CS 313 Lecture 3

Expression grammars
History of programming languages
Expression grammars

• First attempt last time:

\[ E ::= E + E \mid E - E \mid E \times E \mid E / E \mid (E) \mid \text{num} \mid \text{name} \]

• Problem: ambiguous
  • two different parse trees for same string, e.g.:
    \[ \text{num} + \text{num} \times \text{num} \]
Expression grammars

- Idea: build operator precedence into rules

\[
E ::= E + T \mid E - T \mid T \\
T ::= T \ast F \mid T / F \mid F \\
F ::= \text{num} \mid \text{name} \mid (E)
\]

- Unambiguous
- (HW 1 problem 4)
History of programming languages

Outline:
• Early programmable devices
• Programming language generations
• Evolution of high-level languages
• Programming language paradigms
Early programmable devices

Punch cards
• Jacquard loom  1804
• Analytical engine  1834  
  (Charles Babbage and Ada Byron Lovelace)
• US Census data  1890  
  (Herman Hollerith)
Charles Babbage’s Analytic Engine 1834

• Earliest known computer, never fully built
• Operations and variables on separate punch cards
• Conditional jumps via physically jumping over a band of cards
• Collaborator Lady Ada Byron, Countess of Lovelace
• Babbage first computer scientist
  Ada Lovelace first programmer
Von Neumann architecture
1945

• Mathematician John von Neumann
• EDVAC report (Electronic Discrete Variable Arithmetic Computer) describes the first stored-program computer
• Builds on design of ENIAC, one of the first electronic computers
• Computer in his design consists of small CPU, bus, and memory storing both data and instructions
• Single-CPU architecture still referred to as von Neumann architecture
Programming Language Generations

First Generation (late 1940s):

• Machine-level programming languages
  • Fast and efficient, executed directly on the CPU
  • Consists only of 0s and 1s
  • Difficult for humans to read, write, and debug
Programming Language Generations

Second Generation (early 1950s):

• Symbolic assemblers
• Interpreting routines
• Very early compilers

• Assembly languages
  • Simple mnemonic instructions: <opcode> <operands>
• Assembler translates into machine code

.globl selSort
selSort:
pushq %rbx
loopi:
  subq $1, %rsi
  cmpq $0, %rsi
  jle end
  movq $0, %rax
  movq (%rdi, %rax, 8), %rbx
  movq $0, %rcx
loopj:
  cmpq %rbx, (%rdi, %rcx, 8)
  jle skip
  movq %rcx, %rax
  movq (%rdi, %rax, 8), %rbx
skip:
  incq %rcx
  cmpq %rsi, %rcx
  jle loopj
  movq (%rdi, %rsi, 8), %rdx
  movq %rbx, (%rdi, %rsi, 8)
  movq %rdx, (%rdi, %rax, 8)
  jmp loopi
end:
popq %rbx
ret
Programming Language Generations

Third Generation (mid 1950s - present):

• High level, general-purpose

• FORTRAN, LISP, COBOL, ALGOL (Ada, Basic, C, C++, Java, Pascal, Smalltalk, …)

• Easier for humans to read, write, debug
• Compiler translates into machine code before running
• Interpreter translates into machine code at runtime
Programming Language Generations

Fourth Generation (1970s - ):

• Specification languages, query languages, report generators, systems engineering
  • Mathematica, Matlab + Simulink, SPSS, SQL, LabVIEW

Fifth Generation (1980s - ):

• Solve problems using constraints rather than algorithms, AI programming
  • Prolog
Konrad Zuse’s Plankalkül 1945

• First proposal for high-level language

• Anticipated many developments of programming languages
  • Arrays, records
  • Assertions
  • Algorithms for sorting, numerical computations, syntax analysis, and chess
A family tree of languages
Evolution of third-generation Languages

• Begins with FORTRAN in 1954
• Generation of high-level programming languages
• Languages stress expressivity and machine independence
• Includes procedural and functional languages
FORTRAN (1954)

• Designed at IBM to efficiently translate mathematical formulas into IBM 704 machine code
• Language design was secondary to compiler design for optimization
• 1954 Report for a proposed Formula Translating System
• 1957 FORTRAN language manual published
• Translator produced code that in some cases was more efficient than the equivalent hand-coded program
Innovations of Fortran

- language based on variables, expressions, statements
- the form of the arithmetic-assignment statement
- conditional and repetitive branching control structures
- arrays with maximum size known at compile time
- provision for comments
LISP (1958)

