

**Fourth edition**  
**new examples**  
**and updated**  
**material**

Authors: Ian D Chivers & Jane Sleightholme

Date: Tuesday, 16th August, 2022

## Overview

This document has coverage of

- updates of some of the examples in the fourth edition;
- new examples;
- additional material aimed at people attending our courses, but also of use more generally;

The fourth edition is available at

<https://www.fortranplus.co.uk/>

We will be providing the new and changed material in these notes, which will be available from our web site.

There is also additional course material covering

- Compilers used with flags
  - NAG
  - Intel
  - gfortran
  - Cray
  - Nvidia
- Coverage of the Intel and Nvidia toolkit options
  - The Intel Base toolkit option
  - The Intel HPC toolkit option
  - Nvidia and the HPC toolkit
  - Nvidia and the Cuda toolkit
- Development environments
  - NAG Fortran Builder
  - Microsoft Visual Studio
  - Microsoft Visual Code

All files are available on our web site. Here is a link

<https://www.rhymneyconsulting.co.uk/fortran/>

The 4\_edition\_update.tar and 4th\_edition\_update.zip files contains all of the original examples, new examples, and a copy of these notes.

## Table of Contents

<b>1 Fourth edition changes</b> .....	<b>6</b>
1.1 Introduction .....	6
1.2 Example list .....	8
<b>2 Arithmetic</b> .....	<b>17</b>
2.1 Example 18: Update of example ch0402 .....	17
2.2 Computer hardware and real and integer arithmetic in the 1970's and 1980's 18	
2.2.1 CDC .....	18
2.2.2 Cray .....	18
2.2.3 ICL - 1900 series .....	18
2.2.4 IBM .....	18
2.2.5 DEC VAX .....	19
2.3 Using the kind query functions .....	19
2.4 Problems .....	20
<b>3 Introduction to derived types</b> .....	<b>21</b>
3.1 Example 3: Constructor usage .....	21
<b>4 Introduction to pointers</b> .....	<b>22</b>
4.1 Additional technical background .....	22
4.2 New examples .....	22
4.3 Example 7: duplicate of example 1 with c_loc .....	22
4.4 Example 8: duplicate of example 2 with c_loc .....	23
4.5 Example 9: duplicate of example 3 with c_loc .....	25
4.6 Example 10: duplicate of example 4 with c_loc .....	26
4.7 Example 11: duplicate of example 5 with c_loc .....	26
4.8 Example 12: duplicate of example 6 with c_loc .....	28
4.9 Problems .....	28
<b>5 Data structuring in Fortran</b> .....	<b>29</b>
5.1 Example 1: Rewrite of example 1 to use allocatable components .....	29
5.2 Example 2: Rewrite of example 2 to use allocatable components .....	30
5.3 Example 3: Linked lists using move_alloc rather than pointers .....	32
<b>6 C Interop</b> .....	<b>35</b>
6.1 Example 1: passing a one d <vector> from C++ to Fortran .....	35
6.2 Example 2: passing a 1 d <array> between C++ and Fortran .....	36
<b>7 IEEE arithmetic</b> .....	<b>37</b>
7.1 Example 1: inexact summation .....	37
<b>8 Handling missing data using nans</b> .....	<b>39</b>
8.1 Example 1: New C# program .....	39
8.2 Example 2: sed script .....	41
8.3 Example 3: Statistical calculations using NaNs .....	41
<b>9 Miscellaneous new examples</b> .....	<b>47</b>
9.1 Example 1: Adding commas to integer output .....	47
9.2 Example 2: Kahan summation with timing .....	49
9.2.1 Sample output .....	49
9.2.2 Test program .....	50
9.2.3 Kahan summation module .....	51
9.3 Example 3: Memory usage using the Windows API .....	52
9.3.1 Sample output .....	53
9.3.2 Fortran source file .....	53
9.3.3 C source file .....	54
9.4 Example 4: Memory usage using the Linux API .....	55
9.4.1 C source code .....	56
9.4.2 Fortran C interop code .....	56
9.4.3 Fortran test program .....	57
9.4.4 gnu compile script .....	58
9.4.5 Sample output .....	58
9.5 Example 5: Kahan summation with memory usage - Windows .....	59
9.5.1 Sample output .....	61

9.6	Example 6: Kahan summation with memory usage: Linux .....	62
9.6.1	Sample output .....	64
9.7	Example 7: Modified memory leak example with memory checking - Windows 65	
9.7.1	Sample output .....	67
9.8	Example 8: Modified memory leak example with memory checking - Linux 68	
9.8.1	Sample output .....	70
9.9	Compilation details .....	70
<b>10</b>	<b>Compilers used .....</b>	<b>71</b>
10.1	NAG .....	71
10.1.1	Normal compile .....	71
10.1.2	Debug compile .....	71
10.1.3	Optimised compile - simple .....	72
10.1.4	Optimised compile - openmp .....	72
10.1.5	optimised compile - coarray .....	72
10.1.6	optimised compile - mpi .....	72
10.2	Intel .....	72
10.2.1	Normal compile .....	72
10.2.2	Debug compile .....	72
10.2.3	Optimised compile - simple .....	75
10.2.4	Optimised compile - openmp .....	75
10.2.5	optimised compile - coarray .....	75
10.2.6	optimised compile - mpi .....	75
10.3	gfortran .....	75
10.3.1	Normal compile .....	75
10.3.2	Debug compile .....	75
10.3.3	Optimised compile - simple .....	76
10.3.4	Optimised compile - openmp .....	76
10.3.5	optimised compile - coarray .....	76
10.3.6	optimised compile - mpi .....	76
10.4	Nvidia .....	77
10.4.1	Normal compile .....	77
10.4.2	Debug compile .....	77
10.4.3	Optimised compile - simple .....	77
10.4.4	Optimised compile - openmp .....	77
10.4.5	optimised compile - coarray .....	77
10.4.6	optimised compile - mpi .....	77
10.5	HP - nee Cray .....	77
10.5.1	Normal compile .....	77
10.5.2	Debug compile .....	77
10.5.3	Optimised compile - simple .....	77
10.5.4	Optimised compile - openmp .....	77
10.5.5	optimised compile - coarray .....	77
10.5.6	optimised compile - mpi .....	77
10.6	Windows and Linux compile scripts .....	78
10.6.1	Sample Windows batch file .....	78
10.6.2	Sample Linux shell script .....	87
<b>11</b>	<b>Development environments .....</b>	<b>97</b>
11.1	NAG .....	97
11.2	Intel .....	97
11.3	Microsoft Visual Studio .....	97
11.4	Microsoft Visual Code .....	97
<b>12</b>	<b>Intel and Nvidia toolkits .....</b>	<b>99</b>
12.1	Toolkit overview .....	99
12.2	Intel base toolkit .....	99
12.3	Intel HPC toolkit .....	100
12.4	Nvidia HPC toolkit .....	100
12.5	Nvidia CUDA toolkit .....	100
12.6	Nvidia and GPU programming .....	100

12.6.1	Nvidia Fortran.....	100
12.7	Example 1: device driver test program.....	101
12.8	Example 2: basic Cuda Fortran test program - integer type.....	104
12.9	Example 3: 32 bit real variant .....	108
12.10	Example 4: 64 bit real variant .....	111
12.11	Nvidia Cuda .....	115

‘The first thing we do, let’s kill all the language lawyers.’

Henry VI, part II

# 1 Fourth edition changes

## 1.1 Introduction

Here is the list of chapters with changes:

- General changes
  - Added use of the two `iso_fortran_env` functions `compiler_version()` and `compiler_options()` to some examples.
- New examples
  - Chapter 43 - New C interop example which provides function access to the Windows API for memory usage - `GlobalMemoryStatusEx` function which is in `sysinfoapi.h`
  - Chapter 43 - New C interop example which provides function access to the Linux API for memory usage - provided in the `<sys/sysinfo.h>` header file.
  - Chapter 43 - New module to add commas to integers when printing. The original version only handled 64 bit integers. The current version handles 32 and 64 bit integers, and negative integers.
  - Chapter 43 - New example illustrating Kahan summation, with timing.
  - Chapter 43 - Two modified Kahan summation example illustrating the use of the memory api functions on Windows and Linux.
- 1 - Overview - None
- 2 - Introduction to Problem Solving - None
- 3 - Introduction to programming languages - None
- 4 - Introduction to programming - None
- 5 - Arithmetic
  - Added an example in chapter 5 about the use of undefined variables.
  - Added coverage of the NAG compiler flag `-C=undefined` and the Intel flag `/Qtrapuv`
  - Modified `ch0504p.f90` to calculate in seconds.
- 6 - Arrays 1: Some Fundamentals

- Added a problem in chapter 6 about the use of an undefined value, and a repeat of the use of the NAG and Intel flags.
- 7 - Arrays 2: Further Examples - None
- 8 - Whole Array and Additional Array Features - None
- 9 - Output of Results - None
- 10 - Reading in data - None
- 11 - Summary of I/O concepts - None
- 12 - Functions - None
- 13 - Control Structures and execution control
  - Added an explicit forward reference in chapter 13 to the do concurrent example ch3305.f90 in the openMP chapter.
- 14 - Characters - None
- 15 - Complex - None
- 16 - Logical - None
- 17 - Introduction to Derived Types
  - Added an example in chapter 17 showing default constructor usage.
- 18 - An Introduction to Pointers
  - Additional coverage of the differences between variable and pointer status types
  - Deleted example seven
  - Duplicated versions of the first six examples to use the `c_loc` c interop function to provide details of what is happening behind the scenes.
- 19 - Introduction to Subroutines - None
- 20 - Subroutines: 2 - None
- 21 - Modules - None
- 22 - Data structuring in Fortran
  - Added three new examples to chapter 22.
  - The first 2 are rewrites of the linked list examples to use allocatable components rather than pointers.
  - The third linked list example removes pointer usage altogether and uses `move_alloc`.
- 23 - An Introduction to Algorithms and the Big O notation - None
- 24 - Operator overloading - None
- 25 - Generic programming - None
- 26 - Mathematical and numerical examples - None

- 27 - Parameterised derived types (PDTs) in Fortran - None
- 28 - Introduction to Object Oriented Programming - None
- 29 - Additional Object Oriented examples - None
- 30 - Introduction to submodules - None
- 31 - Introduction to parallel programming - None
- 32 - MPI - Message Passing Interface - None
- 33 - OpenMP - None
- 34 - Coarray Fortran - None
- 35 - C Interop
  - Added an example of passing a one d vector from C++ to Fortran. Idea came from some of the people from the UK Met Office attending a Fortran course in June 2022. Many thanks.
- 36 - IEEE Arithmetic
  - Additional explanation in chapter 36 (IEEE arithmetic) of example ch3605 showing incorrect summation by the Intel compiler.
  - Additional material in the IEEE chapter to bring it up to date with the latest IEEE standards.
- 37 - Derived type I/O - None
- 38 - Sorting and searching - None
- 39 - Handling missing data in statistics calculations
  - Updated C# example in chapter 39 to get the Met Office station files
  - New example doing missing data calculations using IEEE nans in chapter 39. This involves multiple versions of some of the files.
  - Added additional explanation in chapter 39 to cover the special processing required for the 3 closed stations
- 40 - Converting from Fortran 77 - None
- 41 - Graphics libraries - simple dislin usage - None
- 42 - Abstract interfaces and procedure pointers - None
- 43 - Miscellaneous additional examples - New chapter

## 1.2 Example list

Here is a complete example list with short description.

Chapter	Example	Number	Example description
4	Example	1	Simple text i/o

Chapter	Example	Number	Example description
4	Example	2	Simple numeric i/o and arithmetic
5	Example	1	Simple arithmetic expressions in Fortran
5	Example	2	Type conversion and assignment
5	Example	3	Integer division and real assignment
5	Example	4	Time taken for light to travel from the Sun to Earth
5	Example	5	Relative and absolute error
5	Example	6	Overflow
5	Example	7	Underflow
5	Example	8	Testing what kind types are available
5	Example	9	Using the numeric inquiry functions with integer types
5	Example	10	Using the numeric inquiry functions with real types
5	Example	11	Literal real constants in a calculation
5	Example	12	Rounding problem
5	Example	13	Binary representation of different integer kind type numbers
5	Example	14	Binary representation of a real number
5	Example	15	Initialisation of physical constants, version 1
5	Example	16	Initialisation of physical constants, version 2
5	Example	17	Initialisation of physical constants, version 3
5	Example	18	Variable status and data objects
6	Example	1	Monthly rainfall
6	Example	2	People's weights and setting the array size with a parameter
7	Example	1	Allocatable arrays
7	Example	2	Two dimensional arrays and a map
7	Example	3	Sensible tabular output
7	Example	4	Average of three sets of values
7	Example	5	Booking arrangements in a theatre or cinema
7	Example	6	Voltage from $-20$ to $+20$ volts
7	Example	7	Longitude from $-180$ to $+180$
7	Example	8	Table of liquid conversion measurements
7	Example	9	Means and standard deviations
8	Example	1	Rank 1 whole arrays in Fortran

Chapter	Example	Number	Example description
8	Example	2	Rank 2 whole arrays in Fortran
8	Example	3	Rank 1 array sections
8	Example	4	Rank 2 array sections
8	Example	5	Rank 1 array initialisation --- explicit values
8	Example	6	Rank 1 array initialisation using an implied do loop
8	Example	7	Rank 1 arrays and the dot_product intrinsic"
8	Example	8	Initialising a rank 2 array
8	Example	9	Rank 1 arrays and a stride of 2
8	Example	10	Rank 1 array and the sum intrinsic function
8	Example	11	Rank 2 arrays and the sum intrinsic function
8	Example	12	Masked array assignment and the where statement"
8	Example	13	Array element ordering
9	Example	1	Twelve times table
9	Example	2	Integer overflow and the I edit descriptor
9	Example	3	Imperial pints and US pints
9	Example	4	Imperial pints and litres
9	Example	5	Narrow field widths and the f edit descriptor
9	Example	6	Overflow and the f edit descriptor
9	Example	7	Simple e edit descriptor example
9	Example	8	Simple g edit descriptor example
9	Example	9	Three ways of generating spaces
9	Example	10	Character output and the a edit descriptor
9	Example	11	Character, integer and real output in a format statement
9	Example	12	Open and close usage
9	Example	13	Timing of writing formatted files
9	Example	14	Timing of writing unformatted files
9	Example	15	Implied do loops and array sections for array output
9	Example	16	Repetition and whole array output
9	Example	17	Choosing the decimal symbol
9	Example	18	Alternative format specification using a string
9	Example	19	Alternative format specification using a character variable
10	Example	1	Reading integer data

Chapter	Example	Number	Example description
10	Example	2	Reading real data
10	Example	3	Reading one column of data from a file
10	Example	4	Skipping lines in a file
10	Example	5	Reading from several files consecutively
10	Example	6	Reading using array sections
10	Example	7	Reading using internal files
10	Example	8	Timing of reading formatted files
10	Example	9	Timing of reading unformatted files
11	Example	1	simple use of the read, write, open, close, unit features
11	Example	2	using iostatus to test for errors
11	Example	3	use of newunit and len_trim
12	Example	1	Simple function usage
12	Example	2	The abs generic function
12	Example	3	Elemental function use
12	Example	4	Simple transformational use
12	Example	5	Intrinsic dot_product use
12	Example	6	Easter
12	Example	7	Simple user defined function
12	Example	8	Recursive factorial evaluation
12	Example	9	Recursive version of gcd
12	Example	10	gcd After removing recursion
12	Example	11	Stirling's approximation
13	Example	1	Quadratic roots
13	Example	2	Date calculation
13	Example	3	Simple calculator
13	Example	4	Counting vowels, consonants, etc.
13	Example	5	Sentinel usage
13	Example	6	The evaluation of e**x
13	Example	7	Wave breaking on an offshore reef
14	Example	1	The * edit descriptor
14	Example	2	The a edit descriptor
14	Example	3	Stripping blanks from a string

Chapter	Example	Number	Example description
14	Example	4	The index character function
14	Example	5	Using len and len_trim
14	Example	6	Finding out about the character set available
14	Example	7	Using the scan function
15	Example	1	Use of cmplx, aimag and conjg
15	Example	2	Polar coordinate example
17	Example	1	Dates
17	Example	2	Variant of example 1 using modules
17	Example	3	Variant of example 2 using default constructor calls
17	Example	4	Address lists
17	Example	5	Nested user defined types
18	Example	1	Illustrating some basic pointer concepts
18	Example	2	The associated intrinsic function
18	Example	3	Referencing pointer variables before allocation or pointer assignment
18	Example	4	Pointer allocation and assignment
18	Example	5	Simple memory leak
18	Example	6	More memory leaks
18	Example	7	Duplicate of 1 with c_loc
18	Example	8	Duplicate of 2 with c_loc
18	Example	9	Duplicate of 3 with c_loc
18	Example	10	Duplicate of 4 with c_loc
18	Example	11	Duplicate of 5 with c_loc
18	Example	12	Duplicate of 6 with c_loc
19	Example	1	Roots of a quadratic equation
20	Example	1	Assumed shape parameter passing
20	Example	2	Character arguments and assumed-length dummy arguments
20	Example	3	Rank 2 and higher arrays as parameters
20	Example	4	Automatic arrays and median calculation
20	Example	5	Recursive subroutines — Quicksort
20	Example	6	Allocatable dummy arrays
20	Example	7	Elemental subroutines

Chapter	Example	Number	Example description
21	Example	1	Modules for precision specification and constant definition
21	Example	2	Modules for globally sharing data
21	Example	3	Person data type
21	Example	4	A module for simple timing of a program
21	Example	5	modules and include statements
22	Example	1	Singly linked list: reading an unknown amount of text
22	Example	2	Reading in an arbitrary number of reals using a linked list and copying to an array
22	Example	3	Ragged arrays
22	Example	4	Ragged arrays and variable sized data sets
22	Example	5	Perfectly balanced tree
22	Example	6	Date class
22	Example	7	Date data type with USA and ISO support
22	Example	8	Singly linked list using allocatable components
22	Example	9	Variation of example ch2202 using allocatable components
22	Example	10	variation of example ch2208 without using pointers
23	Example	1	Order calculations
24	Example	1	Overloading the addition (+) operator
25	Example	1	Sorting reals and integers
25	Example	2	Generic statistics module
26	Example	1	Using linked lists for sparse matrix problems
26	Example	2	Solving first order ODE's using RKM
26	Example	3	A subroutine to extract the diagonal elements of a matrix
26	Example	4	The solution of linear equations using Gaussian Elimination
26	Example	5	Allocatable function results
26	Example	6	Elemental e**x function
26	Example	7	absolute and relative errors involved in subtraction using 32 bit reals
26	Example	8	absolute and relative errors involved in subtraction using 64 bit reals
27	Example	1	Linked list parameterised by real kind

Chapter	Example	Number	Example description
27	Example	2	Ragged array parameterised by real kind type
27	Example	3	specifying len in a PDT"
28	Example	1	The basic shape class
28	Example	2	Base class with private data
28	Example	3	Using an interface to use the class name for the structure constructor
28	Example	4	Simple inheritance
28	Example	5	Polymorphism and dynamic binding
29	Example	1	The base date class
29	Example	2	simple inheritance based on an ISO date format
29	Example	3	using the two date formats and showing polymorphism and dynamic binding
30	Example	1	rewrite of the date class using submodules
30	Example	2	rewrite of the first order RKM ODE solver using modules
32	Example	1	Hello World
32	Example	2	Hello World using send and receive
32	Example	3	Serial solution for pi calculation
32	Example	4	Parallel solution for pi calculation
32	Example	5	Work sharing between processes
33	Example	1	Hello world
33	Example	2	Hello world using default variable data scoping
33	Example	3	Hello world with private thread_number variable
33	Example	4	Parallel solution for pi calculation
33	Example	5	comparing the timing of whole array syntax, simple do loops, do concurrent and an OpenMP solution
34	Example	1	Hello world
34	Example	2	Broadcasting data
34	Example	3	Parallel solution for pi calculation
34	Example	4	Work sharing
35	Example	1	Kind type support
35	Example	2	Fortran calling a C function
35	Example	3	C calling a Fortran function

Chapter	Example	Number	Example description
35	Example	4	C++ calling a Fortran function
35	Example	5	Passing an array from Fortran to C
35	Example	6	Passing an array from C to Fortran
35	Example	7	Passing an array from C++ to Fortran
35	Example	8	Passing a rank 2 array from Fortran to C
35	Example	9	Passing a rank 2 array from C to Fortran
35	Example	10	Passing a rank 2 array from C++ to Fortran
35	Example	11	Passing a rank 2 array from C++ to Fortran and taking care of array storage
35	Example	12	Passing a rank 2 array from C to Fortran and taking care of array storage
35	Example	13	Passing a Fortran character variable to C
35	Example	14	Passing a Fortran character variable to C++
36	Example	1	Testing IEEE support
36	Example	2	Testing what flags are supported
36	Example	3	Overflow
36	Example	4	Underflow
36	Example	5	Inexact summation
36	Example	6	NAN and other specials
37	Example	1	basic syntax, no parameters in call
37	Example	2	extended syntax, passing parameters
37	Example	3	basic syntax with timing
37	Example	4	extended syntax with timing
38	Example	1	Generic recursive quicksort example with timing details
38	Example	2	Non recursive Quicksort example with timing details
38	Example	3	Calling the NAG m01caf sorting routine
38	Example	4	sorting an array of a derived type
38	Example	5	Binary search example
39	Example	1	Program to download and save the data files locally
39	Example	2	The sed script and command file that converts the missing values
39	Example	3	The program to do the statistics calculations
39	Example	4	Met Office Utility program

Chapter	Example	Number	Example description
39	Example	5	sed script changes
39	Example	6	Replacement statistics module to handle nan
39	Example	7	Replacement main driving program
40	Example	1	using the plusFORT tool suite from Polyhedron Software
40	Example	2	leaving as Fortran 77
40	Example	3	Simple conversion to Fortran 90
40	Example	4	Simple syntax conversion to modern Fortran
40	Example	5	date case study
40	Example	6	creating 64 bit integer and 128 bit real sorting subroutines from the Netlib sorting routines
41	Example	1	using dislin to plot Amdahl's Law graph 1 - 8 processors or cores
41	Example	2	using dislin to plot Amdahl's Law graph 2 - 64 processors or cores
41	Example	3	using dislin to plot Gustafson's Law graph 1 - 64 processors or cores
41	Example	4	using dislin to plot tsunami events
41	Example	5	using dislin to plot the Met Office data
42	Example	1	Abstract interfaces and procedure pointers
43	Example	1	Adding commas to integer output
43	Example	2	Kahan summation with timing comparison
43	Example	3	Reporting memory usage under Windows
43	Example	4	Reporting memory usage under Linux
43	Example	5	Kahan summation with memory usage - windows
43	Example	6	Kahan summation with memory usage - linux
43	Example	7	Checking memory availability at run time - Windows
43	Example	8	Checking memory availability at run time - Linux

## 2 Arithmetic

### 2.1 Example 18: Update of example ch0402

The Fortran standard has the following definitions

- data object - object, constant, variable, or subobject of a constant
- defined - data object has a valid value
- undefined - data object does not have a valid value

If a program does not provide an initial value (in a type statement) for a variable then its status is said to be undefined.

