HOW TO CALL THE IMSL FORTRAN LIBRARIES FROM C

by
Jangwon Kim
Linda M. Robertson
Visual Numerics, Inc.

IMSL Technical Report Series
No. 8902

CONTACTS
Visual Numerics, Inc.
Sales Division
1300 W. Sam Houston Pkwy, Suite 150
Houston, Texas 77042, USA.

Corporate Headquarters 713 784-3131 FAX: 713 781-9260
Email : marketing@houston.vni.com
Web: www.vni.com

Trademark Acknowledgments

All terms mentioned in this report that are known to be trademarks or service marks are listed below. Visual Numerics cannot attest to the accuracy of this information. Use of a term in this report should not be regarded as affecting the validity of any trademark or service mark.

Solaris is a trademark of Sun Microsystems, Inc.
UNIX is a registered trademark of AT&T.
VAX, OpenVMS and VMS are trademarks of Compaq.
CRAY T3E and CF90 are trademarks of Cray Inc.
CRAY and UNICOS are registered trademarks of Cray Inc.
HP and HPUX are registered trademarks of Hewlett Packard Company.
IBM and AIX are registered trademarks of International Business Machines Corp.
Microsoft is a registered trademark of Microsoft Corp.
SGI and IRIX are trademarks of Silicon Graphics, Inc.
Linux is a registered trademark of Linus Torvalds.
Red Hat is a trademark of Red Hat, Inc.

How to Call the IMSL Fortran Libraries from C

Jangwon Kim
Linda M. Robertson
Visual Numerics, Inc.

December 15, 1989

Revised September 2000
T. D. Schweizer

Abstract

This report discusses how to call the IMSL Fortran Numerical Libraries from a C program. To use C with the IMSL Fortran Libraries, the C and Fortran compilers must share the same object file format. Interfacing between C and Fortran is dependent on the operating system, and on the compatibility of the compilers and runtime libraries.

This report begins with a discussion on how to overcome the syntax differences between Fortran and C that are common to all the environments tested, followed by a discussion on machine and system dependent differences including compiling and linking procedures. It also provides a complete example for each environment tested.

Introduction

This report describes how to call the IMSL Fortran 90 MP Library from a C program. To use C with the IMSL Fortran library, the C and Fortran compilers must share the same object file format. If the C and Fortran compilers do not share the same object file format, the programs will not link properly. There are many environments for which the IMSL Libraries are available, but we have limited the discussion of our report to the following environments.

Sun Solaris Release 2.6

Compaq Tru64 Unix 5.0

IBM RS6000 AIX Version 4.3

HP UX 10.01

Cray T3E

OpenVMS Version 7.2

Windows NT /95 /98 with Compaq Visual Fortran 6.5

Red Hat Linux

SGI IRIX 6.5

The first section of this report discusses some general differences between C and Fortran.

Sections 2 through 5 discuss machine and system dependent differences between C and Fortran. The environments tested in this report have been categorized into four groups. Several unix operating systems are discussed in the first group, the CRAY UNICOS environment is discussed in the second group, VMS is discussed in the third group, and PC Windows in the fourth. This classification is based on the operating system, C and Fortran compilers, and the runtime libraries. For each group, we discuss how to pass a character string argument, how to pass a function as an argument, and how to compile a C main program and link it with the IMSL Fortran Libraries. At the end of each group, a complete listing of a C program that calls the IMSL linear solver, LSLRG, is provided. This example covers most of the topics discussed in this report.

Section 6, "Other Environments," provides some suggestions for calling the IMSL Fortran Libraries from C in environments not discussed in this report.

1. Syntax Differences Between C and Fortran

Programs that mix C and Fortran can be linked provided the syntax is understood by both languages. If the syntax is not understood by both languages, the differences can usually be overcome with some modifications to the program. This section discusses some general syntax differences between C and Fortran and how to overcome these differences when writing C programs that call the IMSL Fortran Libraries. In particular, data types, data storage, and the argument passing mechanism are discussed.

