# Chapter 35
# C Interop

*We can't solve problems by using the same kind of thinking we used when we created them.*

Einstein

**Aim**
This chapter looks briefly at C interoperability.

## 35.1 Introduction

C is a widely used programming language and there is a considerable amount of software written in C or with a C calling interface. Fortran 2003 introduced a standardised mechanism for interoperating with C.

There were limitations to this interoperability and ISO TS 29113 significantly extended the scope of the interoperation facilities. The TS was published in 2012.

In this chapter we provide a brief coverage of some of the technical details required for interoperability and then have a look at a couple of examples.

## 35.2 The `iso_c_binding` Module

There is an intrinsic module called `iso_c_binding` that contains named constants, derived types and module procedures to support interoperability.

## 35.3 Named Constants and Derived Types in the Module

In Table 35.1 the entities listed in the second column are named constants of type default integer.

**Table 35.1** `iso_c_binding` module - named constants

| Fortran type | Named constant from the iso_c_binding module (kind type parameter is positive if supported) | C type |
|---|---|---|
| integer | c_int | int |
| | c_short | short int |
| | c_long | long int |
| | c_long_long | long long int |
| | c_signed_char | signed char |
| | | unsigned char |
| | c_size_t | size_t |
| | c_int8_t | int8_t |
| | c_int16_t | int16_t |
| | c_int32_t | int32_t |
| | c_int64_t | int64_t |
| | c_int_least8_t | int_least8_t |
| | c_int_least16_t | int_least16_t |
| | c_int_least32_t | int_least32_t |
| | c_int_least64_t | int_least64_t |
| | c_int_fast8_t | int_fast8_t |
| | c_int_fast16_t | int_fast16_t |
| | c_int_fast32_t | int_fast32_t |
| | c_int_fast64_t | int_fast64_t |
| | c_intmax_t | intmax_t |
| | c_intptr_t | intptr_t |
| real | c_float | float |
| | c_double | double |
| | c_long_double | long double |
| complex | c_float_complex | float complex |
| | c_double_complex | double complex |
| | c_long_double_complex | long double complex |
| logical | c_bool | bool |
| character | c_char | char |

## 35.4 Character Interoperability

Table 35.2 shows the mapping between Fortran and C character types. The semantics of these values are explained in 5.2.1 and 5.2.2 of the C International Standard.

**Table 35.2** C Interop character interoperability

| Name | C definition | c_char = −1 | c_char /= −1 |
|---|---|---|---|
| c_null_char | Null character | char(0) | `'\0'` |
| c_alert | Alert | achar(7) | `'\a'` |
| c_backspace | Backspace | achar(8) | `'\b'` |
| c_form_feed | Form feed | achar(12) | `'\f'` |
| c_new_line | New line | achar(10) | `'\n'` |
| c_carriage_return | Carriage return | achar(13) | `'\r'` |
| c_horizontal_tab | Horizontal tab | achar(9) | `'\t'` |
| c_vertical_tab | Vertical tab | achar(11) | `'\v'` |

## 35.5 Procedures in the Module

There are several procedures in this module. In the descriptions below, procedure names are generic and not specific.

A C procedure argument is often defined in terms of a C address. The `c_loc` and `c_funloc` functions are provided so that Fortran applications can determine the appropriate value to use with C facilities.

The `c_associated` function is provided so that Fortran programs can compare C addresses.

The `c_f_pointer` and `c_f_procpointer` subroutines provide a means of associating a Fortran pointer with the target of a C pointer.

More information can be found in Chap. 18 of the Fortran 2018 standard.

## 35.6 Interoperability of Intrinsic Types

Table 35.1 shows the interoperability between Fortran intrinsic types and C types. A Fortran intrinsic type with particular type parameter values is interoperable with a C type if the type and kind type parameter value are listed in the table on the same row as that C type; if the type is character, interoperability also requires that the length type parameter be omitted or be specified by an initialization expression whose value is one. A combination of Fortran type and type parameters that is interoperable with a C type listed in the table is also interoperable with any unqualified C type that is compatible with the listed C type.

The second column of the table refers to the named constants made accessible by the `iso_c_binding` intrinsic module.

A combination of intrinsic type and type parameters is interoperable if it is interoperable with a C type.

The above mentioned C types are defined in the C International Standard, clauses 6.2.5, 7.17, and 7.18.1.

## 35.7 Other Aspects of Interoperability

There are considerable restrictions on other aspects of interoperability. The following provides some brief details of other areas:

### *35.7.1 Interoperability with C Pointer Types*

`c_ptr` and `c_funptr` shall be derived types with private components. `c_ptr` is interoperable with any C object pointer type. `c_funptr` is interoperable with any C function pointer type.

### *35.7.2 Interoperability of Scalar Variables*

A scalar Fortran variable is interoperable if its type and type parameters are interoperable and it has neither the pointer nor the allocatable attribute.

