Templates and Generic Programming

In this presentation we look at the template facilities in three commonly used languages:

- C++
- Java
- C#

and how to implement a generic quicksort in each language.

We then look at how we achieve this in Fortran using currently available facilties.

We finish by looking at a proposal to add a template facility to Fortran, which was presented at an earlier standards meeting, and suggest that it may be a candidate for inclusing in a future Fortran standard.

C++

Templates have been available in C++ since the first standard in 1997. Here is the code.

```
template <class Type>
void swap(Type array[],int i, int j)
{
  Type tmp=array[i];
  array[i]=array[j];
  array[j]=tmp;
}

template <class Type>
void quicksort( Type array[], int l, int r)
{
int i=l;
int j=r;
Type v=array[int((l+r)/2)];
for (;;)
{
     while (array[i] < v) i=i+1;
  while (v < array[j]) j=j-1;
```

```
  if (i<=j) { swap(array,i,j); i=i+1 ; j=j-1; }
  if (i>j) goto ended ;
}
ended: ;
if (l<j) quicksort(array,l,j);
if (i<r) quicksort(array,i,r);
}

template <class Type>
void print(Type array[],int size)
{
  cout << " [ " ;
  for (int ix=0;ix<size; ++ix)
      cout << array[ix] << "  ";
  cout << "] \n";
}

#include <iostream>
using namespace std;

int main()
{
double da[]={1.9,8.2,3.7,6.4,5.5,1.8,9.2,3.6,7.4,5.5};
int    ia[]={1,10,2,9,3,8,4,7,6,5};
int size=sizeof(da)/sizeof(double);

  cout << " Quicksort of double array is \n";
  quicksort(da,0,size-1);
  print(da,size);

  size=sizeof(ia)/sizeof(int);
  cout << " Quicksort of integer array is \n";
```

```
  quicksort(ia,0,size-1);
  print(ia,size);
return(0);
}
```

Java

Templates were introduced into Java in the 1.5 JDK in 2004.

Here is the code.

```
class quicksort
{

  static < Type> void swap(Type[] array, int i, int j)
  {
    Type tmp = array[i];
    array[i] = array[j];
    array[j] = tmp;
  }

  static < Type extends Comparable< Type>> void
      quicksort(Type[] array, int l, int r)
  {
    int i = l;
    int j = r;
    Type v = array[(int) ((l + r) / 2)];
    for (;;)
    {
      while (array[i].compareTo(v) < 0)
      {
        i = i + 1;
      }
      while (v.compareTo(array[j]) < 0)
```

```
    {
      j = j - 1;
    }
    if (i <= j)
    {
      swap(array, i, j);
      i = i + 1;
      j = j - 1;
    }
    if (i > j)
    {
      break;
    }
  }
  if (l < j)
  {
    quicksort(array, l, j);
  }
  if (i < r)
  {
    quicksort(array, i, r);
  }
}

public static void main(String[] args)
{
  int i;
  int l;
  Integer[] iarray =
  {
    10, 1, 9, 2, 8, 3, 7, 4, 6, 5
  };
```

```
Double[] darray =
{
  10.0, 1.0, 9.0, 2.0, 8.0, 3.0, 7.0, 4.0, 6.0, 5.0
};
l = iarray.length;
System.out.printf(" Array is of size %3d\n", l);
System.out.printf(" Original array is\n");
for (i = 0; i < l; i++)
{
  System.out.printf(" %5d ", iarray[i]);
}
System.out.printf("\n");
System.out.println(" Sorted array is \n");
quicksort(iarray, 0, l - 1);
for (i = 0; i < l; i++)
{
  System.out.printf(" %5d ", iarray[i]);
}
System.out.printf("\n");
l = darray.length;
System.out.printf(" Array is of size %3d\n", l);
System.out.printf(" Original array is\n");
for (i = 0; i < l; i++)
{
  System.out.printf(" %7.2f ", darray[i]);
}
System.out.printf("\n");
System.out.println(" Sorted array is \n");
quicksort(darray, 0, l - 1);
for (i = 0; i < l; i++)
{
  System.out.printf(" %7.2f ", darray[i]);
```

```
    }
    System.out.printf("\n");
  }
}
```

C#

Templates were introduced into C# and the .Net frameword in release 2 in 2005.

