Most of the solutions are from Edwin Rudolph (who did an outstanding job on the assignment) with a few modifications.

CS 6311 – Programming Languages I – Fall 2006
Assignment #1 (100 pts)
Due: 7:00pm, Monday 9/25/6

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 40 questions.

1. A system of instructions and data directly understandable by a computer's central processing unit is known as what? (1 pt)

   Machine language.

2. What is the name of the category of programming languages whose structure is dictated by the von Neumann computer architecture? (1 pt)

   Imperative.

3. Although PL/I and Ada were designed to be multi-purpose languages, in fact PL/I was considered to be the "language to end all languages", why is there such difficulty in creating a general purpose programming language applicable to a wide range of areas? (3 pts)

   The difficulty in creating an all purpose language lies in the fact that the variety of problems that we ask computers to solve do not all lend themselves to being ***easily or conveniently*** *expressed in the same way. Numerical processing involves different types of tasks than string processing; a program which computes trajectories of a spacecraft probably does not require extensive string processing facilities, just as a program to search for patterns in news feeds does not require extensive mathematical facilities. Attempting to create languages which provide facilities for solving all types of problems is that the language suffers from a feature bloat, not only making it difficult for programmers to effectively use it, but also making it more difficult to implement compilers and/or translators for. Moreover, since different types of algorithms are often more clearly represented in different ways, having a language (with a single syntax) will inevitably lead to some algorithms being expressed in unnatural ways.*

4. What are the primary tasks of a lexical analyzer? (2 pts)

   The task of a lexical analyzer or lexical analysis phase of compilation/interpretation is to recognize and identify tokens and symbols in the source code (which, prior to lexical analysis, is nothing more than one long string of characters) that represent recognized language entities such as keywords, operators, variable names, block delimiters, etc. This is at a lower level than syntax analysis, however -- the lexical analyzer does not verify whether tokens are being combined in a syntactically correct manner.

5. True or False. The first computing machine - the analytic engine was designed and built by Charles Babbage. (1 pt)

   False (designed, but never built).

6. A disadvantage of programming in a low-level language, such as machine code, is that the resulting code it is not portable to a different architecture. If this is such a big disadvantage, what is the advantage of compiling a program down to the architecture's machine language? (3 pts)

   The advantages of compilation to the architecture's native machine code are speed and efficiency. The fact that the program is being executed directly by the hardware (vs. some intermediate level of interpretation) eliminates bottlenecks inherent in translation from the higher-level language to the machine code during execution. Additionally, good optimizing compilers can further improve the efficiency of the resulting machine code program by eliminating redundant or repetitive code, as well as optimize the code to take advantage of specific features of the target architecture.

7. Explain how a design goal may affect the resulting syntax of a language. (2 pts)

   If a design goal of a language is to provide convenient, built-in features for string processing, then the design of that language's syntax would be heavily influenced by the ability to work with strings. As an example, Perl provides convenient, built-in regular expression capabilities for string matching and substitution, as its original design goal was to provide powerful text manipulation capabilities (perlintro(1) manpage).

8. What language started putting an emphasis on object-oriented programming? (1 pt)

   Smalltalk or C++. Simula 67 introduced the object concept, but it wasn't until Smalltalk and C++ that the emphasis for OOP started become more prevalent.

9. Explain why the C programming language is commonly used for systems programming. (2 pts)

   One reason is the fact that C is "closer" to the hardware machine code level than many other high-level languages. Thus, it is able to perform low-level operations necessary for systems programming that are not available in other languages (for example, direct manipulation of memory, pointer arithmetic, etc.). Additionally, its legacy of being the language used to develop UNIX and UNIX-like operating systems is a major factor in its use in systems programming, since the libraries for interfacing with the OS are also written in C.

10. In C#, integers can be assigned to double variables, but not vice versa. What design principle does this violate? (2 pts)

    Orthogonality, uniformity, and reliability could all be acceptable answers.

    This violates Notational Consistency (more specifically, uniformity). By auto casting one type of variable but not doing so for another, this creates a system in which the user must know which variables are auto-casted and which are not.