An interoperable scalar Fortran variable is interoperable with a scalar C entity if their types and type parameters are interoperable.

### *35.7.3 Interoperability of Array Variables*

An array Fortran variable is interoperable if its type and type parameters are interoperable and it is of explicit shape or assumed size.

### *35.7.4 Interoperability of Procedures and Procedure Interfaces*

A Fortran procedure is interoperable if it has the `bind` attribute, that is, if its interface is specified with a proc-language-binding-spec.

### *35.7.5 Interoperation with C Global Variables*

A C variable with external linkage may interoperate with a common block or with a variable declared in the scope of a module. The common block or variable shall be specified to have the `bind` attribute.

### *35.7.6 Binding Labels for Common Blocks and Variables*

The binding label of a variable or common block is a value of type default character that specifies the name by which the variable or common block is known to the companion processor.

### *35.7.7 Interoperation with C Functions*

A procedure that is interoperable may be defined either by means other than Fortran or by means of a Fortran subprogram, but not both.

Another useful source can be found in the December 2009 edition of Fortran Forum. Details are given at the end of the chapter.

## 35.8 Compilers Used in the Examples

Not all Fortran compilers work with all C and C++ compilers and vice versa.

Table 35.3 has some details of the compilers we have used in the examples that follow.

**Table 35.3** Compilers used

| Main program | Subprogram | Operating system |
|---|---|---|
| gfortran | gcc | cygwin, Windows |
| gfortran | gcc | MinGW-W64, Windows |
| gfortran | gcc | openSuSe Linux |
| Intel Fortran | Microsoft Visual C++ | Windows |
| Intel Fortran | Intel C++ | Windows |
| Nag Fortran | Nag integrated gcc | Windows |
| Nag Fortran | gcc | MinGW-W64, Windows |
| Oracle Fortran | Oracle cc | openSuSe Linux |
| gcc | gfortran | cygwin, Windows |
| gcc | gfortran | openSuSe Linux |
| Intel C | Intel Fortran | openSuSe Linux |
| Nag C | Nag Fortran | Windows |
| Oracle C | Oracle Fortran | openSuSe Linux |
| g++ | gfortran | cygwin, Windows |
| g++ | gfortran | openSuSe Linux |
| Intel C++ | Intel Fortran | openSuSe Linux |
| Intel C++ | Intel Fortran | Windows |
| Microsoft Visual C++ | Intel Fortran | Windows |
| Nag C++ | Nag Fortran | Windows |
| Oracle C++ | Oracle Fortran | openSuSe Linux |

## 35.9 Example 1: Kind Type Support

This example uses Table 35.1 as its basis. It prints out the kind types for each of the kind types in the table. If the value of one of the named constants is positive it will be a valid kind value for the intrinsic type, i.e. the corresponding C type is interoperable with the Fortran intrinsic type of that kind. If the value of one of the named constants is negative then there is no interoperable Fortran kind for that C type.

```
program ch3501
  use iso_c_binding
  implicit none

  print *, 'integer support'
  print *, '  c_int = ', c_int
  print *, '  c_short = ', c_short
  print *, '  c_long = ', c_long
  print *, '  c_long_long = ', c_long_long
  print *, '  c_signed_char = ', c_signed_char
  print *, '  c_size_t = ', c_size_t
  print *, '  c_int8_t = ', c_int8_t
  print *, '  c_int16_t = ', c_int16_t
  print *, '  c_int32_t = ', c_int32_t
  print *, '  c_int64_t = ', c_int64_t
  print *, '  c_int_least8_t = ', c_int_least8_t
  print *, '  c_int_least16_t = ', &
    c_int_least16_t
  print *, '  c_int_least32_t = ', &
    c_int_least32_t
  print *, '  c_int_least64_t = ', &
    c_int_least64_t
  print *, '  c_int_fast8_t = ', c_int_fast8_t
  print *, '  c_int_fast16_t = ', c_int_fast16_t
  print *, '  c_int_fast32_t = ', c_int_fast32_t
  print *, '  c_int_fast64_t = ', c_int_fast64_t
  print *, '  c_intmax_t = ', c_intmax_t
  print *, '  c_intptr_t = ', c_intptr_t
  print *, 'real support'
  print *, '  c_float = ', c_float
  print *, '  c_double = ', c_double
  print *, '  c_long_double = ', c_long_double
  print *, 'complex support'
  print *, '  c_float_complex = ', &
    c_float_complex
  print *, '  c_double_complex = ', &
```

```
    c_double_complex
  print *, '  c_long_double_complex = ', &
    c_long_double_complex
  print *, 'logical support'
  print *, '  c_bool = ', c_bool
  print *, 'character support'
  print *, '  c_char = ', c_char
end program ch3501
```

Table 35.4 summarises support for several compilers.
A negative number means not supported.

