Chapter 4
Introduction to Programming

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

Aims

The aims of the chapter are:

- To introduce the idea that there is a wide class of problems that can be solved with a computer and, further, that there is a relationship between the kind of problem to be solved and the choice of programming language that is used.
- To give some of the reasons for the choice of Fortran.
- To introduce the fundamental components or kinds of statements to be found in a general purpose programming language.
- To introduce the three concepts of name, type and value.
- To illustrate the above with sample programs based on three of the five intrinsic data types:
- character, integer and real.
- To introduce some of the formal syntactical rules of Fortran.

4.1 Introduction

We have seen that an algorithm is a sequence of steps that will solve a part or the whole of a problem. A program is the realisation of an algorithm in a programming language, and there are at first sight a surprisingly large number of programming languages. The reason for this is that there is a wide range of problems that are solved using a computer, e.g., the telephone company generating itemised bills or

I. Chivers and J. Sleightholme, *Introduction to Programming with Fortran*,
https://doi.org/10.1007/978-3-319-75502-1_4

the meteorological centre producing a weather forecast. These two problems make different demands on a programming language, and it is unlikely that the same language would be used to solve both.

The range of problems that you want to solve will therefore strongly influence your choice of programming language. Fortran stands for FORmula TRANslation, which gives a hint of the expected range of problems for which it is suitable.

4.2 Language Strengths and Weaknesses

Some of the reasons for choosing Fortran are:

- It is a modern and expressive language;
- The language is suitable for a wide class of both numeric and nonnumeric problems;
- The language is widely available on a range of hardware and operating system platforms;
- A lot of software already exists that has been written in Fortran. Some 15% of code worldwide is estimated to be in Fortran.

There are a few warts, however. Given that there has to be backwards compatibility with earlier versions some of the syntax is clumsy to say the least. However, a considerable range of problems can now be addressed quite cleanly, if one sticks to a subset of the language and adopts a consistent style.

4.3 Elements of a Programming Language

As with ordinary (so-called natural) languages, e.g., English, French, Gaelic, German, etc., programming languages have rules of syntax, grammar and spelling. The application of these rules in a programming language is more strict. A program has to be unambiguous, since it is a precise statement of the actions to be taken. Many everyday activities are rather vaguely defined — Buy some bread on your way home — but we are generally sufficiently adaptable to cope with the variations which occur as a result. if, in a program to calculate wages, we had an instruction deduct some money for tax and insurance we could have an awkward problem when the program calculated completely different wages for the same person for the same amount of work every time it was run. One of the implications of the strict syntax of a programming language for the novice is that apparently silly error messages will appear when one first starts writing programs. As with many other new subjects you will have to learn some of the jargon to understand these messages.

Programming languages are made up of statements. We will look at the various kinds of statements briefly below.

4.3.1 Data Description Statements

These are necessary to describe the kinds of data that are to be processed. In the wages program, for example, there is obviously a difference between people's names and the amount of money they earn, i.e., these two things are not the same, and it would not make any sense adding your name to your wages. The technical term for this is data type — a wage would be of a different data type (a number) to a surname (a sequence of characters).

4.3.2 Control Structures

A program can be regarded as a sequence of statements to solve a particular problem, and it is common to find that this sequence needs to be varied in practice. Consider again the wages program. It will need to select among a variety of circumstances (say married or single, paid weekly or monthly, etc), and also to repeat the program for everybody employed. So there is the need in a programming language for statements to vary and/or repeat a sequence of statements.

4.3.3 Data-Processing Statements

It is necessary in a programming language to be able to process data. The kind of processing required will depend on the kind or type of data. In the wages program, for example, you will need to distinguish between names and wages. Therefore there must be different kinds of statements to manipulate the different types of data, i.e., wages and names.

4.3.4 Input and Output (I/O) Statements

For flexibility, programs are generally written so that the data that they work on exist outside the program. In the wages example the details for each person employed would exist in a file somewhere, and there would be a record for each person in this file. This means that the program would not have to be modified each time a person left, was ill, etc., although the individual records might be updated. It is easier to modify data than to modify a program, and it is less likely to produce unexpected results. To be able to vary the action there must be some mechanism in a programming language for getting the data into and out of the program. This is done using input and output statements, sometimes shortened to I/O statements.