1.1 Data Types and Data Storage

The length of the data type is environment specific and should be considered when declaring variables. C and Fortran declarations are related as follows:

C	**Fortran**
char	CHARACTER
short int	INTEGER*2
long int	INTEGER*4
float	REAL*4
double	REAL*8

The IMSL Fortran 90 Library routines use 32-bit integers (or larger) and thus, for C compilers that specify the size of int as 16-bits, integers must be declared using **long int.**

- complex data

Complex and double complex data types are not defined in C. However, they are frequently used in the IMSL Fortran 90 Library. To get around this problem, an equivalent structure (at least for all environments in this report) can easily be defined as follows:

```
/* type definition of (single) complex data type */
 typedef struct
       {float r, i;} complex;

/* type definition of double complex data type */
typedef struct
      {double r, i;} double_complex;
```

1.2 Argument Passing Mechanism

Since Fortran receives an argument as its address (call by reference), the user's C program should not pass the value of an argument (call by value). For character strings in the VMS environment, another type, call by description, is used. Below, the argument passing mechanism is explained for the different data storage types such as scalars, arrays, (double) complex types, character strings, and function arguments.

- passing a scalar argument

A scalar variable should be passed by its address; therefore, use the prefix, &, unless it is already declared as an address.

A constant should not be passed to a Fortran subroutine by value. Use its address instead. For example, in order to pass 1 (one) in the Sun environment, use:

```
extern void forsub_ ();
long int one = 1;

forsub_ ( . . ., &one, . . . );
```

- passing an array argument

The name of an array in C is a pointer to the first element of the array. Therefore, the prefix, &, is not needed and the name itself is passed to the Fortran subroutine, unless a subset of the array is to be passed.

For a two (or higher) dimensional array, Fortran is column major while C is row major. Therefore, a two-dimensional array variable must be transposed before it is passed to a Fortran subroutine. The following example transposes a two-dimensional array using the IMSL routine, TRNRR, in the Sun environment.

```
double a[2] [3] = {11., 12., 13.,
                   21., 22., 23.};
main ()
{

  extern void dtrnrr_ ();
  long int nra=3, nca=2, lda=3;
  long int nrb=2, ncb=3, ldb=2;
  double b[3] [2];

  dtrnrr_ (&nra, &nca, a, &lda, &nrb, &ncb, b, &ldb);
}
```

Although matrix a in the previous example is a 2 x 3 matrix, it is received by dtrnrr as a 3 x 2 matrix. The matrix is then transposed using dtrnrr and the output matrix, b, is understood by Fortran to be a 2 x 3 matrix.

Two-dimensional arrays can also be passed by using the transpose option if it exists for the IMSL routine.

Another method of passing a two-dimensional array is to simply declare the transpose of the two-dimensional array in the C code. For example, matrix a in the previous example could be declared as follows:

```
double a[3] [2] = {11., 21.,
                   12., 22.,
                   13., 23.};
```

- passing a complex or double complex data argument

Complex and double complex data types can be declared using the **typedef** statement discussed earlier. The following example repeats the definition of a complex data type and passes z = 1 + 2i to a Fortran subroutine. (This example is for the UNIX environment.)

```
/* type definition of (single) complex data type */
typedef struct
    {float r, i;} complex;

main ()
{
complex z;

z.r = 1.;
z.i = 2.;
forsub_ (...,&z, ...);
}
```

A double complex data type can be defined and used analogously to a complex data type if the Fortran compiler allows DOUBLE COMPLEX data type.

- passing a character string argument

When a character string variable is passed from C to Fortran, it must be passed with information about its length in addition to other attributes of the string. In C, a character string variable (not a character variable) is terminated with the null character, \0. However, in Fortran, the length of a character string is stated explicitly. Since the method for passing a character string is system dependent, this is discussed in each environment section of this report.

- passing a function as an argument

A C function can usually be passed to a Fortran subroutine as an argument. However, a user must be cautious in that Fortran invokes the C function and passes the arguments by address. Passing a function as an argument is system dependent; therefore, it will be discussed in each environment section along with other system dependent function declarations.

2. Unix Platforms

The topics discussed in this section have been tested with the IMSL Fortran 90 MP library version 4.01, using the following environments:

Sun Solaris Release 2.5 with the Sun Workshop 5.0 compilers

Compaq Tru64 Unix Version 5.0 with Compaq C compiler version 6.1-019 and Compaq Fortran version 5.3-915.

IBM RS/6000 AIX Version 4.1.2 with IBM C 3.6.6 and XL Fortran version 6.1.

HP UX 10.01 with HP C 10.01 and HP Fortran 90 1.0

Red Hat Linux 6.1 with GNU C egcs-2.91.66 and Portland Group pgf90 3.1-2

SGI 64 bit IRIX 6.5 with SGI Mipspro C and Fortran 90 version 7.2

The UNIX operating system is case sensitive by nature. Since the IMSL Fortran Libraries were developed in lowercase for the environments specified in this section, all IMSL symbols and identifiers, such as names of IMSL subroutines and functions, and common block identifiers, should be in lowercase.