    It could be argued that this violates both the design principles of orthogonality and reliability. There is a lack of orthogonality, because the legality of these two types of situations is context-dependent and thus results in an exception to the otherwise strong typing of C# which would dictate that the LHS and RHS of an assignment must be of the same type (i.e., a widening *conversion is OK, but not a narrowing conversion). It violates the principle of reliability because strict type checking is not performed – instead, the language assumes that because it is a widening conversion, you will not care if an integer is assigned to a double. However, there may be cases where this is problematic because of precision issues associated with representing floating point numbers (i.e., division operations), or simply undesirable. Yet, because it would not be caught by the compiler, reliability is lessened due to the fact more effort is required from programmer to track it down.*

11. In C, parameters are pass-by-value unless the parameter is an array then it is pass by reference – what design principle does this violate? (2 pts)

    This is a violation of language orthogonality, because the fact that arrays are handled differently is an exception. This exception must be documented and remembered by programmers, thereby increasing the difficulty in learning the language as well as in debugging problems.

12. For a language that allows case-sensitive variable names, some would argue that this violates the readability design principle. Why is that? (2 pts)

    Since, in a case-sensitive language, the variables FooBar *and* Foobar *represent different variables, it could be argued that the language is less readable because additional effort and attention must be given to recognizing the specific case of the letters of a variable name. In addition to the extra effort required to simply distinguish between the two (or more!), it would be next to impossible to easily identify the specific purposes of two or more such variables without examining the code to determine how they are being used.*

13. Many newer languages have an ArrayList class that underneath is just an array. What is purpose of using an ArrayList, if one could instead just use array? (2 pts)

    The ArrayList provides an abstraction of a basic array, with many methods that can be used to make using the class a lot easier and prevent writing your own code.. In the case of the C# implementation, features are provided for iterating over the array as a standard collection, as well as making the array appear to be dynamic in size (since arrays are usually restricted to a fixed size defined at compile time). The ArrayList also provides the familiar method of indexing into the array for specific cells through its own methods.

14. True or False. In C#, when writing a class if you do not define a constructor the code will not compile. (1 pt)

    False

15. In C#, explain the benefit of using a property over creating get/set methods. Given example code that demonstrates the benefit described. (3 pts)

    Properties provide a more compact way of providing field get/set methods. Additionally, they provide the illusion and convenience of working with a field directly, but provide reliability in the fact that the get/set operations are being handled in a controlled manner (fields remain private). Consider the following code snippets, which accomplish the same task:

```
// EXAMPLE 1
class GasStation
```

```
{
  // … some code …

  private int pumpCount;
  // property to get/set the pumpCount field
  public int PumpCount
  {
      get { return pumpCount; }
      set { pumpCount = value; }
  }
}

GasStation gs = new GasStation();
gs.PumpCount = 5;
gs.PumpCount++; // This is where the ease of use of Properties comes in.
Console.WriteLine(gs.PumpCount);

// EXAMPLE 2
class GasStation
{
  // … some code …

  private int pumpCount;
  // method to get the value of the pumpCount field
  public int GetPumpCount
  {
      return pumpCount;
  }
  // method to set the value of the pumpCount field
  public void SetPumpCount(int pumpCount)
  {
      this.pumpCount = pumpCount;
  }
}

GasStation gs = new GasStation();
gs.SetPumpCount(3);
gs.SetPumpCount(gs.GetPumpCount() + 1);  //Compare this to the corresponding increment above
Console.WriteLine(gs.GetPumpCount());
```

While both of these are not drastically different nor does the property version drastically reduce the amount of code required to accomplish the task, the property version provides a more natural way of accessing the pumpCount attribute. Using properties for these tasks also more clearly separates accessor/mutator operations while more narrowly defining methods as the part of classes where more complex types of tasks are performed.

16. Visual Studio development allows method headers to be XML style comments. What is the advantage of writing method headers as XML code? (2 pts)

    One advantage is that it allows Visual Studio to parse and use them to display documentation as part of the coding process, and thereby encourages documentation while coding. Moreover, the fact that these headers use a common set of XML tags ensures consistency in comments across a project as well as for any code developed in Visual Studio. Another advantage is that the XML format, being an open standard, can be easily parsed (using external tools) and displayed in other formats or used for other purposes if necessary.