**Table 35.4** Basic C Interop table

| Compiler vendors | gfortran | Intel | Nag | Sun |
|---|---|---|---|---|
| C interop type | | | | |
| C_INT | 4 | 4 | 4 | 4 |
| C_SHORT | 2 | 2 | 2 | 2 |
| C_LONG | 8 | 4 | 4 | 8 |
| C_LONG_LONG | 8 | 8 | 8 | 8 |
| C_SIGNED_CHAR | 1 | 1 | 1 | 1 |
| C_SIZE_T | 8 | 8 | 8 | 8 |
| C_INT8_T | 1 | 1 | 1 | 1 |
| C_INT16_T | 2 | 2 | 2 | 2 |
| C_INT32_T | 4 | 4 | 4 | 4 |
| C_INT64_T | 8 | 8 | 8 | 8 |
| C_INT_LEAST8_T | 1 | 1 | 1 | 1 |
| C_INT_LEAST16_T | 2 | 2 | 2 | 2 |
| C_INT_LEAST32_T | 4 | 4 | 4 | 4 |
| C_INT_LEAST64_T | 8 | 8 | 8 | 8 |
| C_INT_FAST8_T | 1 | 1 | 1 | 1 |
| C_INT_FAST16_T | 8 | 2 | 2 | 2 |
| C_INT_FAST32_T | 8 | 4 | 4 | 4 |
| C_INT_FAST64_T | 8 | 8 | 8 | 8 |
| C_INTMAX_T | 8 | 8 | 8 | 8 |
| C_INTPTR_T | 8 | 8 | 8 | 8 |
| C_FLOAT | 4 | 4 | 4 | 4 |
| C_DOUBLE | 8 | 8 | 8 | 8 |
| C_LONG_DOUBLE | 10 | 8 | -4 | -3 |
| C_FLOAT_COMPLEX | 4 | 4 | 4 | 4 |
| C_DOUBLE_COMPLEX | 8 | 8 | 8 | 8 |
| C_LONG_DOUBLE_COMPLEX | 10 | 8 | -4 | -3 |
| C_BOOL | 1 | 1 | 1 | 1 |
| C_CHAR | 1 | 1 | 1 | 1 |

## 35.10 Example 2: Fortran Calling a C Function

Here is the Fortran source.

```
program ch3502
  use iso_c_binding
  interface
    real (c_float) function reciprocal(x) &
      bind (c, name='reciprocal')
      use iso_c_binding
      real (c_float), value :: x
    end function reciprocal
  end interface
  real :: x

  x = 10.0
  print *, ' Fortran calling C function'
  print *, x, ' reciprocal = ', reciprocal(x)
end program ch3502
```

Here is the C source.

```
float reciprocal(float x)
{
  return(1.0f/x);
}
```

The first key statement is

```
use iso_c_binding
```

which makes available named constants, derived types and module procedures to support interoperability.

The next part of the program

```
interface
  real (c_float) function reciprocal(x) &
    bind(c,name='reciprocal')
  use iso_c_binding
  real (c_float) , value :: x
  end function reciprocal
end interface
```

provides the compiler with details of the C function that is being called. It is called `reciprocal`, takes an argument of type `real` in Fortran or `float` in C terminology, and returns a value of type `real` in Fortran or `float` in C terminology.

## 35.11 Example 3: C Calling a Fortran Function

Here is the Fortran source.

```
function reciprocal(x) bind (c, name= &
  'reciprocal')
  use iso_c_binding
  implicit none
  real (c_float), intent (in) :: x
  real (c_float) :: reciprocal

  reciprocal = 1.0/x
end function reciprocal
```

Here is the C source.

```
#include <stdio.h>
float reciprocal(float *x);

int main()
{
  float x;
  x=10.0f;
  printf(" C calling a Fortran function\n");
  printf(" (1 / %f ) = %f \n" ,x,reciprocal(&x));
  return(0);
}
```

Let us look at the Fortran code first.

```
function reciprocal(x) bind(c,name=reciprocal)
```

This line tells the compiler that the `reciprocal` function has to have a name and calling convention that is interoperable with C.

```
real (c_float), intent(in) :: x
```

says that the argument `x` is `intent(in)` and is of type `real` in Fortran and type `float` in C.

```
real (c_float) :: reciprocal
```

says that the function will return a value of type `real` in Fortran or `float` in C terminology.

The function prototype

```
float reciprocal(float *x);
```

is required in the C source code to tell the compiler about the `reciprocal` function.

## 35.12 Example 4: C++ Calling a Fortran Function

Here is the Fortran source.

```
function reciprocal(x) bind (c, name= &
  'reciprocal')
  use iso_c_binding
  implicit none
  real (c_float), intent (in) :: x
  real (c_float) :: reciprocal

  reciprocal = 1.0/x
end function reciprocal
```

Here is the C++ source.

```
#include <iostream> using namespace std;
extern "C" { float reciprocal(float *); }
int main()
{
  float x;
  x=10.0f;
  cout << " C++ calling a Fortan function" << endl;
  cout << " x = " << x << " reciprocal = ";
  cout << reciprocal(&x) << endl;
  return(0);
}
```

The Fortran code and explanation is as for the previous example.