Also, on Solaris, Compaq Unix, AIX, SGI, and linux, the object file format requires a trailing underscore (_) at the end of the identifiers. For example, the IMSL routine, DTRNRR is written as dtrnrr_. The underscore is not required on the HP. All examples in this section are shown for a Solaris system.

For HP systems, the examples would be identical to those shown except that the function calls do not have an underscore appended as the last character of the function name.

2.1 Passing a Character String Argument

When passing character string arguments in the environments in this section, explicitly provide the lengths of the strings as additional arguments beyond the last argument of the Fortran subprogram; otherwise, the length is hidden to Fortran. The following example transposes a two dimensional array and prints the result. Note that **lstr** is provided as the length of **char title,** which is ten characters long, even though it does not show up in IMSL subroutines.

```
/*
```

```
* This example transposes a two dimensional matrix
* and prints it out.
*/
double a[2] [3] = {11., 12., 13.,
                   21., 22., 23.};
char title[] = "2x3 matrix";

main ()
{
extern void dtrnrr_(), dwrrrn_();    /* external subroutines */
long int nra=3, nca=2, lda=3;        /* matrix a is nra by nca */
long int nrb=2, ncb=3, ldb=2;        /* matrix b is nrb by ncb */
long int itring=0, lstr=10;          /* triangle option and title length */
double b[3][2];

/* transpose matrix a and save the result in matrix b */
 dtrnrr_ (&nra, &nca, a, &lda, &nrb, &ncb, b, &ldb);

/* write the matrix b with a title */
dwrrrn_ (title, &nrb, &ncb, b, &ldb, &itring, lstr);

}
```

The output from Solaris is:

```
2x3 matrix

              1         2       3
1            11        12      13
2            21        22      23
```

2.2 Passing a Function as an Argument

For the environments in this section, a C function can simply be passed to a Fortran subroutine as an argument. The following example shows how to pass a function to IMSL routine DQDNG. Note that myfunc references x as a pointer because Fortran will invoke myfunc and pass the address of x.

```
/*
* This example shows how to integrate a user supplied function,
* myfunc, at the given interval [0,2].
*/

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

```
/*
* myfunc calculates and returns *x * exp(*x). x is a pointer
* since Fortran calls myfunc and passes the address of x.
*/

 double myfunc (x);
 double *x;
 {

double exp ();
return (*x * exp (*x));
}

main ()
{
extern void dqdng_();
double exp(), fabs(), myfunc();
double lower=0.e0, upper=2.e0, errabs=0.e0, errrel=1.e-7;
double result, errest, exact;

/* IMSL routine dqdng integrates a smooth function
   using a nonadaptive rule. */
dqdng_ (myfunc, &lower, &upper, &errabs, &errrel, &result, &errest);

/* The exact answer is already known and it is 1 + exp(2). */
exact = l.e0 + exp(2.e0);

/* print the result */
printf ("\nIntegral of y = x*exp(x) on [0,2].\n\n");
printf ("Exact answer = %8.5f\n", exact);
printf ("IMSL dqdng_ = %8.5f\n", result);
printf ("Error = %l2.5e\n", fabs (exact - result));
}
```

The output from Solaris is:

```
Integral of y = x*exp(x) on [0,2].

Exact answer = 8.38906
IMSL dqdng_ = 8.38906
Error          =  0.00000e+00
```

2.3 Compilation and Linking

The example in Section 2.4 was successfully compiled and linked with version 4.01 of the IMSL Fortran 90 library, using the UNIX command cc, as shown for the following platforms:

- Sun Solaris 2.5 with Sun Workshop 5.0 Fortran 90 2.0

```
cc -o main main.c  $LINK_F90 –lfui –lfai –lfai2 –lfsumai –lfprodai –lfminlai –lfmaxlai \
 –lfminvai –lfmaxvai –lfsu  -lsunmath -lm
```

- Compaq Unix 5.0

```
cc -o main main.c  $LINK_F90 -lfor -lUfor -lm
```

- IBM RS/6000 AIX 4.3 with IBM C 3.6.6.0 and XL Fortran 6.1.0.0

```
cc -o main main.c  $LINK_F90 -lxlf -lxlf90 -lc -lm
```

- HP UX 10.01 with HP C 10.01 and HP Fortran 90 1.0

