Solutions: Most of the solutions are from Edwin Rudolph's submission as he did an outstanding job on the assignment.

CS 6311 – Programming Languages I – Fall 2006
Assignment #2 (125 pts)
Due: 10:00pm, Wednesday 11/29/6

Directions: The answers to the following questions must be typed in Microsoft Word and submitted as a Word document by the due date. There are a total of 30 questions.

1. Explain how a primitive type such as an integer or a double can be considered an abstract data type. (2 pts)

   Primitive types are abstractions of how data is actually stored in hardware (or the underlying system, such as virtual machine, as may be appropriate). For example, the representation of a signed vs. unsigned integer may be different across platforms, or the particular method of representing floating-point numbers may also be different, but from a high-level language perspective, we are not concerned with these details. Thus, they are a form of abstract data types, albeit simple and limited.

2. Give two advantages of why providing accessors to private data within a class is better than making the data public? (4 pts)

   One advantage relates to the concept of coupling – providing accessors reduces the degree of coupling by ensuring that others who use a class only interact with it using the provided interface methods. By not allowing public/direct access to data members, the details and internals of how data is stored in the class can change without affecting other code that relies on the class – the developer of the class need only ensure that the provided method interfaces remain the same. A second advantage relates to consistency and reliability. By only allowing access to the data of a class via accessors or other methods, the class developer can be certain that such data can only be accessed/modified in known ways (i.e., those ways that are provided via the accessor methods).

3. Give a reason why a language would include the ability to make a data member protected. (Hint: think along the lines of is there an advantage of making the data member protected compared to public or private.) (3 pts)

   Protected data members can be seen(directly accessed) by derived classes inheriting from a base class. By providing a protected attribute it allows a derived class to manipulate the data directly without compromising the information hiding of a data member by making it public. Furthermore the base class is not even required to provide an accessor method so the protected data member can just be an internal data member of the class hierarchy.

4. Short-circuit logic, uses a form of lazy evaluation that only evaluates the Boolean expressions in a statement in a left to right order only until the truth value of the entire expression is known. Short circuit operations can be defined using standard if-else expressions. Rewrite the following Boolean expression as an if-else statement that represent the equivalent short circuit logic where only one subexpression is used per expression, e.g. `if (e1)`. `e1` and `e2` represent Boolean subexpressions.

   4.a. e1 and e2 (2 pts)

   ```
   if ( e1 ) {
   ```

```
    if ( e2 ) { // do something }
}
```

4.b. e1 or e2 (2 pts)

```
if ( e1 ) {
  // do some thing X
} else if ( e2 ) {
  // do some thing X
}
```

5. One of the goals of object-oriented programming is reuse. Explain how dynamic binding supports reuse? (3 pts)

   Dynamic binding supports reuse by enabling a fundamental advantage of OOP: polymorphism. By applying polymorphism, generic code can be written (i.e., a sort method) that can operate on, for example, a generic base class and then have that code apply to any future derivations of that generic base class. If the code written for the generic base class needs to be extended as appropriate to a derived class, the derived class needs only to ensure that it provides the same interface (i.e., via an overridden virtual method) to have those extensions be utilized by the generic code. Since it is dynamic and thus occurring at run-time, it enables extensions to be made not just to one's own code, but also to third-party code for which the source may not be available – in other words, the generic code does not need to be recompiled or modified in any way to take advantage of any such extensions.

6. Should abstract methods always be dynamically bound? Explain. (3 pts)

   Yes - if you are defining an abstract method, then, by its nature the method is not meaningful in the abstract class because it is not implemented; rather, it is just a placeholder or contract for any derived class. Moreover, there would be no way to actually implement a non-virtual abstract method because there would be nothing to execute, again because there is no implementation.

7. It is not possible to create an object of an abstract class, does this mean that variables of an abstract class cannot be declared? Explain. (2 pts)

   No, it does not. One must first differentiate between declaring a variable and instantiating an object. The fact that a class is "abstract" – meaning only that it is not fully implemented – does not affect its ability to represent a "type" in the program. This is in fact one of the major benefits of OOP – the ability to use abstract class types as polymorphic variables.