The

```
extern "C" { float reciprocal(float *); }
```

code is required in the C++ code to tell the compiler about the Fortran function `reciprocal`.

In C++ we have to tell the compiler that the function has C calling semantics.

## 35.13 Example 5: Passing an Array from Fortran to C

Here is the Fortran source.

```
program ch3505
  use iso_c_binding
  interface
    function summation(x, n) bind (c, &
      name='summation')
      use iso_c_binding
      integer (c_int), value :: n
      real (c_float), dimension (1:n), &
        intent (in) :: x
      real (c_float) :: summation
    end function summation
  end interface
  integer, parameter :: n = 10
  real, dimension (1:n) :: x = 1.0

  print *, ' Fortran calling c function'
  print *, ' 1 d array as parameter'
  print *, summation(x, n)
end program ch3505
```

Here is the C source.

```
float summation(float *x,int n)
{
  int i;
  float t;
    t=0.0f;
    for (i=0;i<n;i++)
    {
      t+=x[i];
    }
    return(t);
}
```

The following code

```
interface
  function summation(x,n) bind(c,name=summation)
    use iso_c_binding
    integer (c_int) , value :: n
    real (c_float), dimension(1:n) , intent(in) :: x
    real (c_float) :: summation
  end function summation
end interface
```

is required to tell the Fortran compiler the details of the C function.

Arrays in C are passed as pointers or by address so we have the following signature

```
float summation(float *x,int n)
```

in the C code.

## 35.14 Example 6: Passing an Array from C to Fortran

Here is the Fortran source.

```
function summation(x, n) bind (c, &
  name='summation')
  use iso_c_binding
  implicit none
  integer (c_int), value :: n
  real (c_float), dimension (1:n), &
    intent (in) :: x
  real (c_float) :: summation
  integer :: i

  summation = sum(x(1:n))
end function summation
```

Here is the C source.

```
#include <stdio.h>
float summation(float *x,int n);

int main()
{
  const int n=10;
```

```
    float x[n];
    int i;
    for (i=0;i<n;i++)
      x[i]=1.0;
    printf(" C calling Fortran\n");
    printf(" 1 d array as parameter\n");
    printf(" Sum is = %f \n " ,summation(x,n));
    return(0);
  }
```

The `bind(c)` attribute is required to tell the Fortran compiler that the function will be called from C.

The other declarations provide details of the parameters passed into the function from the C calling routine.

The following function prototype

```
float summation(float *x,int n);
```

is required to tell the C compiler the details of the Fortran function.

## 35.15 Example 7: Passing an Array from C++ to Fortran

Here is the Fortran source.

```
function summation(x, n) bind (c, &
  name='summation')
  use iso_c_binding
  implicit none
  integer (c_int), value :: n
  real (c_float), dimension (1:n), &
    intent (in) :: x
  real (c_float) :: summation
  integer :: i

  summation = sum(x(1:n))
end function summation
```

Here is the C++ source.

```
#include <iostream>
using namespace std;
extern "C" float summation(float *,int );
int main()
{
  const int n=10;
  float *x;
  int i;
  x = new  float[n];
  for (i=0;i<n;i++)
    x[i]=1.0f;
  cout << " C++ calling Fortran" << endl;
  cout << " 1 d array as parameter" << endl;
  cout << " Sum is " << summation(x,n) << endl;
  return(0);
}
```

The explanation of the Fortran source is the same as for the previous example. The following function prototype

```
float summation(float *x,int n);
```

is required to tell the C++ compiler about the Fortran function.

## 35.16 Example 8: Passing a Rank 2 Array from Fortran to C

Here is the Fortran source.

```
program ch3508
  use iso_c_binding
  interface
    subroutine reciprocal(nr, nc, x, y) bind (c, &
      name='reciprocal')
      use iso_c_binding
      integer (c_int), value :: nr
      integer (c_int), value :: nc
      real (c_float), dimension (nr, nc) :: x
      real (c_float), dimension (nr, nc) :: y
    end subroutine reciprocal
  end interface
  integer, parameter :: nr = 2
  integer, parameter :: nc = 6
  integer :: i
  real, dimension (nr, nc) :: x
  real, dimension (nr, nc) :: y
  real, dimension (nr*nc) :: t = [ (i,i=1,nr*nc) &
    ]
  integer :: r
  integer :: c

  x = reshape(t, (/nr,nc/), order=(/2,1/) )
  print *, ' Fortran calling C'
  print *, ' two d array as parameter'
  print *, ' using C 99 VLA'
  do r = 1, nr
    print 100, x(r, 1:nc)
100 format (10(f5.1))
  end do
  call reciprocal(nr, nc, x, y)
  do r = 1, nr
    print 110, y(r, 1:nc)
110 format (10(f6.3))
  end do
end program ch3508
```

Here is the C source.

```
void reciprocal(int nrow,int ncol,
  float matrix1[nrow][ncol],
  float matrix2[nrow][ncol])
{
  int i;
  int j;
  for (i=0;i<nrow;i++)
    for (j=0;j<ncol;j++)
      matrix2[i][j]=1.0f/matrix1[i][j];
}
```

In this example we are using the variable length array syntax that was introduced in the C 99 standard.

