Imperative Programming
The Case of FORTRAN

ICOM 4036
Lecture 4
The Imperative Paradigm

- Computer Model consists of bunch of variables
- A program is a sequence of state modifications or assignment statements that converge to an answer
- PL provides multiple tools for structuring and organizing these steps
  - E.g. Loops, procedures

This is what you have been doing since INGE 3016!
A Generic Imperative Program

START
Initialize Variables
Modify Variables
Converged?
yes
no
END
int fibonacci(int f0, int f1, int n) {
    // Returns the nth element of the Fibonacci sequence
    int fn = f0;
    for (int i=0; i<n; i++) {
        fn = f0 + f1;
        f0 = f1;
        f1 = fn;
    }
    return fn;
}
Examples of (Important) Imperative Languages

• FORTRAN (J. Backus IBM late 50’s)
• Pascal (N. Wirth 70’s)
• C (Kernigham & Ritchie AT&T late 70’s)
• C++ (Stroustrup AT&T 80’s)
• Java (Sun Microsystems late 90’s)
• C# (Microsoft 00’s)
FORTRAN Highlights

• For High Level Programming Language ever implemented
• First compiler developed by IBM for the IBM 704 computer
• Project Leader: John Backus
• Technology-driven design
  – Batch processing, punched cards, small memory, simple I/O, GUI’s not invented yet
Some Online References

• Professional Programmer’s Guide to FORTRAN
• Getting Started with G77

Links available on course web site
Structure of a FORTRAN program

PROGRAM <name>
    <program_body>
END

SUBROUTINE <name> (args)
    <subroutine_body>
END

FUNCTION <name> (args)
    <function_body>
END
Lexical/Syntactactic Structure

• One statement per line
• First 6 columns reserved
• Identifiers no longer than 6 symbols
• Flow control uses numeric labels
• Unstructured programs possible
Hello World in Fortran

PROGRAM TINY
  WRITE(UNIT=*, FMT=*) 'Hello, world'
END
PROGRAM LOAN

WRITE(UNIT=*, FMT=*)'Enter amount, % rate, years'
READ(UNIT=*, FMT=*) AMOUNT, PCRATE, NYEARS
RATE = PCRATE / 100.0
REPAY = RATE * AMOUNT / (1.0 - (1.0+RATE)**(-NYEARS))
WRITE(UNIT=*, FMT=*)'Annual repayments are ', REPAY
END
FORTRAN By Example 2

PROGRAM LOAN
  WRITE (UNIT=*, FMT=*)'Enter amount, % rate, years'
  READ (UNIT=*, FMT=*) AMOUNT, PCRATE, NYEARS
  RATE = PCRATE / 100.0
  REPAY = RATE * AMOUNT / (1.0 - (1.0+RATE)**(-NYEARS))
  WRITE (UNIT=*, FMT=*)'Annual repayments are ', REPAY
END
PROGRAM LOAN
  WRITE(UNIT=*, FMT=*) 'Enter amount, % rate, years'
  READ(UNIT=*, FMT=*) AMOUNT, PCRATE, NYEARS
  RATE = PCRATE / 100.0
  REPAY = RATE * AMOUNT / (1.0 - (1.0+RATE)**(-NYEARS))
  WRITE(UNIT=*, FMT=*) 'Annual repayments are ', REPAY
END
A loop consists of two separate statements

-> Easy to construct **unstructured** programs
FORTRAN Do Loops

A loop consists of two separate statements
-> Easy to construct unstructured programs
PROGRAM REDUCE
WRITE(UNIT=*, FMT=*)'Enter amount, % rate, years'
READ(UNIT=*, FMT=*) AMOUNT, PCRATE, NYEARS
RATE = PCRATE / 100.0
REPAY = RATE * AMOUNT / (1.0 - (1.0+RATE)**(-NYEARS))
WRITE(UNIT=*, FMT=*)'Annual repayments are ', REPAY
WRITE(UNIT=*, FMT=*)'End of Year Balance'
DO 15, IYEAR = 1, NYEARS, 1
   AMOUNT = AMOUNT + (AMOUNT * RATE) - REPAY
   WRITE(UNIT=*, FMT=*)IYEAR, AMOUNT
15 CONTINUE
END