4.4 Example 1: Simple Text I/O

Let us now consider a simple program which will read in somebody's first name and print it out:

```
program ch0401
!
! This program reads in and prints out a name
!
  implicit none
  character *20 :: first_name

  print *, ' type in your first name.'
  print *, ' up to 20 characters'
  read *, first_name
  print *, first_name
end program ch0401
```

There are several very important points to be covered here, and they will be taken in turn:

- Each line is a statement.
- There is a sequence to the statements. The statements will be processed in the order that they are presented, so in this example the sequence is print, read, print.
- The first statement names the program. It makes sense to choose a name that conveys something about the purpose of the program.
- The next three lines are comment statements. They are identified by a `!`. Comments are inserted in a program to explain the purpose of the program. They should be regarded as an integral part of all programs. It is essential to get into the habit of inserting comments into your programs straight away.
- The `implicit none` statement means that there has to be explicit typing of each and every data item used in the program. It is good programming practice to include this statement in every program that you write, as it will trap many errors, some often very subtle in their effect. Using an analogy with a play, where there is always a list of the persona involved before the main text of the play we can say that this statement serves the same purpose.
- The `character*20` statement is a type declaration. It was mentioned earlier that there are different kinds of data. There must be some way of telling the programming language that these data are of a certain type, and that therefore certain kinds of operations are allowed and others are banned or just plain stupid! It would not make sense to add a name to a number, e.g., what does Fred + 10 mean? So this statement defines that the variable `first_name` is to be of type character and only character operations are permitted. The concept of a variable is covered in the next section. character variables of this type can hold up to 20 characters.

- The `print` statements print out an informative message to the screen — in this case a guide as to what to type in. The use of informative messages like this throughout your programs is strongly recommended.
- The `read` statement is one of the I/O statements. It is an instruction to read from the terminal or keyboard; whatever is typed in from the keyboard will end up being associated with the variable `first_name`. Input/output statements will be explained in greater detail in later sections.
- The `print` statement is another I/O statement. This statement will print out what is associated with the variable `first_name` and, in this case, what you typed in.
- The `end program` statement terminates this program. It can be thought of as being similar to a full stop in natural language, in that it finishes the program in the same way that a period (.) ends a sentence. Note the use of the name given in the program statement at the start of the program.
- Note also the use of the asterisk in three different contexts.
- Indentation has been used to make the structure of the program easier to determine. Programs have to be read by human beings and we will look at this in more depth later.
- Lastly, when you do run this program, character input will terminate with the first blank character.

The above program illustrates the use of some of the statements in the Fortran language. Let us consider the action of the `read *` statement in more detail — in particular, what is meant by a variable and a value.

4.5 Variables — Name, Type and Value

The idea of a variable is one that you are likely to have met before, probably in a mathematical context. Consider the following:

$$circumference = 2\pi r \tag{4.1}$$

This is an equation for the calculation of the circumference of a circle. The following represents a translation of this into Fortran:

```
circumference = 2 * pi * radius
```

There are a number of things to note about this equation:

- Each of the variables on the right-hand side of the equals sign (`pi` and `radius`) will have a value, which will allow the evaluation of the expression.
- When the expression is fully evaluated the value is assigned to the variable on the left-hand side of the equals sign.
- In mathematics the multiplication is implied. In Fortran we have to use the `*` operator to indicate that we want to multiply 2 by pi by the radius.
- We do not have access to mathematical symbols like π in Fortran but have to use variable names based on letters from the Roman alphabet.

Table 4.1 Variable name, type and value

Variable name	Data type	Value stored
Temperature	Real	28.55
Number_of_people	Integer	100
First_name	Character	Jane

The whole line is an example of an arithmetic assignment statement in Fortran.

The following arithmetic assignment statement illustrates clearly the concepts of name and value, and the difference in the equals sign in mathematics and computing:

$$i = i + 1 \tag{4.2}$$

In Fortran this reads as take the current value of the variable `i` and add one to it, store the new value back into the variable `i`, i.e., `i` takes the value `i+1`. Algebraically, $i = i + 1$ does not make any sense.

Variables can be of different types. Table 4.1 shows some of those available in Fortran.

Note the use of underscores to make the variable names easier to read.

The concept of data type seems a little strange at first, especially as we commonly think of integers and reals as numbers. However, the benefits to be gained from this distinction are considerable. This will become apparent after you have written several programs.