This feature is not supported in all C compilers.

This enables us to use the following syntax in C.

```
void reciprocal(int nrow,int ncol,
float matrix1[nrow][ncol],
float matrix2[nrow][ncol])
```

## 35.17 Example 9: Passing a Rank 2 Array from C to Fortran

Here is the Fortran source.

```
subroutine reciprocal(nr, nc, x, y) bind (c, &
  name='reciprocal')
  use iso_c_binding
  implicit none
  integer (c_int), value :: nr
  integer (c_int), value :: nc
  real (c_float), dimension (1:nr, 1:nc), &
    intent (in) :: x
  real (c_float), dimension (1:nr, 1:nc), &
    intent (out) :: y

  y = 1.0/x
end subroutine reciprocal
```

Here is the C source.

```
#include <stdio.h>
void reciprocal(int nr,int nc,
                float x[nr][nc],
                float y[nr][nc]);
int main()
{
  const int nr=2;
  const int nc=5;
  float x[nr][nc];
  float y[nr][nc];
  int r;
  int c;
  int i=1;
  for (r=0;r<nr;r++)
    for (c=0;c<nc;c++)
    {
      x[r][c]=(float)(i);
      i++;
    }
  printf(" C calling Fortran\n");
  printf(" 2 d array as parameter\n");
  printf(" C99 vla\n");
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      printf(" %5.2f " , x[r][c]);
    }
    printf("\n");
  }
  reciprocal(nr,nc,x,y);
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      printf(" 1 / %5.2f = %6.3f \n"
  , x[r][c],y[r][c]);
    }
    printf("\n");
  }
  return(0);
}
```

We use C99 VLAs in this example too.

## 35.18 Example 10: Passing a Rank 2 Array from C++ to Fortran

Here is the Fortran source.

```
subroutine reciprocal(nr, nc, x, y) bind (c, &
  name='reciprocal')
  use iso_c_binding
  implicit none
  integer (c_int), value :: nr
  integer (c_int), value :: nc
  real (c_float), dimension (1:nr, 1:nc), &
    intent (in) :: x
  real (c_float), dimension (1:nr, 1:nc), &
    intent (out) :: y

  y = 1.0/x
end subroutine reciprocal
```

Here is the C++ source.

```
#include <iostream>
using namespace std;
extern "C" void reciprocal(int nr,int nc,
                           float *x,float *y);
int main()
```

```
{
  const int nr=2;
  const int nc=5;
  float x[nr][nc];
  float y[nr][nc];
  int r;
  int c;
  int i=1;
  for (r=0;r<nr;r++)
    for (c=0;c<nc;c++)
    {
      x[r][c]=(float)(i);
      i++;
    }
  cout << " C++ calling Fortran" << endl;
  cout << " 2 d array as parameter\n";
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      cout << " " << x[r][c] << " ";
    }
    cout << endl;
  }
  reciprocal(nr,nc,(float*)x,(float*)y);
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
      cout << " 1 / " << x[r][c] << " = "
                << y[r][c] << endl;
  }
  return(0);
}
```

The key syntax in this example is

```
extern "C" void reciprocal(int nr,int nc,
float *x,float *y);
```

where we have to pass pointers to the two d arrays.

## 35.19 Example 11: Passing a Rank 2 Array from C++ to Fortran and Taking Care of Array Storage

Two dimensional arrays are stored by column in Fortran and by row in C++. In this example we take care of the array element ordering changes between C++ and Fortran. We handle the change in the Fortran subroutine.

Here is the C++ calling program.

```
#include <iostream>
#include <iomanip>
using namespace std;
extern "C" void sums(int nr,int nc,
  int *x,int *rsum, int *csum);
int main()
{
  const int nr=2;
  const int nc=6;
  int x[nr][nc];
  int rsum[nr];
  int csum[nc];
  int r;
  int c;
  int i=1;
  for (r=0;r<nr;r++)
    for (c=0;c<nc;c++)
    {
      x[r][c]=i;
      i++;
    }
  for (r=0;r<nr;r++)
    rsum[r]=0;
  for (c=0;c<nc;c++)
    csum[c]=0;
  cout << " C++ calling Fortran" << endl;
  cout << " 2 d array as parameter\n";
  cout << " Original 2 d array" << endl;
  cout << endl;
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      cout << setw(3) << x[r][c] << " ";
    }
    cout << endl;
  }
  cout << endl;
  sums(nr,nc,(int*)x,rsum,csum);
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      cout << setw(3) << x[r][c] << " ";
    }
    cout << " = " << rsum[r] << endl;
  }
  cout << endl;
  for (c=0;c<nc;c++)
    cout << setw(3) << csum[c] << " " ;
  cout << endl;
  return(0);
}
```