Consider the following example, which is a variation on example 2 from chapter 4.

```

program ch0518
!
! Updated version of
! ch0402
!
  implicit none
!
! defined    - data object - has a valid value
!
! undefined - data object - does not have a valid value
!

  real      :: n1
  real      :: n2
  real      :: n3
  real      :: average
  real      :: total
  integer   :: n = 3

  print *, ' Variables have not been assigned values '

  print *,n1
  print *,n2
  print *,n3
  print *,average
  print *,total

  n1        = 1
  n2        = 2
  n3        = 3

  total     = n1 + n2 + n3
  average   = total / n

  print *, 'Total of numbers is ', total
  print *, 'Average of the numbers is ', average

```

```
end program
```

Variables `n1`, `n2`, `n3`, `total` and `average` all have undefined status. The use of variables with undefined status is processor dependent. Care must be taken when writing programs to ensure that your variables have a defined status wherever possible. We look at this topic in several subsequent sections.

## 2.2 Computer hardware and real and integer arithmetic in the 1970's and 1980's

We started working in computer services in the University of London in the 1970's. Here are some of the computer systems that were in use in the 70's and 80's.

### 2.2.1 CDC

These systems were available at Imperial College and the University of London Computer Centre.

The information is taken from

- Assembly Language Programming, Ralph Grishman, Algorithmics Press.

and

[https://en.wikipedia.org/wiki/CDC\\_6600](https://en.wikipedia.org/wiki/CDC_6600)

Word size	60 bit
Integer	48 bit, one's complement
Real	60 bit, sign bit, 11 bit exponent, 48 bit mantissa
Double precision	120 bit, 96 bit mantissa

### 2.2.2 Cray

These systems were available at the University of London Computer Centre.

Information is taken from

<https://en.wikipedia.org/wiki/Cray-1>

Word size	64 bit
Integer	
Real	64 bit
Double precision	128 bit

### 2.2.3 ICL - 1900 series

Information is taken from

[https://en.wikipedia.org/wiki/ICT\\_1900\\_series](https://en.wikipedia.org/wiki/ICT_1900_series)

Word size	24
Integer	Single length, 24 bit two's complement Multi-length, 24 bit first word, second and subsequent 23 bit
Real	two words holding a 24 bit mantissa and 9 bit exponent
Double precision	two words holding a 38 bit mantissa and 9 bit exponent
Additional precision	4 words holding a 75 bit mantissa and 9 bit exponent

### 2.2.4 IBM

Information is taken from

[https://en.wikipedia.org/wiki/IBM\\_System/360\\_architecture#Data\\_formats](https://en.wikipedia.org/wiki/IBM_System/360_architecture#Data_formats)

Word size	32
Integer	two's complement binary halfword or fullword values.
Real	32 bit
Double precision	64 bit
Additional precision	The 360/85 and 360/195 also support 128 bit extended precision floating point numbers

For all three formats, bit 0 is a sign and bits 0-7 are a characteristic (exponent, biased by 64). Bits 8-31 (8-63) are a hexadecimal fraction. For extended precision, the low order doubleword has its own sign and characteristic

### 2.2.5 DEC VAX

The information is taken from

<https://nssdc.gsfc.nasa.gov/nssdc/formats/VAXFloatingPoint.htm>

There are 4 floating point formats.

- F\_floating point numbers have the range of approximately plus or minus  $2.9E-39$  to plus or minus  $1.7E+38$ , with a precision of approximately seven decimal digits.
- D\_floating point numbers have the range of approximately plus or minus  $2.9E-39$  to plus or minus  $1.7E+38$ , with a precision of approximately 16 decimal digits.
- G\_floating point numbers have the range of approximately plus or minus  $5.6E-309$  to plus or minus  $0.9E+308$ , with a precision of approximately 15 decimal digits. The exponent has a bias of 1024 (not 128).
- H\_floating point numbers have the range of approximately plus or minus  $8.4E-4933$  to plus or minus  $5.9E+4931$ , with a precision of approximately 33 decimal digits. The exponent has a bias of 16384 (not 1024).

Word size	32 bits
Integer	32 bits
Real	See above
Double precision	See above

### 2.3 Using the kind query functions

The Fortran 90 standard introduced a variety of kind query functions. Here is a module that illustrates the use of the integer kind query functions.

```

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

```

Here is our current equivalent for real types.

```
module precision_module
  implicit none
  !
  ! Updated with the release of NAG 7 which
  ! supports 16 bit reals.
  !
  ! single, double, quad naming used by lapack.
  ! hence sp, dp, qp
  !
  ! we have used hp as half precision
  !
  integer, parameter :: hp = selected_real_kind( 3,  4)
  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
```

## 2.4 Problems

Compile and run this example with the compilers you have access to.

## 3 Introduction to derived types

Initialisation using constructors was missing from earlier editions.

### 3.1 Example 3: Constructor usage

Here is the source code

```
module date_module

    type date

        integer :: day = 1
        integer :: month = 1
        integer :: year = 2000

    end type

end module

program ch1703

    use date_module

    implicit none

    ! Initialisation via derived type definition

    type (date) :: d1

    ! Intialisation via default compiler
    ! provided constructor at
    ! declaration time

    type (date) :: d2=date(11,2,1952)

    print *, d1%day, d1%month, d1%year
    print *, d2%day, d2%month, d2%year

    ! Intialisation via default compiler
    ! provided constructor at
    ! run time

    d1=date(1,3,1956)

    print *, d1%day, d1%month, d1%year

end program
```

## 4 Introduction to pointers

### 4.1 Additional technical background

A pointer is a variable that has the pointer attribute. A pointer is associated with a target by allocation or pointer assignment. A pointer becomes associated as follows:

- The pointer is allocated as the result of the successful execution of an allocate statement referencing the pointer

or

- The pointer is pointer-assigned to a target that is associated or is specified with the target attribute and, if allocatable, is currently allocated.

A pointer may have a pointer association status of

- associated
- disassociated
- undefined

Its association status may change during execution of a program. Unless a pointer is initialised (explicitly or by default), it has an initial association status of undefined. A pointer may be initialised to have an association status of disassociated.

A pointer shall neither be referenced nor defined until it is associated. A pointer is disassociated following execution of a deallocate or nullify statement, following pointer association with a disassociated pointer, or initially through pointer initialisation.

Examples 1 through 6 highlights some of these issues.

### 4.2 New examples

There are six new examples in this chapter that use the `c_loc` function from the C interop facilities of Fortran. You can now see what is happening behind the scenes with examples 1 to 6 with your compiler.

### 4.3 Example 7: duplicate of example 1 with `c_loc`

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1801

  use iso_c_binding
  use integer_kind_module

  implicit none
  type (c_ptr) :: x
  integer (i64) :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target :: c
  integer, target :: d

  c = 1
  a => c
```

```

c = 2
b => c
d = a + b
print *, a, b, c, d

x = c_loc(a)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(b)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(c)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(d)
x_address = transfer(x,x_address)
print *,x_address

```

end program

Here is some sample output from the NAG compiler under Windows.

```

2 2 2 4
4219008
4219008
4219008
4219032

```

#### 4.4 Example 8: duplicate of example 2 with c\_loc

Here is the source code.

```

include 'integer_kind_module.f90'

program ch1802

  use iso_c_binding
  use integer_kind_module

  implicit none
  type (c_ptr) :: x
  integer (i64) :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target :: c
  integer, target :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)

```

```

x_address = transfer(x,x_address)
print *,x_address
x = c_loc(c)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(d)
x_address = transfer(x,x_address)
print *,x_address

print *, associated(a)
print *, associated(b)
c = 1
a => c
c = 2
b => c
d = a + b
print *, a, b, c, d
print *, associated(a)
print *, associated(b)

x = c_loc(a)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(b)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(c)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(d)
x_address = transfer(x,x_address)
print *,x_address
end program

```

Here is some sample output from the NAG compiler under Windows.

```

0
0
4219856
4219860
F
F
2 2 2 4
T
T
4219856
4219856
4219856
4219860

```

#### 4.5 Example 9: duplicate of example 3 with c\_loc

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1803

  use iso_c_binding
  use integer_kind_module

  implicit none
  type (c_ptr) :: x
  integer (i64) :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target :: c
  integer, target :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(b)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(c)
  x_address = transfer(x,x_address)
  print *,x_address
  x = c_loc(d)
  x_address = transfer(x,x_address)
  print *,x_address

  print *, a
  print *, b

  c = 1
  a => c
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d

end program
```

Here is some sample output from the NAG compiler under Windows with the `-C=all` flag.

```
0
0
4219856
4219860
Runtime Error: ch1809.f90, line 28: Reference to disassoci-
ated POINTER A
```

Program terminated by fatal error

#### 4.6 Example 10: duplicate of example 4 with c\_loc

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1804

  use iso_c_binding
  use integer_kind_module

  implicit none
  type (c_ptr) :: x
  integer (i64) :: x_address
  integer, pointer :: a => null(), b => null()
  integer, target :: c
  integer, target :: d

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address

  allocate (a)

  x = c_loc(a)
  x_address = transfer(x,x_address)
  print *,x_address

  a = 1
  c = 2
  b => c
  d = a + b
  print *, a, b, c, d
  deallocate (a)

end program
```

Here is some sample output from the NAG compiler under Windows.

```
0
141819904
1 2 2 3
```

#### 4.7 Example 11: duplicate of example 5 with c\_loc

Here is the source code.

```
include 'integer_kind_module.f90'

program ch1805
```

```

use iso_c_binding
use integer_kind_module

implicit none
type      (c_ptr)   :: x
integer (i64)      :: x_address
integer, pointer :: a => null(), b => null()
integer, target   :: c
integer, target   :: d

allocate (a)
allocate (b)

x = c_loc(a)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(b)
x_address = transfer(x,x_address)
print *,x_address

a = 100
b = 200

print *, a, b
c = 1
a => c
c = 2
b => c
d = a + b
print *, a, b, c, d

x = c_loc(a)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(b)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(c)
x_address = transfer(x,x_address)
print *,x_address
x = c_loc(d)
x_address = transfer(x,x_address)
print *,x_address

end program

```

Here is some sample output from the NAG compiler under Windows.

```
141819904
141819920
100 200
2 2 2 4
4219848
4219848
4219848
4219852
```

#### **4.8 Example 12: duplicate of example 6 with `c_loc`**

Not available at this time.

#### **4.9 Problems**

Compile and run these examples and examine the output with your compiler.

## 5 Data structuring in Fortran

Under certain circumstances it is possible to replace the use of pointers with allocatable components. Garbage collection is now automatic.

### 5.1 Example 1: Rewrite of example 1 to use allocatable components

Here is the source code.

```

module character_list_module
  type character_list
    character (len=1) :: x
    type (character_list), allocatable :: next
  end type
end module

program ch2208

  use character_list_module
  implicit none

  character (len=80) :: fname
  integer :: io_stat_number = 0

  character :: x
  type (character_list) , allocatable , target :: list
  type (character_list) , pointer :: current
=>null()
  type (character_list) , pointer :: root
=>null()

  integer :: I = 0, n
  character (len=:), allocatable :: string

  fname='ch2208.f90'
  open ( unit=1 , file=fname , status='old' )

  do

    read (unit=1 , fmt='(a)' , advance='no' ,
iostat=io_stat_number) x

    if ( io_stat_number /= -1 ) then

      if (associated(current)) then
        allocate ( current%next , source = charac-
ter_list(x) )
        current => current%next
        i=I+1
      else if ( .not.associated(current) ) then

```

```

        ! First data item, need to anchor the root
        allocate ( list           , source = charac-
ter_list(x) )
        current => list
        root    => list
        I = I + 1
    end if

    else

        exit

    endif

end do

print *, I, ' characters read'

n = I
allocate (character(len=n) :: string)
current => root
do i=1,n
    string(i:i) = current%x
    current      => current%next
end do
print *, 'data read was:'
print 100, string
100 format(a)

end program

```

## 5.2 Example 2: Rewrite of example 2 to use allocatable components

Here is the source code.

```

module real_list_module
    type real_list
        real :: x
        type (real_list), allocatable :: next
    end type
end module

program ch2209

    use real_list_module
    implicit none

    character (len=80) :: fname

```

```

integer :: io_stat_number = 0

real
type (real_list) , allocatable , target      :: x
type (real_list) , pointer                   :: current
=>null()
type (real_list) , pointer                   :: root
=>null()

integer :: I = 0, n
real , allocatable , dimension(:) :: y

fname='ch2209.txt'
open ( unit=1 , file=fname , status='old' )

do

    read (unit=1 , fmt=*
iostat=io_stat_number) x

    if ( io_stat_number /= -1 ) then

        if (associated(current)) then
            allocate ( current%next , source = real_list(x) )
            current => current%next
            i=i+1
        else if ( .not.associated(current) ) then
            ! First data item, need to anchor the root
            allocate ( list
            , source = real_list(x) )
            current => list
            root => list
            I = I + 1
        end if

    else

        exit

    endif

end do

print *, I, ' numbers read read'

n = I
allocate (y(n))
current => root
do i=1,n

```

```

        y(I)      = current%x
        current => current%next
    end do
    print *, 'data read was:'
    print *,y
end program

```

### 5.3 Example 3: Linked lists using move\_alloc rather than pointers

Here is the source code.

```

module character_linked_list_module

    type character_linked_list
        character (len=1) :: c
        type (character_linked_list), allocatable :: next
    end type character_linked_list

contains

    subroutine add_item_to_list(list,new_character)

        type (character_linked_list) , allocatable :: list
        character , intent(in)                ::
new_character

        type (character_linked_list) , allocatable :: t

        call move_alloc(list,t)
        allocate(list,source=character_linked_list(new_charac-
ter))
        call move_alloc(t,list%next)

    end subroutine add_item_to_list

    function return_string(list,n)

        type (character_linked_list) , allocatable :: list
        integer , intent(in)                :: n

        character (len=n)                :: re-
turn_string

        type (character_linked_list) , allocatable :: t
        integer                :: I

        do i=1,n

```

```

        return_string(n-i+1:n-i+1) = list%c
        call move_alloc(list%next,t)
        call move_alloc(t,list)

    end do

end function return_string

end module character_linked_list_module

program ch2210

    integer :: z
    character (len=:), allocatable :: string

    print *, ' Calling subroutine to read the data'
    print *, ' '

    call read_data()

    print *, ' '
    print *, ' Returned from subroutine'
    print *, ' Automatic deallocation of data structures'
    print *, ' '

    print *, 'data read was:'
    print 100, string
    100 format(a)

contains

subroutine read_data()

    use character_linked_list_module
    implicit none

    character (len=80) :: fname
    integer :: io_stat_number = 0

    character :: x
    type (character_linked_list) , allocatable :: list

    integer :: I = 0,
n

    fname='ch2210.f90'
    open (unit=1, file=fname, status='old')

```

```
do
    read (unit=1, fmt='(a)', advance='no',
iostat=io_stat_number) x

    if ( io_stat_number /= -1 ) then

        call add_item_to_list(list,x)
        i=i+1

    else

        exit

    endif

end do

print *, I, ' characters read'
n = I
allocate (character(len=n) :: string)

string = return_string(list,n)

end subroutine

end program
```

## 6 C Interop

There are a small number of additional examples. The idea for the <vector> example came from some people from the UK Met Office attending a Fortran course in June 2022. I added the <array> example for completeness.

### 6.1 Example 1: passing a one d <vector> from C++ to Fortran

This is a variation on example 7.

Here is the Fortran source. It is the same as the original example 7.

```
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
```

Here is the new C++ source.

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

Here are the compile commands for Intel, gfortran and NAG:

Intel - Windows

```
echo on
ifort -c ch3507.f90 -o ch3507_intel.obj
icl ch3507_v.cpp ch3507_intel.obj -o ch3507_v_intel.exe
and
```

gfortran WSL openSuSe

```
echo on
gfortran -c ch3507.f90 -o ch3507_gfortran.o
g++ ch3507_v.cpp ch3507_gfortran.o -o ch3507_v_gfortran.out
```

**NAG - openSuSe**

```
echo on
nagfor -c ch3507.f90 -o ch3507_nag.o
nagfor ch3507_v.cpp ch3507_nag.o -o a.out
```

**Here is the output from Intel, gfortran and NAG**

```
C:\document\met_office>ch3507_v_intel.exe
C++ calling Fortran
1 d vector as parameter
Sum is 10
```

**and**

```
ian@dell-5515:/mnt/c/document/met_office>
./ch3507_v_gfortran.out
C++ calling Fortran
1 d vector as parameter
Sum is 10
```

**and**

```
ian@localhost:~/document/fortran/./a.out
C++ calling Fortran
1 d vector as parameter
Sum is 10
```

## 6.2 Example 2: passing a 1 d <array> between C++ and Fortran

The Fortran source is the same as in the previous 2 one d examples.

Here is the C++ source.

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

## 7 IEEE arithmetic

There are a small number of additional examples.

### 7.1 Example 1: inexact summation

This is a variation on example 5.

Here is the new source code.

```

program ch3605

    use ieee_arithmetic
    use iso_fortran_env
    implicit none

    integer :: I
    real :: computed_sum
    real :: real_sum
    integer :: array_size

    logical :: inexact_happened = .false.
    integer :: allocate_status

    character *13, dimension (3) :: heading = (/ '
10,000,000', ' 100,000,000', '1,000,000,000' /)

    real, allocatable, dimension (:) :: x

    print *,compiler_version()

    if (ieee_support_datatype(x)) then
        print *, ' IEEE support for default precision'
    end if

! 10,000,000

    array_size = 10000000

    do I = 1, 3
        write (unit=*, fmt=100) array_size, heading(I)
100 format (' Array size = ', i15, 2x, a13)
        allocate (x(1:array_size), stat=allocate_status)
        if (allocate_status/=0) then
            print *, ' Allocate fails, program ends'
            stop
        end if
        x = 1.0
        computed_sum = sum(x)
        call ieee_get_flag(ieee_inexact, inexact_happened)
        real_sum = array_size*1.0
    
```

```

        write (unit=*, fmt=110) computed_sum
110 format (' Computed sum = ', e12.4)
        write (unit=*, fmt=120) real_sum
120 format (' Real sum      = ', e12.4)
        if (inexact_happened) then
            print *, ' inexact arithmetic'
            print *, ' in the summation'
            print *, ' program terminates'
            stop 20
        end if
        deallocate (x)
        array_size = array_size*10
    end do

end program

```

Here is the output from the NAG compiler.

```

NAG Fortran Compiler Release 7.0(Yurakucho) Build 7017
IEEE support for default precision
Array size =          10000000          10,000,000
Computed sum =    0.1000E+08
Real sum      =    0.1000E+08
Array size =          100000000         100,000,000
Computed sum =    0.1678E+08
Real sum      =    0.1000E+09
inexact arithmetic
in the summation
program terminates

```

Here is the output from the Intel compiler.

```

Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
(R) 64, Version 2021.5.0 Build 20211109_000000
IEEE support for default precision
Array size =          10000000          10,000,000
Computed sum =    0.1000E+08
Real sum      =    0.1000E+08
Array size =          100000000         100,000,000
Computed sum =    0.1000E+09
Real sum      =    0.1000E+09
inexact arithmetic
in the summation
program terminates

```

In the Intel example the computed sum matches the exact sum!

## 8 Handling missing data using nans

There are several changes to the examples in

- Chapter 39, handling missing data in statistical calculations

We now have

- new C# example to get the data files
- new sed script to convert --- to NANs
- rewrite of statistical program to detect NANs rather than flag values

### 8.1 Example 1: New C# program

Here is the new source file.

