FORTRAN, Part 3

CS4100

February 20, 2012

Reminders

- Assn 2 due Monday, Feb 20th
  - Upload to submission system

DESIGN: Data Structures

- First data structures
  - Suggested by mathematics
    - Primitives
    - Arrays

Primitives

- Primitives are scalars only
  - Integers
  - Floating point numbers
  - Double-precision floating point
  - Complex numbers
  - No text (string) processing

Representations

- Word-oriented
  - Most commonly 32 bits
- Integer
  - Represented on 31 bits + 1 sign bit
- Floating point
  - Using scientific notation: characteristic + mantissa

sm	*sc*	c_7	…	c_0	m_{21}	…	m_0

Arithmetic Operators

- 2 + 3.1 = ?
  - 2 is integer, 3.1 is floating point
- How do we handle this situation?
  - Explicit type-casting: FLOAT(2) + 3.1
    - Type-casting is also called "*coercion*"
  - FORTRAN: Operators are overloaded
  - Automatic type coercion
    - Always coerce to encompassing set
      - Integer + Float → float addition
      - Float * Double → double multiplication
      - Integer – Complex → complex subtraction
    - Types *dominate* their subsets

Example

- X**(1/3) = ?

  1/3 = 0

  1/3.0 = 0.33333

Hollerith Constants

- Early form of character string in FORTRAN
  - 6HCARMEL is a six character string 'CARMEL' (H is for Hollerith)
  - Second-class citizens
    - No operations allowed
    - Can be read into an integer variable, which cannot (should not) be altered
- Problems
  - Integer representing a Hollerith constant may be altered, which makes no sense
- Weak typing
  - No type checking is performed

Constructor: Array

- Constructor
  - Method to build complex data structures from primitive ones
- FORTRAN only has array constructors

  `DIMENSION DTA, COORD(10,10)`
  - Initialization is not required
  - Maximum 3 dimensions

Representation

- Simple, intuitive representation
- Column-major order
  - Most languages do row-major order
  - Addressing equation:
    - $\alpha\{A(2)\} = \alpha\{A(1)\} + 1 = \alpha\{A(1)\} - 1 + 2$
    - $\alpha\{A(i)\} = \alpha\{A(1)\} - 1 + i$
    - $\alpha\{A(i,j)\} = \alpha\{A(1,1)\} + (j - 1)m + i - 1$
    - FORTRAN uses 1-based addressing
      - One addressable slot of each elt

Element	Address
A(1,1)	A
A(2,1)	A + 1
…	
A(m,1)	A + m - 1
A(1,2)	A + m
…	
A(m,2)	A + 2m - 1
…	
A(m,n)	A + nm - 1

Optimizations

- Arrays are mostly associated with loops
  - Most programmers initialize controlled variable to 1, and reference array A(i)
  - Optimization:
    - Initialize controlled variable to address of array element
    - Therefore, we'll increment address itself
    - Dereference controlled variable to get array element

Subscripts

- Subscripts can be expressions
  - A(i+m*c)
  - This defeats above optimization
  - Therefore, subscripts are limited to
    - c and c' are integers, v is an integer variable
    - c
    - v
    - v+c, v-c
    - c*v
    - c*v+c', c*v-c'
  - A(J - 1) ok; A(1+J) not ok
- Optimizations like this sold FORTRAN

DESIGN: Name Structures

- What do name structures structure?
  - Names, of course!
- Primitives bind names to objects
  - INTEGER I, J, K
    - Allocate integers I, J, and K, and bind the names to memory locations
    - Declare: name, type, storage

Declarations

- Declarations are non-executable statements
- Unlike IF, GOTO, etc., which are executable statements
- Static allocation
  - Allocated once, cannot be deallocated for reuse
  - FORTRAN does not do dynamic allocation

Optional Declaration

- FORTRAN does not require variables to be declared
  - First use will declare a variable
- What's wrong with this?
  - COUNT = COUMT + 1
  - What if first use is not assignment?
- Convention:
  - Variables starting with letters i, j, k, l, m, n are integers
  - Others are floating point
  - Bad practice: Encourages funny names (KOUNT, ISUM, XLENGTH…)

Now: Semantics (meaning)

- "They went to the bank of the Rio Grande."
- What does this mean?
- How do we know?
- CONTEXT, CONTEXT, CONTEXT

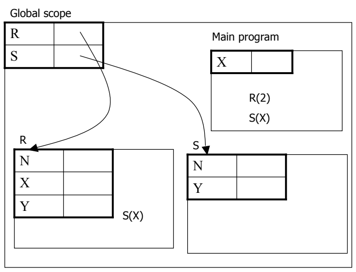
Programming Languages

- X = COUNT(I)
- What does this mean
  - X integer or real
  - COUNT array or function
- Again Context
  - Set of variables visible when statement is seen
- Context is called ENVIRONMENT

