Programming Languages - Lab 1 Answer Key

1. Describe some ways in which programming languages differ from natural languages. Who is the audience for a programming language? Is it the same as the audience for a program?
   Programming languages (PL) differ from natural languages (NL) in a few fundamental ways. Though both are used to communicate ideas, PLs are usually limited to communication of algorithmic ideas, while NL has a very wide range of communication. This difference is deeply connected with two other key differences between PL and NL. First, changes in NL occur though natural processes (i.e. evolution), while changes in PL are deliberate and planned in such a way that typically ensures backward compatibility (old constructs retain their meaning). Second, NLs are extremely flexible, operate outside of a formal grammar, ambiguity is allowed, and a sentence can have multiple meanings. Conversely, PL always operates inside a formalized grammar, are often standardized, and every semantically correct sentence has a single meaning. These differences directly reflect the languages' intended audiences. The audience of a NL is people; their flexible brain allows for the interpretation(s) and ambiguities of NL. The audience of a PL consists of programmers, translators and machines; the latter two designed only for such rigid communication, and the former has learned by choice. A program has a larger audience than a PL. The audience for a program consists of people, programmers, and machines.

2. Explain static versus dynamic memory.
   Static (stack) memory is automatically reserved at the beginning of a program's runtime. The programmer does not explicitly allocate or free memory in the course of the program, as it is allocated for the entire lifetime of a program. Dynamic (heap) memory occurs at runtime and is managed by the programmer (e.g., calloc, maloc, free) during run time. Except in the case of dangling pointers, the lifetime of dynamically allocated memory is determined by whether the block of code responsible for the allocation is still active. ("Difference between static memory").

3. Give some different kinds of examples of abstraction in programming languages.
   Some examples of abstractions are:
   - Types (abstractions of data)
   - Functions (abstractions of processes)
   - Objects (abstractions of associated data and processes)
   - Pointers (abstractions of addresses in memory)
   - Variables (abstractions of values in memory)
   - Control Structures (abstractions of common memory actions)

4. What is meant by *application domain*? A *language paradigm*? How are the two related?
   An application domain is an area in which a computing application could be used to solve a problem. Some examples of unique application domains include scientific computing, management information systems, artificial intelligence, a systems domain, and a web-centric domain.

   A language paradigm is a way in which a language is able to express its algorithms. The problems an application domain will need to solve will dictate which language paradigm will be most useful in solving such problems and will therefore determine which programming languages are best suited for the application domain. The language paradigms include imperative, object-oriented, functional, declarative, event-driven, and concurrent.

5. Why is providing language features to deal with exceptions important?
   It is important for a program to properly handle common errors that could occur during runtime. One way of accomplishing such a task is to implement exception handling. According to Oracle, exceptions have three main advantages. The first is that it allows code that handles errors to be separate from normal code, allowing more organized work ("Advantages of Exceptions"). The second advantage is that is allows errors to be reported only by the method that is interested in them, as long as the methods in which the error occurred has the appropriate throw clause ("Advantages of Exceptions"). The third advantage is that it allows errors to be grouped or differentiated based on their type ("Advantages of Exceptions"). For example all I/O errors can be handled by the proper I/O exception or an exception can be set up to handle only specific errors, such as a read failure ("Advantages of Exceptions").

6. Look at the criteria for language design on p. 15 and the explanations that follow them. Describe orthogonality in your own words and provide some examples (or counterexamples). Pick two additional design criteria to describe and provide a few examples. You *must* go beyond what is provided in the Tucker-Noonan chapter.
   "An orthogonal language has relatively small set of primitive constructs that can combine in a relatively small number of ways to build data and control structures, where every possible structure is legal." (Sebesta, p. 10). An example of a language feature which is not orthogonal is that an array can contain any data type except void in C. Another counterexample is the use of '=' in both assignment and in equality in C. Another counterexample to orthogonality in C is that parameters are passed by value but arrays are passed by reference. Another counterexample is the two IBM mainframe instruction to add integers which reside in either memory or in registers: "A Reg1, memory_cell" computes contents(reg_1) + contents(memory_cell) whereas "AR Reg1, Reg2" computes contents(reg_1) + contents(reg_2). An example of orthogonality would be VAX instructions to add integers with reside in either memory or registers: "ADDL operand1, operand2", which simply takes contents(operand1) + contents(operand2). A lack of orthogonality leads to exceptions to the rules of the language.