17. True or False. The main method in C# must be named: static void Main(). (1 pt)

    False - Main *can also return an* int *and accept a string array for run-time arguments.*

18. Explain why in a declarative language, control constructs like while loops are not necessary. (2 pts)

In declarative languages, the programmer is generally not responsible for program flow control; this is provided by the underlying system. As is indicative of the adjective "declarative," these languages are focused on the "programmer" providing information and facts to the system (declarations of data and rules). Depending on the data provided and the defined rules, the system will determine the appropriate actions to take.

19. What is considered to be the machine language for the following:

    19.a. purely interpreted language. (1 pt)

    The high-level language itself.

    19.b. a hybrid language. (1 pt)

    The "intermediate" code resulting from the high-level language compilation.

    19.c. a compiled language. (1 pt)

    The hardware architecture's native machine language.

20. Compilation is part of the translation process for which of the language translation methods? (3 pts)

    Compiled and hybrid.

21. Syntax errors can occur for which of the language translation methods? (3 pts)

    All methods – compiled, interpreted, and hybrid. If a given statement is not valid syntax, the compiler or interpreter cannot understand it and thus will produce an error message. The difference would only be in the timing of the syntax error being raised: for compiled or hybrid, this would be at compile time, whereas for interpreted, it would be at run-time.

22. In both the purely interpreted and hybrid approach a virtual machine is used to execute the program, but the hybrid approach allows for faster execution – why? (3 pts)

    Hybrid means the high-level language is compiled into an intermediate language. This is faster because the (expensive) source code translation steps of lexical, syntactical, and semantic analysis are performed only once, during compilation. The intermediate code can then be executed directly by the VM. For purely interpreted languages, the expensive translation steps must be performed every time the program is executed. Moreover, repetitive program segments such as loops must be decoded and translated for each loop iteration, adding additional expense to pure interpretation.

23. The disadvantage of using a virtual machine is that the program executes slower compared to a purely compiled program. Why then do so many languages use a virtual machine? (2 pts)

    The primary advantage is portability. Providing a virtual machine means that, for a program to run on multiple software and/or hardware platforms, the language need only provide a virtual machine for each platform. This is in contrast to purely compiled languages, where a complete compiler must exist for each platform and which requires programs to be re-compiled for each platform. It is easier to provide a virtual machine for multiple platforms than it is to provide compilers (and the vast menagerie of system-specific libraries often needed) and to recompile programs for each desired target platform. The speed of modern computers coupled with the wide variety of computing platforms in existence has also resulted in more of an emphasis on this portability over speed/efficiency of execution.

24. Explain how in the hybrid approach to language translation that ahead-of-time compilation can provide for faster execution of the program than just-in-time compilation. (3 pts)

    Ahead-of-time (AOT) compilation would provide for faster execution because portions of the code have been compiled and (and may be cached for future program runs). AOT may just keep around the compiled portions created by the JIT compiler during its execution, so subsequent invocations of the program are faster, or the AOT approach may compile the entire program before execution and keep the compiled portions around as is typically done for embedded systems to reduce the run-time overhead. Whereas, with just-in-time compilation, compilation occurs dynamically at run-time (with portions of the program being compiled as they are encountered, i.e., for loops) and for each time the program is executed.

25. Explain why it is easier to create a debugger for an interpreted language compared to a purely compiled language. (2 pts)

    The reason for this is that in a purely compiled language, the compiler translates the high-level source language into hardware machine code, resulting in a completely new program (albeit semantically equivalent). The difficulty in debugging a compiled program lies in the fact that the machine-level instructions almost never correspond directly to a single instruction in the high-level language (high-level language instruction may correspond to several machine-level instructions), and so there would be no way for a debugger to relate the machine code back to the original source code. The only way to allow for debugging compiled programs is to insert various types of debugging symbols in the resulting machine-level program that would define the relationship to the original high-level source.

    For interpreted languages, the high-level language is the "machine-code" (the virtual machine provided by the interpreter executes the high-level source directly), so there is no need to include any debugging symbols or mappings to relate high-level source to machine-level code since they are one in the same.

26. What is the lifetime for a static variable found within a method? (2 pts)

    The lifetime would be for the complete duration of the program's execution.