8. Many programming languages provide a for loop and a foreach loop, why would a language provide these two looping constructs that are nearly identical? (3 pts)

   Generally speaking, not all problems lend themselves to being expressed in the same format or terms, with alternative but semantically equivalent forms used for different types of problems. In the case of for vs. foreach, the foreach construct often results in a more reliable, and more readable, expression of a looping construct. With a typical application of a for loop, the programmer normally utilizes some counter variable along with a programmer specified terminating condition expression. The chances that a logic error or simple typo could occur in the usage of the counter variable and in the terminating condition are increased due to the fact that they are programmer-specified (a common example being an off-by-one error). With a foreach loop, a counter variable and terminating *condition are not used. Instead, some form of iteration interface is used by the foreach to iterate over a Collection object until the last element of the collection is reached. Thus the reliability is improved since the programmer does not need to maintain a counter and verify the correctness of the terminating condition. On the readability side, iterating over a collection is more natural with a foreach since the current item being accessed is automatically aliased in a reference variable. This removes the need to manually get access to the current item using the counter variable and some indexing notation or other method (also more prone to programmer error).*

9. What is the output of the following Pascal-like program for the following input:
   5, 4, 3, 2, 1
   The parameter x in procedure Q is pass by value and the parameter c is pass by reference. (12 pts)

```
Program P();
var a, b, c, d, e: integer;

        Procedure Q(x: integer; var c: integer)
        var b: integer;
        begin
            a = b - e;
            b = 3 * c;
            x = a + d;
            c = a + x;
            writeln(a, b, c, x);
        end

begin
    readln(a, b, c, d, e);
    Q(a, b);
    Q(b, c);
    Q(d, e);
    writeln(a, b, c, d, e);
end
```

-1 12 0 1
-1 9 0 1
-1 3 0 1
-1 0 0 2 0

10. Given the following C program where arrays are 0-based:

```
int value = 4;  // Declare a global variable and initialize it
int list[5] = {4, 3, 2, 1, 0};

void main(void)
{
    swap(list[2], value);
    swap(list[2], list[3]);
    swap(value, list[value]);
}

void swap(int a, int b)
{
    int temp;
    value = value – 1 ;
    temp = a;
    a = b;
    b = temp;
}
```

What are all the values of the variables value and list after **each** of the three calls to swap using the following parameter-passing methods?

a. Pass by value (4 pts)

value = 3, list = {4,3,2,1,0} // swap 1
value = 2, list = {4,3,2,1,0} // swap 2
value = 1, list = {4,3,2,1,0} // swap 3

b. Pass by reference (4 pts)

value = 2, list = {4,3,3,1,0} // swap 1
value = 1, list = {4,3,1,3,0} // swap 2
value = 3, list = {4,0,1,3,0} // swap 3

c. Pass by value-result (4 pts)

value = 2, list = {4,3,4,1,0} // swap 1
value = 1, list = {4,3,1,4,0} // swap 2
value = 3, list = {4,1,1,4,0} // swap 3

d. Pass by name (4 pts)

value = 2, list = {4,3,3,1,0} // swap 1
value = 1, list = {4,3,1,3,0} // swap 2
value = 4, list = {4,3,1,3,0} // swap 3

11. Given the following main method code in C#:

```
static int Main(void)
{
  A a = new A();
  B b = new B();
  a.R();
  b.R();

  a = b;
  a.R();
}
```

11.a. What is the output when executing Main given the following definitions of the classes A and B? (3 pts)

```
class A
{
  public virtual void P()   {  Console.WriteLine("A::P");  }
  public void Q()  {  Console.WriteLine("A::Q");  }
  public void R()  { P(); Q();  }
};

class B : A
{
  public new void P()  {  Console.WriteLine("B::P"); }
  public new void Q()  {  Console.WriteLine("B::Q");  }

};
```

Output:
A::P
A::Q
A::P
A::Q
A::P
A::Q

11.b. What is the output when executing Main given the following definitions of the classes A and B? (3 pts)

```
class A
{
  public virtual void P()   {  Console.WriteLine("A::P");  }
  public void Q()  {  Console.WriteLine("A::Q");  }

  public void R()  { P(); Q();  }
};

class B : A
{
  public override void P()  {  Console.WriteLine("B::P"); }
  public new void Q()  {  Console.WriteLine("B::Q");  }

};
```

Output:
A::P
A::Q
B::P
A::Q
B::P
A::Q

11.c. Explain what causes the difference in output in A and B? (3 pts)

The difference is due to the use of the "override" modifier for B.P() instead of the "new" modifier. Using "new" effectively disables dynamic binding.

