1

Introduction

Programming Languages and
Programming Paradigms
History of Programming Languages
Functional Programming
Logic Programming
Object-Oriented Programming
Concurrent Programming

©1993 John Mylopoulos

2

Programming Paradigms

Different programming languages reflect different views on how to use a computer to solve a problem.

There are many criteria by which one could classify or compare programming languages:

For example:

- Intended application area
  - data processing, systems,...
- Data structures used
  - arrays, graphs, strings,...
- Operations on data structures
  - w/o pattern matching for strings

.
.
.

©1993 John Mylopoulos

3

What is This Course All About?

Programming "paradigms" for computing

• **What is a program?** A sequence of commands ...
... but there are other answers too!

• **What is programming?** The activity of defining a program to be executed by a computer in order to solve a problem

• **What is a programming paradigm?** A set of concepts for thinking about programs

Every programming language is based on a programming paradigm

©1993 John Mylopoulos

4

Programming Languages

Machine Languages

Every operation is specified in terms of an actual numerical code, and each operand in terms of its absolute address or in terms of a register number or some combination thereof.

The basic architecture of most machines includes:

memory locations	contain data or instructions organized into bytes or words (4K to 2G)
registers	large enough to hold one word of information (usually 1-128bits)
instruction set	numerically encoded commands

©1993 John Mylopoulos

5

Programming Languages

Assembly Languages
Specify each operation and operand with a symbolic name. The assembler has the job of turning assembly language code into machine language code. Macro assemblers allow macros as a convenient shorthand for frequently used sequences of code.

Assembly languages are machine-dependent, i.e., their features depend on the hardware (or, a particular family of CPU's) for which they were intended.

High Level Languages
These are conventional programming languages such as Ada, Pascal and C. They are usually machine-independent but may still be implementation-dependent. For example, Lisp has many dialects such as Franz Lisp, Interlisp and Common Lisp

Very High Level Languages
These are declarative languages that allow one to state what is wanted rather than how to get it. Examples include complex report generation and processing languages for databases. Prolog may be classified as a very high level language.

©1993 John Mylopoulos

6

Programming Languages History

Early computers ('40's to mid-50's) were programmed in machine language and later in assembly language.

First 'real' high level language was Fortran I (mid '50's). Seen as 'automatic programming' when it first came out. Followed by Fortran II, Fortran IV; standard for many years. First versions of COBOL followed soon after, then came Lisp, and Algol. (which appeared around 1960)

Algol was the first block-structured language with `begin-end` blocks. Pascal, Simula were based on Algol.

Structured programming became state-of-the-art as a programming methodology in the late 60's and early 70's. Programs are easier to write and maintain if goto statements are not used; instead a programmer is supposed to use block structures and loop exits.

Different languages were developed for different needs. Attempts to develop a single "good" language led to PL/I, generally considered a failure.

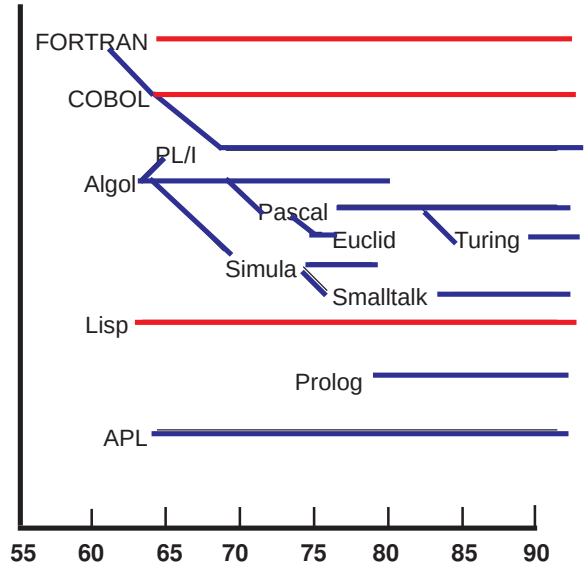
Today popular languages are Cobol, C, Pascal, Basic, Ada, PL/I, Fortran, Lisp, Prolog ...

©1993 John Mylopoulos

7

High Level Programming Languages

Geneology



©1993 John Mylopoulos

8

"Conventional" Programming Languages

Cobol, C, Basic, Fortran, PL/1, Pascal, Ada, Algol, Turing,...

These languages are imperative, in the sense that a program is intended to tell the computer to execute instructions which solve a particular (computational) problem.

