CSE305 Programming Languages

• Notes are based primarily on Sebesta text.
• Quotes are from Sebesta text, unless otherwise noted.
• Sources other than Sebesta will be clearly indicated.
Do you need to study *programming languages*?

- You already know at least two languages: Java and C++.
- Things never change: Java and C++ will always be the major languages.
- There are other languages?
- But nobody really uses them, do they?
Quick survey: what languages have you written programs in (aside from Java and C++)?
“Increased capacity to express ideas.”

- Languages tend to facilitate certain kinds of problem solving.
- Knowing more than one language equips you with different problem solving skills.
  - richness of primitive types, user-defined types, polymorphism, inheritance
  - atomic data, structured data, objects, generics
- The problem solving concepts can be transferred from one language to another.
  - using arrays to simulate records
“Improved background for choosing appropriate languages.”

• Languages tend to facilitate certain kinds of problem solving.
• Knowing what language offers good support for problem solving in a particular domain.
• Fortran is very good for scientific number crunching
• Prolog is very good for rule-based reasoning
“Increased ability to learn new languages.”

- Understanding the essential commonalities aids in learning new languages.
- For example: Structure theorem (Bohm & Jacopini, 1966) tells us that sequencing, selection and iteration are sufficient control structures. Different languages provide these in different ways: syntactic form is not what’s important – look for underlying expressiveness.
“Better understanding of the significance of implementation.”

• Programmers make better use of language features when they understand implementation issues.

• For example: understanding how recursion is implemented leads to judicious use of it.

• For example: understanding how “last call optimization” works allows programmers to write efficient recursive programs, in languages that support this optimization.
“Overall advancement of computing.”

• Deeper understanding of benefits & drawbacks of language design can help promote more sensible language choices over “more comfortable” ones.
Why do you choose to use a particular PL?

- Comfort: you already know it.
- Cost: it is free/inexpensive to acquire.
- Efficiency: it executes quickly.
- Environment: it has good development tools.
- Appropriateness: it is well-suited to your domain.
- No choice: you boss told you to.
- Support: the language is well-supported.
- Robustness: it is easy to write robust programs in it (or: it is difficult to write fragile programs in it).
Things to think about

• Readability
• Writability
• Reliability
• Cost
Readability

• are programs written in a language easy to understand
• a product not only of programmer discipline but also of language facilities (e.g. variable names in early BASIC could be one letter or one letter and one digit).
• Are there too many ways to express the same idea?
  – \(x = x + 1;\ x += 1;\ x++;\ ++x;\)

• Are there many arbitrary restrictions on what you can do? (IBM vs VAX)
  – IBM: two instructions (varies based on type of 2\textsuperscript{nd} arg)
    • A Reg1, memory\_cell
    • AR Reg1, Reg2
  – VAX: one more flexible instruction
    • ADDL op\_1, op\_2

• Context-dependence: \(a + b\) in C
  – meaning is different if a and b are pointers than if they are not.
Writability

• How easy is it to model a problem domain in a language?
• What sorts of abstraction mechanisms does the language support?
  – data abstraction
  – procedure/function abstraction
Reliability

• Type checking
  – static vs. dynamic
  – strong vs. weak

• Exception handling
  – generally considered very useful to deal with unexpected or out-of-the-ordinary runtime events
Cost

• Acquisition cost
• Training cost
• Cost of writing programs in language
• Compilation cost
• Execution cost
• Cost of poor reliability (think of liability for safety-critical applications)
• Cost of maintenance
• Cost of portability