Using C and C++ with Fortran

Liu Hui

April 3, 2006

1 Background

- large existing
- standardize the interface with C and C++(ISO)
- supported by OS & compiler, eg DEC VAX (open) VMS,PC DOS, Unix
- f2c, weakness, I/O, stable, clean, under development

2 run-time differences

- interrupt and trap handling
- I/O
- management of dynamically-allocated heap memory

3 data type differences

3.1 Fortran data types

• first high-level language that designed primarily for numerical programming.

• INTEGER, LOGICAL, REAL, COMPLEX and Hollerith (11HHello,world) (1890, IBM)

• DOUBLE PRECISION (early 1960s)

• COMPLEX*8, COMPLEX*16, INTEGER*2, INTEGER*4, LOGICAL*1 LOGICAL*2, LOGICAL*4, REAL*4, and REAL*8 (1964) IBM System/360, byte-addressable, IBM Fortran compiler

• REAL*16, COMPLEX*32 (1967)

• BYTE, POINTER, even INTEGER*3,INTEGER*6 and REAL*6 (market share, disappear

• CHARACTER*n(ANSI/ISO Fortran 77)

• POINTER(ANSI/ISO Fortran 90/95). ignore byte-length modifiers, defines a new syntax

• NOT define STRUCTURE, COMMON blocks

3.2 C/C++ data types

- int— char, short, int, long (long long) (unsigned/signed)
- floating-point — float, double (long double)
- union, enum, structure, pointers to functions and data
- class (C++)
- NOT define boolean and complex
- bool, complex, double complex (C++98, via classes)
- bool (C99)

3.3 common data types

- int = INTEGER
- float = REAL
- double = DOUBLE PRECISION

4 array

4.1 storage order

- Fortran stores in row order.
- C and C++ store in column. performance
- S1. wrap array-transposition
- S2. program C and C++ matrix code with reversed index order
- S3. preprocessor macro, eg #define A(i,j) a[j][i]

4.2 array indexing

- Fortran A(N,N) 1,N
- C/C++ a[N][N] 0,N-1
- solution: use preprocessor ,eg #define B(i,j) b[j-1][i-1]

5 Function return types

- Fortran :SUBROUTINE , CALL
- Fortran :FUNCTION ,EXPRESSION (CALL, NOT PORTABLE)
- C/C++ :function, return a value or not; scalar values(structure and class(C++))
- C/C++ :one register. complex, complex*16

6 file types

6.1 The Fortran view of file

• FORMATTED(text) and UNFORMATTED(BINARY)

• binary files:compact, fast; data conversion and accuracy loss

• text files:readable, portable, editable

• records.line boundary for text files and special marker for binary files. (records may have different lengths for sequential files.)

• random access:records must have uniform length.

6.2 The C/C++ view of files

- unstructured stream of zero or more bytes
- fgetc(), fputc(), fgets(), fputs().....
- structure
- \n LF(ASCII,10 ,line), CR(ASCII,13)

6.3 Mixed-language file processing

• text

• exponent:use Ew.d, not Dw.d

• blank, eg 0.23E 3, not 0.23E+03. Unix/Linux ×

• two digits. use Ew.dEi. 1PEw.dEi(1.23E+23),0PEw.dEi(0.23E+11)

• avoid using nondecimal numeric data or length suffixes (l, L, f, F), or unsigned numeric data types. (for Fortran read)

• width, separator(blank,comma)

7 Memory management

- malloc, free (C); new, delete (C++)
- pointer: uncommon, not powerful(Fortran). large array

8 Calling sequence

8.1 Argument addressing

• Fortran : address (direct reference or → local variable, copy back)

• C :value

```
CALL FOO(3)
I=3
PRINT *,I
.........


    SUBROUTINE FOO(J)
J=50
END
```

8.2 Routine naming

• C:external name,linker, the same

$cat hello.c
void hello(void)

$gcc -c hello.c && nm foo.o | grep 'T'

• C++:compiler-independent

• Fortran(Unix/Linux), three types

• T1:underscore. world's first Fortran 77 compiler, 1976, Stu Feldman, Bell Lab. f77 in Unix, FreeBSD, NetBSD and OpenBSD, f2c, GNU g77, and Fortran 77, 90 and 95 compilers from Compaq/DEC, the Portland Group, SGI, SUN.....