4.6 Example 2: Simple Numeric I/O and Arithmetic

Let us now consider another program, one that reads in three numbers, adds them up and prints out both the total and the average:

```
program ch0402
!
! This program reads in three numbers and sums
! and averages them
!
  implicit none
  real :: n1, n2, n3, average = 0.0, total = 0.0
  integer :: n = 3
  print *, ' type in three numbers.'
  print *, ' Separated by spaces or commas'
  read *, n1, n2, n3
  total = n1 + n2 + n3
  average = total/n
  print *, 'Total of numbers is ', total
  print *, 'Average of the numbers is ', average
end program ch0402
```

Here are some of the key points about this program.

- This program has declarations for numeric variables and Fortran (in common with most programming languages) discriminates between `real` and `integer` data types.
- The variables `average`, `total` and `n` are also given initial values within the type declaration.
  Variables are initially undefined in Fortran, so the variables `n1`, `n2`, `n3` fall into this category, as they have not been given values at the time that they are declared.
- The first `print` statement makes a text message (in this case what is between the apostrophes) appear at the screen. As was noted earlier, it is good practice to put out a message like this so that you have some idea of what you are supposed to type in.
- The `read` statement looks at the input from the keyboard (i.e., what you type) and in this instance associates these values with the three variables. These values can be separated by commas (,), spaces ( ), or even by pressing the carriage return key, i.e., they can appear on separate lines.
- The next statement actually does some data processing. It adds up the values of the three variables (`n1`, `n2`, and `n3`) and assigns the result to the variable `total`. This statement is called an arithmetic assignment statement.
  and is covered more fully in the next chapter.
- The next statement is another data-processing statement. It calculates the average of the numbers entered and assigns the result to average. We could have actually used the value 3 here instead, i.e., written `average = total/3` and have exactly the same effect. This would also have avoided the type declaration for `n`. However, the original example follows established programming practice of declaring all variables and establishing their meaning unambiguously. We will see further examples of this type throughout the book.
- Indentation has been used to make the structure of the program easier to determine.
- The `sum` and `average` are printed out with suitable captions or headings. Do not write programs without putting captions on the results. It is too easy to make mistakes when you do this, or even to forget what each number means.
- Finally we have the end of the program and again we have the use of the name in the program statement.

4.7 Some More Fortran Rules

There are certain things to learn about Fortran which have little immediate meaning and some which have no logical justification at all, other than historical precedence. Why is a cat called a cat? At the end of several chapters there will be a brief summary of these rules or regulations when necessary. Here are a few:

- Source is free format.
- Lower case letters are permitted, but not required to be recognised.
- Multiple statements may appear on one line and are separated by the semicolon character.
- There is an order to the statements in Fortran. Within the context of what you have covered so far, the order is:

  – Program statement.
  – Type declarations, e.g., implicit, integer, real or character.
  – Processing and I/O statements.
  – End program statement.

- Comments may appear anywhere in the program, after program and before end; they are introduced with a ! character, and can be in line.
- Names may be up to 63 characters in length and include the underscore character.
- Lines may be up to 132 characters.
- Up to 39 continuation lines are allowed (using the ampersand (&) as the continuation character).
- The syntax of the read and print statement introduced in these examples is

  – read format, input-item-list.
  – print format, output-item-list.
    where format is * in the examples and called list directed formatting.
    and input-item-list is a list of variable names separated by commas.
    and output-item-list is a list of variable
    names and/or a sequence of characters enclosed in either “or ” , again separated by commas.

- If the implicit none statement is not used, variables that are not explicitly declared will default to real if the first letter of the variable name is A–H or O–Z, and to integer if the first letter of the variable name is I–N.

4.8 Fortran Character Set

Table 4.2 has details of the Fortran character set.

The default character type shall support a character set that includes the Fortran character set. By supplying non-default character types, the processor may support additional character sets. The characters available in the ASCII and ISO 10646 character sets are specified by ISO/IEC 646:1991 (International Reference Version) and ISO/IEC 10646-1:2000 UCS-4, respectively; the characters available in other non default character types are not specified by the standard, except that one character in each non default character type shall be designated as a blank character to be used as a padding character.

Table 4.2 The Fortran character set

Graphic	Name of character	Graphic	Name of character
Alphanumeric characters			
A–Z	Uppercase letters	0–9	Digits
a–z	Lowercase letters	_	Underscore
Special characters			
	Blank	;	Semicolon
=	Equals	!	Exclamation mark
+	Plus	"	Quotation mark
–	Minus	%	Percent
*	Asterisk	&	Ampersand
/	Slash or oblique	~	Tilde
\	Backslash	<	Less than
(	Left parenthesis	>	Greater than
)	Right parenthesis	?	Question mark
[	Left square bracket	'	Apostrophe
]	Right square bracket	`	Grave accent
{	Left curly bracket	^	Circumflex accent
}	Right curly bracket	|	Vertical bar or line
,	Comma	$	Currency symbol
.	Period or decimal point	#	Number sign
:	Colon	@	Commercial at

Table 4.3 has details of the ASCII character set.

