# Introduction

1. Objectives ........................................................................................................... 2
2. Why are there so many programming languages? ......................... 2
3. What makes a language successful? .......................................................... 2
4. Programming Domains .................................................................................. 3
5. Language and Computer Architecture ..................................................... 4
6. Types of Languages ....................................................................................... 4
7. Compilation vs. Interpretation ....................................................................... 5
8. Phase of Compilation (From Scott’s class notes) .................. 7
9. Language Evaluation ....................................................................................... 8
10. Genealogy of common high-level programming languages ... 9
11. Fortran ............................................................................................................ 10
12. Lisp-1959 ..................................................................................................... 11
13. ALGOL 58 and 60 ......................................................................................... 12
14. Cobol ............................................................................................................. 13
15. PL/I ............................................................................................................... 14
16. Ada ............................................................................................................... 15
17. C++ .............................................................................................................. 16
18. Java .............................................................................................................. 16
19. Python .......................................................................................................... 17
1. Objectives

- Increased capacity to express programming concepts
- Improved background for choosing appropriate languages
- Increased ability to learn new languages
- Understanding the significance of implementation
- Increased ability to design new languages
- Overall advancement of computing

2. Why are there so many programming languages?

- We are trying to improve previous languages
- For certain domains of applications
- For specific hardware
- Don’t agree with the ways certain programming constructs are implemented in certain languages

3. What makes a language successful?

- Easy to learn (BASIC, Pascal, LOGO, Scheme)
- Familiarity: The language looks like the one I know.
- Easy to use
- Easy to implement (BASIC, Forth) possible to compile to very good (fast/small) code (Fortran)
- Who is pushing the language? (COBOL, PL/1, Ada, Visual Basic)
- Popularity (Pascal, Java)
- Are the supporters of Ruby and Python trying to push Java to die?
- Read this article: The Secret of a Successful Programming Language? A Really Great Beard (http://www.wired.com/2012/06/beard-gallery/)

4. Programming Domains

- Scientific applications
- Business applications
- Artificial intelligence
- Systems programming
- Scripting languages
- Special purpose languages
5. Language and Computer Architecture

- Most conventional programming languages can be viewed as abstraction of an underlying Von Neumann architecture:

![Diagram of I/O, Memory, and CPU](image)

- The execution of the following expression: \( x = a + b; \)

![Diagram of Adder](image)

6. Types of Languages

- **Imperative or procedural**: Procedures are the main thing
- **Functional**: Everything is a function call
- **Abstract Data Types**: Like Ada packages
- **Module-Based**: C Program Modules
- **Object-Oriented**: C++/Java Classes
- **Generic**: Like Ada Generics/C++ Templates
- **Declarative**: 
- Functional programming languages
- Logic programming languages

<table>
<thead>
<tr>
<th>Imperative/Algorithmic</th>
<th>Functional Programming</th>
<th>Declarative</th>
<th>Logic Programming</th>
<th>Object-Oriented</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algol</td>
<td>Lisp</td>
<td></td>
<td>Prolog</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>Cobol</td>
<td>Haskell</td>
<td></td>
<td></td>
<td>Simula</td>
</tr>
<tr>
<td>PL/1</td>
<td>ML</td>
<td></td>
<td></td>
<td>C++</td>
</tr>
<tr>
<td>Ada</td>
<td>Miranda</td>
<td></td>
<td></td>
<td>Java</td>
</tr>
<tr>
<td>C</td>
<td>APL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Compilation vs. Interpretation

- Pure Compilation
  - The compiler translates the high-level source program into an equivalent target program (typically in machine language).