```
using System;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Text;
class ch3901
{
    static int Main()
    {
        const int n_sites=37;
        string base_address =
            @"https://www.metoffice.gov.uk/pub/"
            +"data/weather/uk/climate/stationdata/";
        string [] station_name =
        {
            "aberporth",      "armagh",
            "ballypatrick",  "bradford",
            "braemar",        "camborne",
            "cambridge",      "cardiff",
            "chivenor",       "cwmystwyth",
            "dunstaffnage",   "durham",
            "eastbourne",     "eskdalemuir",
            "heathrow",       "hurn",
            "lerwick",        "leuchars",
            "lowestoft",      "manston",
            "nairn",           "newtonrigg",
            "oxford",         "paisley",
            "ringway",        "rossonwye",
            "shawbury",       "sheffield",
            "southampton",    "stornoway",
            "suttonbonington", "tiree",
            "valley",         "waddington",
            "whitby",         "wickairport",
            "yeovilton",
        };
    };
}
```

```

string [] web_address = new string[n_sites];
string last_part="data.txt";
string input_string;
int I;
// create the web address of each file
for (i=0;i<n_sites;i++)
{
    web_address[i]=
    base_address+station_name[i]+last_part;
    System.Console.WriteLine(web_address[i]);
}
string[] local_data_file =
{
    "aberporthdata.txt",          "armaghdata.txt",
    "ballypatrickdata.txt",      "bradforddata.txt",
    "braemardata.txt",           "cambornedata.txt",
    "cambridgedata.txt",         "cardiffdata.txt",
    "chivenordata.txt",          "cwmystwythdata.txt",
    "dunstaffnagedata.txt",      "durhamdata.txt",
    "eastbournedata.txt",        "eskdalemuirdata.txt",
    "heathrowdata.txt",          "hurndata.txt",
    "lerwickdata.txt",           "leucharsdata.txt",
    "lowestoftdata.txt",         "manstondata.txt",
    "nairndata.txt",             "newtonriggdata.txt",
    "oxforddata.txt",            "paisleydata.txt",
    "ringwaydata.txt",           "rossonwyedata.txt",
    "shawburydata.txt",          "sheffielddata.txt",
    "southamptondata.txt",       "stornowaydata.txt",
    "suttonboningtondata.txt",   "tireedata.txt",
    "valleydata.txt",            "waddingtondata.txt",
    "whitbydata.txt",            "wickairportdata.txt",
    "yeoviltondata.txt"
};

StreamWriter output_file;
for (i=0;i<n_sites;i++)
{
    ServicePointManager.Expect100Continue = true;
    ServicePointManager.SecurityProtocol =
SecurityProtocolType.Tls
    | SecurityProtocolType.Tls11
    | SecurityProtocolType.Tls12
    | SecurityProtocolType.Ssl3;
    // create the web addresses
    System.Console.WriteLine(" Create the web addresses");
    HttpWebRequest httpwreq = (HttpWebRequest)
    WebRequest.Create(web_address[i]);
    // set up connection

```

```

System.Console.WriteLine(" Set up the connection");
HttpWebResponse httpwresp = (HttpWebResponse)
httpwreq.GetResponse();
// set up input stream
System.Console.WriteLine(" Set up the input stream");
StreamReader input_stream = new
    StreamReader
        (httpwresp.GetResponseStream(),Encoding.ASCII);
// read the whole file
System.Console.WriteLine(" Read the whole file");
input_string=input_stream.ReadToEnd();
// create the output file
System.Console.WriteLine(" Create the output file");
output_file =
File.CreateText("before_"+local_data_file[i]);
output_file.WriteLine(input_string);
input_stream.Close();
output_file.Close();
System.Console.WriteLine(" Close the files");
}
return(0);
}
}

```

## 8.2 Example 2: sed script

Here is the sed script.

```
s/ ---/ nan/g
```

## 8.3 Example 3: Statistical calculations using NANs

Here is the source file.

```

module statistics_module

    use ieee_arithmetic

    implicit none

contains

    subroutine calculate_month_averages(x, n, n_months, sum_x,
average_x, index_by_month, month_names)

        implicit none

        real, dimension (:), intent (in) :: x
        integer, intent (in) :: n
        integer, intent (in) :: n_months

        real, dimension (1:n_months), intent (inout) :: sum_x

```

```

    real, dimension (1:n_months), intent (inout) :: average_x

    integer, dimension (1:n), intent (in) :: index_by_month
    character *9, dimension (1:n_months), intent (in) ::
month_names

    integer, dimension (1:n_months) :: n_missing
    integer, dimension (1:n_months) :: n_actual

    integer :: m

    sum_x = 0.0
    average_x = 0.0
    n_missing = 0
    n_actual = 0

    do m = 1, n
        if ( ieee_is_nan(x(m)) ) then
            n_missing(index_by_month(m)) = n_missing(index_by_month(m)) + 1
        else
            sum_x(index_by_month(m)) = sum_x(index_by_month(m))
+ x(m)
            n_actual(index_by_month(m)) = n_actual(index_by_month(m)) + 1
        end if
    end do

    do m = 1, n_months
        average_x(m) = sum_x(m)/(n_actual(m))
    end do

    print *, ' Summary of actual      missing'
    print *, '                values      values'
    do m = 1, n_months
        print 100, month_names(m), n_actual(m), n_missing(m)
100    format (2x, a9, 2x, i6, 2x, i6)
    end do

    end subroutine

end module

```

Here is the replacement main driving program.

```

include 'ch3906_statistics_module.f90'
include 'ch3903_met_office_station_module.f90'
include 'timing_module.f90'

```

```

program ch3907

  use met_office_station_module
  use statistics_module
  use timing_module

  implicit none

  ! met office data user defined type

  type (station_type), dimension (:), allocatable :: sta-
tion_data

  ! Temporary variables used on the read

  integer :: year
  integer :: month
  real    :: tmax
  real    :: tmin
  integer :: af_days
  real    :: r_af_days
  real    :: rainfall
  real    :: sunshine

  ! Currently we only calculate the
  ! rainfall sum and averages.

  ! real, dimension (1:n_months) :: sum_tmax
  ! real, dimension (1:n_months) :: sum_tmin
  ! real, dimension (1:n_months) :: sum_af_days
  ! real, dimension (1:n_months) :: sum_rainfall
  ! real, dimension (1:n_months) :: sum_sunshine

  ! real, dimension (1:n_months) :: average_tmax
  ! real, dimension (1:n_months) :: average_tmin
  ! real, dimension (1:n_months) ::
! average_af_days
  ! real, dimension (1:n_months) :: average_rainfall
  ! real, dimension (1:n_months) ::
! average_sunshine

  ! Table to hold the monthly rainfall averages
  ! for all stations.

  real, dimension (1:n_months, 1:n_stations) :: rainfall_ta-
ble = 0

```

```

integer :: n_years

integer :: I, j

call start_timing()

call initialise_station_data()

! Process each station

do j = 1, n_stations

    print *, ' '
    print *, ' Processing ', station_data_file_name(j)
    print *, ' '

    open (unit=100, file=station_data_file_name(j), sta-
tus='old')

!   skip the header lines before starting to
!   read the data

    call skip_header_lines(j)

!   the number of observations at each station
!   is stored in the nl array.

    allocate (station_data(1:nl(j)))

!   Read in the data for each station

    do I = 1, nl(j)
        read (unit=100, fmt=100) year, month, tmax, tmin,
r_af_days, rainfall, sunshine
100   format (2x, i5, 2x, i2, 2x, f5.1, 3x, f5.1, 3x, f5.0,
2x, f6.1, 2x, f6.1)
        if ( ieee_is_nan(r_af_days) ) then
            af_days = -99
        else
            af_days = int(r_af_days)
        end if
        station_data(I) = station_type(year, month, tmax,
tmin, af_days, rainfall, sunshine)
    end do

    close (100)

!   Do the monthly average calculations

```

```

!   for each station

        call calculate_month_averages(station_data%rainfall,
nl(j), n_months, sum_rainfall, average_rainfall, sta-
tion_data%month, &
        month_names)

        n_years = station_data(nl(j))%year - station_data(1)%year
+ 1

        print *, ' '
        print *, ' Start date ', station_data(1)%year, ' ', sta-
tion_data(1)%month
        print *, ' '
        print *, ' Rainfall monthly averages over'
        print 110, n_years
110 format (' ~ ', i5, ' years          mm      ins')
        do I = 1, n_months
            print 120, month_names(I), average_rainfall(I), (aver-
age_rainfall(I)/25.4)
120   format (2x, a9, 8x, f7.2, 2x, f5.2)
            end do
            print 130, sum(average_rainfall), (sum(average_rain-
fall)/25.4)
130 format (' Annual rainfall', /, ' average          ',
f8.2, 2x, f5.2)
            print *, ' '
            print *, ' End date ', station_data(nl(j))%year, ' ',
station_data(nl(j))%month

            rainfall_table(1:n_months, j) = average_rainfall

!   Deallocate the arrays

        deallocate (station_data)

!   move on to next station

end do

        print *, ' '
        print 140, site_name(1:n_stations)
140 format (37(2x,a7))
        print *, ' '

        do I = 1, n_months
            print 150, rainfall_table(I, 1:n_stations)/25.4
150 format (37(2x,f7.2))

```

```
end do  
  
call end_timing()  
  
end program
```

## 9 Miscellaneous new examples

One or more files are required for these examples. All files are available on our web site. Here is a link

<https://www.rhymneyconsulting.co.uk/fortran/>

The tar and zip files contain both all of the fourth edition examples plus all new examples.

### 9.1 Example 1: Adding commas to integer output

The following three include files are required:

- include 'integer\_kind\_module.f90'
- include 'ch4301\_display\_with\_commas\_module.f90'
- include 'ch4301\_display\_with\_commas\_test\_program.f90'

ch4301.f90 has the above three include statements.

Here is sample output.

```

Positive
32 bit
                2147483647                2,147,483,647
                8388607                   8,388,607
                32767                      32,767
                127                        127

Positive
64 bit
 9223372036854775807  9,223,372,036,854,775,807
 36028797018963967   36,028,797,018,963,967
 140737488355327    140,737,488,355,327
 549755813887       549,755,813,887
 2147483647         2,147,483,647
 8388607           8,388,607
 32767             32,767
 127              127

Negative
32 bit
                -2147483647               -2,147,483,647
                -8388607                  -8,388,607
                -32767                     -32,767
                -127                       -127

Negative
64 bit
 -9223372036854775807 -9,223,372,036,854,775,807
 -36028797018963967  -36,028,797,018,963,967
 -140737488355327   -140,737,488,355,327
 -549755813887     -549,755,813,887
 -2147483647      -2,147,483,647
 -8388607        -8,388,607
 -32767         -32,767
 -127          -127

```

The original program only supported positive 64 integers, as we were only interested in producing more readable output in the later memory examples. This version has 32 bit integer support and negative integer support.

Here is the test program.

```
program test
```

```

use integer_kind_module
use display_with_commas_module

integer (i32)      :: x=2147483647
integer (i64)      :: y=9223372036854775807_i64
integer (i32)      :: x1=-2147483647
integer (i64)      :: y1=-9223372036854775807_i64

integer :: I

print *, ' Positive'
print *, ' 32 bit'

do I=1,4

    print 10,x,display_with_commas(x)
    10 format(2x,i22,2x,a)
    x=x/256

end do

print *, ' Positive'
print *, ' 64 bit'

do I=1,8

    print 10,y,display_with_commas(y)
    y=y/256

end do

print *, ' Negative'
print *, ' 32 bit'

do I=1,4

    print 10,x1,display_with_commas(x1)
    x1=x1/256

end do
```

```

print *, ' Negative'
print *, ' 64 bit'

do I=1,8

    print 10,y1,display_with_commas(y1)
    y1=y1/256

end do
end program test

```

The files are on our web site.

## 9.2 Example 2: Kahan summation with timing

The following source files are required.

- include 'integer\_kind\_module.f90'
- include 'precision\_module.f90'
- include 'timing\_module.f90'
- include 'kahan\_summation\_module.f90'

ch4302.f90 is a test program that contains the above include files.

### 9.2.1 Sample output

Here is some sample output.

```

2022/ 5/ 5 13:51:32 76
N =      10000000
Allocate                0.0000000000000000000
Initialise              0.162999868392944336
Intrinsic summation    0.0000000000000000000

5000444.2793215252
Kahan summation        0.062999963760375977

5000444.2793215429
N =      100000000
Allocate                0.0000000000000000000
Initialise              1.616000175476074219
Intrinsic summation    0.108999967575073242

49998117.4713004455
Kahan summation        0.524999856948852539

49998117.4712983146
N =      1000000000
Allocate                0.047000169754028320
Initialise              16.238999843597412109
Intrinsic summation    1.116000175476074219

```

```

499995574.2241585851
Kahan summation          5.306999921798706055

499995574.2241371870
2022/ 5/ 5 13:51:57 720
Total time =                25.629000

```

### 9.2.2 Test program

Here is the test program.

```

include 'integer_kind_module.f90'
include 'precision_module.f90'

include 'timing_module.f90'

include 'kahan_summation_module.f90'

program ch4302

  use timing_module
  use precision_module

  use kahan_summation_module

  implicit none

  integer (i64) :: n = 10000000_i64
  integer :: I
  integer :: j = 3

  real (dp), allocatable, dimension (:)&
                                     :: x
  real (dp) &
                                     :: x_sum = 0.0_dp

  call start_timing()

  do i=1,j

    print 10,n
    10 format(' N = ',i12)

    allocate(x(n))

    print 20,time_difference()
    20 format(' Allocate                ',f22.18)

    call random_number(x)

```

```

print 30,time_difference()
30 format(' Initialise           ',f22.18)

x_sum=sum(x)

print 40, time_difference()
40 format(' Intrinsic summation ',f22.18)

print 100, x_sum
100 format(45x,f20.10)

x_sum=kahan_sum(x,n)

print 50, time_difference()
50 format(' Kahan summation     ',f22.18)

print 100, x_sum

deallocate(x)

n=n*10_i64

end do

call end_timing()

end program ch4302

```

### 9.2.3 Kahan summation module

Here is the Kahan Summation module.

```

module kahan_summation_module

  use integer_kind_module
  use precision_module

contains

  function kahan_sum(x,n)

    implicit none

    real      (dp)  , intent(in) , dimension (:) :: x
    integer  (i64)  , intent(in)           :: n

    real (dp)           :: kahan_sum

```

```

real (dp)           :: sum
real (dp)           :: c
real (dp)           :: y
real (dp)           :: t

integer (i64)      :: I

kahan_sum=0.0_dp
sum           =0.0_dp
c             =0.0_dp

do i=1,n

    y = x(I) - c
    t = sum + y
    c = (t - sum) - y
    sum = t

end do

kahan_sum=sum

end function kahan_sum

end module kahan_summation_module

```

### 9.3 Example 3: Memory usage using the Windows API

Microsoft has an api that provides access to information about memory usage on a Windows system. Here is a link to their documentation.

<https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/ns-sysinfoapi-memorystatusex>

Here is the associated struct.

```

typedef struct _MEMORYSTATUSEX {
    DWORD          dwLength;
    DWORD          dwMemoryLoad;
    DWORDLONG     ullTotalPhys;
    DWORDLONG     ullAvailPhys;
    DWORDLONG     ullTotalPageFile;
    DWORDLONG     ullAvailPageFile;
    DWORDLONG     ullTotalVirtual;
    DWORDLONG     ullAvailVirtual;
    DWORDLONG     ullAvailExtendedVirtual;
} MEMORYSTATUSEX, *LPMEMORYSTATUSEX;

```

In this example we provide a Fortran interface to this information, using the C interop facilities available in Fortran.

Here is a link to the example that was a starting point for our programs.

<https://docs.microsoft.com/en-us/windows/win32/api/sysinfoapi/nf-sysinfoapi-globalmemorystatusex>

### 9.3.1 Sample output

Here is some sample output.

```
ch4303_intel.exe
Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
(R) 64, Version 2021.5.0 Build 20211109_000000
Memory usage                26  %
Total physical               17,179,127,808
Available physical          12,598,317,056
Total page file              22,816,272,384
Available page file          17,433,796,608
Total virtual                140,737,488,224,256
Available virtual            140,733,142,515,712
```

Here is some sample output from the NAG compiler on the same system.

```
ch4303_nag.exe
NAG Fortran Compiler Release 7.0(Yurakucho) Build 7017
Memory usage                27  %
Total physical               17,179,127,808
Available physical          12,488,970,240
Total page file              22,816,272,384
Available page file          17,327,640,576
Total virtual                140,737,488,224,256
Available virtual            140,733,001,248,768
```

### 9.3.2 Fortran source file

Here is the Fortran source file.

```
include 'integer_kind_module.f90'
include 'display_with_commas_module.f90'
include 'memory_module_windows.f90'

program ch4303

  use iso_fortran_env
  use memory_module_windows
  use display_with_commas_module

  print *,compiler_version()
  print *,' Memory usage                ',MemoryLoad(), ' %'
  print *,' Total physical              ',&
  display_with_commas(TotalPhysical())
  print *,' Available physical          ',&
  display_with_commas(AvailablePhysical())
  print *,' Total page file              ',&
  display_with_commas(TotalPageFile())
```

```

print *, ' Available page file  ', &
display_with_commas(AvailablePageFile())
print *, ' Total virtual          ', &
display_with_commas(TotalVirtual())
print *, ' Available virtual      ', &
display_with_commas(AvailableVirtual())

```

end program ch4303

### 9.3.3 C source file

You also require the following C source file

ch4303\_memory\_module\_windows.c

which is shown below.

```

#include <windows.h>

int memory_load()
{
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    return(statex.dwMemoryLoad);
}

long long int total_physical()
{
    long long int t;

    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    t=statex.ullTotalPhys;
    return(t);
}

long long int available_physical()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);
    t=statex.ullAvailPhys;
    return(t);
}

```

```
long long int total_page_file()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);
    t=statex.ullTotalPageFile;
    return(t);
}
```

```
long long int available_page_file()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);
    t=statex.ullAvailPageFile;
    return(t);
}
```

```
long long int total_virtual()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    t=statex.ullTotalVirtual;
    return(t);
}
```

```
long long int available_virtual()
{
    long long int t;
    MEMORYSTATUSEX statex;
    statex.dwLength = sizeof (statex);
    GlobalMemoryStatusEx (&statex);

    t=statex.ullAvailVirtual;
    return(t);
}
```

#### 9.4 Example 4: Memory usage using the Linux API

Here is a link to the Linux api.

<https://man7.org/linux/man-pages/man2/sysinfo.2.html>

Here is the struct.

```

struct sysinfo {
    long uptime;
        /* Seconds since boot */
    unsigned long loads[3];
        /* 1, 5, and 15 minute load averages */
    unsigned long totalram;
        /* Total usable main memory size */
    unsigned long freeram;
        /* Available memory size */
    unsigned long sharedram; /* Amount of shared memory */
    unsigned long bufferram; /* Memory used by buffers */
    unsigned long totalswap; /* Total swap space size */
    unsigned long freeswap;
    /* Swap space still available */
    unsigned short procs;
    /* Number of current processes */
    char _f[22]; /* Pads structure to 64 bytes */
};

```

#### 9.4.1 C source code

Here is our C code.

```

#include <stdio.h>
#include <sys/sysinfo.h>

unsigned long total_ram()
{
    struct sysinfo si;
    sysinfo (&si);
    return( si.totalram ) ;
}

unsigned long free_ram()
{
    struct sysinfo si;
    sysinfo (&si);
    return( si.freeram ) ;
}

```

#### 9.4.2 Fortran C interop code

Here is our Fortran C interop code.

```

module memory_module_linux

    use :: iso_c_binding
    use :: integer_kind_module

contains

    function totalram()

```

```

use :: iso_c_binding

interface
  function total_ram() bind (c, name='total_ram')
    use :: integer_kind_module
    integer (i64) :: total_ram
  end function total_ram
end interface

integer (c_long_long) :: totalram

totalram = total_ram()

end function totalram

function freeram()

  use :: iso_c_binding

  interface
    function free_ram() bind (c, name='free_ram')
      use :: integer_kind_module
      integer (i64) :: free_ram
    end function free_ram
  end interface

  integer (c_long_long) :: freeram

  freeram = free_ram()

end function freeram

end module

```

### 9.4.3 Fortran test program

Here is the driving program.

```

include 'integer_kind_module.f90'
include 'ch4304_memory_module_linux.f90'
include 'display_with_commas_module.f90'

program ch4304

  use iso_fortran_env
  use memory_module_linux
  use display_with_commas_module

  print *, compiler_version()

```

```

    print *, ' Total ram  ', totalram(), ' ', display_with_com-
mas(totalram())
    print *, ' Free  ram  ', freeram() , ' ', display_with_com-
mas(freeram())

```

end program ch4304

#### 9.4.4 gnu compile script

Here is the gnu Fortran compile script.

```

gcc -c ch4304_memory_module_linux.c
    -o ch4304_memory_module_linux.o
gfortran ch4304.f90 ch4304_memory_module_linux.o
    -o ch4304.out

```

#### 9.4.5 Sample output

Here is some sample output.

```

./ch4304.out
GCC version 11.2.1 20220420
[revision 691af15031e00227ba6d5935c1d737026cda4129]
  Total ram 13393960960   13,393,960,960
  Free  ram 13218791424   13,218,791,424

```

Here is some more sample output.

```

GCC version 11.2.1 20220127 (Red Hat 11.2.1-9)
  Total ram                4703784960
4,703,784,960
  Free  ram                1982140416
1,982,140,416

```

Here is another Linux run.

```

Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
(R) 64, Version 2021.5.0 Build 20211109_000000
  Total ram                4703784960
4,703,784,960
  Free  ram                1981882368
1,981,882,368

```

Both of these are taken from a Redhat 9 system under Hyper-V.

Here are two more sample outputs.

```

GCC version 11.2.1 20220420 [revision
691af15031e00227ba6d5935c1d737026cda4129]
  Total ram    16776175616          16,776,175,616
  Free  ram    12548960256          12,548,960,256

```

and

```

Intel(R) Fortran Intel(R) 64 Compiler Classic for applica-
tions running on Intel
(R) 64, Version 2021.5.0 Build 20211109_000000
Total ram      16776175616      16,776,175,616
Free  ram      12674674688      12,674,674,688

```

from a native openSuSe linux distribution.

Here is another sample output from the NAG compiler on a native openSuSe install.