Enter amount, % rate, years
2000, 9.5, 5
Annual repayments are 520.8728
End of Year Balance
  1 1669.127
  2 1306.822
  3 910.0968
  4 475.6832
  5 2.9800416E-04

• optional increment (can be negative)
• final value of index variable
• index variable and initial value
• end label
FORTRAN Functions I

PROGRAM TRIANG
    WRITE(UNIT=*,FMT=*)'Enter lengths of three sides:'
    READ(UNIT=*,FMT=*) SIDEA, SIDEB, SIDE C
    WRITE(UNIT=*,FMT=*)'Area is ', AREA3(SIDEA, SIDEB, SIDE C)
END

FUNCTION AREA3(A, B, C)
*     Computes the area of a triangle from lengths of sides
    S = (A + B + C)/2.0
    AREA3 = SQRT(S * (S-A) * (S-B) * (S-C))
END

• No recursion
• Parameters passed by reference only
• Arrays allowed as parameters
• No nested procedure definitions – Only two scopes
• Procedural arguments allowed
• No procedural return values

Think: why do you think FORTRAN designers made each of these choices?
REAL FUNCTION AREA3(A, B, C)
* Computes the area of a triangle from lengths of its sides.
* If arguments are invalid issues error message and returns
* zero.
REAL A, B, C
S = (A + B + C)/2.0
FACTOR = S * (S-A) * (S-B) * (S-C)
IF(FACTOR .LE. 0.0) THEN
   STOP 'Impossible triangle'
ELSE
   AREA3 = SQRT(FACTOR)
END IF
END
FORTRAN ARRAYS

Subroutines are analogous to void functions in C

Parameters are passed by reference

SUBROUTINE MEANSD(X, NPTS, AVG, SD)
    INTEGER NPTS
    REAL X(NPTS), AVG, SD
    SUM = 0.0
    SUMSQ = 0.0
    DO 15, I = 1,NPTS
        SUM = SUM + X(I)
        SUMSQ = SUMSQ + X(I)**2
    15 CONTINUE
    AVG = SUM / NPTS
    SD = SQRT(SUMSQ - NPTS * AVG)/(NPTS-1)
END
subroutine checksum(buffer,length,sum32)
C       Calculate a 32-bit 1's complement checksum of the input buffer, adding
C       it to the value of sum32. This algorithm assumes that the buffer
C       length is a multiple of 4 bytes.
C       a double precision value (which has at least 48 bits of precision)
C       is used to accumulate the checksum because standard Fortran does not
C       support an unsigned integer datatype.
C       buffer - integer buffer to be summed
C       length - number of bytes in the buffer (must be multiple of 4)
C       sum32 - double precision checksum value (The calculated checksum
C       is added to the input value of sum32 to produce the
C       output value of sum32)

integer buffer(*),length,i,hibits
double precision sum32,word32parameter (word32=4.294967296D+09)
C                 (word32 is equal to 2**32)
C       LENGTH must be less than 2**15, otherwise precision may be lost
C       in the sum
if (length .gt. 32768)then
   print *, 'Error: size of block to sum is too large'
   return
end if

   do i=1,length/4
      if (buffer(i) .ge. 0)then
         sum32=sum32+buffer(i)
      else
         C               sign bit is set, so add the equivalent unsigned value
         sum32=sum32+(word32+buffer(i))
      end if
   end do

   fold any overflow bits beyond 32 back into the word
10   hibits=sum32/word32
    if (hibits .gt. 0)then
       sum32=sum32-(hibits*word32)+hibits
       go to 10
    end if
end if
end subroutine checksum
• WhiteBoard Exercises