   Simplicity and readability speaks to how easy a program is to read or write. For example, a PL based on whitespace is not readable. Arguably, both have more to do with the styles and skills of individual programmers than the language as a whole, but to some extent, certain languages tend to be easier to read and write than others.

   Clarity about binding deals with pairing of two elements. All the data types of a language are bound to keywords. For example, character is bound to *char* and integer is bound to *int*. Hence, when the keywords *char* or *int* are used in Java, the compiler knows you are referring to a character and integer respectively. Another example of binding is when you are implementing a function in a programming language. In order to call your function, you give it a name that binds it to the function. Hence, whenever you wish to call that particular function, you simply refer to the name you have given it.

   Reliability ensures behavior is consistent and independent of the hardware. The use of type checking and well-defined syntax and semantics are examples.

   Support in a PL is determined by how accessible it is. It can include, for example, clear user documentation, tutorials, APIs, and a well-formed community. Additional examples include the financial cost to use and whether it is supported on multiple platforms.

7. What is meant by *nonconformant* when a standard discusses a language feature supported by a particular compiler? Give an example.
   A nonconformant language feature of a compiler is its capability to compile code that does not meet the standard requirements of the language ("Nonstandard Behavior"). An example of this is Microsoft Visual

Studio 2013. It is a compiler that has both nonconformant language features and extensions upon the C and C++ standards, allowing it to compile code that other compilers would reject ("Nonstandard Behavior"; "Microsoft Extensions to C and C++"). Although this is beneficial to projects that will be compiled only on Visual Studio 13, it also means that the source code is not portable to any other compiler.

8. Look at the following large list of programming languages. For each of the following languages, use the provided link or other resource to learn something about it. Write, in your own words, a brief summary of its features, paradigms, and relationship with languages that preceded or followed it.
    - Java

      Java is an object-oriented and event-driven language. It uses a Java Virtual Machine as an intermediary between the Java compiler and an architecture-specific interpreter. This makes changes to the language easier, as there is a single compiler. The use of the Java Virtual Machine also makes the language usable on a wide range of platforms. Due to the languages wide use, it has a very large number of libraries available. ("Learn About Java Technology")

    - Scheme

      Scheme is a functional programming language. Scheme was an innovator in implementation of powerful computing ideas like recursion and continuations. Schemes is characterized by its relatively concise syntax and wide expressive power. The use of functions and recursion is integral to programming in Scheme. Scheme was preceded by Lisp, and is a predecessor of Common Lisp.

    - Another programming language of your choice

      Prolog, short for "Programming in Logic", was developed in the 1970's by Alain Comerauer with the idea in mind of combining logic and programming. Originally intended to be used in natural language processing, it has also found uses in expert systems, theorem proving and other areas of artificial intelligence. One interesting feature of Prolog is it only has a single data type, a term. Prolog's uniqueness comes from its logical/declarative paradigm. That is, Prolog describes what is to be accomplished, rather than how it is to be accomplished. Prolog accomplishes this by describing relations set by facts and rules. Despite creation in the 1970's, Prolog is still used today. It has left a legacy in extensions, uses as middleware, and influence on similar declarative languages, like Visual Prolog

Stack Overflow. *Difference Between Static Memory Allocation And Dynamic Memory Allocation*. [online] Available at: https://stackoverflow.com/questions/8385322/difference-between-static-memory-allocation-and-dynamic-memory-allocation. Accessed 26 July 2020.

Oracle: Java Documentation. *Advantages of Exceptions*. [online] Available at https://docs.oracle.com/javase/tutorial/essential/exceptions/advantages.html. Accessed 26 July 2020.

Sebesta, Robert W. 2010. *Concepts of programming language, 9th edition.* Boston, Addison Wesley.

Microsoft Developer Network. *Microsoft Extensions to C and C++.* [online] Available at https://docs.microsoft.com/en-us/cpp/build/reference/microsoft-extensions-to-c-and-cpp?view=vs-2019. Accessed 26 July 2020.

Microsoft Developer Network. *Nonstandard Behavior.* [online] Available at https://docs.microsoft.com/en-us/cpp/cpp/nonstandard-behavior?view=vs-2019 Accessed 26 July 2020

Phillips, Winfred. *Introduction to Prolog.* [online] Available at http://www.mind.ilstu.edu/curriculum/protothinker/prolog_intro.php Accessed 26 july 2020.