```
cc -o main -Aa main.c  $LINK_F90_STATIC  -lm  -lcl –W1,-L,/opt/fortran90/lib –lF90 \
–lisamstub –lU77
```

- Red Hat Linux 6.1 with GNU C egcs-2.91.66 and Portland Group pgf90 3.1-2

```
gcc –o main main.c $LINK_F90 –lpgf90 –lpgf90_rpm1 –lpgf902 –lpgf90rtl –lpgftnrtl  \
  -lm –lpgc
```

- SGI 64 bit IRIX 6.5 with SGI Mipspro C and Fortran 90 version 7.2

```
cc -o main $F90FLAGS main.c $LINK_F90 -lm -lfortran -lftn
```

where main.c is a user written main C program

LINK_F90 and F90FLAGS are environment variables set by the cttsetup.csh shell script, which is supplied with IMSL Fortran 90 MP Library version 4.01.

2.4 Example

```
/*
* This example solves a linear system of 100 by 100 using
* the IMSL routine dlslrg_.
*/
 #include <stdio.h>
 #include <math.h>
```

```
 #include <string.h>

#define MAXN 100


main ()
{

extern void dlslrg_(), dmurrv_(),  rnset_(), rnun_();

 int            i, nerr;
 long int       n=MAXN, nn=MAXN*MAXN, lda=MAXN;
 long int       ipath=2, seed=123457;
 double         ermax=1.e-12;
 double         a[MAXN][MAXN], b[MAXN], x[MAXN], ans[MAXN];


/* generate matrix a with uniform random numbers */
rnset_ (&seed);
drnun_ (&nn, a);

/* generate a pre-determined answer, ans */
for (i=0; i<n; i++) {
   ans[i] = (double) (i%5) + 1.;
   }

/* calculate b <- a*ans */
dmurrv_ (&n, &n, a, &lda, &n, ans, &ipath, &n, b);

/* calculate x in a*x = b using the IMSL routine dlslrg_ */
dlslrg_ (&n, a, &lda, b, &ipath, x);

/* compare the pre-determined answer (ans) and the result (x) */
nerr = 0;
for (i = 0; i < n; i++)
   if (fabs(ans[i] - x[i]) > ermax) {
       nerr++;
       printf ("ans[%2d]= %g and x[%2d] = %g are different.\n",
            i, ans[i], i, x[i]);
    }


/* print the result */
if (nerr)
   printf ("IMSL: %2 differences occurred.\n", nerr);
else
   printf ("IMSL: dlslrg_ is correct with ermax = %g.\n", ermax);
}
```

The output from Solaris is:

IMSL: dlslrg_ is correct with ermax = le-12.

3. Cray

The Cray UNICOS environment has been tested using the CRAY T3E computer system running the Cray operating system UNICOSMK Revision 2.0.0.42 with Revision 6.0.1.3 of the Cray C compiler, Revision 3.0.1.2 of the Cray CF90 Fortran compiler, and version 3.0 of the IMSL Fortran 90 MP Library. Since double in Cray C is 64-bits long by default, DOUBLE PRECISION in Cray Fortran (128 bits) has no matching data type in Cray C. Therefore, the single precision version of the IMSL Fortran Libraries is demonstrated in conjunction with double precision of Cray C.

As in other UNIX systems, this environment is case sensitive. The IMSL Fortran Libraries are written in uppercase for Fortran and compiled as they are. Therefore, the IMSL symbols and identifiers, such as names of IMSL subroutines and functions, and common block identifiers, should be in uppercase. Unlike the other UNIX systems however, the object file format does not append an underscore (_) at the end of those identifiers (e.g., TRNRR should be typed as it is).

3.1 Passing a Character String Argument

The Cray C character pointer is incompatible with the Cray Fortran character type. Therefore, character values should not be passed between Cray C and Cray Fortran routines (Cray Research 1986). Instead, use the macro _cptofcp, which is defined in fortran.h, to describe the structure of a C character string pointer, and give its length, so that it can be understood by Fortran. The following example transposes the two-dimensional matrix, a, and prints the result with a title.

