problem! Since Base's destructor is not virtual, it's Base that is called and not Derived. Therefore, all the important cleanup is not performed, which can create several resource leaks.

12. What is the best and worst case efficiency (in Big-O notation) of the following algorithms? When does the best and worst case happen? **(6 points)**

(a) Binary search over a sorted array
Best case: O(1) – the first item reached is the search item.
Worst case: O($log_2 n$) – any other case.

(b) Insertion sort
Best case: O(n) – when input is in sorted order.
Worst case: O(n^2) – Input in reverse sorted order.

(c) Bubble Sort
Best case: O(n) – when input is in sorted order.
Worst case: O(n^2) – Input in reverse sorted order.

13. Write a sorting algorithm for insertion sort over an array 'A' of size 'n'. **(5 points)**

`int insertionSort(A, n)`

```
int insertionSort(A, n)
{
    int i, j, temp;

    for(i=1 ; i<size ; i++){
        temp = A[i];
        for(j=i-1 ; j>=0 && A[j]>temp ; j--)
            A[j+1] = A[j];
        A[j+1] = temp;
    }

}
```