Here is the code.

```
using System;
using System.Collections;

public static class utility
{

  public static void swap< T >( T[] array,int i, int j)
  {
    T tmp=array[i];
    array[i]=array[j];
    array[j]=tmp;
  }

  public static void quicksort < T > ( T[] array, int l, int
r)
    where T : IComparable< T >
  {
    int i=l;
    int j=r;
    T v=array[(int)((l+r)/2)];
    for (;;)
    {
      while ( array[i].CompareTo(v) < 0 ) i=i+1;
```

```
      while ( v.CompareTo(array[j]) < 0 ) j=j-1;
      if (i<=j) { swap(array,i,j); i=i+1 ; j=j-1; }
      if (i>j) goto ended ;
    }
    ended: ;
    if (l<j) quicksort(array,l,j);
    if (i<r) quicksort(array,i,r);
  }

  public static void print(IEnumerable x,string heading)
  {
    Console.WriteLine(" {0} ",heading) ;
    foreach (var i in x)
      Console.Write(" {0} " ,i);
    Console.WriteLine("\n") ;
  }

}

public static class test_quicksort
{
  public static int Main()
  {
    string heading;
    double[] da={1.9,8.2,3.7,6.4,5.5,1.8,9.2,3.6,7.4,5.5};
    int[]    ia={1,10,2,9,3,8,4,7,6,5};
    int size = da.Length;
      heading = "Original double array";
      utility.print(da,heading);
      heading = "After quicksort";
      utility.quicksort(da,0,size-1);
      utility.print(da,heading);
```

```
        size=ia.Length;
        heading = "Original integer array";
        utility.print(ia,heading);
        heading = "After quicksort";
        utility.quicksort(ia,0,size-1);
        utility.print(ia,heading);
        return(0);
    }
}
```

Fortran

Here is the Fortran code to achieve this for the 4 integer data types and 3 real types supported by most Fortran compilers. Several files are used in this example.

- ch2501.f90
- integer_kind_module.f90
- precision_module.f90
- quicksort_include.f90
- sort_data_module.f90

Integer kind module

```
module integer_kind_module
  implicit none
  integer , parameter :: &
    i8 = selected_int_kind(2)
  integer , parameter :: &
    i16 = selected_int_kind(4)
  integer , parameter :: &
    i32 = selected_int_kind(9)
  integer , parameter :: &
    i64 = selected_int_kind(15)
end module integer_kind_module
```

Precision module

```
module precision_module
```

```
  implicit none
  integer, parameter :: &
    sp = selected_real_kind( 6,  37)
  integer, parameter :: &
    dp = selected_real_kind(15, 307)
  integer, parameter :: &
    qp = selected_real_kind(30, 291)
end module precision_module
```