```
/*
   This example transposes a two dimensional matrix
   and prints it out.
*/
#include <fortran.h>
#include <string. h>

double a[2] [3] = {11., 12., 13.,
                   21., 22., 23.};
char title[] = “2x3 matrix”;
main ()
{

long int nra=3, nca=2, lda=3;          /* matrix a is nra by nca */
long int      nrb=2, ncb=3, ldb=2;    /* matrix b is nrb by ncb */
long int      itring=0, lstr=10;         /* triangle option */
double       b[3][2];
_fcd          fcd;       /* Fortran character definition */
```

```
/* _cptofcd transforms a C character pointer to a Fortran
* character definition */

 fcd = _cptofcd (title, strlen(title));

/* transpose matrix a and save the result in matrix b */
TRNRR (&nra, &nca, a, &lda, &nrb, &ncb, b, &ldb);

/* write the matrix b with a title */
WRRRN (fcd, &nrb, &ncb, b, &ldb,&itring);
}
```

The output from CRAY is:

2x3 matrix

	1	2	3
1	11	12	13
2	21	22	23

3.2 Passing a Function as an Argument

In Cray C, a function can be passed to a Fortran subroutine as an argument, as in other UNIX systems. The following example shows how to pass a function to IMSL routine QDNG. Note that myfunc references x as a pointer because Fortran will invoke myfunc and pass the address of x. Also note that single precision is used with the IMSL Libraries, and double precision is used with C.

```
#include <stdio.h>
#include <math.h>
#include <fortran.h>

/*
* myfunc calculates and returns *x * exp(*x). x is a pointer
* since Fortran calls myfunc and passes x as an address.
*/
double myfunc (x)
double *x;
{
   double exp ();
   return (*x * exp (*x));
}
```

```
main ()
{

double exp(), fabs(), myfunc();
double lower=0.e0, upper=2.e0, errabs=0.e0, errrel=l.e-7;
double result, errest, exact;

/* IMSL QDNG integrates a smooth function using
* a nonadaptive rule. */
QDNG (myfunc, &lower, &upper, &errabs, &errrel, &result, &errest);

  /* The exact answer is already known and it is 1 + exp(2). */
  exact = l.0e0 + exp (2.e0);

/* print the result */
printf ("\nIntegral of y = x*exp(x) at [0,2].\n\n");
printf ("Exact answer = %8.5f\n", exact);
printf ("IMSL QDNG = %8.5f\n", result);
printf ("Error              = %12.5e\n", fabs (exact - result));
}
```

The output from CRAY is:

```
Integral of y = x*exp(x) at [0,2].

Exact answer = 8.38906
IMSL QDNG = 8.38906
Error          =    1.70530e-13
```

3.3 Compilation and Linking

To compile and link a Cray C program with version 3.0 of the IMSL Fortran 90 MP library, enter:

```
cc -c main.c $LINK_F90
```

where main.c is a user written C program and LINK_F90 is an environment variable set by the iptsetup.csh shell script, which is supplied with IMSL Fortran 90 Libraries version 3.0.

3.4 Example

```
/*
   This example solves a linear system of 100 by 100
   using the IMSL routine LSLRG.
*/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <fortran.h>

#define MAXN 100


main()
{

 int            i, nerr;
 long int       n=MAXN, nn=MAXN*MAXN, lda=MAXN;
 long int       ipath=2, seed=123457;
 double         ermax=l.e-7;
 double         a[MAXN][MAXN], b[MAXN], x[MAXN], ans[MAXN];

/* generate matrix a with uniform random numbers */
 RNSET (&seed);
 RNUN (&nn, a);

/* generate a pre-determined answer, ans */
for (i = 0; i < n; i++) {
     ans[i] = (double) (i%5) + 1.;
}

/* calculate b <- a*ans */
MURRV (&n, &n, a, &lda, &n, ans, &ipath, &n, b);

/* calculate x in a*x = b by IMSL dlslrg */
LSLRG (&n, a, &lda, b, &ipath, x) ;

/* compare the pre-determined answer, ans, and the result, x */
nerr = 0;
for (i = 0; i < n; i++) {
     if (fabs(ans[i] - x[i]) > ermax) {
     nerr++;
     printf ("ans[%2d]= %g and x[%2d] = %g are different.\n",
               i, ans[i], i, x[i]);
      }
}
```

```
/* print the result */
if (nerr)
     printf ("IMSL: %2 differences occurred.\n", nerr);
else
     printf ("IMSL: LSLRG is correct with ermax = %g.\n", ermax);
}
```

The output from CRAY is:

```
IMSL: LSLRG is correct with ermax = 1e-07.
```

4. OpenVMS

The OpenVMS environment has been tested using OpenVMS Version 7.2 with version 6.0 of the Digital C compiler, version 7.2 of the Compaq Fortran compiler, and version 4.01 of the IMSL Fortran 90 Library. Although the C language is inherently case sensitive, VMS allows you to mix the cases in the common language environment (interlanguage environment). For example, the name of the IMSL routine, DTRNRR, can be interchanged with dtrnrr.

4.1 Passing a Character String Argument

VMS Fortran passes character string variables as descriptors (pass by descriptor); therefore, C must pass character strings to Fortran only after inserting them in a descriptor structure. VMS C provides a macro for this purpose called $DESCRIPTOR, which is defined in the #include descrip header file (Digital Equipment Corporation 1987). The following example shows how to use this macro to pass a character string (title) to the IMSL routine, DWRRRN, and print the matrix.

