Aidan Leuck

Homework 1

P16:3 – Why are there so many programming languages?

There are many reasons why there are so many different programming languages in existence. First of all we have found better ways to do things over time. For this reason older languages that did not implement these new features have fallen out of common use. Secondly, different programming languages are good for different things. Some languages may be easy to compute complex math problems while another may be really good for scripting. Because there are so many different problems one may want to solve by using a computer, many different flavors have arisen in order to make solving said problem easier. Personal preference is yet another reason why there are so many languages, every person has their own way of thinking through a problem and certain languages cater to different ways of thinking. There are a few other reasons why there are so many programming languages, but those listed above have the most impact in my opinion.

P16:4 – What makes a language more successful?

As with the previous question there is not a clear cut reason why one language is more successful than another. One of the most obvious reasons are that a language may have a more complete feature set than one another. An example would be a functional language such as C vs Java which has object oriented programming. Secondly, you have portability, a language that can run on many different types of hardware is clearly a better choice than a language that can only run on one or a few different chips. This portability goes hand in hand with standardization of the language, meaning does it implement libraries that are deemed essential (I/O, strings). The quality of the compiler is also another reason that a language could be more powerful over another. Clearly using a language that generates more efficient machine code would be beneficial due to the decreased run times. Sometimes it even comes down to just economics. In other words who sponsored or was using the language. If a influential company or government agency is using a language to develop their products then programmers may have no choice but to use the language. Also, for products that already have a large code base in a certain language it is not worth the money or time to rewrite the language in a more modern language.

P 16:6 – What distinguishes declarative languages from imperative ones?

The difference between a declarative language and imperative language are in how a programmer specifies something to be done. Declarative programming isn’t really concerned with how something is implemented all of that happens in the back end of the language. You are telling the computer what you want and it gives you those results. SQL and HTML are two good examples of a declarative language. Using SQL as an example you say SELECT * FROM TABLE. You aren’t telling the computer how to select that data, but you are telling it what you want it to select. In an imperative language the programmer implements how something is done. An example is Java when you say if(condition) then do this. You are telling the computer what to do if a condition is met.

P25:11 – Distinction between interpretation and compilation (Advantage/disadvantages of both)

The difference between interpretation and compilation are that compilation is when a program is transferred from a high level programming language to its target program. This is usually machine code. Interpretation on the other hand does not convert the source program into machine language. Instead it uses a virtual machine to run the program in the same language that a program was written. The main advantage of a compiler over an interpreter is efficiency because once a program is compiled no decisions have to be made at run time. An interpreters' main benefits are that is often easier to debug programs using an interpreter because the interpreter is ran in the same programming language.

P25:12 Is Java Compiled or Interpreted?

Java is both compiled and interpreted. Java compiles its program into bytecode which is interpreted by the Java Virtual Machine. The compilation into bytecode is why Java is able to be used cross platform so easily, but without the JVM the bytecode would not be able to be understood by the machine.

P36:24 - Describe the form in which the scanner is passed from the scanner to the parser; from the parser to the semantic analyzer; from the semantic analyzer to the intermediate code generator.

The scanner reads the literal characters of the program and groups these characters into tokens. It also removes the extraneous information such as comments. The overall goal of the scanner is to make it easier for the parser. These tokens are passed on to the parser which figures out the order in which these characters should be interpreted. Additionally it will find errors if a token is incorrectly written according to the programming languages syntax. The semantic analyzer on the other hand uses the information from the parser to actually discover what a program is supposed to do. It does more advanced error checking that goes beyond enforcing the languages syntax. The semantic analyzer uses the parse tree in order to do this work by traversing it inorder. The result of the semantic analyzer is an annotated syntax tree that is passed on to the code generator which uses this tree in order to generate assembly language.

p38:1.1: 1.1 Errors in a computer program can be classified according to when they are de-
tected and, if they are detected at compile time, what part of the compiler detects
them. Using your favorite imperative language, give an example of each of the
following.

a) Lexical Error – An example of a lexical error could be doing something like int y = 4x. The scanner would discover this as a lexical error because you can not assign an integer non numeric values.

b) Syntax Error – An example of a syntax error in java could be while (x != y { do something here } This is a syntax error because the conditional statement is not enclosed in parentheses. A while loop is always while (condition) {code block}.

c) Static semantic error – An example of a static semantic error is could be int x; and then doing x++; This is a static semantic error because it is uninitialized.

d) Dynamic semantic error – Dynamic semantic errors can't be discovered until run time. An example of this in java is an index out of bounds exception when accessing an array element.

e) Error that can't be found by compiler (Not a bug). One thing that comes to mind for this question would be trying to modify a string in Java. Once a string has been set it is impossible to directly access and modify that string. For example String s = "This is a string" and doing s[1] = "D". This is not possible to do in Java but it won’t necessarily throw an error in the compiler because the statement is technically valid.