CS 2SC3 and SE 2S03 Fall 2009

01 Programming Languages

William M. Farmer

Department of Computing and Software
McMaster University

24 September 2009

Software Development Phases

1. Requirements: What is the problem that needs to be solved? What are the product requirements that need to be satisfied?
2. Design: How will the problem be solved? How will the product requirements be satisfied?
3. Implementation: What is a solution to the problem? What is an executable implementation of the design?
4. Verification: What behavior does the product exhibit? Is the behavior correct?
5. Delivery and Maintenance: How will the product be delivered? What needs to be maintained? How will it be maintained?

Software Life Cycle Models

- Waterfall model:
  - Development follows the logical order of the phases given above in a linear fashion.
  - Is an idealization of the software development process that is rarely realized.
- Other life cycle models:
  - Refinement
  - Incremental
  - Spiral
  - Prototyping

What is a Program?

- A program is the executable part of a software product.
- A program is most often viewed as a sequence of instructions for a machine.
  - An understanding of a program requires an understanding of the machine.
- A machine language program is a sequence of instructions for a physical machine.
  - Usually represented as a sequence of 0s and 1s.
  - Not intelligible to humans.
- A high-level language program can be viewed as a sequence of instructions for a high-level abstract machine.
  - Easier to understand because the machine is simpler.
  - Ultimately executed on a physical machine.

Other Ways of Viewing Programs

- As a small abstract machine.
  - Good because the machine can be simple.
- As a function that maps inputs to outputs.
  - Good if the program has no side-effects.
- As an expression in a formal language.
  - The syntax of the expression is the program.
  - The semantics of the expression is the behavior of the program.
  - Good if the language is well behaved.
- As a constructive proof of an existential formula.
  - Very impractical with today's technology.

Ways of Classifying Programs

- Sequential vs. concurrent.
- Terminating vs. nonterminating.
- Subject-invoked vs. event-triggered.
- Applicative vs. systemic.

CS 3SC3 / SE 2S03 focuses on programs that are sequential, terminating, subject-invoked, and applicative.

Programming Languages

- Programming languages are intended to facilitate program implementation but not necessarily program design.
- Program languages have a syntax and a semantics:
  - The syntax concerns the structure of the programs.
  - The semantics concerns the behavior of the programs.
  - Most programming language have a precise syntax; few have a precise semantics.
- Programming languages support various programming styles called programming paradigms.
- Implementations of programming languages support various modes of execution.
- Ideally, the design of a program should not be restricted by the programming language chosen for implementing the design.

Programming Paradigms

Chief programming paradigms:

1. Imperative. Program statements modify a program state.
2. Object Oriented. Data and procedures are organized into units called objects.
3. Functional. Function applications are evaluated without modifying a program state.
4. Logical. Answers to questions are deduced from logical statements.

Some other programming paradigms:

1. Visual.
2. Constraint.
3. Scripting.
4. Language Oriented.

Modes of Program Execution

1. The program can be compiled into native machine code.
   - Advantage: The machine code is optimized to run fast.
   - Disadvantage: Code development is more difficult.
   - Compiled languages: C, C++, Fortran, Lisp, OCaml.
2. The program can be interpreted directly line by line.
   - Advantage: Supports interactive development and debugging of code.
   - Disadvantage: Interpreting code is generally slower than executing compiled code.
   - Interpreted languages: Lisp, Smalltalk, Python, OCaml.
3. The program can be compiled into bytecode for a virtual machine that is either interpreted or compiled.
   - Advantage: Programs are more portable.
   - Languages compiled into bytecode: Java, Perl, Python, OCaml.

Objective Caml (OCaml)

- Developed in 1996 at INRIA in France.
- A member of the ML family of programming languages.
  - ML stands for metalanguage.
- A multiparadigm programming language: imperative, object-oriented, functional.
- Three modes of execution: compilation to native machine code, interpretation, compilation to bytecode.
- Notable characteristics:
  - Powerful type system with type inference.
  - Automatic garbage collection.
  - Syntax matching.
  - Exception handling.
  - High execution speed.
  - Modules and functors (parametric modules).

Executing OCaml: Toplevel System

- The toplevel system for OCaml is an interactive read-eval-print loop.
- The toplevel system is started by the command `ocaml`.
- OCaml phrases are repeatedly read, type-checked, compiled, executed, and then the results of the execution are printed.
- A list of OCaml phrases can be executed as a script.

Executing OCaml: Native Code Compilation

- The OCaml native-code compiler `ocamlopt` compilers OCaml source code files to native code object files and links these object files to produce standalone executables.
- Example: `ocamlopt -o nc-prog prog.ml`
- Native code compilation results in slower compilation time, faster run time.

Executing OCaml: Bytecode Compilation

- The OCaml bytecode compiler `ocamlc` compiles OCaml source files to bytecode object files and links these object files to produce standalone bytecode files.
- Example: `ocamlc -o bc-prog prog.ml`
- A standalone bytecode file can be executed by the OCaml bytecode interpreter `ocamlrun`.
- Bytecode compilation results in faster compilation time, slower run time.

The C Programming Language

- Developed by Dennis Ritchie in 1972 at AT&T Bell Labs.
- Intermediate level language designed for system programming for the Unix operating system.
- A single paradigm programming language: imperative.
- Usually has a single mode of execution: compilation to native machine code.
- Notable characteristics:
  - Weak typing.
  - Low-level access to memory.
  - Extensive use of explicit pointers.
  - Preprocessor for macro definitions.
  - Major functionality provided by library routines.
  - Very high execution speed.