11.d. What variable(s) in the Main method is(are) polymorphic? (2 pts)

The variable "a" is polymorphic since it can refer to instances of class A as well as any classes derived from A - in this case the class B. The variable "b" is not polymorphic since it can only represent instances of B.

12. Explain how dynamic binding of a method is enabled in C#? (3 pts)

    The method in the base class is declared with the "virtual" modifier, with the method (of the same signature) in the derived class declared using the "override" modifier. BOTH of these must be done in order for dynamic binding to work.

13. In C#, explain how a reference variable within a method is related to the stack and to the heap. (3 pts)

    A reference variable is a variable that stores (as its "value") a reference (memory location) to an object allocated on the heap. For a reference variable within a method, the reference variable would be located on the stack, but its "value" – the reference to a heap object – points to a memory location containing the data in the heap. Thus, when the method returns, the reference variable is popped off the stack. If the reference variable within the method was the only reference to the memory on the heap, then that heap memory would become garbage. If other references exist to that same heap memory, however (for example, a reference variable located in the calling method), then that heap memory would remain allocated.

14. Explain why it is not possible to create an object of an abstract class. (3 pts)

    Instantiation of an object of an abstract class type would not make sense due the nature of an abstract class; abstract classes are not fully implemented (by definition they are intended to be derived from rather than instantiated) and thus rely on derived classes for actual implementation.

15. For the programming assignment #4, some solutions represented the colors ("Red", "Yellow", etc) as an enumerated type while others put the colors in a string array by doing the following:

```
string[] colorArray = { "Red", "Yellow", "Blue", "Green" };
```

15.a. For the situation described, give two advantages of using an enumerated type over a string array. (4 pts)

Use of an enumerated type would provide strong type-checking (ensuring that only defined values are used, whereas, in this case any string value could be used whether it is in the array or not). An enumerated type also provides a set of constant values – with the string array, there is nothing to prevent the program from purposely or accidentally modifying the contents of the array.

15.b. If one was really adamant about using the string array approach, how could the string[] declaration be modified to replicate one of the advantages of the enumerated type given in your answer to step a. (2 pts)

Declare the array as a constant.

16. Suppose you wish to write a method that prints a heading on a new output page, along with a page number that is 1 on the first activation and that increases by 1 with each subsequent activation. Can

this be done without parameters and without reference to nonlocal variables in: (Please explain your answer.)

16.a. C# ? (2 pts)

No, because C# does not provide static function variables, which would be necessary to do this without parameters or non-local data.

16.b. Python? (2 pts)

No, because Python also does not provide static function variables.

17. C# and Java are very similar languages, however C# supports out mode parameters whereas Java does not.

17.a. Give a benefit provided by C# since it allows out mode parameters. (2 pts)

O*ut mode parameters provide a way for a method to "return" multiple values by assigning values to those out mode parameters.*

17.b. Give a potential drawback of providing out mode parameters. (2 pts)

One potential problem with out mode parameters would involve a case in which the same variable was passed in to a method that takes two out mode parameters. The resulting value that ends up in the variable passed in would depend on the implementation of the method and which of the two parameters is assigned a value last. This is an example of where use of out mode parameters could lead to difficult to detect side effects (particularly if the method is part of a library for which the source is unavailable).