- Interactive functional language
- Designed for IBM 704 by John McCarthy at Dartmouth 1956-1958
- Implemented at MIT, first reference manual published in 1960
- Language based on lambda calculus (mathematical notation for expressing functions.)
- LISP (LiST Processor) was designed for symbolic formula manipulation
- Was long standard language of the AI community
Innovations of LISP

- the function as the basic program unit
- the list as the basic data structure
- dynamic data structures
- facilities for "garbage collection" of unused memory
- use of symbolic expressions as opposed to numbers
- recursion and the conditional expression as control structures
- the "eval" function for interactive evaluation of LISP statements
ALGOL (1958)

- **ALGO**rithmic Language, designed by international team
- Several revisions:
  - ALGOL58, ALGOL60, ALGOL68
- ALGOL60 had profound influence on programming language design and on computer science; Pascal carries on tradition
- ALGOL68 was a huge, general purpose language, not widely accepted
- Language description published in ALGOL60 report
  - First appearance of Backus-Naur Form for programming language definition
- Widely used as a publication language for algorithms
Innovations of ALGOL60

- block structure and localized data environments
- nesting of program units
- free format program code
- explicit type declarations
- dynamic memory allocation
- parameter passing by value and by name
COBOL (1960)

- US Dept of Defense wanted “common” PL for data processing
- CODASYL committee (Conference on Data Systems Languages)
- Result was COBOL in 1960 (COmmon Business-Oriented Language)
- Grace Hopper was involved in development and wrote 1st compiler
- Designed to be machine independent, unlike FORTRAN
- Influenced by Fortran, ALGOL58, and English
- Example: Multiply A by B giving C
  Perform <loop body>
    Varying J from 2 by 1
    Until J > N.

- Major revisions standardized and released in 1968, 1974, and 1985
Innovations of COBOL

- the record data structure
- file description and manipulation facilities
- machine independence of data and program descriptions
- influence of English
- relatively natural language style, including extra words for readability
- effort toward a language that would produce self-documenting program code
APL (early 1960s)

- A Programming Language
- Based on notation developed by Ken Iverson at Harvard 1957-1962
- Functional, interactive, science-oriented language that assumes the array as the default data structure
- Suitable for applications with a heavy use of numerical data in large multi-dimensional arrays
- Used special symbols requiring special keyboard / printer

```
life←{⊥1 w v.∧3 4=+/,-1 0 10. e-1 0 10. φ=ω}
```
BASIC (1964)

• Developed at Dartmouth in 1960’s by Tom Kurtz, John Kemeny, and a succession of undergraduates; first ran in 1964
• Beginner’s All-purpose Symbolic Instructional Code
• Intended to introduce students in non-scientific disciplines to computing
• Influenced by FORTRAN and ALGOL
• Major goal to simplify user interface:
  • Simplicity chosen over efficiency; time sharing over punched cards
  • Clear error messages; distinctions such as int vs real eliminated
  • Automatic defaults for declarations, values, arrays, output format, etc.
  • Students had access to computers at all times
• No universal BASIC standard:
  • ANSI (American National Standards Institute) is a minimal standard
  • True Basic – Kemeny’s company
PL/1 (1964)

• Planned and designed by IBM as an extension to FORTRAN
• “Extension” departed from FORTRAN specs and was first released as NPL; renamed PL/1 (Programming Language 1)
• Of interest in academic community because it had every element of language design
• Too big and complicated
• Compiler sold separately from machine
• COBOL and FORTRAN already had huge user bases
Innovations of PL/1

- multitasking
- programmer-defined exception handling
- explicit use of pointers and list processing
- wide variety of alternatives for storage allocation (static, automatic, controlled)
- consideration of problems arising from interacting with operating system
ALGOL68

• ALGOL committee produced considerably revised and extended version of ALGOL in 1968
• Huge, general-purpose language, very different from ALGOL60
• Not widely accepted, but influenced many other languages
• ALGOL68 introduced:
  • User-defined data type
  • Pointer type
  (Both significant features of Pascal)
Pascal (1970)

• Designed by Niklaus Wirth
  (member of ALGOL committee; he proposed
  a revision known as ALGOL-W in 1965)
• Pascal first implemented in 1970
• In opposition to trend of PL/1 – ALGOL68 – Ada
• Named after 17th century French philosopher
  and mathematician Blaise Pascal
• Simple and elegant
• Was widely used in academic community
• Interesting features:
  • Case statement
  • Facility for user-defined data types
  • Record structure
C (1972)