27. When and where in memory does the address binding occur for an explicit heap dynamic variable within a method? (4 pts)

With explicit-heap dynamic variables, it is necessary to distinguish between the pointer or reference variable that stores the memory address from the heap (or a reference to a class instance which is allocated from the heap) and the actual memory allocated from the heap. During execution, when a method containing an explicit heap-dynamic variable is entered, a pointer or reference variable would be placed on the stack. However, the binding of a memory address from the heap to this reference variable would not occur until the statement which explicitly requests heap memory (i.e., C malloc(), *C#/Java* new*) allocation is encountered (which might or might not be part of the same statement which declares the variable).*

28. Give an example of a problem that can occur with dynamic type binding? (2 pts)

In the case where a programmer chooses (unwisely) to use single-letter variable names, a simple typographical error in coding could result in an undetected data loss. Since a language with dynamic typing would allow the assignment of a string value to a variable containing a numeric value, there would be no way for this error to be detected by the interpreter and thus adds to the difficulty on the programmer's part in debugging a program and discovering such an error.

29. Explain how to access the classes within a namespace in both C# (2 pts) and Python (2 pts).

In C#, the programmer may choose to provide a fully qualified reference, as in the following statement which utilizes the WriteLine() method from the System.Console namespace:

```
System.Console.WriteLine("Foo bar splat.");
```

Alternatively, the programmer may choose to utilize a "using" statement, which instructs the C# compiler to automatically "import" all of the classes within a given namespace into the current referencing scope, thus removing the necessity of providing a fully-qualified reference, as in the following statements:

```
using System;

/// ... more code here ...

Console.WriteLine("Foo bar splat.");
```

In Python, there is only one way of accessing members of a namespace:

```
import math
print math.sqrt(4)
```

Or to have an abbreviated syntax you can do the following:

```
from math import sqrt
```

Then you have direct access to sqrt
```
print sqrt(4)
```

If you want to have direct access to all the methods within a module, in this instance the math module, you do the following:

```
from math import *
```

30. In Python, all attributes are public by default, explain the mechanism that creates “private” attributes, and why this approach doesn’t strictly enforce privacy. (3 pts)

Python provides "private" attributes via means of variable name mangling. When a attribute variable is named with two underscore characters prefixing the name, Python will perform this name mangling such that it adds additional text to the name. Within methods of the class (but not outside the class), the variable may be accessed using the original name (i.e., self.__fooBar), and Python will perform the translation automatically. However, there is no such thing as a truly private attribute, since a determined programmer may directly access the variable outside the class, provided she is familiar with the specifics of Python's mangling scheme (which are, of course, publicly available since Python is open-source).

As in other aspects of Python programming, the "private" name mangling feature mainly serves to help keep programmers honest and assist them in following the accepted conventions.

31. Explain why dynamic type binding is so expensive? (2 pts)

Dynamic type binding is expensive because the interpreter must perform type checking dynamically at run-time. Additionally, internal "house-keeping" operations must be performed at run-time for the interpreter to keep track of the current type of the variable as well as handle the differing memory allocation and de-allocation needs when the variable's type changes (consider, for example, switching from an integer type to an arbitrarily long string type). Depending on the frequency of such changes, this could potentially be extremely costly.

32. Describe a situation where the lifetime of a variable exceeds its scope, i.e., describe how a variable may go out of scope, but still be allocated to a memory address. (2 pts)

A common example of such a situation would be that of a method or function call in a language such as C or Java. Suppose the function foo()*, which contains an integer variable* quux*, calls the function* bar()*. Since* quux *is a local function variable of* foo()*, it is outside the scope of* bar()*. The lifetime of* quux *is however long* foo() *takes to complete. Since* quux *is a variable allocated off the stack, it will remain allocated and in memory while* bar() *is executing, and will continue to be in memory until foo() completes.*

```
void foo(void) {
  int quux = 1;
  bar();
  // quux is still in memory
  printf("%d", quux);
  return;
}
void bar(void) {
  // some code... (quux not in scope since it is local to foo())
  return;
}
```

33. Both implicit type binding and dynamic type binding can cause problems - if you assume you think you know what the type of a variable is. Explain how this problem manifests itself in both situations. (4 pts)