18. Explain why naming encapsulations are important for developing large applications. (2 pts)

Large applications, or more likely, systems consisting of various components from potentially various sources/vendors are more likely to have names (classes, methods, modules, constants, etc.) that overlap. Encapsulating such entities within some form of name "space" (through a package, class, name-space, or even a naming convention) essentially qualifies individual names and thereby eliminates any ambiguities. This is functionally equivalent to any other type of naming encapsulation, such as surnames, area codes, Internet domain names, etc. whose purpose is to add specificity to a potentially ambiguous name.

19. What dangers are avoided in C# and Python by having implicit garbage collection? (2 pts)

Implicit garbage collection removes the "human factor" from the problem of cleaning up memory which is no longer needed. By handling this task automatically, the system does not rely on the programmer to explicitly deallocate memory (reliability) and thereby prevents memory leaks as well as preventing a program from (potentially) using all available memory.

20. Explain how the parent version of an inherited method that is overridden in a derived class can be called in the derived class in C#? (2 pts)

Use the base *reserved word. Suppose a method named myMethod in class P is overriden by a* *myMethod in class C. C.myMethod can perform its specific tasks and then call P.myMethod if needed by the statement base.myMethod().*

21. Explain the difference between method overloading and method overriding. (3 pts)

    Overloading a method means providing multiple implementations of a method with different signatures, i.e. one with no parameters, one with 1 parameter, another with 2 parameters. Overriding, on the other hand, would be a more specific (or replacement) implementation of a method, with the same signature, in a derived class. So a method in class P with the signature myMethod(int a) would be overridden by defining a method in a derived class C with the same signature, myMethod(int a).

22. True or False. An exception can be explicitly raised in C#. (1 pt)

    True.

23. Explain how an exception handler can be written in C# so that it handles any thrown exception. (2 pts)

    There are two ways: use a catch block with no arguments (meaning catch any exception), or use a catch block that specifies "Exception" as the argument (since Exception is the base class for all exceptions).

24. In C#, the code in the finally block is executed no matter what (whether an exception is thrown or not). Therefore, it would seem that one could just eliminate the finally block completely and put the code that would have been placed in the finally block after the last catch block. Explain the problem with this approach (i.e., why is there a need for a finally block structure?) (4 pts)

    If an exception is thrown for which no catch block exists (i.e., there exists only a catch block for a FormatException, but a DivideByZeroException is thrown), then execution will be interrupted and the exception will propagate up the call stack. In this case, any code following the try/catch block will never be reached. So, the code following the try/catch would only be executed if either no exception is thrown, or a handled exception is thrown. Thus the need for a finally block in the case where there is code that absolutely must be executed regardless of whether or not an exception is thrown, or whether or not a thrown exception is handled.

25. Given the following C# code snippets:

```
public void MethodA(int[] myArray)
{
        try
        {
1.              MethodB(myArray)
2.              PrintResults(myArray);
        }
        catch (FormatException ex)
        {
3.              Console.WriteLine(ex.Message);
        }
        catch
        {
```

```
4.                  Console.WriteLine(“Houston, we have a problem.”);
            }

5.          Console.WriteLine(“Size of array is:” + myArray.Length);
    }

    public void MethodB(int[] array)
    {
            try
            {
6.                  for (int i=0; i<array.Length; i++)
                    {
                            try
                            {
7.                                  Console.WriteLine(“Enter number to divide by: “);
8.                                  int divisor = Int32.Parse(Console.ReadLine());
9.                                  array[i] /= divisor;
                            }
                            catch (DivideByZeroException)
                            {
10.                                 Console.WriteLine(“Division by zero exception.”);
                            }

                    }
            }
            catch(ArgumentNullException)]
            {
11.                 Console.WriteLine(ex.StackTrace);
            }
    }
```

25.a. Give the line numbers of the next five lines that would be executed if line 9 threw a DivideByZeroException. (2 pts).

10, 6, 7, 8, 9 OR 10, 6, 2, 5if the array had no more items in it

25.b. Give the line numbers of the next five lines that would be executed if line 8 threw a FormatException. (2 pts).

3, 5

25.c. Can the code at line 4 in MethodA could ever be executed as a result of an exception being thrown in MethodB? Explain. (2 pts)

Yes. If an exception is thrown in MethodB that is neither a DivideByZeroException or an ArgumentNullException, then that exception would be propagated up to MethodA. For any exception other than a FormatException, the catch on line 4 would be executed.