```

NAG Fortran Compiler Release 7.1(Hanzomon) Build 7103
Total ram      25201889280      25,201,889,280
Free  ram      21974958080      21,974,958,080

```

The following is from a cygwin installation.

```

./ch4304_cygwin.out
GCC version 11.2.0
Total ram      4194123          4,194,123
Free  ram      3286242          3,286,243

```

The following is from an openSuSe system under WSL.

```

GCC version 11.2.1 20220420 [revision
691af15031e00227ba6d5935c1d737026cda4129]
Total ram      13393960960      13,393,960,960
Free  ram      13217333248      13,217,333,248

```

## 9.5 Example 5: Kahan summation with memory usage - Windows

Here is the Fortran source for ch4305.

```

include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

include 'display_with_commas_module.f90'

include 'kahan_summation_module.f90'
include 'memory_module_windows.f90'

program ch4305

  use timing_module
  use precision_module

  use kahan_summation_module

  use memory_module_windows
  use display_with_commas_module

  implicit none

  integer (i64) :: n = 10000000_i64
  integer :: I

```

```

integer :: j = 4
integer (i64) , parameter :: sixty_four_bit=8_i64
integer (i64) :: nbytes

real (dp), allocatable, dimension (:) :: x
real (dp)                                :: x_sum =
0.0_dp

character (len=20) :: heading = 'call memory usage  '
integer :: lu=6
call start_timing()

do i=1,j

  nbytes=n*sixty_four_bit
  print *, ' Problem size'
  print *, display_with_commas(n)
  print *, display_with_commas(nbytes), ' bytes'

  if ( AvailablePhysical() < nbytes ) then
    print *, ' Insufficient memory '
    print *, ' Memory usage           ', &
      MemoryLoad(), ' %'
    print *, ' Total physical         ', &
      display_with_commas(TotalPhysical())
    print *, ' Available physical     ', &
      display_with_commas(AvailablePhysical())
    print *, ' Program terminates'
    stop 20
  end if

  allocate(x(n))

  print 20,time_difference()
  20 format(' Allocate                ',f22.18)

  call random_number(x)

  print 30,time_difference()
  30 format(' Initialise                ',f22.18)

  x_sum=sum(x)

  print 40, time_difference()
  40 format(' Intrinsic summation ',f22.18)

  print 100, x_sum
  100 format(45x,f20.10)

```

```

x_sum=kahan_sum(x,n)

print 50, time_difference()
50 format(' Kahan summation          ',f22.18)

print 100, x_sum

print *,' Memory usage                ',&
      MemoryLoad(),' %'
print *,' Total physical              ',
display_with_commas(TotalPhysical())
print *,' Available physical          ',&
display_with_commas(AvailablePhysical())

deallocate(x)

n=n*10_i64

end do

call end_timing()

end program ch4305

```

### 9.5.1 Sample output

Here is some sample output.

```

14:43:14 D:\fortran_web_site > ch4305_intel
2022/ 5/ 5 14:43:19 474
  Problem size
                10,000,000
                80,000,000 bytes
Allocate                0.015000104904174805
Initialise              0.167999982833862305
Intrinsic summation    0.008999824523925781

5000444.2793215252
Kahan summation        0.047000169754028320

5000444.2793215429
  Memory usage                20 %
  Total physical              17,179,127,808
  Available physical          13,604,413,440
  Problem size
                100,000,000
                800,000,000 bytes
Allocate                0.014999866485595703
Initialise              1.644000053405761719

```

```

Intrinsic summation      0.108999967575073242

49998117.4713004455
Kahan summation          0.535000085830688477

49998117.4712983146
Memory usage              24 %
Total physical            17,179,127,808
Available physical        12,886,278,144
Problem size
    1,000,000,000
    8,000,000,000 bytes
Allocate                  0.047999858856201172
Initialise                16.639000177383422852
Intrinsic summation      1.218999862670898438

499995574.2241585851
Kahan summation          5.355000019073486328

499995574.2241371870
Memory usage              66 %
Total physical            17,179,127,808
Available physical        5,768,028,160
Problem size
    10,000,000,000
    80,000,000,000 bytes
Insufficient memory
Memory usage              19 %
Total physical            17,179,127,808
Available physical        13,841,661,952
Program terminates

20

```

We can use the memory functions to detect that there is insufficient memory to make the allocation and terminate the program, providing memory usage figures.

## 9.6 Example 6: Kahan summation with memory usage: Linux

Here is the source code.

```

include 'integer_kind_module.f90'
include 'precision_module.f90'
include 'timing_module.f90'

include 'display_with_commas_module.f90'

include 'kahan_summation_module.f90'
include 'memory_module_linux.f90'

program ch4306

```

```

use timing_module
use precision_module

use kahan_summation_module

use memory_module_linux
use display_with_commas_module

implicit none

integer (i64) :: n = 10000000_i64
integer :: I
integer :: j = 4
integer (i64) , parameter :: sixty_four_bit=8_i64
integer (i64) :: nbytes

real (dp), allocatable, dimension (:) :: x
real (dp) :: x_sum =
0.0_dp

character (len=20) :: heading = 'call memory usage '
integer :: lu=6
call start_timing()

do i=1,j

    nbytes=n*sixty_four_bit
    print *, ' Problem size'
    print *, display_with_commas(n)
    print *, display_with_commas(nbytes), ' bytes'

    if ( freeram() < nbytes ) then
        print *, ' Insufficient memory '
        print *, ' Number of bytes = ', display_with_com-
mas(nbytes)
        print *, ' Total physical ', display_with_com-
mas(freeram())
        print *, ' Available physical ', display_with_com-
mas(totalram())
        stop 20
    end if

    allocate(x(n))

    print 20,time_difference()
    20 format(' Allocate ',f22.18)

```

```

call random_number(x)

print 30,time_difference()
30 format(' Initialise                ',f22.18)

x_sum=sum(x)

print 40, time_difference()
40 format(' Intrinsic summation ',f22.18)

print 100, x_sum
100 format(45x,f20.10)

x_sum=kahan_sum(x,n)

print 50, time_difference()
50 format(' Kahan summation          ',f22.18)

print 100, x_sum

print *,' Total physical                ',display_with_com-
mas(freeram())
print *,' Available physical          ',display_with_com-
mas(totalram())

deallocate(x)

n=n*10_i64

end do

call end_timing()

```

end program ch4306

### 9.6.1 Sample output

Here is some sample output.

```

./ch4306_gnu.out
2022/ 5/ 5 14:49:30 381
  Problem size
                10,000,000
                80,000,000 bytes
Allocate                0.0001017000000001870
Initialise              0.1150717000000001442
Intrinsic summation    0.0156284999999998269

5000669.2088570781
Kahan summation        0.0559422999999998113

```

```

5000669.2088573920
  Total physical          13,136,986,112
  Available physical      13,393,960,960
  Problem size
      100,000,000
      800,000,000 bytes
  Allocate                0.0006683000000000954
  Initialise              1.1106556000000001186
  Intrinsic summation    0.1495850000000001855

49997221.2586892843
  Kahan summation        0.528356199999997500

49997221.2587036043
  Total physical          12,415,856,640
  Available physical      13,393,960,960
  Problem size
      1,000,000,000
      8,000,000,000 bytes
  Allocate                0.003667399999997656
  Initialise              12.188676499999999692
  Intrinsic summation    1.5120210000000004278

500006058.1260393262
  Kahan summation        5.293246199999998680

500006058.1257698536
  Total physical          5,201,567,744
  Available physical      13,393,960,960
  Problem size
      10,000,000,000
      80,000,000,000 bytes
  Insufficient memory
  Number of bytes =      80,000,000,000
  Total physical          13,216,075,776
  Available physical      13,393,960,960
STOP 20

```

## 9.7 Example 7: Modified memory leak example with memory checking - Windows

Here is the Fortran source.

```

include 'integer_kind_module.f90'
include 'memory_module_windows.f90'
include 'display_with_commas_module.f90'

! Update of ch1806 to give a diagnostic information

```

```

! about the run time memory behaviour
! of the program

program ch4307

  use iso_fortran_env

  use integer_kind_module
  use memory_module_windows
  use display_with_commas_module

!
! This is a variation on
! the pointer example in chapter
! 18 that has a memory leak.
!
! We use the memory query functions to provide
! warning messages as memory usage goes up.
!

  implicit none

  integer (i64)          :: n = 100000000_i64
  integer (i64)          :: I=0

  integer                :: allocate_status = 0

  integer (i64) , dimension ( : ) , pointer :: x
  integer (i64) , dimension (1:5) , target  :: y

  real                  :: avail-
able
  real                  :: physi-
cal
  real                  :: per-
centage_free

  Print *, ' Program starts'
  print *, compiler_version()
  print *, ' Memory usage           ', MemoryLoad(), ' %'
  print *, ' Total physical         ', display_with_com-
mas(TotalPhysical())
  print *, ' Available physical     ', display_with_com-
mas(AvailablePhysical())
  print *, ' Total page file        ', display_with_com-
mas(TotalPageFile())
  print *, ' Available page file    ', display_with_com-
mas(AvailablePageFile())

```

```

    print *, ' Total virtual          ', display_with_com-
mas(TotalVirtual())
    print *, ' Available virtual      ', display_with_com-
mas(AvailableVirtual())

    physical = real(AvailablePhysical())
    print *, ' '
    print *, ' Loop starts'
    print *, ' '

do

    allocate (x(1:n), stat=allocate_status)

    if (allocate_status>0) then
        print *, ' allocate failed. program ends.'
        stop
    end if

    x = 1_i64

    y = 10_i64

    x => y !           x now points to y

    i=i+1

    available = real(AvailablePhysical())

    percentage_free = (available/physical)*100

    if (percentage_free < 5.0) then
        print *, ' Memory usage over 95%'
        print *, ' Program terminates'
        print *, ' Iteration count was ', i
        stop 20
    end if

end do

end program

```

### 9.7.1 Sample output

Here is some sample output.

```

ch4307_nag.exe
  Program starts
NAG Fortran Compiler Release 7.0(Yurakucho) Build 7048
  Memory usage           13 %

```

```

Total physical          33,663,741,952
Available physical     28,979,843,072
Total page file        38,764,015,616
Available page file    32,489,934,848
Total virtual          140,737,488,224,256
Available virtual      140,733,000,802,304

```

Loop starts

Memory usage over 95%

Program terminates

Iteration count was 35

STOP: 20

## 9.8 Example 8: Modified memory leak example with memory checking - Linux

Here is the source file.

```

include 'integer_kind_module.f90'
include 'memory_module_linux.f90'
include 'display_with_commas_module.f90'

! Update of ch1806 to give a diagnostic information
! about the run time memory behaviour
! of the program

program ch4308

  use iso_fortran_env

  use integer_kind_module
  use memory_module_linux
  use display_with_commas_module

!
! This is a variation on
! the pointer example in chapter
! 18 that has a memory leak.
!
! We use the memory query functions to provide
! warning messages as memory usage goes up.
!

  implicit none

  integer (i64)          :: n = 100000000_i64
  integer (i64)          :: I=0

```

```

integer                                :: allocate_status = 0

integer (i64) , dimension ( : ) , pointer :: x
integer (i64) , dimension (1:5) , target  :: y

real                                     :: avail-
able
real                                     :: physi-
cal
real                                     :: per-
centage_free

Print *, ' Program starts'
print *, compiler_version()
print *, ' Total ram  ', totalram(), ' ', display_with_com-
mas(totalram())
print *, ' Free  ram  ', freeram() , ' ', display_with_com-
mas(freeram())

physical = real(totalram())
print *, ' '
print *, ' Loop starts'
print *, ' '

do

    allocate (x(1:n), stat=allocate_status)

    if (allocate_status>0) then
        print *, ' allocate failed. program ends.'
        stop
    end if

    x = 1_i64

    y = 10_i64

    x => y !                x now points to y

    i=i+1

    available = real(freeram())

    percentage_free = (available/physical)*100

    if (percentage_free < 5.0) then
        print *, ' Memory usage over 95%'
        print *, ' Program terminates'
    end if
end do

```

```
        print *, ' Iteration count was ', I
        stop 20
    end if

end do

end program
```

### 9.8.1 Sample output

Here is some sample output.

```
./ch4308_gnu.out
  Program starts
GCC version 11.2.1 20220420 [revision
691af15031e00227ba6d5935c1d737026cda4129]
  Total ram                33663741952
33,663,741,952
  Free ram                 29230915584
29,230,993,408

  Loop starts

  Memory usage over 95%
  Program terminates
  Iteration count was          35
STOP 20
```

### 9.9 Compilation details

Examples 1 and 2 can just be simply compiled from the command line.

Examples 3, 4, 5, 6, 7 and 8 involve Fortran and C. We have provided batch files and shell scripts to help out here.

‘Though this be madness, yet there is method in’t’

Shakespeare.

‘Plenty of practice’ he went on repeating, all the time that Alice was getting him on his feet again. ‘plenty of practice.’

The White Knight, Through the Looking Glass and What Alice Found There, Lewis Carroll.

## 10 Compilers used

In this chapter we will look at the compilers we use with simple, debug and run time compilation flags. A later chapter has details of the diagnostic capabilities of the compilers we use. We recommend teaching and code development with the best diagnostic compiler available. We also recommend developing with a minimum of three compilers.

We currently use the following compilers

- NAG
- Intel
- gfortran
- nvidia
- Cray

The NAG and Intel compilers we use natively on both Windows and Linux.

The gfortran compiler we use on Linux primarily. We use it on a native install (openSuSe linux), under Hyper-V (openSuSe, Redhat, ubuntu), and also under WSL (openSuSe and Ubuntu).

The Nvidia compiler we use under Linux. There is no Windows version at the moment. We use it under a native install (openSuSe), under Hyper-V (openSuSe and Redhat) and under WSL (openSuSe).

The Cray compiler we use on the HPC systems at Edinburgh.

### 10.1 NAG

This is the compiler we teach with. It has consistently been one of the best diagnostic compilers available. The compiler is available for Windows, Linux and the Mac.

#### 10.1.1 Normal compile

nagfor

#### 10.1.2 Debug compile

One or more of

- -C=all
- -C=undefined
- -f2018 or -f2008
- -g
- -gline

- -ieee=stop
- -info
- -mtrace=verbose
- -thread\_safe

### 10.1.3 Optimised compile - simple

nagfor -O4

### 10.1.4 Optimised compile - openmp

nagfor -O4 -fopenmp

### 10.1.5 optimised compile - coarray

nagfor -O4 -coarray

### 10.1.6 optimised compile - mpi

Not available by default.

## 10.2 Intel

This compiler is available for both Windows and Linux.

### 10.2.1 Normal compile

ifort

### 10.2.2 Debug compile

One or more of

- /check:all
- /debug:all
- /fpe:0
- /gen-interfaces
- /standard-semantic
- /traceback
- /warn:all

Here are some extracts from the help files.

/check:all - enables the following

- check arg\_temp\_created
  - Enables run-time checking on whether actual arguments are copied into temporary storage before routine calls. If a copy is made at run-time, an informative message is displayed.
- check assume
  - Enables run-time checking on whether the scalar-Boolean-expression in the ASSUME directive is true and that the addresses in the ASSUME\_ALIGNED directive are aligned on the specified byte boundaries. If the

test is `.FALSE.`, a run-time error is reported and the execution terminates.

- check bounds
  - Enables compile-time and run-time checking for array subscript and character substring expressions. An error is reported if the expression is outside the dimension of the array or the length of the string. For array bounds, each individual dimension is checked. For arrays that are dummy arguments, only the lower bound is checked for a dimension whose upper bound is specified as `*` or where the upper and lower bounds are both 1. For some intrinsics that specify a `DIM=` dimension argument, such as `LBOUND`, an error is reported if the specified dimension is outside the declared rank of the array being operated upon. Once the program is debugged, omit this option to reduce executable program size and slightly improve run-time performance. It is recommended that you do bounds checking on unoptimized code. If you use option `check bounds` on optimized code, it may produce misleading messages because registers (not memory locations) are used for bounds values.
- check contiguous
  - Tells the compiler to check pointer contiguity at pointer-assignment time. This will help prevent programming errors such as assigning contiguous pointers to non-contiguous objects.
- check format
  - Issues the run-time `FORVARMIS` fatal error when the data type of an item being formatted for output does not match the format descriptor being used (for example, a `REAL*4` item formatted with an `I` edit descriptor). With `check noformat`, the data item is formatted using the specified descriptor unless the length of the item cannot accommodate the descriptor (for example, it is still an error to pass an `INTEGER*2` item to an `E` edit descriptor).
- check `output_conversion`
  - Issues the run-time `OUTCONERR` continuable error message when a data item is too large to fit in a designated format descriptor field without loss of significant digits. Format truncation occurs, the field is filled with asterisks (`*`), and execution continues.
- check pointers

- Enables run-time checking for disassociated or uninitialized Fortran pointers, unallocated allocatable objects, and integer pointers that are uninitialized.
- `check stack`
  - Enables checking on the stack frame. The stack is checked for buffer overruns and buffer underruns. This option also enforces local variables initialization and stack pointer verification. This option disables optimization and overrides any optimization level set by option `O`.
- `check uninit`
  - Enables run-time checking for uninitialized variables. If a variable is read before it is written, a run-time error routine will be called. Only local scalar variables of intrinsic type `INTEGER`, `REAL`, `COMPLEX`, and `LOGICAL` without the `SAVE` attribute are checked. To detect uninitialized arrays or array elements, please see option `[Q]init` or see the article titled: Detection of Uninitialized Floating-point Variables in Intel® Fortran, which is located in <https://software.intel.com/articles/detection-of-uninitialized-floating-point-variables-in-intel-fortran>
- `/debug:all`
  - Generates complete debugging information. It produces symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It is the same as specifying `/debug` with no keyword. If you specify `/debug:full` for an application that makes calls to C library routines and you need to debug calls into the C library, you should also specify `/dbglibs` to request that the appropriate C debug library be linked against.
- `/fpe:0`
  - Floating-point invalid, divide-by-zero, and overflow exceptions are enabled throughout the application when the main program is compiled with this value. If any such exceptions occur, execution is aborted. This option causes denormalized floating-point results to be set to zero.
- `/gen_interfaces`
  - Tells the compiler to generate an interface block for each routine in a source file.
- `/standard_semantics`

- Determines whether the current Fortran Standard behaviour of the compiler is fully implemented.
- /traceback
  - Tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time.
- /warn:all

### 10.2.3 Optimised compile - simple

ifort -O3 -heap-arrays /Qparallel -fp=strict

### 10.2.4 Optimised compile - openmp

ifort -O3 -heap-arrays /Qparallel -fp=strict -fopenmp

### 10.2.5 optimised compile - coarray

ifort -O3 -heap-arrays /Qparallel -fp=strict /Qcoarray

### 10.2.6 optimised compile - mpi

A batch file is available under Windows and a shell script under Linux

- mpif90

## 10.3 gfortran

This compiler is available for a wide variety of platforms. It is free.

### 10.3.1 Normal compile

gfortran

### 10.3.2 Debug compile

One or more of

- -fbacktrace
- -fcheck:all
- -ffpe-trap=zero,overflow,underflow
- -g
- -O
- -pedantic-errors
- -std=f2008
- -Wall
- -Wunderflow

Here are some extracts from the help files.

- -fbacktrace
  - trace back in the event of a run time error, i.e. the Fortran runtime library tries to output a backtrace of the error
- -fcheck

- Enable the generation of run-time checks; the argument shall be a comma-delimited list of the following keywords. all Enable all run-time test of -fcheck
- -ffpe-trap=list
  - Specify a list of floating point exception traps to enable
- -pedantic
  - Issue warnings for uses of extensions to Fortran 95
- -std
  - standard conformance
- -Wall
  - Enables commonly used warning options pertaining to usage that we recommend avoiding and that we believe are easy to avoid. This currently includes -Waliasing, -Wampersand, -Wconversion, -Wsurprising, -Wc-binding-type, -Wintrinsics-std, -Wtabs,-Wintrinsic-shadow, -Wline-truncation, -Wtarget-lifetime, -Wreal-q-constant and -Wunused.
- -Wunderflow
  - Produce a warning when numerical constant expressions are encountered, which yield an UNDERFLOW during compilation. Enabled by default
- -Wrealloc-lhs
  - Warn when the compiler might insert code to for allocation or reallocation of an allocatable array variable of intrinsic type in intrinsic assignments

### 10.3.3 Optimised compile - simple

gfortran -O3

### 10.3.4 Optimised compile - openmp

gfortran -O3 -fopenmp

### 10.3.5 optimised compile - coarray

gfortran -O3 -fcoarray=

One of

- none
- single
- library

### 10.3.6 optimised compile - mpi

Separate install.

## 10.4 Nvidia

The primary platform that this compiler is available for is Linux. Additional information about the Nvidia offerings is given later in this chapter.

### 10.4.1 Normal compile

nvfortran

### 10.4.2 Debug compile

One or more of

- -C
- -g
- -Mbounds
- -Mchkptr
- -Mchkstk
- -traceback

### 10.4.3 Optimised compile - simple

nvfortran -O4

### 10.4.4 Optimised compile - openmp

-fopenmp

### 10.4.5 optimised compile - coarray

NA

### 10.4.6 optimised compile - mpi

NA

## 10.5 HP - nee Cray

This is available on HPC systems. We use it on the UK HPC systems at Edinburgh.

### 10.5.1 Normal compile

ftn

### 10.5.2 Debug compile

-G - debug level

-R - runtime checks

### 10.5.3 Optimised compile - simple

Default options

### 10.5.4 Optimised compile - openmp

Default options

### 10.5.5 optimised compile - coarray

Default options

### 10.5.6 optimised compile - mpi

Default options

## 10.6 Windows and Linux compile scripts

A small number of batch files (Windows) and shell scripts (linux) are available:

- nag\_4.bat - Windows NAG file
- intel\_4.bat - Windows Intel file
- nag\_4.sh - Linux NAG file
- intel\_4.sh - Linux Intel file
- gfortran\_4.sh - Linux gfortran file
- nvidia\_4.sh - Linux nvfortran file

### 10.6.1 Sample Windows batch file

Here is a listing of the nag\_4.bat file.

