Homework 1: Essay

Introduction and Overview

A critical part of being a professional computer scientist is communication at many levels, from technical discussions with peers to non-technical exposition for clients and others outside of the field. This course will include several essays, which will be technical in nature, but 25% of your grade on each essay will be grammar and professional writing. Grading on professionalism is at the discretion of the instructor or the other computer science faculty in the department. Our grading standard for grammar in this class will be the Purdue OWL online grammar guide. You are free to use grammar checking software similar to spell checking software (and many modern editors have this built-in already), however, the grammar software is not authoritative for the purposes of grading.

In addition to addressing the question asked, please also try to learn something new and to enjoy conveying this information! This ability to communicate technical ideas in an interesting and understandable tone is one of the strongest skills you will use in a future career.

Assignment Details

For this assignment, you will write a 1-2 page paper on one of the following topics. I will expect at least 3 sources beyond your textbook, with proper citations for relevant facts from those references. Of course, for this assignment, you are welcome to use web or book resources of any type; just be sure to use reputable sources and include them in your bibliography. Your paper and your citations should adhere to the MLA style guide, and again our reference source for grading will be the Purdue OWL MLA Formatting and Style Guide

Please note that since all of these topics have at least some component that is opinion-based, I expect a clear thesis statement which summarizes your opinion or finding, which must then be supported by relevant facts and reasonable arguments in the body of your essay.

- Consider the programming language with which you are most familiar, and list three things about it that you find frustrating or counterintuitive and wish had been differently designed. Now look into its history a bit: why was it designed the way it was? How would you fix these issues if you had a chance to do it over? Would there be any negative consequences, for example in terms of compiler complexity or program execution speed, or do you think other languages show these changes are work it? Overall, do you think it is worth redesigning it, or are there good reasons to keep things the way they are?

- Familiarize yourself with the history of Java and C# using online (or other) sources, including the conflict between Sun and Microsoft over Java standardization. Some have claimed that C#, at least partially, is Microsoft's attempt to kill Java. Defend or refute this claim, with relevant facts to back up your opinion.

- C has a reputation for being a relatively "unsafe" high-level language. Why is this the case? Is this a feature, or a design flaw? What utilities exist to enforce safer practices in C? Are these sufficient "patches" to C's security vulnerabilities, or does the language deserve its reputation?
- Java contains *checked exceptions* as a language feature, which are supposed to make code safer. The lead architect of C#, Anders Hejlsberg, deliberately did not want to include them in C#. Why would the language architect want to make the language less safe?
- Both C++ and Java support generic types. C++ supports generics with templates. Java supports generics with parameterized or generic types. Both approaches only enforce generic type safety at compile time, and neither approach incurs any runtime overhead. However, the two approaches are quite different. How are they different, and more importantly, why did the C++ designers implement generics one way while the Java designers implement generics another way?
- Rust is often characterized as a "safe" alternative to C for systems programming. What does Rust do differently than C, and how is safety enforced? What are the tradeoffs Rust makes to do this?