Sort data module

```
module sort_data_module

  use precision_module
  use integer_kind_module

  interface sort_data
    module procedure sort_real_sp
    module procedure sort_real_dp
    module procedure sort_real_qp
    module procedure sort_integer_8
    module procedure sort_integer_16
    module procedure sort_integer_32
    module procedure sort_integer_64
  end interface sort_data

contains

  subroutine sort_real_sp(raw_data, how_many)
    use precision_module
    implicit none
    integer, intent (in) :: how_many
    real (sp), intent (inout), dimension (:) :: raw_data
```

```
  call quicksort(1, how_many)

contains

  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    real (sp) :: v, t
    include 'quicksort_include.f90'
  end subroutine quicksort

end subroutine sort_real_sp

subroutine sort_real_dp(raw_data, how_many)
  use precision_module
  implicit none
  integer, intent (in) :: how_many
  real (dp), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)
contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    real (dp) :: v, t
    include 'quicksort_include.f90'
  end subroutine quicksort
end subroutine sort_real_dp
```

```
subroutine sort_real_qp(raw_data, how_many)
  use precision_module
  implicit none
  integer, intent (in) :: how_many
  real (qp), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)
contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    real (qp) :: v, t
    include 'quicksort_include.f90'
  end subroutine quicksort
end subroutine sort_real_qp

subroutine sort_integer_8(raw_data, how_many)
  use integer_kind_module
  implicit none
  integer, intent (in) :: how_many
  integer (i8), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)
contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    integer (i8) :: v, t
    include 'quicksort_include.f90'
  end subroutine quicksort
```

```
end subroutine sort_integer_8

subroutine sort_integer_16(raw_data, how_many)
  use integer_kind_module
  implicit none
  integer, intent (in) :: how_many
  integer (i16), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)
contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    integer (i16) :: v, t
    include 'quicksort_include.f90'
  end subroutine quicksort
end subroutine sort_integer_16

subroutine sort_integer_32(raw_data, how_many)
  use integer_kind_module
  implicit none
  integer, intent (in) :: how_many
  integer (i32), intent (inout), dimension (:) :: raw_data

  call quicksort(1, how_many)
contains
  recursive subroutine quicksort(l, r)
    implicit none
    integer, intent (in) :: l, r
    integer :: i, j
    integer (i32) :: v, t
```

```
      include 'quicksort_include.f90'
    end subroutine quicksort
  end subroutine sort_integer_32

  subroutine sort_integer_64(raw_data, how_many)
    use integer_kind_module
    implicit none
    integer, intent (in) :: how_many
    integer (i64), intent (inout), dimension (:) :: raw_data

    call quicksort(1, how_many)
  contains
    recursive subroutine quicksort(l, r)
      implicit none
      integer, intent (in) :: l, r
      integer :: i, j
      integer (i64) :: v, t
      include 'quicksort_include.f90'
    end subroutine quicksort

  end subroutine sort_integer_64

end module sort_data_module
```

Quicksort include file

```
      i = l
      j = r
      v = raw_data(int((l+r)/2))
      do
        do while (raw_data(i)<v)
          i = i + 1
```

```
        end do
        do while (v<raw_data(j))
          j = j - 1
        end do
        if (i<=j) then
          t = raw_data(i)
          raw_data(i) = raw_data(j)
          raw_data(j) = t
          i = i + 1
          j = j - 1
        end if
        if (i>j) exit
      end do
      if (l<j) then
        call quicksort(l, j)
      end if
      if (i<r) then
        call quicksort(i, r)
      end if
```

The main program - ch20501

```
include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'sort_data_module.f90'

program ch2501

  use precision_module
  use integer_kind_module
  use sort_data_module
```

```
  implicit none
  integer, parameter :: n = 1000000
  real     (sp), allocatable, dimension (:) :: x
  integer (i32), allocatable, dimension (:) :: y

  integer :: allocate_status

  allocate_status = 0

  print *, ' Program starts'
  allocate (x(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, ' Allocate failed.'
    print *, ' Program terminates'
    stop 10
  end if
  print *, ' Real allocate complete'
  call random_number(x)
  print *, ' Real array initialised'
  call sort_data(x, n)
  print *, ' Real sort ended'
  print *, ' First 10 reals'
  write (unit=*, fmt=100) x(1:10)
100 format (5(2x,e12.6))
  allocate (y(1:n), stat=allocate_status)
  if (allocate_status/=0) then
    print *, ' Allocate failed.'
    print *, ' Program terminates'
    stop 10
  end if
  y = int(x*1000000)
  deallocate(x)
```

```
  print *, ' Integer array initialised'
  call sort_data(y, n)
  print *, ' Sort ended'
  print *, ' First 10 integers'
  write (unit=*, fmt=110) y(1:10)
110 format (5(2x,i10))
  deallocate (y)
  print *, ' Deallocate'
  print *, ' Program terminates'

end program ch2501
```

Summary

The key point point is that with C++, Java and C# the template is used by the compiler to generate the code for each type of array we call the quicksort routine with.

In Fortran we, the programmer, have to write or duplicate the code for each type of array we are interested in sorting.

The code examples used in the above can be found on the Fortranplus website.

Visit

```
http://www.fortranplus.co.uk/
fortranplus-books/3rd-edition-book-examples/
```

The generic examples are in chapter 25.

Fortran template standard proposal

This can be found at

```
http://j3-fortran.org/doc/meeting/172/05-181r1.txt
```