```
/*
  This example transposes a two-dimensional matrix
  and prints it out.
*/
#include <string.h>
#include descrip

double a[2] [3] = {11., 12., 13.,
                   21., 22., 23.};
char title[] = "2x3 matrix";

main ( )

{

  extern void dtrnrr(), dwrrrn();   /* external IMSL subroutines */
  long int nra=3, nca=2, lda=3;   /* matrix a is nra by nca */
  long int nrb=2, ncb=3, ldb=2;   /* matrix b is nrb by ncb */
  long int itring=0;                  /* triangle option */
  double b[3] [2];

  $DESCRIPTOR (name_desc, title);
  dtrnrr (&nra, &nca, a, &lda, &nrb, &ncb, b, &ldb);
  dwrrrn (&name_desc, &nrb, &ncb, b, &ldb, &itring);

}
```

The output from VMS is:

```
     2x3 matrix

          1     2     3
1        11    12    13
2        21    22    23
```

Note that the header file, descrip, is not surrounded with < and > and the structure identifier, name_desc, is declared, not defined by the macro.

4.2 Passing a Function as an Argument

A VMS C function can be passed to a Fortran subroutine as an argument. The following example shows how to pass a function to IMSL routine DQDNG. Note that myfunc references x as a pointer because Fortran will invoke myfunc and pass the address of x.

```
 #include <stdio.h>
 #include <math.h>
 /*
* myfunc calculates and returns *x * exp (*x). x is a pointer
* since Fortran calls myfunc and passes an address of x.
*/

double myfunc (x)
double *x;
{
      double exp ();
       return (*x * exp (*x));
}

main()
{
 extern void dqdng ();
 double exp(), fabs(), myfunc();
 double lower=0.e0, upper=2.e0, errabs=0.e0, errrel=l.e-7;
 double result, errest, exact;

/* IMSL dqdng integrates a smooth function using a
   nonadaptive rule. */
dqdng (myfunc, &lower, &upper, &errabs, &errrel, &result, &errest);

    /* The exact answer is already known and it is 1 + exp(2). */
    exact = l.0e0 + exp (2. e0);

printf ("\nIntegral of y = x*exp(x) at [0,2].\n\n");
printf ("Exact answer = %8.5f\n", exact);
printf ("IMSL dqdng = %8.5f\n", result);
printf ("Error = %l2.5e\n", fabs (exact - result));
```

```
}
```

The output from VMS is:

```
Integral of y = x*exp(x) at [0,2].

Exact answer = 8.38906
IMSL dqdng = 8.38906
Error = 2.22045e-16
```

4.3 Compilation and Linking

The following steps show how to compile a VMS C program and link it with the IEEE IMSL Fortran 90 library using the default options.

```
! Compile the main C program.
$ cc/float=ieee main.c

! Link the library using logicals defined by cttsetup.com

$ link main, link_f90/OPT
```

4.4 Example

```
/* This example solves a linear system of 100 by 100
* using the IMSL routine dlslrg.
*/
#include <stdio.h>
#include <math.h>
#include <string.h>

#define MAXN 100


main ()
{
extern void dlslrg(), drnun(), dmurrv(), rnset();
 int            i, nerr;
 long int       n=MAXN, nn=MAXN*MAXN, lda=MAXN;
 long int       ipath=2, seed=123457;
 double         ermax=1.e-12;
 double         a[MAXN][MAXN], b[MAXN], x[MAXN], ans[MAXN];


/* generate a random matrix, a */
 rnset (&seed);
```

```
 drnun (&nn, a);

/* generate a pre-determined answer, ans */
for (i = 0; i < n; i++) {
       ans[i] = (double) (i%5) + 1.;
 }

/* calculate b <- a*ans */
dmurrv (&n, &n, a, &lda, &n, ans, &ipath, &n, b);

/* calculate x in a*x =b by IMSL dlslrg */
dlslrg (&n, a, &lda, b, &ipath, x);