26. Given the following C# code snippet:

```
List<string> pirateMovies = new List<string>();
```

```
string title = “: The Curse of the Black Pearl (2003)”;
pirateMovies.Add(title);
title = “: Dead Man’s Chest (2006)”;
pirateMovies.Add(title);
title = “: At the World’s End (2007)”;
pirateMovies.Add(title);

string name = “Pirates of the Caribbean ”;
Console.WriteLine(name);

string longTitle;
for (int i=0; i<pirateMovies.Count; i++)
{
    longTitle = name + (i+1);
    longTitle = longTitle + pirateMovies[i];
    Console.WriteLine(longTitle);
}

// HERE
```

26.a. How many objects are created by the above code? (3 pts)

A total of 11 objects are created.

5 objects are created from the code preceding the loop (pirateMovies list, 4 strings). 2 objects are created for each iteration of the for loop (as a result of the 2 string concatenations within it). With 3 items in the pirateMovies list, this will mean that a total of 6 objects are created as a result of the for loop.

26.b. How many objects are garbage at the // HERE line when the above code executes? (2 pts)

A *total of 5 objects are garbage at //HERE.*

The code preceding the loop does not result in any garbage objects since the string values from the title re-assignments are not lost due to the fact that they are added to the list. The first iteration of the loop results in 1 object as garbage, since initially there is no string object for longTitle. However, for each subsequent iteration, 2 objects will be garbage since the next iteration will result in the object currently in longTitle (from the previous iteration) being lost. Since the loop will execute 3 times, this will result in 5 objects as garbage.

27. C# doesn’t allow the assignment of default values for its parameters. Explain how in C# this functionality can still be incorporated into a program, even though it is not directly allowed by the programming language. (3 pts)

Use method overloading. Suppose there is a method FindItemsByPrice(int min, int max). If you wanted a method for which the parameters min and max default to 0 and 1000, respectively, then just overload the method with no parameters, i.e. FindItemsByPrice(), and have this version call the more specific one, passing those "default" values to it, i.e., FindItemsByPrice(0, 1000).

28. In computer science, design considerations often have a time/space tradeoff issue. Explain how this manifests itself in the eager and lazy approach to garbage collection. (3 pts)

With the eager approach, the trade-off is to favor the use of some additional time and space throughout execution to ensure that memory is freed as soon as it is no longer needed, thus attempting to keep as much memory available as possible, at all times. With the lazy approach, the trade-off is to favor the use of significant time, in the hope that garbage collection will occur infrequently (because modern systems include substantial amounts of memory), and then perform garbage collection at one time with the system having to perform the significant task of searching for unreferenced memory (with the two pass) or shifting memory blocks around to "pack" in-use memory (one pass).

29. Given a C# program, describe a specific situation where you would want a class to be sealed. (Do not tell me what a sealed class is, I want a specific situation where a sealed class is needed.) (3 pts)

A class that is simply a collection of static methods would be one place where a sealed class might be needed. In such a case, the fact that all methods are static means that the class is nothing more than a "module" – just a collection of methods and that is not taking advantage of any object-oriented features. Making the class sealed would prevent accidental (or malicious!) overriding of such methods. A example where this might be very important would be something like a library of mathematical routines (i.e., System.Math) where it would not make sense (and even dangerous) to re-define the method for calculating the square root.

30. Given the following C# code and a List of student objects called students:

```
class HeightComparer : IComparer<Student>   {
   public enum SortOrder { Ascending, Descending };

   private SortOrder order = SortOrder.Ascending;

   public HeightComparer(SortOrder theOrder) {
      order = theOrder;
   }

   public int Compare(Student stu1, Student stu2) {
      if (order = = SortOrder.Ascending)
         return stu1.Height - stu2.Height;
      else
         return stu2.Height - stu1.Height;
  }
}
```

and a List of Student objects called students. Give the line(s) of code to sort the students in descending order according to height. (3 pts)

students.Sort(new HeightComparer(HeightComparer.SortOrder.Descending));