Here is the Fortran subroutine.

```
subroutine sums(nr, nc, x, rsum, csum) bind (c, &
  name='sums')
! g++ needs -lgfortran to link
  use iso_c_binding
  implicit none
  integer (c_int), value :: nr
  integer (c_int), value :: nc
  integer (c_int), dimension (1:nr, 1:nc), &
    intent (in) :: x
  integer (c_int), dimension (1:nr), &
    intent (out) :: rsum
  integer (c_int), dimension (1:nc), &
    intent (out) :: csum
  integer (c_int), dimension (1:nc, 1:nr) :: t

  t = reshape(x, (/nc,nr/) )
  rsum = sum(t, dim=1)
  csum = sum(t, dim=2)
end subroutine sums
```

The key syntax in the C++ code is shown below.

```
extern "C" void sums(int nr,int nc,
int *x,int *rsum, int *csum);
```

where all arrays are passed by address.

The key statements in the Fortran are

```
t=reshape(x,(/nc,nr/))
```

where we use the `reshape` intrinsic to transform from row storage to column storage.

The `reshape` intrinsic and the following statements

```
rsum=sum(t,dim=1)
csum=sum(t,dim=2)
```

show the power and expressiveness of array handling in Fortran compared to the C family of languages (C, C++, C# and Java).

Here is some sample output.

```
 C++ calling Fortran
 2 d array as parameter
 Original 2 d array

  1   2   3   4   5   6
  7   8   9  10  11  12

  1   2   3   4   5   6  = 21
  7   8   9  10  11  12  = 57

  8  10  12  14  16  18
```

### 35.19.1 Compiler Switches

We now have to ensure that we include the necessary components of the Fortran run time system.

Here are details of how to make this work with the following compiler combinations.

```
gfortran and g++, openSuSe Linux and Windows

gfortran -c ch3511.f90 -o ch3511_f.o
g++         ch3511.cxx    ch3511_f.o -lgfortran


ifort and icc, openSuSe Linux
ifort -c ch3511.f90 -o ch3511_f.o
icc      ch3511.cxx    ch3511_f.o


nagfor, openSuSe linux

nagfor -c ch3511.f90 -o ch3511_nag.o
nagfor    ch3511.cxx    ch3511_nag.o


sunf90 and sunCC, openSuSe Linux

sunf90 -c ch3511.f90 -o ch3511_f.o
sunCC     ch3512.cxx    ch3511_c.o -xlang=f90
```

## 35.20 Example 12: Passing a Rank 2 Array from C to Fortran and Taking Care of Array Storage

Two dimensional arrays are stored by column in Fortran and by row in C. In this example we take care of the array element ordering changes between C and Fortran. We handle the change in the Fortran subroutine.

Here is the C calling program.

```
#include <stdio.h>
void sums(int nr,int nc,int x[nr][nc],
  int * rsum, int * csum);
int main()
{
  const int nr=2;
  const int nc=6;
  int x[nr][nc];
  int rsum[nr];
  int csum[nc];
  int r;
  int c;
  int i=1;
  for (r=0;r<nr;r++)
    rsum[r]=0;
  for (c=0;c<nc;c++)
    csum[c]=0;
  for (r=0;r<nr;r++)
    for (c=0;c<nc;c++)
    {
      x[r][c]=i;
      i++;
    }
  printf(" C calling Fortran\n");
  printf(" 2 d array as parameter\n");
  printf(" c99 vla\n");
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      printf(" %3d " , x[r][c]);
    }
    printf("\n");
  }
  printf("\n");
  sums(nr,nc,x,rsum,csum);
  for (r=0;r<nr;r++)
  {
    for (c=0;c<nc;c++)
    {
      printf(" %3d " , x[r][c]);
    }
    printf(" %3d ",rsum[r]);
    printf("\n");
  }
  printf("\n");
  for (c=0;c<nc;c++)
    printf(" %3d ",csum[c]);
  printf("\n");
  return(0);
}
```

Here is the Fortran subroutine.

```
subroutine sums(nr, nc, x, rsum, csum) bind (c, &
  name='sums')
! gcc requires -lgfortran
  use iso_c_binding
  implicit none
  integer (c_int), value :: nr
  integer (c_int), value :: nc
  integer (c_int), dimension (1:nr, 1:nc), &
    intent (in) :: x
  integer (c_int), dimension (1:nr), &
    intent (out) :: rsum
  integer (c_int), dimension (1:nc), &
```

```
     intent (out) :: csum
   integer (c_int), dimension (1:nc, 1:nr) :: t

   t = reshape(x, (/nc,nr/) )
   rsum = sum(t, dim=1)
   csum = sum(t, dim=2)
 end subroutine sums
```

Here is some sample output.

```
 C calling Fortran
 2 d array as parameter
 c99 vla
   1    2    3    4    5    6
   7    8    9   10   11   12

   1    2    3    4    5    6   21
   7    8    9   10   11   12   57

   8   10   12   14   16   18
```

### 35.20.1 Compiler Switches

In this example we are calling a Fortran subroutine from C++ and the subroutine calls the `reshape` intrinsic function.