```

echo on
rem Windows
rem
rem .obj
rem _nag.exe
rem .mod
rem
rem nagfor = fortran compiler
rem nag      = vendor
rem nagfor = c++ compiler
rem %4 = openmp
rem
rem created 29/04/2022
rem
rem
rem introduction
nagfor      ch0401.f90 -o ch0401_nag.exe
nagfor      ch0402.f90 -o ch0402_nag.exe
rem arithmetic
nagfor      ch0501.f90 -o ch0501_nag.exe
nagfor      ch0502.f90 -o ch0502_nag.exe
nagfor      ch0503.f90 -o ch0503_nag.exe
nagfor      ch0504.f90 -o ch0504_nag.exe
rem nagfor      ch0504p.f90 -o ch0504p_nag.exe
nagfor      ch0505.f90 -o ch0505_nag.exe
nagfor      ch0506.f90 -o ch0506_nag.exe
nagfor      ch0507.f90 -o ch0507_nag.exe
nagfor      ch0508.f90 -o ch0508_nag.exe
nagfor      ch0509.f90 -o ch0509_nag.exe
nagfor      ch0510.f90 -o ch0510_nag.exe
nagfor      ch0511.f90 -o ch0511_nag.exe
nagfor      ch0512.f90 -o ch0512_nag.exe
nagfor      ch0513.f90 -o ch0513_nag.exe
nagfor      ch0514.f90 -o ch0514_nag.exe
nagfor      ch0515.f90 -o ch0515_nag.exe
nagfor      ch0516.f90 -o ch0516_nag.exe
nagfor      ch0517.f90 -o ch0517_nag.exe
nagfor      ch0518.f90 -o ch0518_nag.exe
rem nagfor      ch05_base_conversion.f90 -o ch05_base_conversion_nag.exe
rem nagfor      ch05_expression_equivalence.f90 -o ch05_expression_equivalence_nag.exe
rem nagfor      ch05_subtract.f90 -o ch05_subtract_nag.exe
rem arrays 1
nagfor      ch0601.f90 -o ch0601_nag.exe
nagfor      ch0602.f90 -o ch0602_nag.exe
rem arrays 2
nagfor      ch0701.f90 -o ch0701_nag.exe
nagfor      ch0702.f90 -o ch0702_nag.exe
nagfor      ch0703.f90 -o ch0703_nag.exe
nagfor      ch0704.f90 -o ch0704_nag.exe
nagfor      ch0705.f90 -o ch0705_nag.exe
nagfor      ch0706.f90 -o ch0706_nag.exe
nagfor      ch0707.f90 -o ch0707_nag.exe
nagfor      ch0708.f90 -o ch0708_nag.exe

```

```

nagfor      ch0709.f90 -o ch0709_nag.exe
rem arrays 3
nagfor      ch0801.f90 -o ch0801_nag.exe
nagfor      ch0802.f90 -o ch0802_nag.exe
nagfor      ch0803.f90 -o ch0803_nag.exe
nagfor      ch0804.f90 -o ch0804_nag.exe
nagfor      ch0805.f90 -o ch0805_nag.exe
nagfor      ch0806.f90 -o ch0806_nag.exe
nagfor      ch0807.f90 -o ch0807_nag.exe
nagfor      ch0808.f90 -o ch0808_nag.exe
nagfor      ch0809.f90 -o ch0809_nag.exe
nagfor      ch0810.f90 -o ch0810_nag.exe
nagfor      ch0811.f90 -o ch0811_nag.exe
nagfor      ch0812.f90 -o ch0812_nag.exe
nagfor      ch0813.f90 -o ch0813_nag.exe
rem output
nagfor      ch0901.f90 -o ch0901_nag.exe
nagfor      ch0902.f90 -o ch0902_nag.exe
nagfor      ch0903.f90 -o ch0903_nag.exe
nagfor      ch0904.f90 -o ch0904_nag.exe
nagfor      ch0905.f90 -o ch0905_nag.exe
nagfor      ch0906.f90 -o ch0906_nag.exe
nagfor      ch0907.f90 -o ch0907_nag.exe
nagfor      ch0908.f90 -o ch0908_nag.exe
nagfor      ch0909.f90 -o ch0909_nag.exe
nagfor      ch0910.f90 -o ch0910_nag.exe
nagfor      ch0911.f90 -o ch0911_nag.exe
nagfor      ch0912.f90 -o ch0912_nag.exe
nagfor      ch0913.f90 -o ch0913_nag.exe
nagfor      ch0914.f90 -o ch0914_nag.exe
nagfor      ch0915.f90 -o ch0915_nag.exe
nagfor      ch0916.f90 -o ch0916_nag.exe
nagfor      ch0917.f90 -o ch0917_nag.exe
nagfor      ch0918.f90 -o ch0918_nag.exe
nagfor      ch0919.f90 -o ch0919_nag.exe
rem input
nagfor      ch1001.f90 -o ch1001_nag.exe
nagfor      ch1002.f90 -o ch1002_nag.exe
nagfor      ch1003.f90 -o ch1003_nag.exe
nagfor      ch1004.f90 -o ch1004_nag.exe
nagfor      ch1005.f90 -o ch1005_nag.exe
nagfor      ch1006.f90 -o ch1006_nag.exe
nagfor      ch1007.f90 -o ch1007_nag.exe
nagfor      ch1008.f90 -o ch1008_nag.exe
nagfor      ch1009.f90 -o ch1009_nag.exe
rem i/o
nagfor      ch1101.f90 -o ch1101_nag.exe
nagfor      ch1102.f90 -o ch1102_nag.exe
nagfor      ch1103.f90 -o ch1103_nag.exe
rem functions
nagfor      ch1201.f90 -o ch1201_nag.exe
nagfor      ch1202.f90 -o ch1202_nag.exe
nagfor      ch1203.f90 -o ch1203_nag.exe
nagfor      ch1204.f90 -o ch1204_nag.exe
nagfor      ch1205.f90 -o ch1205_nag.exe
nagfor      ch1206.f90 -o ch1206_nag.exe
nagfor      ch1207.f90 -o ch1207_nag.exe
nagfor      ch1208.f90 -o ch1208_nag.exe
nagfor      ch1209.f90 -o ch1209_nag.exe
nagfor      ch1210.f90 -o ch1210_nag.exe
nagfor      ch1211.f90 -o ch1211_nag.exe
nagfor -c ch12_gcd_module.f90 -o ch12_gcd_module_nag.obj
rem control structures
nagfor      ch1301.f90 -o ch1301_nag.exe
nagfor      ch1302.f90 -o ch1302_nag.exe
nagfor      ch1303.f90 -o ch1303_nag.exe
nagfor      ch1304.f90 -o ch1304_nag.exe
nagfor      ch1305.f90 -o ch1305_nag.exe
nagfor      ch1306.f90 -o ch1306_nag.exe
nagfor      ch1307.f90 -o ch1307_nag.exe
rem characters
nagfor      ch1401.f90 -o ch1401_nag.exe
nagfor      ch1402.f90 -o ch1402_nag.exe

```

```

nagfor      ch1403.f90 -o ch1403_nag.exe
nagfor      ch1404.f90 -o ch1404_nag.exe
nagfor      ch1405.f90 -o ch1405_nag.exe
nagfor      ch1406.f90 -o ch1406_nag.exe
nagfor      ch1407.f90 -o ch1407_nag.exe
rem complex
nagfor      ch1501.f90 -o ch1501_nag.exe
nagfor      ch1502.f90 -o ch1502_nag.exe
rem 16 logical
rem derived types
nagfor      ch1701.f90 -o ch1701_nag.exe
nagfor      ch1702.f90 -o ch1702_nag.exe
nagfor      ch1703.f90 -o ch1703_nag.exe
nagfor      ch1704.f90 -o ch1704_nag.exe
nagfor      ch1705.f90 -o ch1705_nag.exe
rem pointers
nagfor      ch1801.f90 -o ch1801_nag.exe
nagfor      ch1802.f90 -o ch1802_nag.exe
nagfor      ch1803.f90 -o ch1803_nag.exe
nagfor      ch1804.f90 -o ch1804_nag.exe
nagfor      ch1805.f90 -o ch1805_nag.exe
nagfor      ch1806.f90 -o ch1806_nag.exe
nagfor      ch1807.f90 -o ch1807_nag.exe
nagfor      ch1808.f90 -o ch1808_nag.exe
nagfor      ch1809.f90 -o ch1809_nag.exe
nagfor      ch1810.f90 -o ch1810_nag.exe
nagfor      ch1811.f90 -o ch1811_nag.exe
nagfor      ch1812.f90 -o ch1812_nag.exe
rem subroutines 1
nagfor      ch1901.f90 -o ch1901_nag.exe
rem subroutines 2
nagfor      ch2001.f90 -o ch2001_nag.exe
nagfor      ch2002.f90 -o ch2002_nag.exe
nagfor      ch2003.f90 -o ch2003_nag.exe
nagfor      ch2004.f90 -o ch2004_nag.exe
nagfor      ch2005.f90 -o ch2005_nag.exe
nagfor      ch2006.f90 -o ch2006_nag.exe
nagfor      ch2007.f90 -o ch2007_nag.exe
rem modules
nagfor      ch2101.f90 -o ch2101_nag.exe
nagfor      ch2102.f90 -o ch2102_nag.exe
nagfor      ch2103.f90 -o ch2103_nag.exe
rem
rem module exe
rem
nagfor -c ch2104_timing_module.f90 -o ch2104_timing_module_nag.obj
nagfor      ch2105.f90 -o ch2105_nag.exe
nagfor -c ch21_maths_module.f90 -o ch21_maths_module_nag.obj
nagfor -c ch21_precision_module.f90 -o ch21_precision_module_nag.obj
rem data structuring
nagfor      ch2201.f90 -o ch2201_nag.exe
nagfor      ch2202.f90 -o ch2202_nag.exe
nagfor      ch2203.f90 -o ch2203_nag.exe
nagfor      ch2204.f90 -o ch2204_nag.exe
nagfor      ch2205.f90 -o ch2205_nag.exe
nagfor      ch2206.f90 -o ch2206_nag.exe
nagfor      ch2207.f90 -o ch2207_nag.exe
nagfor -c ch2207_date_module.f90 -o ch2207_date_module_nag.obj
nagfor      ch2208.f90 -o ch2208_nag.exe
nagfor      ch2209.f90 -o ch2209_nag.exe
nagfor      ch2210.f90 -o ch2210_nag.exe
rem algorithms and the big o
nagfor      ch2301.f90 -o ch2301_nag.exe
rem operator overloading
nagfor      ch2401.f90 -o ch2401_nag.exe
rem generic programming
nagfor      ch2501.f90 -o ch2501_nag.exe
nagfor      ch2502.f90 -o ch2502_nag.exe
rem
nagfor      ch25_generic_cs.cs -o ch25_generic_cs.cs
rem
nagfor      ch25_generic_cxx.cxx -o ch25_generic_cxx.cxx
rem
nagfor -c ch25_integer_kind_module.f90 -o ch25_integer_kind_module_nag.obj

```

```

nagfor -c ch25_sort_data_module.f90 -o ch25_sort_data_module_nag.obj
nagfor -c ch25_statistics_module.f90 -o ch25_statistics_module_nag.obj
rem mathematical examples
nagfor ch2601.f90 -o ch2601_nag.exe
nagfor ch2602.f90 -o ch2602_nag.exe
nagfor -c ch2602_fun1_module.f90 -o ch2602_fun1_module_nag.obj
nagfor -c ch2602_rkm_module.f90 -o ch2602_rkm_module_nag.obj
nagfor ch2603.f90 -o ch2603_nag.exe
nagfor ch2604.f90 -o ch2604_nag.exe
nagfor ch2605.f90 -o ch2605_nag.exe
nagfor ch2606.f90 -o ch2606_nag.exe
nagfor ch2607.f90 -o ch2607_nag.exe
nagfor ch2608.f90 -o ch2608_nag.exe
rem pdts
nagfor ch2701.f90 -o ch2701_nag.exe
nagfor -c ch2701_link_module.f90 -o ch2701_link_module_nag.obj
nagfor ch2702.f90 -o ch2702_nag.exe
nagfor -c ch2702_ragged_module.f90 -o ch2702_ragged_module_nag.obj
nagfor ch2703.f90 -o ch2703_nag.exe
nagfor -c ch2703_matrix_module.f90 -o ch2703_matrix_module_nag.obj
rem object oriented
nagfor -c ch2801_shape_module.f90 -o ch2801_shape_module_nag.obj
nagfor ch2801.f90 -o ch2801_nag.exe
rem
rem OOP - 1
rem
nagfor -c ch2802_shape_module.f90 -o ch2802_nag.obj
nagfor ch2803.f90 -o ch2803_nag.exe
nagfor -c ch2803_shape_module.f90 -o ch2803_shape_module_nag.obj
nagfor ch2804.f90 -o ch2804_nag.exe
nagfor -c ch2804_circle_module.f90 -o ch2804_circle_module_nag.obj
nagfor -c ch2804_rectangle_module.f90 -o ch2804_rectangle_mod-
ule_nag.obj
nagfor ch2805.f90 -o ch2805_nag.exe
nagfor -c ch2805_display_module.f90 -o ch2805_display_mod-
ule_nag.obj
nagfor -c ch2805_shape_module.f90 -o ch2805_shape_module_nag.obj
nagfor -c ch2805_shape_wrapper_module.f90 -o ch2805_shape_wrapper_mod-
ule_nag.obj
nagfor -c ch2806_shape_module.f90 -o ch2806_shape_module_nag.obj
rem
rem OOP - 2
rem
nagfor ch2901.f90 -o ch2901_nag.exe
nagfor -c ch2901_date_module.f90 -o ch2901_date_module_nag.obj
nagfor -c ch2901_day_and_month_name_module.f90 -o ch2901_day_and_month_name_mod-
ule_nag.obj
nagfor ch2902.f90 -o ch2902_nag.exe
nagfor -c ch2902_iso_date_module.f90 -o ch2902_iso_date_mod-
ule_nag.obj
nagfor ch2903.f90 -o ch2903_nag.exe
nagfor -c ch2903_date_wrapper_module.f90 -o ch2903_date_wrapper_mod-
ule_nag.obj
nagfor ch2904.f90 -o ch2904_nag.exe
nagfor -c ch2905_valid_date_module.f90 -o ch2905_valid_date_mod-
ule_nag.obj
rem
rem submodules
rem
nagfor ch3001.f90 -o ch3001_nag.exe
nagfor ch3002.f90 -o ch3002_nag.exe
rem
rem mpi
rem
rem requires an mpi installation
rem replace normal compile with mpif90
rem
rem must be compiled one at a time as the mpif90
rem command is a batch file
rem
rem mpif90 ch3201.f90 -o ch3201_nag.exe
rem mpif90 ch3202.f90 -o ch3202_nag.exe
rem mpif90 ch3203.f90 -o ch3203_nag.exe

```

```

rem mpif90 ch3204.f90 -o ch3204_nag.exe
rem mpif90 ch3205.f90 -o ch3205_nag.exe
rem
rem openmp
rem
rem add compiler specific flag
rem
nagfor -fopenmp      ch3301.f90 -o ch3301_nag.exe
nagfor -fopenmp      ch3302.f90 -o ch3302_nag.exe
nagfor -fopenmp      ch3303.f90 -o ch3303_nag.exe
nagfor -fopenmp      ch3304.f90 -o ch3304_nag.exe
nagfor -fopenmp      ch3305.f90 -o ch3305_nag.exe
rem
rem coarrays
rem
rem add compiler specific flag
rem
nagfor -coarray      ch3401.f90 -o ch3401_nag.exe
nagfor -coarray      ch3402.f90 -o ch3402_nag.exe
nagfor -coarray      ch3403.f90 -o ch3403_nag.exe
nagfor -coarray      ch3404.f90 -o ch3404_nag.exe
rem
rem c interop
rem
rem main program comes second
rem
rem nag requires
rem
rem nagfor -c ch3502.c   -o ch3502_nag.obj
rem
rem kind type query
rem
nagfor      ch3501.f90                -o ch3501_nag.exe
rem fortran calling c
nagfor -c   ch3502.c                  -o ch3502_nag.obj
nagfor      ch3502.f90      ch3502_nag.obj -o ch3502_nag.exe
rem c calling fortran
nagfor -c   ch3503.f90                -o ch3503_nag.obj
nagfor      ch3503.c      ch3503_nag.obj -o ch3503_nag.exe
rem c++ calling fortran
nagfor -c   ch3504.f90                -o ch3504_nag.obj
nagfor      ch3504.cpp      ch3504_nag.obj -o ch3504_nag.exe
rem fortran calling c
rem passing arrays
nagfor -c   ch3505.c                  -o ch3505_nag.obj
nagfor      ch3505.f90      ch3505_nag.obj -o ch3505_nag.exe
rem c calling fortran
rem passing arrays
nagfor -c   ch3506.f90                -o ch3506_nag.obj
nagfor      ch3506.c      ch3506_nag.obj -o ch3506_nag.exe
rem c++ calling fortran
rem passing arrays
nagfor -c   ch3507.f90                -o ch3507_nag.obj
nagfor      ch3507.cpp      ch3507_nag.obj -o ch3507_nag.exe
rem fortran calling c
rem 2 d arrays
nagfor -c   ch3508.c                  -o ch3508_nag.obj
nagfor      ch3508.f90      ch3508_nag.obj -o ch3508_nag.exe
rem c calling fortran
rem 2 d arrays
nagfor -c   ch3509.f90                -o ch3509_nag.obj
nagfor      ch3509.c      ch3509_nag.obj -o ch3509_nag.exe
rem c++ calling fortran
rem 2 d arrays
nagfor -c   ch3510.f90                -o ch3510_nag.obj
nagfor      ch3510.cpp      ch3510_nag.obj -o ch3510_nag.exe
rem c++ calling fortran
rem 2 d arrays
nagfor -c   ch3511.f90                -o ch3511_nag.obj
nagfor      ch3511.cpp      ch3511_nag.obj -o ch3511_nag.exe
rem c calling fortran
rem 2 d arrays
rem ch3512.c ch3512.c

```

```

nagfor -c      ch3512.f90                -o ch3512_nag.obj
nagfor        ch3512.c                    ch3512_nag.obj -o ch3512_nag.exe
rem fortran calling c
rem character passing
nagfor -c      ch3513.c                    -o ch3513_nag.obj
nagfor        ch3513.f90                  ch3513_nag.obj -o ch3513_nag.exe
rem fortran calling c++
rem character data
nagfor -c      ch3514.cpp                  -o ch3514_nag.obj
nagfor        ch3514.f90                  ch3514_nag.obj -o ch3514_nag.exe
rem ieee arithmetic
nagfor        ch3601.f90 -o ch3601_nag.exe
nagfor        ch3602.f90 -o ch3602_nag.exe
nagfor        ch3603.f90 -o ch3603_nag.exe
nagfor        ch3604.f90 -o ch3604_nag.exe
nagfor        ch3605.f90 -o ch3605_nag.exe
nagfor        ch3606.f90 -o ch3606_nag.exe
rem dtio
nagfor        ch3701.f90 -o ch3701_nag.exe
nagfor -c     ch3701_person_module.f90    -o                ch3701_person_module_nag.obj
nagfor        ch3702.f90 -o ch3702_nag.exe
nagfor -c     ch3702_person_module.f90    -o                ch3702_person_module_nag.obj
nagfor        ch3703.f90 -o ch3703_nag.exe
nagfor        ch3704.f90 -o ch3704_nag.exe
rem sorting and searching
rem
rem include 'integer_kind_module.f90'
rem include 'precision_module.f90'
rem include 'sort_data_module.f90'
rem include 'timing_module.f90'
rem
nagfor        ch3801.f90 -o ch3801_nag.exe
rem
rem dsort.f ch3802_dsort.f
rem isort.f ch3802_ssort.f
rem ssort.f ch3802_isort.f
rem
nagfor -c     ch3802_dsort.f                -o                ch3802_dsort.o
nagfor -c     ch3802_isort.f                -o                ch3802_isort.o
nagfor -c     ch3802_ssort.f                -o                ch3802_ssort.o
nagfor        ch3802.f90 ch3802_dsort.o ch3802_ssort.o ch3802_isort.o -o ch3802_nag.exe
rem
rem nag library call
rem
nagfor -c     ch3803.f90                    -o                ch3803_nag.obj
rem
rem include 'ch3804_date_module.f90'
rem include 'ch3804_generic_sort_module.f90'
rem include 'timing_module.f90'
rem
nagfor        ch3804.f90 -o ch3804_nag.exe
rem
nagfor -c     ch3804_date_module.f90        -o                ch3804_date_module_nag.obj
nagfor -c     ch3804_generic_sort_module.f90 -o            ch3804_generic_sort_mod-
ule_nag.obj
nagfor        ch3805.f90 -o ch3805_nag.exe
rem
rem missing data and statistics
rem
rem ch3901.cs c get data files
rem
rem ch3901.cs
rem
rem ch3902      sed file to convert --- to -999 flag values
rem
rem ch3902.sed
rem
rem
nagfor        ch3903.f90 -o ch3903_nag.exe
rem
nagfor -c     ch3903_met_office_station_module.f90 -o        ch3903_met_office_sta-
tion_module_nag.obj

```