• the same with SUBROUTINE. Unix Fortran compilers from Hewlett-Packard and IBM....

• UPPERCASE. compilers form Ardent, Stellar, and Stardent.....

9 COMMON blocks

• reuse; share,information hiding

• INCLUDE, nonstandard

• initialize: direct assignment at runtime; DATA statements in a BLOCK DATA routine.

• R1:A named COMMON block may be initialized with DATA statements in only a single BLOCK DATA routine.
• R2:It is illegal to initialize blank COMMON variables with DATA statements.
• R3:It is illegal to initialize COMMON variables with DATA statements inside a SUBROUTINE or FUNCTION.

• BLOCK DATA cannot be called. explicitly loaded by the linker; garbage; libraries cannot use BLOCK DATA. no equivalent of Fortran COMMON blocks in C and C++

• aligned at memory addresses; order:
COMPLEX*32
REAL*16 AND COMPLEX*16
COMPLEX*8 REAL*8 AND DOUBLE PRECISION
REAL INTEGER AND LOGICAL
INTEGER*2 AND LOGICAL*2
BYTE INTEGER*1 AND LOGICAL*1

- external name : BLOCK DATA and blank COMMON

```
#include <stdio.h> #include <stdlib.h>

#include "common.h"    /* for FORTRAN() macro */

#define C_xyzcb FORTRAN(xyzcb,XYZCB) #define C_uvwcb
FORTRAN(uvwcb,UVWCB)

extern struct {
    int u;
    int v;
    int w;
} C_uvwcb;

extern struct {
    double x;
    double y;
    double z;
} C_xyzcb;

int main() {
    (void)printf("%6d     %6d     %6d\n",   C_uvwcb.u, C_uvwcb.v, C_uvwcb.w);
    (void)printf("%11.3le%11.3le%11.3le\n", C_xyzcb.x, C_xyzcb.y, C_xyzcb.z);
    return (EXIT_SUCCESS);
}
```

- common.h

```
$ cat common.f
      REAL A,B,C
      COMMON / / A,B,C

      INTEGER U,V,W
      COMMON /UVWCB/ U,V,W

      DOUBLE PRECISION X,Y,Z
      COMMON /XYZCB/ X,Y,Z

      A = 1.0
      B = 2.0
      C = 3.0
```

```
      WRITE (6,'(3(F6.3, 5X))') A,B,C
      WRITE (6,'(3(I6,   5X))') U,V,W
      WRITE (6,'(1P3E11.3E3)')  X,Y,Z

      END

*************************************

      BLOCK DATA BDXYZ

      DOUBLE PRECISION X,Y,Z
      COMMON /XYZCB/ X,Y,Z

      DATA X,Y,Z / 1.111111111111111D+100,
     X             2.2222222222222222D+200,
     X             3.3333333333333333D+300 /

      END

*************************************

      BLOCK DATA

      INTEGER U,V,W
      COMMON /UVWCB/ U,V,W

      DATA U,V,W / 123456, 234567, 345678 /

      END
```

10 Recursion

• call itself, 1970s.

```
int factorial(int num)
{
    if(num==0) return 1;
    else return num * factorial(num-1);

}
```

• Tail Recursion Elimination

```
int factorial(int num, int factor)
{
    if(num==0) return factor;
    else return factorial(num - 1,num * factor);

}
```

11 Run-time array dimension

• passed to routines and used there without having to know at compile time what their dimensions .

```
      INTEGER NA, NB, NC
      PARAMETER (NA = 3, NB = 4, NC = 17)
      REAL X(NA,NB,NC)
      ...
      CALL SOLVE(X, NA, NB, NC, MA, MB, MC)
      ...
      SUBROUTINE SOLVE(Y, MAXYA, MAXYB, MAXYC, KYA, KYB, KYC)
      INTEGER KYA, KYB, KYC, MAXYA, MAXYB, MAXYC,
      REAL Y(MAXYA, MAXYB, MAXYC)
      INTEGER I, J, K

      ...
      DO 30 K = 1, KYC
          DO 20 J = 1, KYB
              DO 10 I = 1, KYA
              ... Y(I,J,K) ...
   10         CONTINUE
   20     CONTINUE
   30 CONTINUE
      END
```