We now have to ensure that we include the necessary components of the Fortran run time system.

Here are details of how to make this work with the following compiler combinations.

```
gfortran and gcc, openSuSe Linux and Windows

gfortran -c ch3512.f90 -o ch3512_f.o
gcc         ch3512.c      ch3512_f.o -lgfortran

ifort and icc, openSuSe Linux

ifort -c ch3512.f90 -o ch3512_f.o
icc      ch3512.c      ch3512_f.o

nagfor, openSuSe linux and Windows

nagfor -c ch3512.f90 -o ch3512_nag.o
nagfor    ch3512.c      ch3512_nag.o

sunf90 and sunc99, openSuSe Linux

sunf90 -c ch3512.f90 -o ch3512_f.o
sunc99 -c ch3512.c   -o ch3512_c.o
sunf90    ch3512_f.o    ch3512_c.o
```

## 35.21 Example 13: Passing a Fortran Character Variable to C

A Fortran character variable normally has a length type parameter. In this example we will pass a Fortran character variable to three C routines.

We use a module to provide functions that help convert from Fortran style character variables to C style character variables.

Here is the C source.

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

void print_string(char * string)
{
  printf("                    %s\n",string);
}

void replace_string(char * string)
{
  strcpy(string,"Hello Hello");
}

void concatenate_string(char * string)
{
  strcat(string," Hello Hello");
}
```

Here is the Fortran source. The font size has been reduced to fit the page width.

```
include 'c_interop_module.f90'

program ch3513

  use iso_c_binding

  use c_interop_module
  implicit none

  interface

    subroutine print_string(x)       bind (c, name='print_string')
      use iso_c_binding
      character (c_char) :: x(*)
    end subroutine print_string

    subroutine replace_string(x)     bind (c, name='replace_string')
      use iso_c_binding
      character (c_char)  :: x(*)
    end subroutine replace_string

    subroutine concatenate_string(x) bind (c, name='concatenate_string')
      use iso_c_binding
      character (c_char)  :: x(*)
    end subroutine concatenate_string

  end interface

  integer , parameter :: line_length=80

  character ( len=line_length )                :: fortran_string
  character ( len=line_length , kind=c_char ) :: c_string

  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' print_string '
  call print_string( c_string )

  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' replace_string '
  call replace_string( c_string )
  fortran_string = c_to_f( c_string )
  print *, ' After              ' , fortran_string

  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' concatenate_string '
  call concatenate_string( c_string )
  fortran_string = c_to_f( c_string )
  print *, ' After              ' , fortran_string

end program ch3513
```

Here is the module that has the functions that help converting from Fortran style string variables to C style string variables.

```
module c_interop_module

  use iso_c_binding
  implicit none

  integer , parameter :: n=80

  contains

    function f_to_c(fortran_string)
      implicit none
      character (len=n,kind=c_char) :: f_to_c
      character (len=n)             :: fortran_string
      integer :: f_length
      f_length = len_trim(fortran_string)
      if (f_length >= n) then
        f_length = 79
      end if
      f_to_c = fortran_string(1:f_length) // c_null_char
    end function f_to_c

    function c_to_f(c_string)
      implicit none
      character (len=n)              :: c_to_f
      character (len=n,kind=c_char)  :: c_string
      integer :: c_length
      integer :: i
      c_length = 1
      c_to_f = ' '
      do i=1,n
        if ( c_string(i:i) == c_null_char ) exit
        c_length = c_length +1
      end do
      c_length = c_length -1
      c_to_f = c_string(1:c_length)
    end function c_to_f

end module c_interop_module
```

Here is the sample output.

```
 print_string
                   Hello
 replace_string
 After             Hello Hello
 concatenate_string
 After             Hello Hello Hello
```

## 35.22 Example 14: Passing a Fortran Character Variable to C++

This is a C++ version of the previous one.

Here is the Fortran source. The font size has been reduced to fit the page width.