/* compare the pre-determined answer, ans, and the result, x */
nerr = 0;
for (i = 0; i < n; i++)
      if (fabs(ans[i] - x[i]) > ermax) {
         nerr++;
         printf ("ans[%2d]= %g and x[%2d] = %g are different.\n",
                  i, ans[i], i, x[i]);
}

/* print the result */
if (nerr)
       printf ("IMSL: %2 differences occurred.\n", nerr);
else
       printf ("IMSL: dlslrg is correct with ermax = %g.\n", ermax);
}
```

The output from VMS is:

```
IMSL: dlslrg is correct with ermax = l.000000e-12.
```

5. WINDOWS

The Windows environment has been tested using Windows NT 4.0 and Windows 95. The compilers used are Microsoft Visual C++ 6.0 and Compaq Visual Fortran 6.5. Version 4.01 of the IMSL Fortran 90 MP Library was used.

Microsoft Visual C++ and Compaq Fortran handle stack bookkeeping for a module differently. Microsoft C, for example, pushes the first argument of the argument list to the end (first-in-first-out), which allows the implementation of one of the C features, the variable length argument list. Compaq Fortran, meanwhile, uses the first-in-last-out scheme.

When a function or subroutine is referenced in a module of a language, it is assumed that the stack manipulation is one that fits the language it is written in. Therefore, when a module written in Fortran is referenced in C, it should be declared with the __stdcall keyword, so that the Compaq compiler can handle the stack problem properly. For example:

```
extern void __stdcall IVPRK (long *, long *, void * , float *, float *, float *,  float *, float *);
```

Header files containing these declarations are provided with F90 4.01. If these header files are included in the program, the programmer does not have to explicitly provide the “extern” statement for each subroutine used. For example:

```
#include <maths.h>  /* include this header file for all single precision subroutines used*/
#include <mathd.h> /* include this header file for all double precision subroutines used*/
```

Note also that the F90 library names must be referenced in upper case.

5.1 Passing a Character String Argument

To pass variable-length strings to Compaq Fortran from Microsoft Visual C++, you must pass both the string and the length of the string to the Fortran subroutine as separate arguments.

The following example illustrates how to pass a character string, "2x3 matrix", to the IMSL routine.

```
/*
* This example transposes a two-dimensional matrix
* and prints it out.
*/
#include <mathd.h>
double a[2][3] = {11., 12., 13.,
                  21., 22., 23.};
char title[] = "2x3 matrix";
unsigned int lstr=10;

main ()

{

long int nra=3, nca=2, lda=3;          /* matrix a is nra by nca */
long int nrb=2, ncb=3, ldb=2;          /* matrix b is nrb by ncb */
long int itring=0;     /* triangle option */
double b[3][2];


DTRNRR (&nra, &nca, a, &lda, &nrb, &ncb, b, &ldb);
DWRRRN (title, lstr, &nrb, &ncb, b, &ldb, &itring);

}
```

The output from Visual C++ is:

2x3 matrix

	1	2	3
1	11.00	12.00	13.00
2	21.00	22.00	23.00

5.2 Passing a Function as an Argument

Microsoft Visual C++ and Compaq Fortran use different methods for passing function arguments. Therefore, it is necessary to declare the C function with __stdcall so that it is recognized by Fortran.

The following example uses the IMSL routine, QDNG, which integrates a smooth function supplied by the user (myfunc is a function that the user supplies to be integrated).

```
#include <stdio.h>
# include <math.h>
#include <mathd.h>

/*
* myfunc calculates and returns *x * exp(*x). x is a pointer
* since Fortran calls myfunc and passes an address of x.
*/
double __stdcall myfunc (x)
double *x;
{
double exp ();
return (*x * exp (*x));
}

main ()
{
double exp(), fabs();
double lower=0.e0, upper=2.e0, errabs=0.e0, errrel=1.e-7;
double result, errest, exact;

/* IMSL dqdng integrates a smooth function using a nonadaptive rule. */
DQDNG (myfunc, &lower, &upper, &errabs, &errrel, &result, &errest);

/* The exact answer is already known and it is 1 + exp(2). */
exact = 1.0e0 + exp (2.0e0);

printf ("\nIntegral of y = x*exp(x) at [0,2].\n\n");
printf ("Exact answer = %8.5f\n", exact);
printf ("IMSL dqdng = %8.5f\n", result);
printf ("Error = %12.5e\n", fabs (exact - result));
}
```

The output from Visual C++ is:

