FORTRAN, Part 1

CS4100

February 14, 2011

Reminders

- Jeopardy tournament with Watson starts today
- Assn 2 due today
  - Hard copy now
  - Upload to submission system
- Project proposals due Friday
  - Please upload to submission system

Highlights of Psuedo-Code

- Virtual computer
  - More regularity
  - Higher level
- Decreased chance of errors
  - Automate tedious and error-prone tasks
- Increased security
  - Error checking
- Simplify debugging
  - trace

Now: FORTRAN
The First Generation

- Early 1950s
  - Simple assemblers and libraries of subroutines were tools of the day
  - Automatic programming was considered unfeasible
  - Good coders liked being masters of the trade
- Laning and Zierler at MIT in 1952
  - Algebraic language

Backus at IBM

- Visionary at IBM
- Recognized need for faster coding practice
- Need “language” that allows decreasing costs to linear, in size of the program
- Speedcoding for IBM 701
  - Language based on mathematical notation
  - Interpreter to simulate floating point arithmetic

Backus at IBM

- Goals
  - Get floating point operations into hardware: IBM 704
    - Exposes deficiencies in pseudo-code
  - Decrease programming costs
    - Programmers to write in conventional mathematical notation
    - Still generate efficient code
- IBM authorizes project
  - Backus begins outlining FORTRAN
    - IBM Mathematical FORmula TRANslating System
  - Has few assistants
  - Project is overlooked (greeted with indifference and skepticism according to Dijkstra)

Meanwhile

- Grace Hopper organizes Symposia via Office of Naval Research (ONR)
- Backus meets Laning and Zierler
- Later (1978) Backus says:
  - “As far as we were aware we simply made up the language as we went along. We did not regard language design as a difficult problem, merely as a simple prelude to the real problem: designing a compiler which could produce efficient programs.”
- FORTRAN compiler works!

FORTRAN timeline

- 1954: Project approved
- 1957: FORTRAN
  - First version released
- 1958: FORTRAN II and III
  - Still many dependencies on IBM 704
- 1962: FORTRAN IV
  - “ANS FORTRAN” by American National Standards Institute
  - Breaks machine dependence
  - Few implementations follow the specifications
- We’ll look at 1966 ANS FORTRAN

FORTRAN

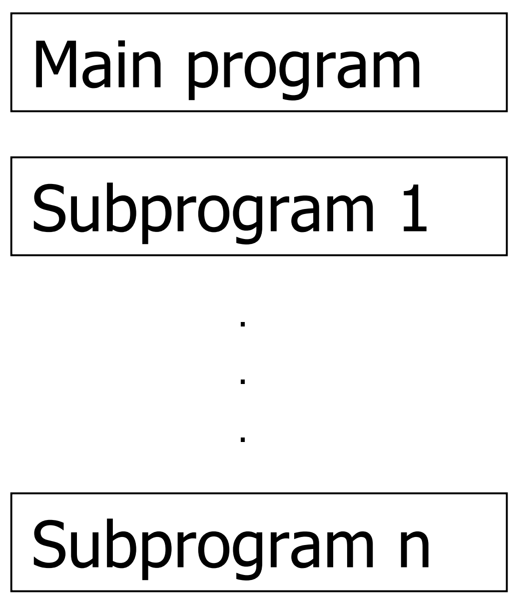
- Goals
  - Decrease programming costs (to IBM)
  - Efficiency

Sample FORTRAN program

```
        DIMENSION DTA(900)
        SUM 0.0
        READ 10, N
10      FORMAT(I3)
        DO 20 I = 1, N
        READ 30, DTA(I)
30      FORMAT(F10.6)
        IF (DTA(I)) 25, 20, 20
25      DTA(I) = -DTA(I)
20      CONTINUE
        …
```

Structural Organization

- Preliminary specification did not include subprograms (like in pseudo-code)
- FORTRAN I, however, already included subprograms



Constructs

- Declarative constructs
  - (First part in pseudo-code: data initialization)
  - Declare facts about the program, to be used at compile-time
- Imperative constructs
  - (Second part in pseudo-code: program)
  - Commands to be executed during run-time

Declarative Constructs

- Declarations include
  - Allocate area of memory of a specified size
  - Attach symbolic name to that area of memory
  - Initialize the memory
- FORTRAN example
  - `DIMENSION DTA (900)`
  - `DATA DTA, SUM / 900*0.0, 0.0`
    - initializes DTA to 900 zeroes
    - SUM to 0.0

Imperative Constructs

- Categories:
  - Computational
    - E.g.: Assignment, Arithmetic operations
    - FORTRAN: `AVG = SUM / FLOAT(N)`
  - Control-flow
    - E.g.: comparisons, loop
    - FORTRAN:
      - `IF`-statements
      - `DO` loop
      - `GOTO`
  - Input/output
    - E.g.: read, print
    - FORTRAN: Elaborate array of I/O instructions (tapes, drums, etc.)