```
 include 'c_interop_module.f90'

program ch3514

  use iso_c_binding

  use c_interop_module

  implicit none

  interface

    subroutine print_string(x)       bind (c, name='print_string')
      use iso_c_binding
      character (c_char) :: x(*)
    end subroutine print_string

    subroutine replace_string(x)     bind (c, name='replace_string')
      use iso_c_binding
      character (c_char)  :: x(*)
    end subroutine replace_string

    subroutine concatenate_string(x) bind (c, name='concatenate_string')
      use iso_c_binding
      character (c_char)  :: x(*)
    end subroutine concatenate_string

  end interface

  integer , parameter :: line_length=80

  character ( len=line_length )               ::  fortran_string
  character ( len=line_length , kind=c_char ) :: c_string

  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' print_string '
  call print_string( c_string )

  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' replace_string '
  call replace_string( c_string )
  fortran_string = c_to_f( c_string )
  print *, ' After              ' , fortran_string
  fortran_string = 'Hello'
  c_string       = f_to_c(fortran_string)

  print *, ' concatenate_string '
  call concatenate_string( c_string )
  fortran_string = c_to_f( c_string )
  print *, ' After              ' , fortran_string

end program ch3514
```

Here is the C++ source.

```
#include <cstring>
#include <cstdio>

using namespace std;

extern "C"
{
void print_string(char *);
}

extern "C"
{
void replace_string(char *);
}

extern "C"
{
void concatenate_string(char *);
}

void print_string(char * string)
{
  printf("                         %s\n",string);
}

void replace_string(char * string)
{
  strcpy(string,"Hello Hello");
}

void concatenate_string(char * string)
{
  strcat(string," Hello Hello");
}
```

We use the same module.
Here is the sample output.

```
print_string
                   Hello
replace_string
After              Hello Hello
concatenate_string
After              Hello Hello Hello
```

## 35.23 `c_loc` Examples on Our Web Site

We have examples of using the `c_loc` function on our web site for both 32 bit and 64 bit operating systems.

```
https://www.fortranplus.co.uk/
```

Here is some background technical information on `c_loc` from the Fortran 2008 standard.

### *35.23.1 `c_loc(x)` Description*

Description: Returns the C address of the argument.
Class: Inquiry function.
Argument: `x` shall either

- (1) have interoperable type and type parameters and be
  - (a) a variable that has the `target` attribute and is interoperable,
  - (b) an allocated allocatable variable that has the `target` attribute and is not an array of zero size, or
  - (c) an associated scalar pointer, or
- (2) be a nonpolymorphic scalar, have no length type parameters, and be
  - (a) a nonallocatable, nonpointer variable that has the `target` attribute,
  - (b) an allocated allocatable variable that has the `target` attribute, or
  - (c) an associated pointer.

Result Characteristics: Scalar of type `c_ptr`.
Result Value: The result value will be described using the result name `cptr`.

- (1) If x is a scalar data entity, the result is determined as if c_ptr were a derived type containing a scalar pointer component px of the type and type parameters of x and the pointer assignment

  ```
  cptr%px => x
  ```

  were executed.
- (2) If x is an array data entity, the result is determined as if c_ptr were a derived type containing a scalar pointer component px of the type and type parameters of x and the pointer assignment of cptr%px to the first element of x were executed.

If x is a data entity that is interoperable or has interoperable type and type parameters, the result is the value that the C processor returns as the result of applying the unary & operator (as defined in the C International Standard, 6.5.3.2) to the target of cptr

The result is a value that can be used as an actual cptr argument in a call to c_f_pointer where fptr has attributes that would allow the pointer assignment

```
fptr => x
```

Such a call to c_f_pointer shall have the effect of the pointer assignment

```
fptr => x
```

NOTE 15.6 - Where the actual argument is of noninteroperable type or type parameters, the result of c_loc provides an opaque "handle" for it. In an actual implementation, this handle may be the C address of the argument; however, portable C functions should treat it as a void (generic) C pointer that cannot be dereferenced (6.5.3.2 in the C International Standard).

The key issues are that we must take care with the argument to the function, the return value is of type c_ptr, and that this is an opaque type. Let us now look at some examples using this function.

## Bibliography

Standardized Mixed Language Programming for Fortran and C, Bo Einarsson, Richard J. Hanson and Tim Hopkins. Fortran Forum, Volume 28, Number 3, December 2009.

The C VLA information was taken from the standard.

ISO/IEC 9899:2011, Programming languages C. The official standard.

```
http://www.iso.org/iso/iso_catalogue/
catalogue_tc/catalogue_detail.htm?csnumber=57853
```

Harbison S.P., Steele G.L., A C Reference Manual, Prentice-Hall, 2002.

Kernighan B.W., Ritchie D.M., The C programming Language, Prentice-Hall; first edition 1978; second edition 1988.

Both of the following texts cover C++11.

Josuttis N.M., The C++ Standard Library, second edition, Addison Wesley, 2012.

Stroustrup B., The C++ Programming Language, 4th edition, Addison Wesley, 2013.

ISO/IEC 14882:2011, Programming Languages - C++.

```
http://www.iso.org/iso/home/store/catalogue_ics/
  catalogue_detail_ics.htm?csnumber=50372
```

## 35.24 Problem

**35.1** Compile and run the example programs in this chapter with your compiler and examine the output.

**Acknowledgements** Thanks to Steve Clamage of Oracle and Themos Tsikas of Nag for technical advice with some of C interop examples.