These languages are founded on John von Neuman's view of computers and computing.

Based on

Declarations of variables
+ Commands (Statements)
- `assign` value to variable
- `call` procedure with parameters
- `return` from procedure
- `goto label` or location

+ Composite Commands
- Sequence `begin ... end`
- Conditional `if ... then ... else ...`
- Iteration `loop ... end`

©1993 John Mylopoulos

9

Programming Language Design

Many tradeoffs have to be taken into account in designing a programming language

Computational Considerations

- language designed must admit efficient compilation and/or interpretation (space- and time-wise, during compilation and execution)
- syntax of the language must be unambiguous, and simple to specify
- must be able to express (almost) anything the computer can do.

Human Factors Considerations

- easy to learn, to read and understand
- clear in intended meaning; orthogonal with respect to features
- easy to express things that are frequently done
- easy to write correct programs, helpful in protecting the programmer against silly, but human mistakes.

©1993 John Mylopoulos

10

Compilation versus Interpretation

Most programming languages allow programs to be compiled for efficient execution. i.e., to be translated into machine language. The translation is not necessarily direct. For example:

C++ => C => 68000 assembler => 68000 machine

Some languages are interpreted. An interpreter is a program that takes another program as input, and performs the actions specified by that other program. Interpretation is generally much less efficient than execution of compiled code
(rule of thumb: 10-100 less efficient)

If a program is changed often, but is not run very often, then interpretation may be more appropriate.

©1993 John Mylopoulos

11

"Unconventional" Programming Languages

"Unconventional" programming languages may reject one or more of the features of conventional programming languages, for example

- no Declarations of variables
- no Commands
- no (explicit) Assignment of values to variables
- no (explicit) Passing of arguments to functions
  (instead, ask an object to execute a method)
- no (explicit) Return from functions
- no (explicit) Gotos
- no (explicit) Iteration
  (use recursion, or backtracking instead)

Rejected features must be replaced by new ones, thereby leading to a

New views of programming!!

©1993 John Mylopoulos

12

What is This Course All About?

Unconventional programming paradigms and the features of languages based on these paradigms

Lecture material adopts a "theoretician's" view of languages; we won't deal with particular language implementations (say,Common Lisp vs Scheme)

Practical details required for your programming assignments will be provided during tutorials.

What will the course cover?

Functional Programming: Programming using Lisp
Logic Programming. Programming using Prolog
Object-Oriented Programming using the Common Lisp Object System (CLOS)

©1993 John Mylopoulos

13

Functional Programming and Lisp

Given a program, the question is
not "What does it do?"
but "What does it evaluate to?"

Everything amounts to evaluation of functional expressions

The emphasis is on values derived by applying functions to arguments.

Distinguishing Features:

- A program consists of a set of functions
- Recursion does most of the work during function definitions
- Execution amounts to function application to an appropriate set of arguments
- Pure functional programming means no variables, no side-effects
- Storage allocation and management is implicit
- Programs can be used as data and data as programs

Lisp is mainly used in AI, symbolic math and CAD/CAM.

©1993 John Mylopoulos

14

Examples of Functions

"Age" as in "Age(Mary) = 15"
Age takes one argument (a person) and returns that person's age (a positive integer).

"Father" as in "Father(Mary) = Fred"
Father takes one argument (a person) and returns that person's father (a person)

"!" as in "5! = 120"
Factorial takes one argument (a positive integer) and returns its factorial (a positive integer).

"+" as in "2+3=5"
Addition takes two arguments (each a real value) and returns their sum (a real value).

Function	Domain	Range
Age	Person	positive integer (Z^+)
Father	Person	Person
!	Z^+	Z^+
+	$R \times R$	R (real number)

©1993 John Mylopoulos

15

The Ackermann Function

Here is a highly non-trivial function:

f(0,x,y) = y + x
f(1,x,y) = y + ... + y (x times) = y*x
f(2,x,y) = y * y * ... * y (x times) = y^x
f(3,x,y) = y to the power y, to the power y,...(x times) = ^xy

(call ^xy y to the superpower x)

f(4,x,y) = y to the super power y, to the superpower y,...(x times) = $_xy$

etc.

This function grows very-very fast with respect to its first argument!

16

Logic Programming and Prolog

Given a program the question is
not "What does the program do?"
but "What can be inferred from the facts constituting the program?"