```

nagfor -c      ch3903_statistics_module.f90          -o      ch3903_statistics_mod-
ule_nag.obj
nagfor        ch3904.f90                            -o ch3904_nag.exe
nagfor -c      ch3904_site_description_module.f90    -o      ch3904_site_descrip-
tion_module_nag.obj
rem
rem          ch3905_convert_to_nan.sed
nagfor -c      ch3906_statistics_module.f90          -o      ch3906_nag.obj
nagfor        ch3907.f90 -o ch3907_nag.exe
rem
rem converting from fortran 77
rem
rem simple subroutine
rem
rem There are three versions of this subroutine
rem f66
rem f77
rem f90
rem
nagfor -c      ch4001_f66.for          -o ch4001_66_nag.obj
nagfor -c      ch4001_f77.for          -o ch4001_77_nag.obj
nagfor -c      ch4001.f90              -o ch4001_90_nag.obj
rem fortran 77
nagfor -c      ch4002_dsort.f          -o      ch4002_dsort_nag.obj
nagfor -c      ch4002_isort.f          -o      ch4002_isort_nag.obj
nagfor -c      ch4002_ssort.f          -o      ch4002_ssort_nag.obj
rem metcalf
nagfor -c      ch4003.f90              -o ch4003_nag_obj
rem nag polish
nagfor -c      ch4004.f90              -o ch4004_nag_obj
rem
rem ch4005 - date conversion
rem
nagfor -c      ch4005.for              -o ch4005_for_nag.obj
nagfor -c      ch4005.f90              -o ch4005_f90_nag.obj
rem
rem ch4006 - example 6 - misnumbered subroutines
rem
nagfor -c      ch4006_i64.f90          -o ch4006_i64_nag.obj
nagfor -c      ch4006_qp.f90          -o ch4006_qp_nag.obj
rem
rem dislin graphics
rem
rem You need a compiler with dislin bindings.
rem
rem nagfor      ch41_dislin_01.f90
rem nagfor      ch41_dislin_02.f90
rem nagfor      ch41_dislin_03.f90
rem nagfor      ch41_dislin_04.f90
rem nagfor      ch41_dislin_05.f90
rem
rem abstract interfaces and procedure pointers
rem
nagfor          ch4201.f90 -o ch4201_nag.exe
rem
rem ch43 - miscellaneous examples
rem
rem          add commas module
rem
nagfor -c      ch4301.f90              -o      ch4301_nag.exe
rem
rem          kahan summation with timing
rem
nagfor          ch4302.f90              -o      ch4302_nag.exe
rem
rem          reporting memory usage under windows
rem
nagfor -c      ch4303_memory_module_windows.c -o ch4303_memory_module_windows_nag.obj
nagfor          ch4303.f90              ch4303_memory_module_windows_nag.obj -o
ch4303_nag.exe
rem
rem          reporting memory usage under linux
rem

```

```

rem nagfor -c ch4304_memory_module_linux.c -o ch4304_memory_module_linux_nag.o
rem nagfor ch4304.f90 ch4304_memory_module_linux_nag.o -o
ch4304_nag.out
rem
rem kahan summation with memory usage - windows
rem
nagfor -c ch4303_memory_module_windows.c -o ch4303_memory_module_windows_nag.obj
nagfor ch4305.f90 ch4303_memory_module_windows_nag.obj -o
ch4305_nag.exe
rem
rem kahan summation with memory usage - Linux
rem
rem nagfor -c ch4304_memory_module_linux.c -o ch4304_memory_module_windows_linux.o
rem nagfor ch4306.f90 ch4304_memory_module_windows_linux.o
-o ch4306_nag.out
rem
rem memory leak with memory test - windows
rem
nagfor -c ch4303_memory_module_windows.c -o ch4303_memory_module_windows_intel.obj
nagfor ch4307.f90 ch4303_memory_module_windows_intel.obj -o
ch4307_nag.exe
rem
rem memory leak with memory test - linux
rem
rem nagfor -c ch4304_memory_module_linux.c -o ch4304_memory_module_linux_intel.o
rem nagfor ch4308.f90 ch4304_memory_module_linux_intel.o -o
ch4308_nag.out
rem
rem modules
rem
nagfor -c c_interop_module.f90 -o c_interop_mod-
ule_nag.obj
nagfor -c date_module_implementation.f90 -o date_module_implemen-
tation_nag.obj
nagfor -c date_module_interface.f90 -o date_module_inter-
face_nag.obj
nagfor -c day_and_month_name_module.f90 -o
day_and_month_name_module_nag.obj
nagfor -c integer_kind_module.f90 -o integer_kind_mod-
ule_nag.obj
nagfor -c maths_module.f90 -o maths_mod-
ule_nag.obj
nagfor -c precision_module.f90 -o precision_mod-
ule_nag.obj
nagfor -c sort_data_module.f90 -o sort_data_mod-
ule_nag.obj
nagfor -c statistics_module.f90 -o statistics_mod-
ule_nag.obj
nagfor -c subl_module.f90 -o subl_module_nag.obj
nagfor -c timing_module.f90 -o timing_mod-
ule_nag.obj
rem
rem modules that are used
rem
rem use abstract_function_interface_module
rem use add_commas_module
rem use ch3701_person_module
rem use ch3702_person_module
rem use character_binary_search_module
rem use character_linked_list_module
rem use character_list_module
rem use c_interop_module
rem use circle_module
rem use date_module
rem use date_sub
rem use date_wrapper_module
rem use display_module
rem use etox_module
rem use factorial_module
rem use fun01
rem use fun02
rem use fun1_module
rem use gcd_module

```

```

rem use ge_module
rem use generic_sort_module
rem use integer_kind_module
rem use interact_module
rem use kahan_summation_module
rem use link_module
rem use maths_module
rem use matrix_module
rem use md_module
rem use met_office_station_module
rem use pdt_matrix_module
rem use personal_module
rem use precision_module
rem use print_data_module
rem use print_tree_module
rem use ragged_module
rem use read_data_module
rem use read_module
rem use real_list_module
rem use rectangle_module
rem use rkm_module
rem use running_average_module
rem use shape_module
rem use shape_wrapper_module
rem use site_description_module
rem use solve_module
rem use sort_data_module
rem use sparse_vector_module
rem use statistics_module
rem use sub1_module
rem use subs_module
rem use swap_module
rem use timing_module
rem use t_position
rem use tree_module
rem use tree_node_module
rem use tree_node_module
rem use us_date_module_01
rem use windows_memory_status_module
rem
rem source files that are included
rem
rem include 'add_commas_module.f90'
rem include 'ch2207_date_module.f90'
rem include 'ch2602_fun1_module.f90'
rem include 'ch2602_rkm_module.f90'
rem include 'ch2701_link_module.f90'
rem include 'ch2702_ragged_module.f90'
rem include 'ch2703_matrix_module.f90'
rem include 'ch2801_shape_module.f90'
rem include 'ch2803_shape_module.f90'
rem include 'ch2804_circle_module.f90'
rem include 'ch2804_rectangle_module.f90'
rem include 'ch2805_display_module.f90'
rem include 'ch2805_shape_module.f90'
rem include 'ch2805_shape_wrapper_module.f90'
rem include 'ch2901_date_module.f90'
rem include 'ch2901_day_and_month_name_module.f90'
rem include 'ch2902_iso_date_module.f90'
rem include 'ch2903_date_wrapper_module.f90'
rem include 'ch3002_fun1_module.f90'
rem include 'ch3002_rkm_implementation_module.f90'
rem include 'ch3002_rkm_interface_module.f90'
rem include 'ch3701_person_module.f90'
rem include 'ch3702_person_module.f90'
rem include 'ch3804_date_module.f90'
rem include 'ch3804_generic_sort_module.f90'
rem include 'ch3903_met_office_station_module.f90'
rem include 'ch3903_statistics_module.f90'
rem include 'ch3904_site_description_module.f90'
rem include 'ch4301_date_module.f90'
rem include 'c_interop_module.f90'
rem include 'date_module_implementation.f90'

```

```

rem include 'date_module_interface.f90'
rem include 'day_and_month_name_module.f90'
rem include 'integer_kind_module.f90'
rem include 'kahan_summation_module.f90'
rem include 'maths_module.f90'
rem include 'precision_module.f90'
rem include 'sort_data_module.f90'
rem include 'statistics_module.f90'
rem include 'sub1_module.f90'
rem include 'timing_module.f90'
rem include 'windows_memory_status_module.f90'
rem
rem _include_code files
rem
rem ch25_statistics_module_code.f90 ch25_statistics_module_include_code.f90
rem ch2801_shape_module_code.f90   ch2801_shape_module_include_code.f90
rem date_module_code.f90           date_module_include_code.f90
rem quicksort_code.f90             quicksort_include_code.f90
rem shape_module_code.f90          shape_module_include_code.f90
rem statistics_module_code.f90     statistics_module_include_code.f90

```

## 10.6.2 Sample Linux shell script

Here is the Intel shell script.

```

echo off
## rem Linux
## rem
## rem .o
## rem _intel.out
## rem .mod
## rem
## rem ifort = fortran compiler
## rem intel = vendor
## rem icc = c++ compiler
## rem %4 = openmp
## rem
## rem created 29/04/2022
## rem
## rem
## rem introduction
ifort ch0401.f90 -o ch0401_intel.out
ifort ch0402.f90 -o ch0402_intel.out
## rem arithmetic
ifort ch0501.f90 -o ch0501_intel.out
ifort ch0502.f90 -o ch0502_intel.out
ifort ch0503.f90 -o ch0503_intel.out
ifort ch0504.f90 -o ch0504_intel.out
## rem ifort ch0504p.f90 -o ch0504p_intel.out
ifort ch0505.f90 -o ch0505_intel.out
ifort ch0506.f90 -o ch0506_intel.out
ifort ch0507.f90 -o ch0507_intel.out
ifort ch0508.f90 -o ch0508_intel.out
ifort ch0509.f90 -o ch0509_intel.out
ifort ch0510.f90 -o ch0510_intel.out
ifort ch0511.f90 -o ch0511_intel.out
ifort ch0512.f90 -o ch0512_intel.out
ifort ch0513.f90 -o ch0513_intel.out
ifort ch0514.f90 -o ch0514_intel.out
ifort ch0515.f90 -o ch0515_intel.out
ifort ch0516.f90 -o ch0516_intel.out
ifort ch0517.f90 -o ch0517_intel.out
ifort ch0518.f90 -o ch0518_intel.out
## rem ifort ch05_base_conversion.f90 -o ch05_base_conversion_intel.out
## rem ifort ch05_expression_equivalence.f90 -o ch05_expression_equiva-
lence_intel.out
## rem ifort ch05_subtract.f90 -o ch05_subtract_intel.out
## rem arrays 1
ifort ch0601.f90 -o ch0601_intel.out
ifort ch0602.f90 -o ch0602_intel.out
## rem arrays 2
ifort ch0701.f90 -o ch0701_intel.out
ifort ch0702.f90 -o ch0702_intel.out
ifort ch0703.f90 -o ch0703_intel.out

```

```

ifort      ch0704.f90 -o ch0704_intel.out
ifort      ch0705.f90 -o ch0705_intel.out
ifort      ch0706.f90 -o ch0706_intel.out
ifort      ch0707.f90 -o ch0707_intel.out
ifort      ch0708.f90 -o ch0708_intel.out
ifort      ch0709.f90 -o ch0709_intel.out
### rem arrays 3
ifort      ch0801.f90 -o ch0801_intel.out
ifort      ch0802.f90 -o ch0802_intel.out
ifort      ch0803.f90 -o ch0803_intel.out
ifort      ch0804.f90 -o ch0804_intel.out
ifort      ch0805.f90 -o ch0805_intel.out
ifort      ch0806.f90 -o ch0806_intel.out
ifort      ch0807.f90 -o ch0807_intel.out
ifort      ch0808.f90 -o ch0808_intel.out
ifort      ch0809.f90 -o ch0809_intel.out
ifort      ch0810.f90 -o ch0810_intel.out
ifort      ch0811.f90 -o ch0811_intel.out
ifort      ch0812.f90 -o ch0812_intel.out
ifort      ch0813.f90 -o ch0813_intel.out
### rem output
ifort      ch0901.f90 -o ch0901_intel.out
ifort      ch0902.f90 -o ch0902_intel.out
ifort      ch0903.f90 -o ch0903_intel.out
ifort      ch0904.f90 -o ch0904_intel.out
ifort      ch0905.f90 -o ch0905_intel.out
ifort      ch0906.f90 -o ch0906_intel.out
ifort      ch0907.f90 -o ch0907_intel.out
ifort      ch0908.f90 -o ch0908_intel.out
ifort      ch0909.f90 -o ch0909_intel.out
ifort      ch0910.f90 -o ch0910_intel.out
ifort      ch0911.f90 -o ch0911_intel.out
ifort      ch0912.f90 -o ch0912_intel.out
ifort      ch0913.f90 -o ch0913_intel.out
ifort      ch0914.f90 -o ch0914_intel.out
ifort      ch0915.f90 -o ch0915_intel.out
ifort      ch0916.f90 -o ch0916_intel.out
ifort      ch0917.f90 -o ch0917_intel.out
ifort      ch0918.f90 -o ch0918_intel.out
ifort      ch0919.f90 -o ch0919_intel.out
### rem input
ifort      ch1001.f90 -o ch1001_intel.out
ifort      ch1002.f90 -o ch1002_intel.out
ifort      ch1003.f90 -o ch1003_intel.out
ifort      ch1004.f90 -o ch1004_intel.out
ifort      ch1005.f90 -o ch1005_intel.out
ifort      ch1006.f90 -o ch1006_intel.out
ifort      ch1007.f90 -o ch1007_intel.out
ifort      ch1008.f90 -o ch1008_intel.out
ifort      ch1009.f90 -o ch1009_intel.out
### rem i/o
ifort      ch1101.f90 -o ch1101_intel.out
ifort      ch1102.f90 -o ch1102_intel.out
ifort      ch1103.f90 -o ch1103_intel.out
### rem functions
ifort      ch1201.f90 -o ch1201_intel.out
ifort      ch1202.f90 -o ch1202_intel.out
ifort      ch1203.f90 -o ch1203_intel.out
ifort      ch1204.f90 -o ch1204_intel.out
ifort      ch1205.f90 -o ch1205_intel.out
ifort      ch1206.f90 -o ch1206_intel.out
ifort      ch1207.f90 -o ch1207_intel.out
ifort      ch1208.f90 -o ch1208_intel.out
ifort      ch1209.f90 -o ch1209_intel.out
ifort      ch1210.f90 -o ch1210_intel.out
ifort      ch1211.f90 -o ch1211_intel.out
ifort -c ch12_gcd_module.f90 -o ch12_gcd_module_intel.o
### rem control structures
ifort      ch1301.f90 -o ch1301_intel.out
ifort      ch1302.f90 -o ch1302_intel.out
ifort      ch1303.f90 -o ch1303_intel.out
ifort      ch1304.f90 -o ch1304_intel.out
ifort      ch1305.f90 -o ch1305_intel.out

```

```

ifort    ch1306.f90 -o ch1306_intel.out
ifort    ch1307.f90 -o ch1307_intel.out
## rem characters
ifort    ch1401.f90 -o ch1401_intel.out
ifort    ch1402.f90 -o ch1402_intel.out
ifort    ch1403.f90 -o ch1403_intel.out
ifort    ch1404.f90 -o ch1404_intel.out
ifort    ch1405.f90 -o ch1405_intel.out
ifort    ch1406.f90 -o ch1406_intel.out
ifort    ch1407.f90 -o ch1407_intel.out
## rem complex
ifort    ch1501.f90 -o ch1501_intel.out
ifort    ch1502.f90 -o ch1502_intel.out
## rem 16 logical
## rem derived types
ifort    ch1701.f90 -o ch1701_intel.out
ifort    ch1702.f90 -o ch1702_intel.out
ifort    ch1703.f90 -o ch1703_intel.out
ifort    ch1704.f90 -o ch1704_intel.out
ifort    ch1705.f90 -o ch1705_intel.out
## rem pointers
ifort    ch1801.f90 -o ch1801_intel.out
ifort    ch1802.f90 -o ch1802_intel.out
ifort    ch1803.f90 -o ch1803_intel.out
ifort    ch1804.f90 -o ch1804_intel.out
ifort    ch1805.f90 -o ch1805_intel.out
ifort    ch1806.f90 -o ch1806_intel.out
ifort    ch1807.f90 -o ch1807_intel.out
ifort    ch1808.f90 -o ch1808_intel.out
ifort    ch1809.f90 -o ch1809_intel.out
ifort    ch1810.f90 -o ch1810_intel.out
ifort    ch1811.f90 -o ch1811_intel.out
ifort    ch1812.f90 -o ch1812_intel.out
## rem subroutines 1
ifort    ch1901.f90 -o ch1901_intel.out
## rem subroutines 2
ifort    ch2001.f90 -o ch2001_intel.out
ifort    ch2002.f90 -o ch2002_intel.out
ifort    ch2003.f90 -o ch2003_intel.out
ifort    ch2004.f90 -o ch2004_intel.out
ifort    ch2005.f90 -o ch2005_intel.out
ifort    ch2006.f90 -o ch2006_intel.out
ifort    ch2007.f90 -o ch2007_intel.out
## rem modules
ifort    ch2101.f90 -o ch2101_intel.out
ifort    ch2102.f90 -o ch2102_intel.out
ifort    ch2103.f90 -o ch2103_intel.out
## rem
## rem module out
## rem
ifort -c ch2104_timing_module.f90 -o ch2104_timing_module_intel.o
ifort    ch2105.f90 -o ch2105_intel.out
ifort -c ch21_maths_module.f90 -o ch21_maths_module_intel.o
ifort -c ch21_precision_module.f90 -o ch21_precision_module_intel.o
## rem data structuring
ifort    ch2201.f90 -o ch2201_intel.out
ifort    ch2202.f90 -o ch2202_intel.out
ifort    ch2203.f90 -o ch2203_intel.out
ifort    ch2204.f90 -o ch2204_intel.out
ifort    ch2205.f90 -o ch2205_intel.out
ifort    ch2206.f90 -o ch2206_intel.out
ifort    ch2207.f90 -o ch2207_intel.out
ifort -c ch2207_date_module.f90 -o ch2207_date_module_intel.o
ifort    ch2208.f90 -o ch2208_intel.out
ifort    ch2209.f90 -o ch2209_intel.out
ifort    ch2210.f90 -o ch2210_intel.out
## rem algorithms and the big o
ifort    ch2301.f90 -o ch2301_intel.out
## rem operator overloading
ifort    ch2401.f90 -o ch2401_intel.out
## rem generic programming
ifort    ch2501.f90 -o ch2501_intel.out
ifort    ch2502.f90 -o ch2502_intel.out

```

```

## rem
ch25_generic_cs.cs ch25_generic_cs.cs
## rem
ch25_generic_cxx.cxx ch25_generic_cxx.cxx
ifort -c ch25_integer_kind_module.f90 -o ch25_integer_kind_module_intel.o
ifort -c ch25_sort_data_module.f90 -o ch25_sort_data_module_intel.o
ifort -c ch25_statistics_module.f90 -o ch25_statistics_module_intel.o
## rem mathematical examples
ifort ch2601.f90 -o ch2601_intel.out
ifort ch2602.f90 -o ch2602_intel.out
ifort -c ch2602_fun1_module.f90 -o ch2602_fun1_module_intel.o
ifort -c ch2602_rkm_module.f90 -o ch2602_rkm_module_intel.o
ifort ch2603.f90 -o ch2603_intel.out
ifort ch2604.f90 -o ch2604_intel.out
ifort ch2605.f90 -o ch2605_intel.out
ifort ch2606.f90 -o ch2606_intel.out
ifort ch2607.f90 -o ch2607_intel.out
ifort ch2608.f90 -o ch2608_intel.out
## rem pdts
ifort ch2701.f90 -o ch2701_intel.out
ifort -c ch2701_link_module.f90 -o ch2701_link_module_intel.o
ifort ch2702.f90 -o ch2702_intel.out
ifort -c ch2702_ragged_module.f90 -o ch2702_ragged_module_intel.o
ifort ch2703.f90 -o ch2703_intel.out
ifort -c ch2703_matrix_module.f90 -o ch2703_matrix_module_intel.o
## rem oect oriented
ifort -c ch2801_shape_module.f90 -o ch2801_shape_module_intel.o
ifort ch2801.f90 -o ch2801_intel.out
## rem
## rem OOP - 1
## rem
ifort -c ch2802_shape_module.f90 -o ch2802_intel.o
ifort ch2803.f90 -o ch2803_intel.out
ifort -c ch2803_shape_module.f90 -o ch2803_shape_module_intel.o
ifort ch2804.f90 -o ch2804_intel.out
ifort -c ch2804_circle_module.f90 -o ch2804_circle_module_intel.o
ifort -c ch2804_rectangle_module.f90 -o ch2804_rectangle_mod-
ule_intel.o
ifort ch2805.f90 -o ch2805_intel.out
ifort -c ch2805_display_module.f90 -o ch2805_display_module_intel.o
ifort -c ch2805_shape_module.f90 -o ch2805_shape_module_intel.o
ifort -c ch2805_shape_wrapper_module.f90 -o ch2805_shape_wrapper_mod-
ule_intel.o
ifort -c ch2806_shape_module.f90 -o ch2806_shape_module_intel.o
## rem
## rem OOP - 2
## rem
ifort ch2901.f90 -o ch2901_intel.out
ifort -c ch2901_date_module.f90 -o ch2901_date_module_intel.o
ifort -c ch2901_day_and_month_name_module.f90 -o ch2901_day_and_month_name_mod-
ule_intel.o
ifort ch2902.f90 -o ch2902_intel.out
ifort -c ch2902_iso_date_module.f90 -o ch2902_iso_date_module_intel.o
ifort ch2903.f90 -o ch2903_intel.out
ifort -c ch2903_date_wrapper_module.f90 -o ch2903_date_wrapper_mod-
ule_intel.o
ifort ch2904.f90 -o ch2904_intel.out
ifort -c ch2905_valid_date_module.f90 -o ch2905_valid_date_mod-
ule_intel.o
## rem
## rem submodules
## rem
ifort ch3001.f90 -o ch3001_intel.out
ifort ch3002.f90 -o ch3002_intel.out
ifort -c ch3002_fun1_module.f90 -o ch3002_fun1_module_intel.o
ifort -c ch3002_rkm_implementation_module.f90 -o ch3002_rkm_implementation_mod-
ule_intel.o
ifort -c ch3002_rkm_interface_module.f90 -o ch3002_rkm_interface_mod-
ule_intel.o
## rem
## rem mpi
## rem
## rem requires an mpi installation

```