• Y(I,J,K) is found at addressof(Y(1,1,1)) + (I - 1) + (J - 1)*MAXYA + (K-1)*MAXYA*MAXYB)

• Y(MAXYA, MAXYB, 1); Y(MAXYA, MAXYB, *)

• C/C++:no hidden information about dimensions; not provide run-time dimensioning ; preprocessor

```
#define Y(i,j,k) ((i)*maxyb*maxyc + (j)*maxyc + (k))
            /* assuming C/C++ storage order */
void solve(float y[][][],
           int maxya, int maxyb, int maxyc,
           int kya, int kyb, int kyc)
{
    ... Y(i,j,k) ...
}
```

• GNU C and C++ compilers, gcc and g++, suport for run-time array dimensions in the 1990s. C99:variable-length array, just C standard, not C++.

```
void solve2(int maxya, int maxyb, int maxyc,
            int kya, int kyb, int kyc,
            float y[maxya][maxyb][maxyc])
{
    ... y[i][j][k] ...
}
```

12 CHARACTER*n arguments

• Hollerith, count charactors, no standard data type to represent routine argument of Hollerith type. then, integer array (portable), eg

```
CALL S(12HHello, world, 12)
END
SUBROUTINE S(MSG,N)
INTEGER K, N, M
INTEGER MSG(1)
M = (N + 3) / 4
WRITE (6,'(20A4)') (MSG(K), K = 1,M)
END
```

• quoted strings, eg 7HO'Neill could be written as 'O' 'Neill'
• lowercase avalable, 1970s
• BYTE, not universal; word-addressable architecture ×

• CHARACTOR*n (Fortran 77), nonportable, horrible....

• S1. fixed-length

• S2. no empty string; C/C++ '\0'; blanks, Fortran

• S3. long string → shorter string, truncates.....

• S4. short string → longer string, pads blanks

• S5. substring facility cannot be applied to string constants.

• S6. no FORMAT item support for centered or left-aligned

• S7. CHARACTOR*n functions return strings of fixed length

• S8. no primitives for reverse searching, for letter-case conversion, for comparisons ignoring letter case in the native and ASCII character sets, for translation, for subset and regular-expression matching, or for tokenizing.....

• S9. unprintable charactor, write yourself by using ASCII, eg C string "Hello\tWorld\n"→ 'Hello' // char(9) // 'world' // char(10).

• S10. overlaping is illegal. eg
S=STR(2:19), T=STR(5:22), S=T ×, you have to write as a loop.

```
      DO 10 I = 5, 22
          TMP = T(I:I)
          S(I-3:I-3) = TMP
   10 CONTINUE
```

• S11. CHARACTOR*n cannot reside in a COMMON block with data of any other type.

• S12. connot overlay data of other types

• S13. length unvisible. current size ×, no sizeof ()

• Solution : Stu Feldman's UNIX f77, supplies a additional argument, the string length , for each CHARACTOR*n argument,but extra arguments are placed at the end of the argument list.
works for old Fortran with quoted string and Fortran 77 data type, for example

```
      CALL FOO(123, 'Hello', 3.14, 'World')
      ...

      SUBROUTINE FOO(A, B, C, D)
      INTEGER A
      INTEGER B(1)
      REAL C
      INTEGER D(1)
      ...
      END

      SUBROUTINE FOO(A, B, C, D)
      INTEGER A
      CHARACTER*(*) B
      REAL C
      CHARACTER D(*)
      ...
      END
```

```
void foo_(int *a, char *b, float *c, char *d,
         int *_len_b, int *_len_d)
{ }
```

• Other solution. Hewlett-Packard, IBM; depend on OS

References

[1] Richard Heathfield, Lawrence Kirby, *C unleashed* , 2002

[2] comp.lang.c *http://www.faqs.org/faqs/C-faq/faq/*, 1999

[3] Nelson H. F. Beebe, *http://www.math.utah.edu/software/c-with-fortran.html*, 2001

Thanks