If you live and work outside of the USA and UK you may well have problems with your keyboard when programming. There is a very good entry in Wikipedia on keyboards, that is well worth a look at for the curious.

Table 4.3 ASCII character set

Decimal	Character	Decimal	Character	Decimal	Character	Decimal	Character
0	nul	32	&	64	@	96	'
1	soh	33	!	65	A	97	a
2	stx	34	"	66	B	98	b
3	etx	35	#	67	C	99	c
4	eot	36	$	68	D	100	d
5	enq	37	%	69	E	101	e
6	ack	38	&	70	F	102	f
7	bel	39	'	71	G	103	g
8	bs	40	(	72	H	104	h

(continued)

Table 4.3 (continued)

9	ht	41	)	73	I	105	i
10	lf	42	*	74	J	106	j
11	vt	43	+	75	K	107	k
12	ff	44	,	76	L	108	l
13	cr	45	-	77	M	109	m
14	so	46	.	78	N	110	n
15	si	47	/	79	O	111	o
16	dle	48	0	80	P	112	p
17	dc1	49	1	81	Q	113	q
18	dc2	50	2	82	R	114	r
19	dc3	51	3	83	S	115	s
20	dc4	52	4	84	T	116	t
21	nak	53	5	85	U	117	u
22	syn	54	6	86	V	118	v
23	etb	55	7	87	W	119	w
24	can	56	8	88	X	120	x
25	em	57	9	89	Y	121	y
26	sub	58	:	90	Z	122	z
27	esc	59	;	91	[	123	{
28	fs	60	<	92	\	124	|
29	gs	61	=	93	]	125	}
30	rs	62	>	94	^	126	~
31	us	63	?	95	_	127	del

4.9 Good Programming Guidelines

The following are guidelines, and do not form part of the Fortran language definition:

- Use comments to clarify the purpose of both sections of the program and the whole program.
- Choose meaningful names in your programs.
- Use indentation to highlight the structure of the program. Remember that the program has to be read and understood by both humans and a computer.
- Use `implicit none` in all programs you write to minimise errors.
- Do not rely on the rules for explicit typing, as this is a major source of errors in programming.

4.10 Compilers Used

A number of hardware platforms, operating systems and compilers have been used when writing this book and earlier books. The following have been used in the production of this edition of the book:

- NAG Fortran Builder 6.1 and 6.2 for Windows.
- NAG Fortran Compiler 6.1 and 6.2 for Windows.
- NAG Fortran Compiler 6.1 and 6.2 for Linux.
- Intel Fortran 16.x, 17.x, 18.x for Windows.
- Intel Fortran 16.x, 18.x for Linux.
- gnu gfortran 4.8.x, 4.9.x, 4.10.x, 5.4.x, 7.x, 8.0.x for Windows.
- gnu gfortran 4.8.x, 6.3.x for Linux.
- Cray Fortran: Version 8.x.x - Cray Archer service.
- Oracle Solaris Studio 12.6 for Linux.

Our recommendation is that you use at least two compilers in the development of your code. Moving code between compilers and platforms teaches you a lot.

The following were used in the production of the third edition of the book:

- NAG Fortran Builder 6.0 for Windows.
- NAG Fortran compiler 6.0 for Windows.
- NAG Fortran Compiler 6.0 for Linux.
- NAG Fortran Builder 5.3.1 for Windows.
- Nag Fortran compiler 5.3.1 and 5.3.2 for Windows.
- Intel Fortran 14.x, 15.x for Windows.
- Intel Fortran 15.x for Linux.
- gnu gfortran 4.8.x, 4.9.x, 4.10.x for Windows.
- gnu gfortran 4.8.x for Linux.
- Cray Fortran: Version 8.2.1 - Cray Archer service.
- Oracle Solaris Studio 12.4 for Linux.

The following were used in the production of earlier editions.

