CMPS 401 – Survey of Programming Languages

Current Course Description: Prerequisite: Computer Science 390. Involves the formal study of programming languages, specification, and analysis in terms of data types and structures.

Minimum Topics:
- History of programming languages
- Brief survey of programming paradigms
  - Procedural languages
  - Object-oriented languages
  - Functional languages
  - Declarative, non-algorithmic languages
  - Scripting languages
- The effects of scale on programming methodology
- The concept of a virtual machine
- Intermediate languages
- Comparison of interpreters and compilers
- Machine-dependent and machine-independent aspects of translation
- Introduction to syntax and semantics
- The conception of types as a set of values with together with a set of operations
- Declaration models (binding, visibility, scope, and lifetime)
- Overview of type-checking
- Procedures, functions, and iterators as abstraction mechanisms
- Parameterization mechanisms (reference vs. value)
- Type parameters and parameterized types
- Data types
- Expressions and assignment statements
- Control statements (selection and iterative)
- Subprograms in programming languages

Learning Objectives: Students will be able to:
- Demonstrate understanding of the evolution of programming languages and relate how this history has led to the paradigms available today.
- Identify at least one outstanding and distinguishing characteristic for each of the programming paradigms covered in this unit.
- Evaluate the tradeoffs between the different paradigms, considering such issues as space efficiency, time efficiency (of both the computer and the programmer), safety, and power of expression.
- Describe the importance and power of abstraction in the context of virtual machines.
- Explain the benefits of intermediate languages in the compilation process.
- Evaluate the tradeoffs in performance vs. portability.
- Compare and contrast compiled and interpreted execution models, outlining the relative merits of each.
- Describe the phases of program translation from source code to executable code and the files produced by these phases.
- Explain the differences between machine-dependent and machine-independent translation and where these differences are evident in the translation process.
- Explain formal methods of describing syntax (backus-naur form, context-free grammars, and parser tree).
- Describe the meanings of programs (dynamic semantics, weakest precondition).
• Identify and describe the properties of a variable such as its associated address, value, scope, persistence, and size.
• Discuss type incompatibility.
• Explain data types: primitive types, character string types, user-defined ordinal types, array types, associative arrays, record type, union types, point and reference types
• Demonstrate different forms of binding, visibility, scoping, and lifetime management.
• Defend the importance of types and type-checking in providing abstraction and safety.
• Explain how abstraction mechanisms support the creation of reusable software components.
• Demonstrate the difference between overridden and overloaded subprograms
• Explain functional side effects.
• Demonstrate the difference between pass-by-value, pass-by-result, pass-by-value-result, pass-by-reference, and pass-by-name parameter passing.
• Explain the difference between the static binding and dynamic binding.

Relevance to Program Learning Outcomes and Evaluation:

b. An ability to analyze a problem, and identify and define the computing requirements appropriate to its solution
   Justification: Implementing various projects in programming languages such as COBOL, Fortran, C, Perl, PHP, and C#
   Measured by: programming assignments and tests

c. An ability to design, implement and evaluate a computer-based system, process, component, or program to meet desired needs
   Justification: Coding in various programming languages in different platforms such as OpenVMS, FreeBSD, and Windows
   Measured by: assignments and tests

h. Recognition of the need for, and an ability to engage in, continuing professional development
   Justification: Research projects are required to obtain certain functions of programming languages that can lead to the solutions sought. Also, emphasis is placed on fundamentals of languages to support future ability to follow developments.
   Measured by: research projects

i. An ability to use current techniques, skills, and tools necessary for computing practice
   Justification: Coding in various programming languages using modern techniques, skills and tools.
   Measured by: assignments and tests

Units Covered:
PL1 Overview of programming languages (2/2)
PL2 Virtual machines (1/1)
PL3 Introduction to language translation (2/2)
PL4 Declarations and types (3/3)
PL5 Abstraction mechanisms (3/3)