```

## rem replace normal compile with mpif90
## rem
## rem must be compiled one at a time as the mpif90
## rem command is a batch file
## rem
## rem mpif90 ch3201.f90 -o ch3201_intel.out
## rem mpif90 ch3202.f90 -o ch3202_intel.out
## rem mpif90 ch3203.f90 -o ch3203_intel.out
## rem mpif90 ch3204.f90 -o ch3204_intel.out
## rem mpif90 ch3205.f90 -o ch3205_intel.out
## rem
## rem openmp
## rem
## rem add compiler specific flag
## rem
ifort -fopenmp      ch3301.f90 -o ch3301_intel.out
ifort -fopenmp      ch3302.f90 -o ch3302_intel.out
ifort -fopenmp      ch3303.f90 -o ch3303_intel.out
ifort -fopenmp      ch3304.f90 -o ch3304_intel.out
ifort -fopenmp      ch3305.f90 -o ch3305_intel.out
## rem
## rem coarrays
## rem
## rem add compiler specific flag
## rem
ifort -coarray      ch3401.f90 -o ch3401_intel.out
ifort -coarray      ch3402.f90 -o ch3402_intel.out
ifort -coarray      ch3403.f90 -o ch3403_intel.out
ifort -coarray      ch3404.f90 -o ch3404_intel.out
## rem
## rem c interop
## rem
## rem main program comes second
## rem
## rem Intel requires
## rem
## rem icc -c ch3502.c -o ch3502_intel.o
## rem
## rem kind type query
## rem
ifort      ch3501.f90                -o ch3501_intel.out
## rem fortran calling c
icc -c     ch3502.c                  -o ch3502_intel.o
ifort     ch3502.f90      ch3502_intel.o -o ch3502_intel.out
## rem c calling fortran
ifort -c   ch3503.f90                -o ch3503_intel.o
icc       ch3503.c      ch3503_intel.o -o ch3503_intel.out
## rem c++ calling fortran
ifort -c   ch3504.f90                -o ch3504_intel.o
icc       ch3504.cxx    ch3504_intel.o -o ch3504_intel.out
## rem fortran calling c
## rem passing arrays
icc -c     ch3505.c                  -o ch3505_intel.o
ifort     ch3505.f90      ch3505_intel.o -o ch3505_intel.out
## rem c calling fortran
## rem passing arrays
ifort -c   ch3506.f90                -o ch3506_intel.o
icc       ch3506.c      ch3506_intel.o -o ch3506_intel.out
## rem c++ calling fortran
## rem passing arrays
ifort -c   ch3507.f90                -o ch3507_intel.o
icc       ch3507.cxx    ch3507_intel.o -o ch3507_intel.out
## rem fortran calling c
## rem 2 d arrays
icc -c     ch3508.c                  -o ch3508_intel.o
ifort     ch3508.f90      ch3508_intel.o -o ch3508_intel.out
## rem c calling fortran
## rem 2 d arrays
ifort -c   ch3509.f90                -o ch3509_intel.o
icc       ch3509.c      ch3509_intel.o -o ch3509_intel.out
## rem c++ calling fortran
## rem 2 d arrays
ifort -c   ch3510.f90                -o ch3510_intel.o

```

```

icc          ch3510.cxx      ch3510_intel.o    -o  ch3510_intel.out
## rem c++ calling fortran
## rem 2 d arrays
ifort -c    ch3511.f90                -o ch3511_intel.o
icc          ch3511.cxx      ch3511_intel.o    -o  ch3511_intel.out
## rem c calling fortran
## rem 2 d arrays
## rem ch3512.c ch3512.c
ifort -c    ch3512.f90                -o ch3512_intel.o
icc          ch3512.c        ch3512_intel.o    -o  ch3512_intel.out
## rem fortran calling c
## rem character passing
icc -c      ch3513.c                -o ch3513_intel.o
ifort      ch3513.f90      ch3513_intel.o    -o  ch3513_intel.out
## rem fortran calling c++
## rem character data
icc -c      ch3514.cxx                -o ch3514_intel.o
ifort      ch3514.f90      ch3514_intel.o    -o  ch3514_intel.out
## rem ieee arithmetic
ifort      ch3601.f90 -o ch3601_intel.out
ifort      ch3602.f90 -o ch3602_intel.out
ifort      ch3603.f90 -o ch3603_intel.out
ifort      ch3604.f90 -o ch3604_intel.out
ifort      ch3605.f90 -o ch3605_intel.out
ifort      ch3606.f90 -o ch3606_intel.out
## rem dtio
ifort      ch3701.f90 -o ch3701_intel.out
ifort -c   ch3701_person_module.f90    -o          ch3701_person_module_intel.o
ifort      ch3702.f90 -o ch3702_intel.out
ifort -c   ch3702_person_module.f90    -o          ch3702_person_module_intel.o
ifort      ch3703.f90 -o ch3703_intel.out
ifort      ch3704.f90 -o ch3704_intel.out
## rem sorting and searching
## rem
## rem include 'integer_kind_module.f90'
## rem include 'precision_module.f90'
## rem include 'sort_data_module.f90'
## rem include 'timing_module.f90'
## rem
ifort      ch3801.f90 -o ch3801_intel.out
## rem
## rem dsort.f ch3802_dsort.f
## rem isort.f ch3802_ssort.f
## rem ssort.f ch3802_isort.f
## rem
ifort -c   ch3802_dsort.f                -o          ch3802_dsort.o
ifort -c   ch3802_isort.f                -o          ch3802_isort.o
ifort -c   ch3802_ssort.f                -o          ch3802_ssort.o
ifort      ch3802.f90 ch3802_dsort.o ch3802_ssort.o ch3802_isort.o -o ch3802_intel.out
## rem
## rem nag library call
## rem
ifort -c   ch3803.f90                    -o          ch3803_intel.o
## rem
## rem include 'ch3804_date_module.f90'
## rem include 'ch3804_generic_sort_module.f90'
## rem include 'timing_module.f90'
## rem
ifort      ch3804.f90 -o ch3804_intel.out
## rem
ifort -c   ch3804_date_module.f90        -o          ch3804_date_module_intel.o
ifort -c   ch3804_generic_sort_module.f90 -o          ch3804_generic_sort_module_intel.o
ifort      ch3805.f90 -o ch3805_intel.out
## rem
## rem missing data and statistics
## rem
## rem ch3901.cs c get data files
## rem
## rem ch3901.cs
## rem
## rem ch3902 sed file to convert --- to -999 flag values
## rem
## rem ch3902.sed

```

```

## rem
ifort      ch3903.f90 -o ch3903_intel.out
## rem
ifort -c   ch3903_met_office_station_module.f90 -o      ch3903_met_office_sta-
tion_module_intel.o
ifort -c   ch3903_statistics_module.f90      -o      ch3903_statistics_mod-
ule_intel.o
ifort      ch3904.f90      -o ch3904_intel.out
ifort -c   ch3904_site_description_module.f90 -o      ch3904_site_description_mod-
ule_intel.o
## rem
## rem      ch3905_convert_to_nan.sed
ifort -c   ch3906_statistics_module.f90      -o      ch3906_intel.o
ifort      ch3907.f90 -o ch3907_intel.out
## rem
## rem converting from fortran 77
## rem
## rem simple subroutine
## rem
## rem There are three versions of this subroutine
## rem f66
## rem f77
## rem f90
## rem
ifort -c   ch4001_f66.for      -o ch4001_66_intel.o
ifort -c   ch4001_f77.for      -o ch4001_77_intel.o
ifort -c   ch4001.f90      -o ch4001_90_intel.o
## rem fortran 77
ifort -c   ch4002_dsort.f      -o      ch4002_dsort_intel.o
ifort -c   ch4002_isort.f      -o      ch4002_isort_intel.o
ifort -c   ch4002_ssort.f      -o      ch4002_ssort_intel.o
## rem metcalf
ifort -c   ch4003.f90      -o ch4003_intel.o
## rem nag polish
ifort -c   ch4004.f90      -o ch4004_intel.o
## rem
## rem ch4005 - date conversion
## rem
ifort -c   ch4005.for      -o ch4005_for_intel.o
ifort -c   ch4005.f90      -o ch4005_f90_intel.o
## rem
## rem ch4006 - example 6 - misnumbered subroutines
## rem
ifort -c   ch4006_i64.f90      -o ch4006_i64_intel.o
ifort -c   ch4006_qp.f90      -o ch4006_qp_intel.o
## rem
## rem dislin graphics
## rem
## rem You need a compiler with dislin bindings.
## rem
## rem ifort      ch41_dislin_01.f90
## rem ifort      ch41_dislin_02.f90
## rem ifort      ch41_dislin_03.f90
## rem ifort      ch41_dislin_04.f90
## rem ifort      ch41_dislin_05.f90
## rem
## rem abstract interfaces and procedure pointers
## rem
ifort      ch4201.f90 -o ch4201_intel.out
## rem
## rem ch43 - miscelaneous examples
## rem
## rem      add commas module
## rem
ifort -c   ch4301.f90      -o      ch4301_intel.o
## rem
## rem      kahan summation with timing
## rem
ifort      ch4302.f90      -o      ch4302_intel.out
## rem
## rem      reporting memory usage under windows
## rem

```

```

## rem icc -c          ch4303_memory_module_windows.c          -o ch4303_memory_module_win-
dows_intel.o
## rem ifort          ch4303.f90                                ch4303_memory_mod-
ule_windows_intel.o -o ch4303_intel.out
## rem
## rem          reporting memory usage under linux
## rem
icc -c          ch4304_memory_module_linux.c          -o ch4304_memory_module_linux_intel.o
ifort          ch4304.f90                                ch4304_memory_mod-
ule_linux_intel.o -o ch3404_intel.out
## rem
## rem          kahan summation with memory usage - windows
## rem
## rem icc -c          ch4303_memory_module_windows.c          -o ch4303_memory_module_win-
dows_intel.o
## rem ifort          ch4305.f90                                ch4303_memory_mod-
ule_windows_intel.o -o ch4305_intel.out
## rem
## rem          kahan summation with memory usage - Linux
## rem
icc -c          ch4304_memory_module_linux.c          -o ch4304_memory_module_windows_linux.o
ifort          ch4306.f90                                ch4304_memory_module_win-
dows_linux.o -o ch4306_intel.out
## rem
## rem          memory leak with memory test - windows
## rem
## rem icc -c ch4303_memory_module_windows.c -o ch4303_memory_module_windows_intel.o
## rem ifort          ch4307.f90                                ch4303_memory_module_windows_intel.o
-o ch4307_intel.out
## rem
## rem          memory leak with memory test - linux
## rem
icc -c ch4304_memory_module_linux.c -o ch4304_memory_module_linux_intel.o
ifort          ch4308.f90                                ch4304_memory_module_linux_intel.o -o
ch4308_intel.out
## rem
## rem modules
## rem
ifort -c          c_interop_module.f90                    -o          c_interop_mod-
ule_intel.o
ifort -c          date_module_implementation.f90          -o          date_module_implemen-
tation_intel.o
ifort -c          date_module_interface.f90                -o          date_module_inter-
face_intel.o
ifort -c          day_and_month_name_module.f90            -o          day_and_month_name_mod-
ule_intel.o
ifort -c          integer_kind_module.f90                  -o          integer_kind_mod-
ule_intel.o
ifort -c          maths_module.f90                          -o          maths_module_intel.o
ifort -c          precision_module.f90                      -o          precision_mod-
ule_intel.o
ifort -c          sort_data_module.f90                      -o          sort_data_mod-
ule_intel.o
ifort -c          statistics_module.f90                     -o          statistics_mod-
ule_intel.o
ifort -c          sub1_module.f90                           -o          sub1_module_intel.o
ifort -c          timing_module.f90                         -o          timing_module_intel.o
## rem
## rem modules that are used
## rem
## rem use abstract_function_interface_module
## rem use add_commas_module
## rem use ch3701_person_module
## rem use ch3702_person_module
## rem use character_binary_search_module
## rem use character_linked_list_module
## rem use character_list_module
## rem use c_interop_module
## rem use circle_module
## rem use date_module
## rem use date_sub
## rem use date_wrapper_module
## rem use display_module

```

```

## rem use etox_module
## rem use factorial_module
## rem use fun01
## rem use fun02
## rem use fun1_module
## rem use gcd_module
## rem use ge_module
## rem use generic_sort_module
## rem use integer_kind_module
## rem use interact_module
## rem use kahan_summation_module
## rem use link_module
## rem use maths_module
## rem use matrix_module
## rem use md_module
## rem use met_office_station_module
## rem use pdt_matrix_module
## rem use personal_module
## rem use precision_module
## rem use print_data_module
## rem use print_tree_module
## rem use ragged_module
## rem use read_data_module
## rem use read_module
## rem use real_list_module
## rem use rectangle_module
## rem use rkm_module
## rem use running_average_module
## rem use shape_module
## rem use shape_wrapper_module
## rem use site_description_module
## rem use solve_module
## rem use sort_data_module
## rem use sparse_vector_module
## rem use statistics_module
## rem use subl_module
## rem use subs_module
## rem use swap_module
## rem use timing_module
## rem use t_position
## rem use tree_module
## rem use tree_node_module
## rem use tree_node_module
## rem use us_date_module_01
## rem use windows_memory_status_module
## rem
## rem source files that are included
## rem
## rem include 'add_commas_module.f90'
## rem include 'ch2207_date_module.f90'
## rem include 'ch2602_fun1_module.f90'
## rem include 'ch2602_rkm_module.f90'
## rem include 'ch2701_link_module.f90'
## rem include 'ch2702_ragged_module.f90'
## rem include 'ch2703_matrix_module.f90'
## rem include 'ch2801_shape_module.f90'
## rem include 'ch2803_shape_module.f90'
## rem include 'ch2804_circle_module.f90'
## rem include 'ch2804_rectangle_module.f90'
## rem include 'ch2805_display_module.f90'
## rem include 'ch2805_shape_module.f90'
## rem include 'ch2805_shape_wrapper_module.f90'
## rem include 'ch2901_date_module.f90'
## rem include 'ch2901_day_and_month_name_module.f90'
## rem include 'ch2902_iso_date_module.f90'
## rem include 'ch2903_date_wrapper_module.f90'
## rem include 'ch3002_fun1_module.f90'
## rem include 'ch3002_rkm_implementation_module.f90'
## rem include 'ch3002_rkm_interface_module.f90'
## rem include 'ch3701_person_module.f90'
## rem include 'ch3702_person_module.f90'
## rem include 'ch3804_date_module.f90'
## rem include 'ch3804_generic_sort_module.f90'

```