![Diagram showing compilation process]

- Most compilers produce an intermediate set of virtual instructions, also known as P-code.
- Other Language translate the source code to another source code:
  - Early C++ implementations generates an intermediate program in C, instead of an assembly language
- Pure Interpretation
  - Interpreter stays around for the execution of the program
  - Interpreter is the locus of control during execution

- Mixing Compiler & Interpreter:
  - Some languages are semi-compiled like Java & C#
8. Phase of Compilation (From Scott’s class notes)

Character stream \rightarrow Scanner (lexical analysis)

Token stream \rightarrow Parser (syntax analysis)

Parse tree \rightarrow Semantic analysis and intermediate code generation

Abstract syntax tree or other intermediate form \rightarrow Machine-independent code improvement (optional)

Modified intermediate form \rightarrow Target code generation

Assembly or machine language, or other target language \rightarrow Machine-specific code improvement (optional)

Modified target language

Symbol table
9. Language Evaluation

- Overall simplicity: Readability, Writability
- Orthogonality: Readability
- Support for abstraction: Writability
- Expressivity: Writability
- Reliability
  - Type checking:
  - Exception handling
- Maintainability
  - **Factoring**: The ability to group related features into a single unit. Use subroutines to group related computations units so they can be re-used in different parts of the application.
  - **Locality**: The ability to implement information hiding so that changes to a grouping (either control or data) are transparent.

- Cost
  - Programmer training
  - Software creation
  - Compilation
  - Execution
  - Compiler cost
  - Poor reliability
  - Maintenance

- Others: portability and generality
10. Genealogy of common high-level programming languages
11. **Fortran**

- Impact of environment on design of FORTRAN I
  - No need for dynamic storage
  - Need good array handling and counting loops
  - No string handling, decimal arithmetic, or powerful input/output (commercial stuff)

- First implemented version of FORTRAN
  - Names could have up to six characters
  - Posttest counting loop (DO)
  - Formatted i/o
  - User-defined subprograms
  - Three-way selection statement (arithmetic IF)
  - No data typing statements
  - No separate compilation
  - Compiler released in April 1957, after 18 worker/years of effort
  - Programs larger than 400 lines rarely compiled correctly, mainly due to poor reliability of the 704
  - Code was very fast
  - Quickly became widely used

FORTRAN II - 1958
- Independent compilation
- Fix the bugs

FORTRAN IV - 1960-62
- Explicit type declarations
- Logical selection statement
- Subprogram names could be parameters
- ANSI standard in 1966

- FORTRAN 77 - 1978
  - Character string handling
  - Logical loop control statement
  - IF-THEN-ELSE statement

- FORTRAN 90 - 1990
  - Modules
  - Dynamic arrays
• Pointers
• Recursion
• CASE statement
• Parameter type checking

• FORTRAN Evaluation
  • Dramatically changed forever the way computers are used

12. Lisp-1959
  • List Processing language (Designed at MIT by McCarthy)AI research needed a language that: Process data in lists (rather than arrays)
  2. Symbolic computation (rather than numeric)
• Only two data types: atoms and lists
• Syntax is based on lambda calculus
• Pioneered functional programming
  No need for variables or assignment
  o Control via recursion and conditional expressions
• Still the dominant language for AI
• COMMON LISP and Scheme are contemporary dialects of LISP
• ML, Miranda, and Haskell are related languages
13. ALGOL 58 and 60

- Genealogy of ALGOL 60

- Environment of development:
  1. FORTRAN had (barely) arrived for IBM 70x
  2. Many other languages were being developed, all for specific machines
  3. No portable language; all were machine-dependent
  4. No universal language for communicating algorithms

- Goals of the language:
  - Close to mathematical notation
  - Good for describing algorithms
  - Must be translatable to machine code

- ALGOL 58 Language Features:
  - Concept of type was formalized
  - Names could have any length
  - Arrays could have any number of subscripts
  - Parameters were separated by mode (in & out)
  - Subscripts were placed in brackets
  - Compound statements (begin ... end)
  - Semicolon as a statement separator
  - Assignment operator was :=
  - if had an else-if clause
  - no i/o - “would make it machine dependent”
• ALGOL 60
  ▪ Modified ALGOL 58 at 6-day meeting in Paris
  ▪ New Features:
    o Block structure (local scope)
    o Two parameter passing methods
    o Subprogram recursion
    o Stack-dynamic arrays
  ▪ Still no i/o and no string handling