Logic programming languages are declarative.
The emphsis is on expressing facts and rules.
Based on (restricted) first-order logic

Prolog (PROgramming in LOGic):: Distinguishing Features

Programs consist of facts and rules
Execution amounts to searching for a proof for a statement posed by the program user.
Values are retrieved through pattern matching with program facts and rules.
Programs can be inverted, i.e., given an output find one or more inputs that lead to that output
Storage allocation and management are implicit
Programs can be used as data and vice versa

Prolog is used mainly in AI, expert system shells and some database work.

17

Examples of Logic Statements

Facts:

"Toronto is exciting"
"Clyde is an elephant"
"Ann has big ears"
"Ann has a long trunk"
"Ann is an animal"

Rules:

If *x* is an animal, and
x has big ears, and
x has a long trunk,
Then *x* is an elephant.

Query	**Answer**
" Toronto is exciting?"	yes
"Who is an elephant?"	Clyde , Ann
"Is Marc an elephant?"	no

©1993 John Mylopoulos

18

Object-Oriented Programming

Given a program the question is
not "What does the program do?"
but "What behaviours are supported by the (active) objects constituting the program?"

The emphasis is on objects (not on processes). Every object inherits a behaviour, defined in terms of a set of methods, from the class it is an instance of

Distinguishing Features

Every object is an instance of some class that determines the characteristics and behaviour of its instances.

Classes can be subclasses of other classes.

The specific method used to evaluate an incoming message for a particular object depends on the object's class

Objects are at the same time programs and data and can be encapsulated and abstracted.

Object-Oriented Programming is used in AI systems, Databases, Software Engineering, Window Management systems and Graphics

©1993 John Mylopoulos

19

Object-Oriented Programming Example

Classes and Methods:

Square with Side
- Perimeter is Side × 4
- Area is Side × Side

Circle with Radius
- Perimeter is 3.1416 × Radius
- Area is 3.1416 × Radius × Radius

Triangle with Height and Width
- Area is 0.5 × Height × Width

Determine Area and Perimeter of Instance X:

X is a Square with Side=3 Area of X is 9
X is a Circle with Radius=1 Area of X is 3.1416
X is a Square with Side=3 Perimeter of X is 12
X is a Triangle with Height=2 and Width=3
Perimeter of X is *undefined*

To evaluate a generic function , say Area, for an object, an object-oriented language determines the *instance class* of the object, and then asks the class for the required method

©1993 John Mylopoulos

20

(Not Covered this Term)
Concurrent Programming

Given a program the question is
- **not** "What does the program do?"
- **but** "How do the subprograms communicate and coordinate their actions to solve a given problem?"

The emphasis now is on subprograms (or, processes). Processes may run concurrently (hence the name of this programming paradigm.)

Distinguishing Features

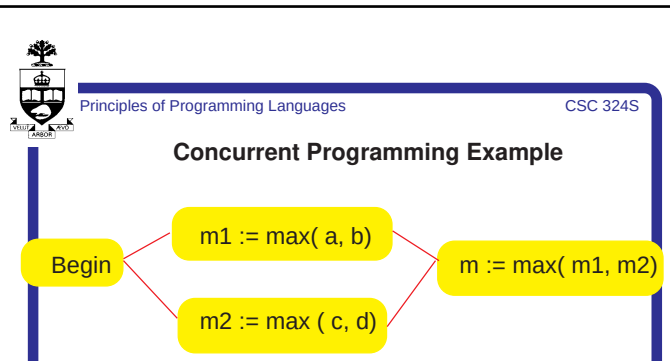
Concurrent processes use shared resources. The key issue in this programming paradigm is to coordinate the processes so that they make effcient use of the shared resources and don't step on each others' toes. For example, it is inefficient to have a process grab a resource which is not needed immediately, because this could prevent another process from completing its task.

Issues

Facilities for specifying concurrency
Communication among processes
Synchronization among processes
Correct concurrent execution of processes

©1993 John Mylopoulos

21



Concurrent Programming Example

;; in Turing+: Find the largest value of { a, b, c, d}

```
cobegin
    m1 := max( a, b)
    m2 := max( c, d)
coend
m := max( m1, m2)
```

Evaluation is either:

m1 := max(a, b)		m2 :- max(c, d)
m2 := max(c, d)	**or**	m1 :- max(a, b)
m :- max(m1, m2)		m :- max(m1, m2)

Assignments to m1, m2 could be done in parallel, if this process was executing on parallel hardware

©1993 John Mylopoulos