```
## rem include 'ch3903_met_office_station_module.f90'  
## rem include 'ch3903_statistics_module.f90'  
## rem include 'ch3904_site_description_module.f90'  
## rem include 'ch4301_date_module.f90'  
## rem include 'c_interop_module.f90'  
## rem include 'date_module_implementation.f90'  
## rem include 'date_module_interface.f90'  
## rem include 'day_and_month_name_module.f90'  
## rem include 'integer_kind_module.f90'  
## rem include 'kahan_summation_module.f90'  
## rem include 'maths_module.f90'  
## rem include 'precision_module.f90'  
## rem include 'sort_data_module.f90'  
## rem include 'statistics_module.f90'  
## rem include 'sub1_module.f90'  
## rem include 'timing_module.f90'  
## rem include 'windows_memory_status_module.f90'  
## rem  
## rem _include_code files  
## rem  
## rem ch25_statistics_module_code.f90 ch25_statistics_module_include_code.f90  
## rem ch2801_shape_module_code.f90 ch2801_shape_module_include_code.f90  
## rem date_module_code.f90 date_module_include_code.f90  
## rem quicksort_code.f90 quicksort_include_code.f90  
## rem shape_module_code.f90 shape_module_include_code.f90  
## rem statistics_module_code.f90 statistics_module_include_code.f90
```

# 11 Development environments

We cover some of the development environment options in this chapter. Most Fortran compilers don't come with a bundled IDE. In this chapter we look at some options.

## 11.1 NAG

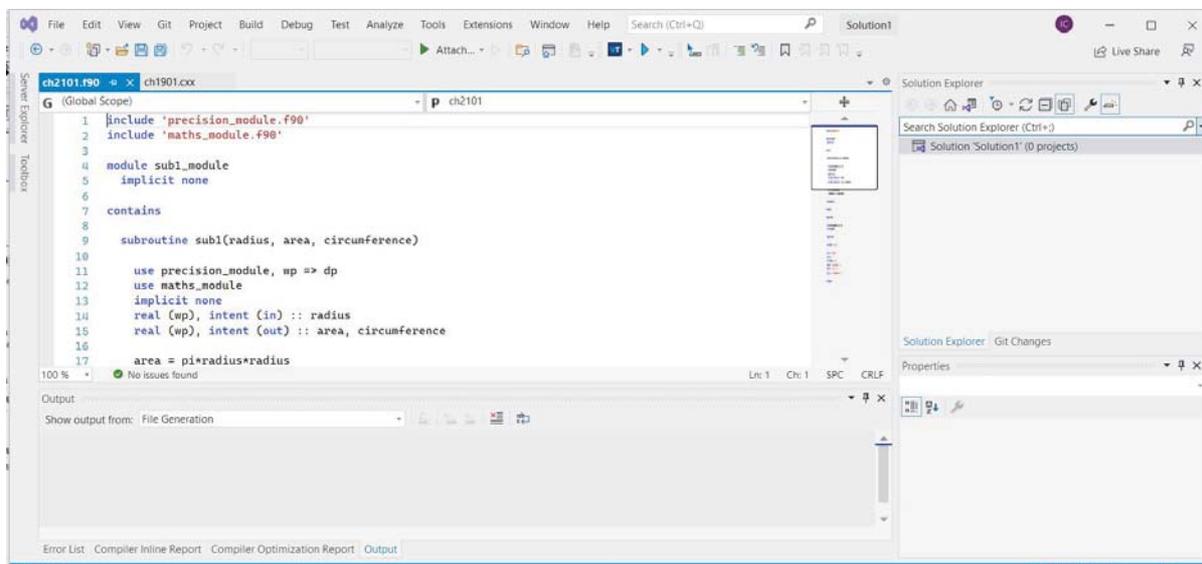
NAG provide Fortran Builder. We prepared an article for the August 2015 edition of Fortran Forum on Fortran Builder. It is a separate chapter in these notes.

## 11.2 Intel

On a Windows platform Intel integrates into Visual Studio. See the next section.

## 11.3 Microsoft Visual Studio

Here is a screen shot of a recent version of Visual Studio.



We recommend installing Visual Studio Community Edition before installing the Intel compiler suite. Visit

<https://visualstudio.microsoft.com/vs/community/>

for more details of Visual Studio.

Visit

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.zbt6x0>

for details of the Intel toolkits. We recommend installing the Intel base toolkit plus the Intel HPC toolkit.

We recommend installing a range of products including the Microsoft C++ compiler and C# compiler.

## 11.4 Microsoft Visual Code

Microsoft also make Visual Code available.

Here is some blurb taken from their site.

- Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (such as C++, C#, Java, Python, PHP, Go) and runtimes (such as .NET and Unity).

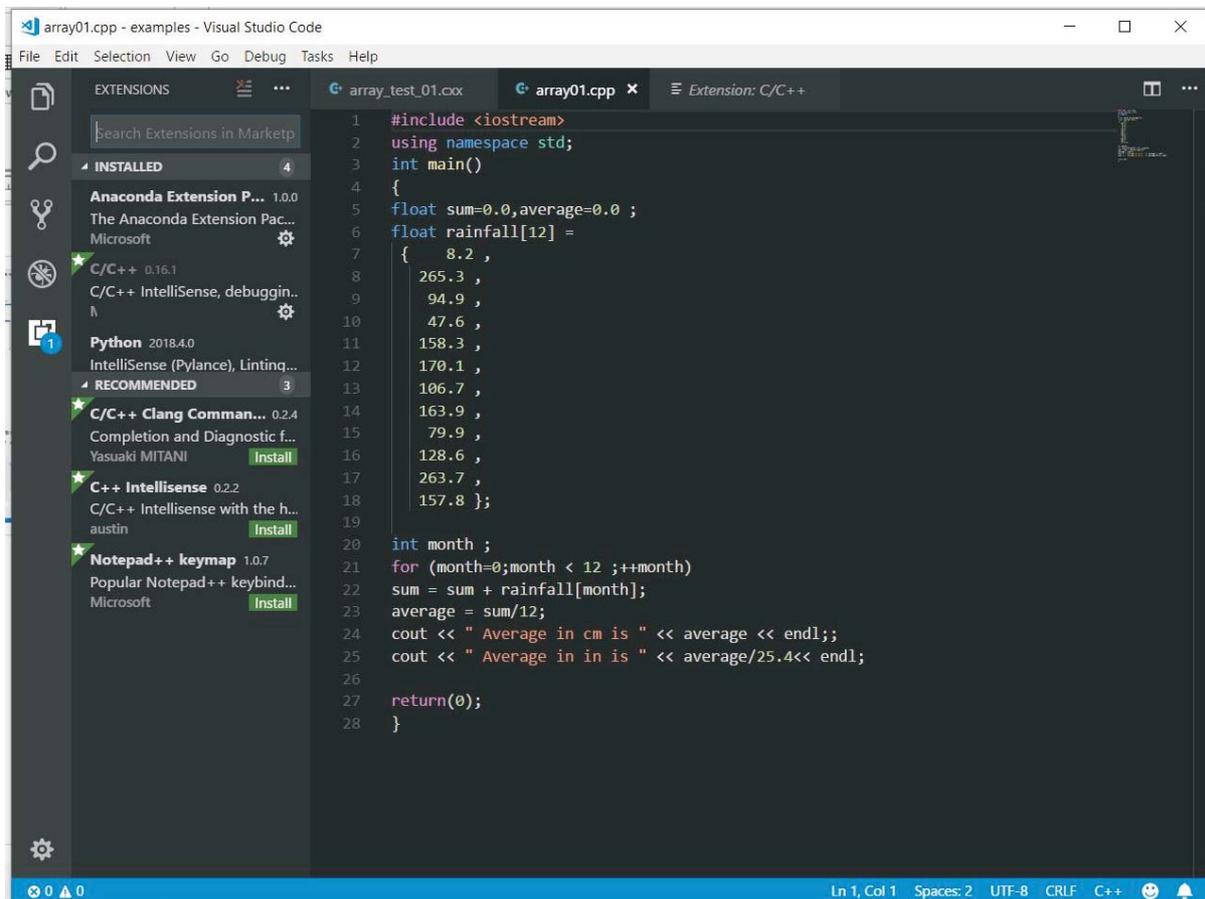
Here is a link

<https://code.visualstudio.com/>

Versions are available for

- Windows
- Linux
- Apple Mac

Here is a screen shot. taken from a Windows installation.



# 12 Intel and Nvidia toolkits

Both Intel and Nvidia toolkits offer the possibility of developing code that can run on both CPUs and GPUs, i.e. with a system with a cpu and graphics card it is possible to do processing on both the CPU and GPU.

## 12.1 Toolkit overview

Intel and Nvidia make their compilers available via a variety of toolkits: Here is the Intel link.

<https://www.intel.com/content/www/us/en/developer/tools/oneapi/toolkits.html#gs.597yak>

They make the following toolkits available:

- Intel® oneAPI Base Toolkit
- Intel oneAPI HPC Toolkit
- Intel® AI Analytics Toolkit
- Intel® Distribution of OpenVINO toolkit (Powered by oneAPI)
- Intel® oneAPI Rendering Toolkit
- Intel oneAPI IoT Toolkit
- Intel® System Bring-up Toolkit

The two Intel toolkits we have looked at are:

- Intel Base toolkit
- Intel HPC toolkit

Nvidia make the following toolkits available.

- Nvidia HPC toolkit
- Nvidia Cuda toolkit

The HPC toolkit can be found at

<https://developer.nvidia.com/hpc-sdk>

and the Cuda toolkit can be found at

<https://developer.nvidia.com/cuda-downloads>

More detailed coverage is given below.

## 12.2 Intel base toolkit

Intel recommend installing this toolkit first. You can take the default install (which is large) or choose a subset. We normally omit the Python component. We have used this toolkit on Windows, Linux (various distributions) and the Mac. Here are the components as of July 2022.

- Intel® oneAPI Collective Communications Library
- Intel® oneAPI Data Analytics Library
- Intel® oneAPI Deep Neural Networks Library
- Intel® oneAPI DPC++/C++ Compiler
- Intel® oneAPI DPC++ Library

- Intel® oneAPI Math Kernel Library
- Intel® oneAPI Threading Building Blocks
- Intel® oneAPI Video Processing Library
- Intel® Advisor
- Intel® Distribution for GDB\*
- Intel® Distribution for Python\*
- Intel® DPC++ Compatibility Tool
- Intel® FPGA Add-on for oneAPI Base Toolkit
- Intel® Integrated Performance Primitives
- Intel® VTune™ Profile

This is about 40 GB.

### **12.3 Intel HPC toolkit**

We recommend installing all of this toolkit. We have used this toolkit on Windows, Linux (various distributions) and the Mac. It has the following components as of July 2022.

- Intel oneAPI DPC++/C++ Compiler
- Intel® C++ Compiler Classic
- Intel® Cluster Checker
- Intel® Fortran Compiler
- Intel® Fortran Compiler Classic
- Intel® Inspector
- Intel® MPI Library
- Intel® Trace Analyzer and Collector

This is about 17 GB.

### **12.4 Nvidia HPC toolkit**

We have used it on a variety of Linux platforms. This is not available currently on a Windows platform.

### **12.5 Nvidia Cuda toolkit**

This is available for both Windows and Linux. We have used it on both platforms.

### **12.6 Nvidia and GPU programming**

In addition to conventional Fortran and C++ programming we are also trying out usage of the GPU, and have started running their examples and writing our own parallel examples.

#### **12.6.1 Nvidia Fortran**

For Fortran (using nvfortran) we have tried the following

- native linux
- hyper-v and a linux distro
- wsl and a linux distro

They all require access via an Nvidia driver to the GPU. The only one we have got to work is a native ubuntu 20.04.4 install on the Dell T5820.

There is no Windows HPC toolkit at this time.

We have had a number of problems with different versions of the sdk. We have used the following versions

- 21.3
- 21.9
- 21.11
- 22.3
- 22.5

We are currently using 22.5. We recommend using the latest version. This version compiles most of the examples.

The following are general compilation messages, which apply to all versions:

- a warning about detection of integer overflow;
- kind type errors with 128 bit reals - Nvidia support 32 and 64 bit reals only;
- no support for allocatable components;
- C compilation errors due to lack of conformance to the latest C standards;

These diagnostic messages are not a real issue.

We get problems with malloc and loader warnings with the 21.3 version. We get illegal instruction generation with the 21.9 and 21.11 versions on an AMD hardware platform.

We are currently using the following system setups:

- Dell T5280, Nvidia graphics card
  - ubuntu 20.04.4, native install, 22.5
  - openSuSe 15.3, hyper-v install, 22.5
  - ubuntu 20.04.4, wsl install, 22.5
- Dell 5515, Intel graphics card
  - openSuSe 15.3, hyper-v, 22.3
  - ubuntu 20.04.4, wsl, 22.5
- Dell 7100, Nvidia graphics card
  - openSuSe 15.3, native install,
  - ubuntu 20.04.4, wsl, 22.5

Obviously if you don't have an Nvidia graphics card you can't run and try out some of the examples.

## **12.7 Example 1: device driver test program**

This program is provided by Nvidia and tests out access to the GPU. This should be the first program you try out. Here is some sample output.

### Dell 5820, native ubuntu 20.04.4

```
Device Number: 0
Device Name: Quadro RTX 4000
Total Global Memory: 8.347 Gbytes
sharedMemPerBlock: 49152 bytes
regsPerBlock: 65536
warpSize: 32
maxThreadsPerBlock: 1024
maxThreadsDim: 1024 x 1024 x 64
maxGridSize: 2147483647 x 65535 x 65535
ClockRate: 1.545 GHz
Total Const Memory: 65536 bytes
Compute Capability Revision: 7.5
TextureAlignment: 512 bytes
deviceOverlap: 1
multiProcessorCount: 36
integrated: 0
canMapHostMemory: 1
ECCEnabled: 0
UnifiedAddressing: 1
L2 Cache Size: 4194304
maxThreadsPerSMP: 1024
```

### Dell 5515, WSL ubuntu 20.04.4

One run on this system generated a 684,306 line file. In the light of this the original example has been rewritten to test the stats of the device query. Here is the rewritten version.

```
! An example of getting device properties in CUDA Fortran
! Build with
!
!   nvfortran cufinfo.cuf
!
! Rewritten to test for the return status
! of the device query.
! Also added implicit none
!
program cufinfo

  use cudafor
  implicit none

  integer istat, num, numdevices
  type(cudaDeviceProp) :: prop
  istat = cudaGetDeviceCount(numdevices)
  if (istat /=0) then
    print *, ' istat = ', istat
    print *, ' numdevices = ', numdevices
    print *, ' Error in cudaGetDeviceCount'
```

```

        print *, ' Program terminates'
        stop 10
    end if
    do num = 0, numdevices-1
        istat = cudaGetDeviceProperties(prop, num)
        call printDeviceProperties(prop, num)
    end do
end
!
subroutine printDeviceProperties(prop, num)
use cudafor
type(cudadeviceprop) :: prop
integer num
ilen = verify(prop%name, ' ', .true.)
write (*,900) "Device Number: "          ,num
write (*,901) "Device Name: "           ,prop%name(1:ilen)
write (*,903) "Total Global Memory:
",real(prop%totalGlobalMem)/1e9," Gbytes"
write (*,902) "sharedMemPerBlock: "
,prop%sharedMemPerBlock," bytes"
write (*,900) "regsPerBlock: "          ,prop%regsPerBlock
write (*,900) "warpSize: "              ,prop%warpSize
write (*,900) "maxThreadsPerBlock: "    ,prop%maxThreadsPerBlock
write (*,904) "maxThreadsDim: "         ,prop%maxThreadsDim
write (*,904) "maxGridSize: "           ,prop%maxGridSize
write (*,903) "ClockRate: "
,real(prop%clockRate)/1e6," GHz"
write (*,902) "Total Const Memory: "    ,prop%totalConstMem,"
bytes"
write (*,905) "Compute Capability Revision: ",prop%ma-
jor,prop%minor
write (*,902) "TextureAlignment: "      ,prop%textureAlignment,"
bytes"
write (*,906) "deviceOverlap: "         ,prop%deviceOverlap
write (*,900) "multiProcessorCount: "    ,prop%multiProcessorCount
write (*,906) "integrated: "            ,prop%integrated
write (*,906) "canMapHostMemory: "      ,prop%canMapHostMemory
write (*,906) "ECCEnabled: "            ,prop%ECCEnabled
write (*,906) "UnifiedAddressing: "     ,prop%unifiedAddressing
write (*,900) "L2 Cache Size: "         ,prop%l2CacheSize
write (*,900) "maxThreadsPerSMP: "
,prop%maxThreadsPerMultiProcessor
900 format (a,i0)
901 format (a,a)
902 format (a,i0,a)
903 format (a,f5.3,a)
904 format (a,2(i0,1x,'x',1x),i0)
905 format (a,i0,'.',i0)

```

```
906 format (a,10)
return
end
```

Here is the output from this version of the program.

```
  istat =                35
 numdevices =            32529
 Error in cudaGetDeviceCount
 Program terminates
  10
```

## 12.8 Example 2: basic Cuda Fortran test program - integer type

Our example is based on an example in Chapter 2 of the Cuda Fortran Programming Guide. It illustrates the 6 basic steps involved in Cuda Fortran programming.

- Initialize and select the GPU to run on. Often this is implicit in the program and defaults to NVIDIA device 0.
- Allocate space for data on the GPU.
- Move data from the host to the GPU, or in some cases, initialize the data on the GPU.
- Launch kernels from the host to run on the GPU.
- Gather results back from the GPU for further analysis our output from the host program.
- Deallocate the data on the GPU allocated in step 2. This might be implicitly performed when the host program exits.

Here is the new source code. We have add timing information.

```
!
! This example is based on the
! first example
! in chapter 2 of the
! Cuda Fortran Programming Guide
!
! That example works with default integer type
!
! We also have 32 and 64 bit real versions
! of this example.
!
! We also make all arrays allocatable
! and start with an array size of
!
!      1,000,000 and loop to
! 1,000,000,000
!
! We also add timing information.
!
! We have also added implicit none
!
```

```

! compiler switches
!
! -fast
! -cudalib=cublas
!
include 'precision_module.f90'
include 'timing_module.f90'
module mytests
  implicit none
  contains
    attributes(global) subroutine test1( a )
      implicit none
      integer, device :: a(*)
      integer          :: I
      I = threadIdx%x
      a(I) = I
      return
    end subroutine test1
end module mytests
program t1
  use precision_module
  use timing_module

  use cudafor
  use mytests

  implicit none

  integer                                :: n =
1000000

  integer , allocatable , dimension(:)   :: x
  integer , allocatable , dimension(:) , device :: y

  integer                                ::
istat
  integer                                :: I
  integer                                :: j

  character (20)                          ::
heading
  heading = 'Program starts'
  call start_timing()
  istat = cudaSetDevice(0)
! initialise and select device
  print *, 'cudaSetDevice return value = ', istat

  do j=1,4

```

```

    print *, ' n = ', n
    allocate(x(1:n))

    heading = 'x allocation '
    print 10, heading, time_difference()
    10 format(a, 2x, f10.6)
    allocate(y(1:n))
! allocate data on GPU

    heading = 'y allocation '
    print 10, heading, time_difference()

    x=1

    heading = 'x=1 '
    print 10, heading, time_difference()

    y=x
! initialise the data

    heading = 'y=x '
    print 10, heading, time_difference()

    call test1<<<1,n>>> (y)
! launch the kernels

    heading = 'call test1 '
    print 10, heading, time_difference()

    x = y
! gather the results

    heading = 'x=y '
    print 10, heading, time_difference()

    print *, 'sum x array = ', sum(x)

    heading = 'sum '
    print 10, heading, time_difference()

    deallocate(x)

    heading = 'deallocate x '
    print 10, heading, time_difference()

    deallocate(y)
! deallocate the data

```

```

        heading = 'deallocate y '
        print 10,heading,time_difference()

        n=n*10

    end do

end program t1

```

Here is sample output from the Dell 5820.

```

2022/ 7/12 12:55:20 181
cudaSetDevice return value = 0
  n =          1000000
x allocation          0.048451
y allocation          0.170867
x=1                   0.001379
y=x                   0.000491
call test1            0.000008
x=y                   0.000519
  sum x array =      1000000
sum                   0.000267
deallocate x          0.000122
deallocate y          0.000115
  n =          10000000
x allocation          0.000009
y allocation          0.000132
x=1                   0.013051
y=x                   0.004534
call test1            0.000003
x=y                   0.003853
  sum x array =      10000000
sum                   0.002443
deallocate x          0.000969
deallocate y          0.000116
  n =          100000000
x allocation          0.000009
y allocation          0.001347
x=1                   0.130267
y=x                   0.045940
call test1            0.000003
x=y                   0.035294
  sum x array =      100000000
sum                   0.024353
deallocate x          0.008739
deallocate y          0.000313
  n =          1000000000
x allocation          0.000010

```

```

y allocation          0.004230
x=1                  1.304921
y=x                  0.457374
call test1           0.000007
x=y                  0.358499
  sum x array =      1000000000
sum                  0.250289
deallocate x         0.128961
deallocate y         0.002425

```

## 12.9 Example 3: 32 bit real variant

Here is the source code.

```

!
! This example is based on the first example
! in chapter 2 of the
! Cuda Fortran Programming Guide
! and reference
!
! That example works with default integer type
!
! We also have 32 and 64 bit real versions
! of this example.
!
! We also make all arrays allocatable
! and start with an array size of
!
!     1,000,000 and loop to
! 1,000,000,000
!
! We also add timing information.
!
! We have also added implicit none
!
include 'precision_module.f90'
include 'timing_module.f90'
module mytests
  use precision_module , wp => sp
  implicit none
  contains
    attributes(global) subroutine test1( a )
      implicit none
      real (wp) , device :: a(*)
      integer          :: I
      I = threadIdx%x
      a(I) = I
      return
    end subroutine test1
end module mytests

```

```

        end subroutine test1
end module mytests
program t1
  use precision_module , wp => sp
  use timing_module

  use cudafor
  use mytests

  implicit none

  integer :: n =
1000000

  real (wp) , allocatable , dimension(:) :: x

  real (wp) , allocatable , dimension(:) , device :: y

  integer ::
istat
  integer :: I
  integer :: j

  character (20) ::
heading
  heading = 'Program starts'
  call start_timing()
  istat = cudaSetDevice(0)
  print *, 'cudaSetDevice return value = ', istat

  do j=1,4

    print *, ' n = ', n
    allocate(x(1:n))

    heading = 'x '
    print 10, heading, time_difference()
    10 format(a, 2x, f10.6)
    allocate(y(1:n))

    heading = 'y '
    print 10, heading, time_difference()

    x=1

    heading = 'x=1 '
    print 10, heading, time_difference()

```

```

y=x

heading = 'y=x '
print 10,heading,time_difference()

call test1<<<1,n>>> (y)

heading = 'call test1 '
print 10,heading,time_difference()

x = y

heading = 'x=y '
print 10,heading,time_difference()

print *,'sum x array = ',sum(x)

heading = 'sum '
print 10,heading,time_difference()

deallocate(x)

heading = 'deallocate x '
print 10,heading,time_difference()

deallocate(y)

heading = 'deallocate y '
print 10,heading,time_difference()

n=n*10

end do

end program t1

```

Here is sample output.

```

2022/ 7/12 16:24:25 752
cudaSetDevice return value = 0
n = 1000000
x 0.050264
y 0.162845
x=1 0.001302
y=x 0.000502
call test1 0.000008
x=y 0.000518
sum x array = 1000000.
sum 0.000281

```

```

deallocate x          0.000134
deallocate y          0.000125
  n =      10000000
x                    0.000011
y                    0.000143
x=1                  0.012925
y=x                  0.004685
call test1           0.000003
x=y                  0.003839
  sum x array =      1.0000000E+07
sum                  0.002505
deallocate x          0.000990
deallocate y          0.000122
  n =      100000000
x                    0.000009
y                    0.001350
x=1                  0.130838
y=x                  0.046889
call test1           0.000003
x=y                  0.034959
  sum x array =      1.0000000E+08
sum                  0.024786
deallocate x          0.008942
deallocate y          0.000341
  n =      1000000000
x                    0.000011
y                    0.004231
x=1                  1.306275
y=x                  0.453667
call test1           0.000008
x=y                  0.356646
  sum x array =      5.3687091E+08
sum                  0.251174
deallocate x          0.088194
deallocate y          0.002399

```

## 12.10 Example 4: 64 bit real variant

Here is the source code.

```

!
! This example is based on the first example
! in chapter 2 of the
! Cuda Fortran Programming Guide
! and reference
!
! That example works with default integer type
!
```

```

! We also have 32 and 64 bit real versions
! of this example.
!
! We also make all arrays allocatable
! and start with an array size of
!
!     1,000,000 and loop to
! 1,000,000,000
!
! We also add timing information.
!
! We have also added implicit none
!
include 'precision_module.f90'
include 'timing_module.f90'
module mytests
  use precision_module , wp => dp
  implicit none
  contains
    attributes(global) subroutine test1( a )
      implicit none
      real (wp) , device :: a(*)
      integer          :: I
      I = threadIdx%x
      a(I) = I
      return
    end subroutine test1
end module mytests
program t1
  use precision_module , wp => dp
  use timing_module

  use cudafor
  use mytests

  implicit none

  integer                               :: n =
1000000

  real (wp) , allocatable , dimension(:) :: x

  real (wp) , allocatable , dimension(:) , device :: y

  integer                               ::
istat

  integer                               :: I
  integer                               :: j

```

```

integer :: al-
locate_status
character (20) ::
heading
  heading = 'Program starts'
  call start_timing()
  istat = cudaSetDevice(0)
  print *, 'cudaSetDevice return value = ', istat

  do j=1,4

    print *, ' n = ', n
    allocate(x(1:n), stat=allocate_status)
    if (allocate_status /=0) then
      print *, ' cpu allocation failed. Program termi-
nates'
      stop 10
    end if

    heading = 'x '
    print 10, heading, time_difference()
    10 format(a, 2x, f10.6)
    allocate(y(1:n), stat=allocate_status)

    if (allocate_status /=0) then
      print *, ' GPU allocation failed. Program termi-
nates'
      stop 20
    end if
    heading = 'y '
    print 10, heading, time_difference()

    x=1

    heading = 'x=1 '
    print 10, heading, time_difference()

    y=x

    heading = 'y=x '
    print 10, heading, time_difference()

    call test1<<<1,n>>> (y)

    heading = 'call test1 '
    print 10, heading, time_difference()

    x = y

```

```

    heading = 'x=y '
    print 10,heading,time_difference()

    print *,'sum x array = ',sum(x)

    heading = 'sum '
    print 10,heading,time_difference()

    deallocate(x)

    heading = 'deallocate x '
    print 10,heading,time_difference()

    deallocate(y)

    heading = 'deallocate y '
    print 10,heading,time_difference()

    n=n*10

end do

end program t1

```

Here is sample output.

```

2022/ 7/12 16:24:34 584
cudaSetDevice return value = 0
  n =      1000000
x                0.048970
y                0.159012
x=1              0.002601
y=x              0.000904
call test1      0.000008
x=y              0.000877
  sum x array = 1000000.000000000
sum              0.000534
deallocate x     0.000227
deallocate y     0.000123
  n =      10000000
x                0.000010
y                0.000276
x=1              0.026062
y=x              0.008828
call test1      0.000003
x=y              0.007593
  sum x array = 10000000.000000000
sum              0.005124

```

```
deallocate x          0.001918
deallocate y          0.000155
  n =      100000000
x                    0.000010
y                    0.002105
x=1                  0.262281
y=x                  0.086092
call test1           0.000003
x=y                  0.070654
  sum x array =      100000000.0000000
sum                  0.049349
deallocate x          0.017720
deallocate y          0.000536
  n =      1000000000
x                    0.000010
  GPU allocation failed. Program terminates
```

## 12.11 Nvidia Cuda

We have got this to work on Windows at this time using Microsoft VS 2022. This provides C++ based parallel programming.