```
Integral of y = x*exp(x) at [0,2].
```

Exact answer = 8.38906
IMSL dqdng = 8.38906
Error = 1.77636e-015

5.3 Compilation and Linking

- Compaq Visual Fortran 6.5

This is the DOS command used to compile and link a program with the single precision math library which is bundled with Compaq Visual Fortran 6.5:

```
cl main.c %link_f90% dfor.lib dfconsol.lib
```

Libraries dfor.lib and dfconsol.lib are supplied with Compaq Visual Fortran.

5.4 Example

```
/*
* This example solves a linear system of l00x100 by using IMSL dlslrg.
*/
#include <mathd.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

#define MAXN 100

main ( )
{

 int            i, nerr;
 long int       n=MAXN, nn=MAXN*MAXN, lda=MAXN;
 long int       ipath=2, seed=123457;
                double ermax=1.e-12;
 double         a[MAXN] [MAXN], b[MAXN], x[MAXN], ans[MAXN];

/* generate a random matrix, a */
RNSET (&seed);
DRNUN (&nn, a);

/* generate a pre-determined answer, ans */
for (i=0; i < n; i++) {
     ans[i] = (double) (i%5) + 1.;
}

/* calculate b <- a*ans */
DMURRV (&n, &n, a, &lda, &n, ans, &ipath, &n, b);

/* calculate x in a*x = b by IMSL dlslrg */
DLSLRG (&n, a, &lda, b, &ipath, x);

/* compare the pre-determined answer, ans, and the result, x */
nerr = 0;
for (i = 0; i < n; i++) {
     if (fabs(ans [i] - x [i] ) > ermax) {
        nerr++;
        printf ("ans[%2d]= %g and x[%2d] = %g are different.\n",
             i, ans[i], i, x[i]);
     }
}

/* print the result */
```

```
 if (nerr)
      printf ("IMSL: %2 differences occurred.\n", nerr);
else
      printf ("IMSL: dlslrg is correct with ermax = %g.\n”, ermax);
}
```

The output from Visual C++ is:

```
IMSL: dlslrg is correct with ermax = le-012.
```

6. Other Environments

This section discusses how to call the IMSL Fortran Libraries from C in environments not discussed in the previous sections of this report.

The "Introduction" section of this report can be applied to most computing environments with the exception of the following topics.

- case sensitivity

Although C is case sensitive for grammar, the operating system or the library manager may not be case sensitive. If the external references of the IMSL Fortran Libraries are case sensitive, then try switching case.

- naming conventions

As seen in previous sections, many UNIX systems append a trailing underscore (_) at the end of the routine referenced in order to share the same object file format. Some UNIX systems and other operating systems may or may not have this requirement.

- data storage type

Even in an integer data type, the length of a data type in one language may not be the same as in another. Therefore, short int and long int are preferred to int to ensure the data types match (unless different optimizers are used that require a machine dependent data storage type). When using INTEGER in Fortran, it is usually safer to use long int in C, rather than int.

- character string arguments

Many C compilers handle character strings differently than Fortran compilers. C reads a character string until it reaches the null character, \0, while Fortran generally specifies the length of the character string in advance. In C, a structure is usually used to hold the information of a character string passed by descriptor.

- passing functions and subroutines as arguments

Passing a function or subroutine as an argument is system dependent. The technical documents for your environment should be consulted.

- compilation and linking

The user should compile C and Fortran modules separately before linking. When linking object files and libraries, both C and Fortran runtime libraries must be explicitly specified along with other necessary libraries, since the default libraries may or may not be linked.

For additional information on mixing C and Fortran, it is recommended that you refer to the C and Fortran manuals provided with your operating system for a discussion on mixing languages.

References

Cray Research, Inc. (1986), *Cray C Reference Manual,* Publication SR2024, Revision C, Cray Research, Mendota Heights, Minnesota.

Digital Equipment Corporation (1987), *Digital Guide to VAX C,* Digital Equipment Corporation, Maynard, Massachusetts.

Visual Numerics, Inc. (1994), *IMSL MATH/LIBRARY FORTRAN Subroutines for mathematical applications,* Visual Numerics, Inc., Houston, Texas.

Kernighan, Brian W. and Dennis M. Ritchie (1978), *The C Programming Language,* Prentice Hall, Inc., Englewood Cliffs, New Jersey.

Sun *Microsystems,* Inc. (1986), *FORTRAN Programmer's Guide,* Sun Microsystems, Mountain View, California.