SCOPE

- Scope of a binding of a name
  - Region of program where binding is visible
- In FORTRAN
  - Subprogram names GLOBAL
    - Can be called from anywhere
  - Variable names LOCAL
    - To subprogram where declared

Contour Diagram



Once we have subprograms…

- We need to find a way to share data
  - Parameters
    - Pass by reference
    - Pass by value-result
      - Caller copies value of actual to formal variable
      - On return, caller copies result value to actual
        - » Omit for constants or expressions as actuals

Once we have subprograms…

- Share Data With Just Parameters?
  - Cumbersome, and hard to maintain
  - Produces long list of parameters
  - If data structure changes, there are many changes to be made
  - Violates information hiding

Sharing Data

- FORTRAN's solution:
- COMMON blocks allow more flexibility
  - Allows sharing data between subprograms
  - Scope rules necessitation this
- Consider a symbol table

```
SUBROUTINE ARRAY2 (N, L, C, D1, D2)
COMMON /SYMTAB/ NAMES(100), LOC(100), TYPE(100)
...
SUBROUTINE VAR (N, L, C)
COMMON /SYMTAB/ NAMES(100), LOC(100), TYPE(100)
```

COMMON Problems

- Tedious to write
- Unreadable
- Virtually impossible to change AND
- COMMON permits aliasing, which is dangerous
  - If COMMON specifications don't agree, misuse is possible

Aliasing

- The ability to have more than one name for the same memory location
- Very flexible!

```
COMMON /B/ M, A(100)

COMMON /B/ X, K, C(50), D(50)
```

EQUIVALENCE

- Since dynamic memory allocation is not supported, and memory is scarce, FORTRAN has EQUIVALENCE

```
DIMENSION INDATA(10000), RESULT(8000)
EQUIVALENCE INDATA(1), RESULT(8)
```

- Allows a way to explicitly alias two arrays to the same memory

EQUIVALENCE

- This is only to be used when usage of INDATA and RESULT do not overlap
- Allows access to different data types (float as if it was integer, etc.)
- Has same dangers as COMMON

DESIGN: Syntactic Structures

- Languages are defined by lexics and syntax
  - Lexics
    - Way to combine characters to form words or symbols
    - E.g. Identifier must begin with a letter, followed by no more than 5 letters or digits
  - Syntax
    - Way to combine symbols into meaningful instructions
- Syntactic analysis:
  Lexical analyzer (scanner)
  Syntactic analyzer (parser)

Fixed Format Lexics

- Still using punch-cards!
- Particular columns had particular meanings
- Statements (columns 7-72) were free format

Columns	Purpose
1-5	Statement number
6	Continuation
7-72	Statement
73-90	Sequence number

Blanks Ignored

- FORTRAN ignored spaces (not just white spaces)
- Thisisveryunfortunate!

```
DIMENSION INDATA(10000), RESULT(8000)
D I M E N S I O N I N D A T A (1 0 0 0 0), R E S U L T (8000)
DIMENSIONINDATA(10000),RESULT(8000)
```

- Lexing and parsing such a language is very difficult

Blanks Ignored

- In combination with other features, it promoted mistakes

```
DO 20 I = 1. 100
DO 20 I = 1, 100
DO20I = 1.100
```

- Variable DO20I is unlikely, but . and , are next to each other on the keyboard…

No Reserved Words

- FORTRAN allows variable named IF

```
DIMENSION IF(100)
```

- How do you read this?

```
IF (I - 1) = 1 2 3
IF (I - 1) 1, 2, 3
```

- The compiler does not know what `IF (I - 1)` will be
  - Needs to see , or = to decide

Algebraic Notation

- One of the main goals was to facilitate scientific computing
  - Algebraic notation had to look like math
  - (-B + SQRT(B**2 – 4*AA*C))/(2*A)
  - Very good, compared to our pseudo-code
- Problems
  - How do you parse and execute such a statement?

Operators Need Precedence

- $b^2 - 4ac == (b^2) - (4ac)$
- $ab^2 == a(b^2)$
- Precedence rules
  1. Exponentiation
  2. Multiplication and division
  3. Addition and subtraction
- Operations on the same level are associated to the left (read left to right)
- How about unary operators (-)?

Some Highlights

- Integer type is overworked
  - Integer
  - Character strings
  - Addresses
- Weak typing
- Combine the two and we have a security loophole
  - Meaningless operations can be performed without warning

Some Highlights

- Arrays
  - Only data structure
  - Data constructor
  - Static
  - Limited to three dimensions
  - Restrictions on index expressions
  - Optimized
  - Column major order for 2-dimensional
  - Not required to be initialized