In the case of implicit type binding, simple oversights on the part of the programmer are more *difficult to detect. For example, in FORTRAN, a variable with the name* I *is implicitly typed as an integer value. Then if you explicitly assigned I to be another type, e.g., a string, if you make an assumption that it is an integer, it could create programmer errors, for example, you may end up printing out the wrong data, a string instead of an integer value.*

For dynamic type binding, there is a similar situation, although perhaps even more amplified than with implicit binding. A language such as Python, which is dynamically typed, would allow a variable that was initially assigned an integer value to later be assigned a string value. Again, a simple programmer typographical error could result in a loss of data that would go undetected by the interpreter.

In both cases, additional burden is placed on the programmer to be careful about assignment statements.

34. In C#, explain how the static modifier for a method name differs from than the static modifier for a variable. (3 pts)

    There is no difference in C# between the meaning of a static method or static variable (field), since they are both ***class*** *methods/variables (must be referenced by referring to the class name) vs.* ***object*** *methods/variables (which must be accessed from a class instance).*

35. Explain why in a dynamically-scoped language it is not possible to perform static type checking. (3 pts)

    Static type checking is not possible because there is no way for the type of a particular variable to be determined before run-time, since the variable X may refer to integer as a result of one particular execution path, but for another execution path, it may refer to a string. Thus, because the specific "X" variable in the current scope depends on the order of execution, there is no way to perform static type checking.

36. In what part of memory are constants bound? (1 pt)

    Data segment.

37. In C#, you define constants using the const or readonly attribute, explain a situation where one would want to define a constant using readonly instead of const. (2 pts)

    Since the value of a const *variable is defined at compile-time, it will always be the same for every invocation of the program.* readonly *allows for a constant variable whose value is defined at run-time. So, a* readonly *constant would be appropriate where an enforced constant variable is needed but where that constant needs to be initialized at run-time rather than at compile time.*

38. Both constants and enumerated types can make a program more readable. The enumerated type is essentially defined as a set of constants. If this is the case, they why use an enumerated type. Give at least two reasons. (4 pts)

    Enumerated types offer at least 2 advantages over regular constants:

    - Additional readability. The fact that a set of enumerated values are qualified (i.e., EnumTypeName.ConstantValue) adds to the readability of enumerated values because the type name

for the enumeration provides some indication as to what it is intended to represent. Also, they are more readable since the value itself is meaningful.

- Reliability. Constant values are just regular data types such as integers or doubles. Thus, it is not possible to restrict the range of acceptable values. For example, a method expecting a value from a particular set of named constants has no way of ensuring that the values passed for a particular parameter are from that set of constant values unless it explicitly checks the parameter for each possible value. Enumerated types provide this restriction since they are user-defined types and thus type checking can be performed by the compiler to ensure that only values within the enumeration are used.

39. Given the following C# method:

```
void fun(void)
{
    int a, b; /* Definition 1 */
    …
    while (…)
    {
            int a, c, d; /* Definition 2 */
            … ← Point 1
            while (…)
            {
                    int c, d, e; /* Definition 3*/
                    … ← Point 2
            }

    } // end while
    … ← Point 3
}
```

For each of the four marked points in this function, list each visible variable, along with the number of the definition state that defines it. (6 pts)

Point 1: a (2), b (1), c (2), d (2)
Point 2: a (2), b (1), c (3), d (3), e (3)
Point 3: a (1), b (1)

40. Given the following Ada-like program:

```
procedure Main is
    X : Integer;
    procedure C; -- This is a forward declaration of C which allows A to call it

    procedure A is
      X : Integer;
      procedure B is
        begin -- B
        …
         end; -- of B

      begin -- A
      …
      end; of A
```

```
procedure C is
  begin -- C
  …
  end; -- of C
begin – main
…
end; -- main
```

Given the execution is in the following order: main calls A, A calls B, and B calls C - give the declaration of X used in A, B, and C with static scoping (3 pts) and dynamic scoping (3 pts).

For static scoping:
X declared in A is used for A, X declared in A used for B, X declared in Main used for C

For dynamic scoping:
X declared in A is used for A, B, and C