- Designed by Ken Thompson and Dennis Ritchie at Bell Labs in 1972
- Designed for coding the routines of the UNIX operating system
- “High level” systems programming language which created the notion of a portable operating system
- Concise syntax – programs somewhat hard to read, understand, debug
- No built-in operations for handling composite data types such as strings, sets, and lists
- Not strongly typed; no run-time type checking
- Easily leads to programming errors
- Provides ability to code low-level operations in a high-level language
Ada

• Designed according to specifications developed by US Dept of Defense

• Requirements stressed structural programming methodology and readability over writability

• Development period 1975 – 1985
  1975: first requirements documents
  1980: complete language proposed
  1983: final standardized version
  1985: working usable compilers appeared

• Contains virtually all elements of PL design
  • Exception handling
  • Parallel processing
  • Abstract data types
# Programming Language Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>Imperative: tell machine how to change its state</td>
</tr>
<tr>
<td>Object-oriented</td>
<td></td>
</tr>
<tr>
<td>Functional</td>
<td>Declarative: specify properties of desired result</td>
</tr>
<tr>
<td>Logic</td>
<td></td>
</tr>
<tr>
<td>[Multi-paradigm]</td>
<td></td>
</tr>
</tbody>
</table>
Programming Language Paradigms

• Procedural: procedures, sequential execution of code are basic building blocks of program

• FORTRAN (FORmula TRANslation System; John Backus, IBM, 1950s)
• ALGOL (ALGOrithmic Language, 1958)
• COBOL (COmmon Business Oriented Language, 1960)
• BASIC (Beginner's All-purpose Symbolic Instruction Code, John Kemeny and Thomas Kurtz, Dartmouth, 1963)
• Pascal (Niklaus Wirth, 1970)
• C (Dennis Ritchie, Bell Labs, 1972)
...
Programming Language Paradigms

• Object-Oriented: Program is designed around the objects required to solve the problem

  • Smalltalk (Alan Kay, Xerox PARC, 1971)
  • C++ (Bjarne Stroustrup, Bell Labs, 1983)
  • Java (James Gosling, Sun Microsystems, 1995)
  • C# (Microsoft, 2000)

...
Programming Language Paradigms

• Functional: Program is designed around the evaluation of *functions*, rather than modifying state

  • **LISP** (John McCarthy, MIT, 1958)
    • Common Lisp
    • Dylan
    • Logo
    • Scheme
  • **ML** (Robin Milner et al, Edinburgh, 1970s)
  • **Haskell** (purely functional language, 1990)
...
Programming Language Paradigms

• Logic: Program is declarative, based on mathematical logic

• Prolog (Alain Colmerauer, 1972)

A program lists facts and rules, program execution is controlled deduction to answer a query
Programming Language Paradigms

• Scripting and multi-paradigm languages

• **awk** (Aho, Weinberger, Kerningham, Bell labs, 1978)
• **Perl** (Larry Wall, NASA, 1987)
• **Tcl/Tk** (John Ousterhout, 1988)
• **Python** (Guido van Rossum, CWI, 1991)
• **PHP** (Rasmus Lerdorf, 1995)
• **Ruby** (Yukihiro “Matz” Matsumoto, 1996)
...
Summary

• Expression grammars
• Early programmable devices / punch cards
• Programming language generations
• Evolution of 3rd-generation languages
• Programming language paradigms
Sources

• Sethi, Programming Languages, 2nd edition
• Sebasta, Concepts of Programming Languages, 8th edition
• https://en.wikipedia.org/wiki/Punched_card
• https://en.wikipedia.org/wiki/Jacquard-machine
• https://en.wikipedia.org/wiki/Analytical_Engine
• https://history-computer.com/Babbage/AnalyticalEngine.html
• http://sydneypadua.com/2dgoggles/
• https://en.wikipedia.org/wiki/Von_Neumann_architecture
• https://cacm.acm.org/magazines/2020/1/241712-von-neumann-thought-turings-universal-machine-was-simple-and-neat/fulltext
• https://en.wikipedia.org/wiki/Programming_language_generations
• https://en.wikipedia.org/wiki/History_of_programming_languages
• https://en.wikipedia.org/wiki/Timeline_of_programming_languages
• https://en.wikipedia.org/wiki/Plankalk%C3%BCl
• https://en.wikipedia.org/wiki/Programming_paradigm
• Wikipedia (individual programming languages, images)