CPSC 5011: Object-Oriented Concepts

Lecture 1: OO Overview & Programming Languages

Examine Core Concepts

Abstraction is Key!

- Abstraction elevates the task of problem-solving
- Low-level details hidden
- Hardware obscured
- Software more portable, scalable and maintainable
- Abstractions for modern software include
  - Operating systems
  - Memory hierarchies
  - Compilers
    - High-level languages
    - Data Types, …

Operating System

- Handles I/O
  - loading of software for execution
  - retrieval of data
  - assignment of memory.
- Schedules the CPU
  - processes, threads, multi-threading, multi-core, …
- Handles ancillary devices
  - printers, memory, audio, …
- Platform-specific details
  - no longer the responsibility of software designers.

Compilers

- Translate source code to executable code
- Embody software reuse
- Partitioned
  - front end (HLL targeted)
  - back end (processor targeted)
- Complex
  - code processed across
  - multiple stages of interpretation and translation.

Compilers profoundly enhanced the productivity of software developers

High-Level Languages (HLL)

- Intent is to be readable (like a natural language)
  - Not binary, not machine code
- Promote abstraction: little attention given to
  - specific CPU registers
  - memory locations
  - bit strings.
- Allow software designer to focus
  - control flow
  - data structures
  - software components, etc.,

Structured Programming

- 1970s on
- Addressed ‘spaghetti code’
  - Unstructured code that was:
    - Hard to decompose
    - Expensive to maintain
    - Convoluted => “Job security”
  - Excessive and poor use of GOTO
- Focus on
  - Control flow
  - Functional decomposition

Role of Type in Structured Programming

- Built-in (primitives)
  - Predefined operations
- User-defined types
  - Defined by programmer
  - Composite of primitives
- ADT (Abstract Data Type)
  - Separation of interface & implementation
  - Operations associated with data

=> Object

  - Encapsulated ADT

Notion of Type

- Data types built into HLLs
  - Integer (whole number), Float (real number), …
  - String (alphanumeric), …
- Software Designer may define own data types
  - Composites
  - ADT/Class structure
  - Class hierarchies
  - Templates
- Programmers
  - Need not track type representation (low-level details)

Type Support

- Compiler
  - allocates the correct size of memory for data
  - checks the legitimacy of data operations
  - selects appropriate algorithm
    - Real addition vs integer addition vs string concatenation
- Software designer
  - define typeless templates
    - Type specifies according to need
    - e.g. Set of integers, set of strings, set of dates, …
  - design type extensibility

=> promote software reuse.

OO Programming

- 1980s perspective
  - Data centric analysis
  - Focus on objects
  - Lots of verbiage
- **Modern perspective**
  - Driven by large-scale software development
  - Expectations of fast development & ease of maintenance
    - **CODE REUSE**
  - Key attributes
    - Substitutability
    - Dynamic behavior (function selection)
    - Extensibility

Structured vs. OO

- Not mutually exclusive
  - Drivers (high-level process) structured
  - Object functionality structured
- Structured programming still useful
  - Complex control flow & simple data types
  - Performance critical
    - Cost of generality (vs. efficiency of specificity)
    - OO constructs avoided

Coupling & Cohesion

- COUPLING
  - How tightly interrelated are two or more modules (objects)?
  - *Degree of dependence*
- COHESION
  - How integral is a module's (object's) definition? Any extraneous members?
  - How well does it 'stick' together?
  - *Singularity of purpose*

Coupling & Cohesion

- Tight COUPLING
  - High maintenance cost
  - *Cascading changes*
- Low COUPLING (minimal dependency)
  - Promotes reusable code
  - Lowers maintenance costs
- High COHESION
  - Clarity in design
  - Isolated functionality minimizes impact of change

C

- Kernighan & Ritchie, ATT Bell labs, 1971
- Efficiency primary goal
  - Omission of run-time checks
  - Static Binding default
  - Simple memory model
    - Stack-based allocation
    - No garbage collection
  - Pointer arithmetic
- Concise representation
- Quickly became industry standard
- Can write unstructured code
  - GOTO construct (break, return,exit)
- Can mimic OO constructs with difficulty
  - Not extensible

C++

- Bjarne Stroustrup, ATT Bell labs, 1985
- OO extension of C
- Backward compatible with C
  - Support all C features
  - C code compiled by C++ compiler
- Efficiency still primary goal
  - “pay only for what you use”
  - Stack allocation default
  - Static binding default
- HL abstraction with low-level manipulations permitted
- Quickly became industry standard
- Not considered safe
  - Omission of run-time checks
  - Memory management problems

Java

- Sun Microsystems, 1994
- Primary goals
  - Portability
    - Java byte codes (compiler & interpreter)
    - Each primitive type always same size (platform-independent)
  - Safety (Managed Code)
    - No pointer construct for programmer
    - Garbage collection
    - Run-time checks
- Quickly gained market share
- Significant performance problems
  - Overhead of dynamic binding & heap allocation
  - GC imperfect process

C#

Microsoft, 2001

- Proprietary Java (precursor: Visual J++)
- .NET platform

Goals similar to Java

– Portability
– Programmer Productivity
– Safety (Managed Code)
  - Garbage collection
  - Run-time checks
  - Pointer construct only viable in UNSAFE code
    - `delegate` construct (like function pointer)

Small differences from Java

- Limited operator overloading
- Getter/setter (properties)
- No checked exceptions

C++ 11

- Backward compatible with C++ 98
- Emphasis on improved support for
  - Performance
  - Multithreading
  - Generics
- Appeal (move closer to Python)
  - Support for lambda and regular expressions
- Software Construction for Longevity
  - Self-documenting code & design consistency

Language Choice?

- Not strictly independent
  - Legacy code
  - Utilities
  - UI, famiiliarity, etc.
- What are design priorities?
  - Safety
  - Performance
  - Extensibility
    - Maintainability
  - Portability
  - Scalability
  - Fast Development Time?

No Silver Bullets

- Frederick Brooks, landmark SE paper
  - Addressed choice of programming language
  - Different PL fit different needs

⇒ Different software design paradigms target different goals

- Use OO appropriately