Building a FORTRAN Program

- Interpretation unacceptable, since the selling point is speed
- Need the following stages to build:
  1. Compilation
     Translate code to relocatable object code
  2. Linking
     Incorporating libraries (resolving external dependencies)
  3. Loading
     Program loaded into memory; converted from relocatable to absolute format
  4. Execution
     Control is turned over to the processor

Compilation

- Compilation has 3 phases
  - Syntactic analysis
    - Classify statements, constructs and extract their parts
  - Optimization
    - FORTRAN has considerable optimizations, since that was the selling point
  - Code synthesis
    - Put together parts of object code instructions in relocatable format

DESIGN: Control Structures

- Control structures control flow in the program
- Most important statement in FORTRAN:
  - Assignment Statement

DESIGN: Control Structures

- Machine Dependence (1st generation)
- In FORTRAN, these were based on native IBM 704 branch instructions
  - "Assembly language for IBM 704"

FORTRAN II statement	IBM 704 branch operation
`GOTO n`	`TRA k   (transfer direct)`
`GOTO n, (n1, n2,…,nm)`	`TRA i   (transfer indirect)`
`GOTO (n1, n2,…,nm), n`	`TRA i,k (transfer indexed)`
`IF (a) n1, n2, n3`	`CAS k`
`IF ACCUMULATOR OVERFLOW n1, n2`	`TOV k`
…	…

Arithmetic IF-statement

- Example of machine dependence
  - `IF (a) n1, n2, n3`
  - Evaluate a: branch to
    - n1: if -,
    - n2: if 0,
    - n3: if +
  - CAS instruction in IBM 704
- More conventional IF-statement was later introduced
  - `IF (X .EQ. A(I)) K = I - 1`

Principles of Programming

- The Portability Principle
  - Avoid features or facilities that are dependent on a particular computer or a small class of computers.

GOTO

- Workhorse of control flow in FORTRAN
- 2-way branch:

```
    IF (condition) GOTO 100
          case for false
    GOTO 200
100       case for true
200
```

- Equivalent to *if-then-else* in newer languages

Reversing TRUE and FALSE

- To get *if-then-else* –style if:

```
    IF (.NOT. (condition)) GOTO 100
        case for true
    GOTO 200
100     case for false
200
```

n-way Branching with Computed GOTO

```
    GOTO (L1, L2, L3, L4 ), I
10  case 1
     GOTO 100
20  case 2
     GOTO 100
30  case 3
     GOTO 100
40  case 4
     GOTO 100
100
```

- Transfer control to label L_k if I contains k
- Jump Table

n-way Branching with Computed GOTO

```
    GOTO (10, 20, 30, 40 ), I
10  case 1
     GOTO 100
20  case 2
     GOTO 100
30  case 3
     GOTO 100
40  case 4
     GOTO 100
100
```

- IF and GOTO are *selection statements*

Loops

- Loops are implemented using combinations of IF and GOTOs
- Trailing-decision loop:

```
100 …body of loop…
     IF (loop not done) GOTO 100
```

- Leading-decision loop:

```
100 IF (loop done) GOTO 200
     …body of loop…
    GOTO 100
200 …
```

- Readable?

But wait, there's more!

- Mid-decision loop:

```
100 …first half of loop…
     IF (loop done) GOTO 200
     …second half of loop…
    GOTO 100
200 …
```

Hmmm…

- Very difficult to know what control structure is intended
- Spaghetti code
- Very powerful
- Must be a principle in here somewhere

Principles of Programming

- The Structure Principle (Dijkstra)
  – The static structure of the program should correspond in a simple way to the dynamic structure of the corresponding computations.
- What does this mean?
  – Should be able to visualize behavior of program based on written form

GOTO: A Two-Edged Sword

- Very powerful
  - Can be used for good or for evil
- But seriously is GOTO good or bad?
  - Good: very flexible, can implement elaborate control structures
  - Bad: hard to know what is intended
  - Violates the structure principle