  ▪ Successes:
    o It was the standard way to publish algorithms
    o for over 20 years
    o All subsequent imperative languages are
    o based on it
    o First machine-independent language
    o First language whose syntax was formally defined (BNF)

  ▪ Failure:
    o Never widely used, especially in U.S.
    o Reasons:
      1. No i/o and the character set made programs non-portable
      2. Too flexible--hard to implement
      3. Lack of support of IBM

14. Cobol

• First Design Meeting (Pentagon) - May 1959
  o Design goals:
    1. Must look like simple English
    2. Must be easy to use, even if that means it will be less powerful
    3. Must broaden the base of computer users
4. Must not be biased by current compiler problems
   o Design committee members were all from computer manufacturers and DoD branches
   o Design Problems:
     1. Arithmetic expressions?
     2. Subscripts?
     3. Fights among manufacturers

   • Contributions:
     o First macro facility in a high-level language
     o Hierarchical data structures (records)
     o Nested selection statements
     o Long names (up to 30 characters), with hyphens
     o Separate data division

   • Comments:
     o First language required by DoD; would have failed without DoD
     o Still the most widely used business applications language

15. PL/I

   • Genealogy of PL/I

<table>
<thead>
<tr>
<th>COBOL (1960)</th>
<th>ALGOL 60 (1960)</th>
<th>FORTRAN IV (1962)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PL/I (1965)</td>
<td></td>
</tr>
</tbody>
</table>

   • Goals:
     o Scientific users began to need more elaborate i/o, like COBOL had;
- Business users began to need fl. pt. and arrays (MIS)
- Design a new language to do both kinds of applications.

- Designed by IBM in five months by the 3 X 3 Committee
- **PL/I contributions:**
  1. First unit-level concurrency
  2. First exception handling
  3. First pointer data type
  4. First array cross sections

- Comments:
  - Many new features were poorly designed
  - Too large and too complex
  - Was (and still is) actually used for both scientific and business applications

### 16. Ada

- Huge design effort, involving hundreds of people, much money, and about eight years.
- **Environment:** More than 450 different languages being used for DOD embedded systems (no software reuse and no development tools)
- **Contributions:**
  1. Packages - support for data abstraction
  2. Exception handling - elaborate
  3. Generic program units
  4. Concurrency - through the tasking model

- Comments:
  - Included all that was then known about software engineering and language design
First compilers were very difficult; the first really usable compiler came nearly five years after the language design was completed

- Ada 95 (began in 1988)
  - Support for OOP through type derivation
  - Better control mechanisms for shared data (new concurrency features)
  - More flexible libraries

17. C++

- Developed at Bell Labs by Stroustrup
- Evolved from C and SIMULA 67
- Facilities for object-oriented programming, taken partially from SIMULA 67, were added to C
- Also has exception handling
- A large and complex language, in part because it supports both procedural and OO programming
- Rapidly grew in popularity, along with OOP
- ANSI standard approved in November, 1997

18. Java

- Developed at Sun in the early 1990s
- Based on C++
- Significantly simplified (does not include struct, union, enum, pointer arithmetic, and half of the assignment coercions of C++)
- Supports only OOP
- Has references, but not pointers
- Includes support for applets and a form of concurrency
19. Python

- Visit: https://www.python.org/
- Invented in the Netherlands, early 90s by Guido van Rossum
- Open sourced from the beginning
- Considered a scripting language, but is much more
- Scalable, object oriented and functional from the beginning
- Used by Google from the beginning
- Translation to bytecode (like Java)
- Dynamic typing (like LISP)
- Higher-order function (LISP, ML)
- Garbage-collected, no ptrs (LISP, SNOBOL4)