- NAG Fortran Builder 5.1, 5.2, 5.3 for Windows.
- NAG Fortran Compiler 5.1, 5.2, 5.3 for Linux.
- Intel Fortran 11.x, 12.x, 13.x for Windows.
- Intel Fortran 12.x for Linux.
- gnu gfortran 4.x for Windows.
- gnu gfortran 4.x for Linux.
- Cray Fortran: Version 7.3.1 - Cray Hector service.
- g95 for Linux.
- pgi 10.x - Cray Hector service.
- IBM XL Fortran for AIX, V13.1 (5724-X15), Version: 13.01.0000.0002.
- Oracle Solaris Studio 12.0, 12.1, 12.2 for Linux.

The following have been used with earlier books:

- DEC VAX under VMS and later OPEN VMS with the NAG Fortran 90 compiler.
- DEC Alpha under OPEN VMS using the DEC Fortran 90 compiler.
- Sun Ultra Sparc under Solaris:
  - NAGACE F90 compiler.
  - NAGWare F95 compiler.
  - Sun (Release 1.x) F90 compiler.
  - Sun (Release 2.x) F90 compiler.
- PCs under DOS and Windows:
  - DEC/Compaq Fortran 90 and Fortran 95 compilers.
  - Intel Compiler (7.x, 8.x).
  - Lahey Fujitsu Fortran 95 (5.7).
  - NAG Fortran 95 Compiler.
  - NAG Salford Fortran 90 Compiler.
  - Salford Fortran 95 Compiler.
- PCs under Linux:
  - Intel Compiler.
  - Lahey Fujitsu Fortran 95 Pro (6.1).
  - NAG Fortran 95 (4.x, 5.x).

It is very illuminating to use more than one compiler whilst developing programs.

4.11 Compiler Documentation

The compiler may come with documentation. Here are some details for a number of compilers.

4.11.1 gfortran

Manuals are available at

```
http://gcc.gnu.org/wiki/GFortran\#manuals
```

The following

```
http://gcc.gnu.org/onlinedocs/
gcc-4.5.2/gfortran.pdf
```

is a 236 page pdf.

4.11.2 IBM

Here is a starting point. The urls have been split as the lines are too long.

```
http://www-03.ibm.com/software/
products/en/fortcompfami/
```

Here is a starting point for the XLF for AIX system.

```
http://www-01.ibm.com/support/
docview.wss?uid=swg27036673
```

and the starting point for the pdf version of the documentation is.

```
http://www-01.ibm.com/support/
docview.wss?uid=swg27036673
```

They provide

- Getting Started with XL Fortran for AIX 15.1 This book introduces you to XL Fortran for Linux and its features, including features new for 15.1.
- Installation Guide - XL Fortran for AIX 15.1 This book contains information for installing XL Fortran and configuring your environment for basic compilation and program execution.
- Compiler Reference - XL Fortran for AIX 15.1 This book contains information about the many XL Fortran compiler options and environment variables that you can use to tailor the XL Fortran compiler to your application development needs.
- Language Reference - XL Fortran for AIX 15.1 This book contains information about the Fortran programming language as supported by IBM, including language extensions for portability and conformance to non-proprietary standards, compiler directives and intrinsic procedures.
- Optimization and Programming Guide - XL Fortran for AIX 15.1 This book contains information on advanced programming topics, such as application porting, inter language calls, floating-point operations, input/output, application optimization and parallelization, and the XL Fortran high-performance libraries.

4.11.3 Intel

Windows. The following will end up available after a complete install.

- Intel MKL
  - Release notes

  - Reference Manual
  - User Guide

- Parallel Debugger Extension

  - Release Notes

- Compiler

  - Reference Manual, Visual Studio Help files or html.
  - User Guide, Visual Studio Help files or html.

  Intel also provide the following

```
https://software.intel.com/en-us/articles/
intel-software-technical-documentation/
```

4.11.4 Nag

Windows

- Fortran Builder Help

  - Fortran Builder Tutorial - 44 pages
  - Fortran Builder Operation Guide - 67 pages
  - Fortran Language Guide - 115 pages
  - Compiler Manual - 149 pages
  - LAPACK Guide - 70 pages (440 MB as PDF!)
  - GTK+ Library - 201 pages
  - OpenGL/GLUT Library - 38 pages
  - SIMDEM Library - 78 pages

4.11.5 Oracle/Sun

Oracle make available a range of documentation. From within Oracle Solaris Studio

- Help

  - Help Contents
  - Online Docs and Support
  - ..
  - ..
  - Quick Start Guide

and you will get taken to the Oracle site by some of these entries.

You can also download a 300+ MB zip file which contains loads of Oracle documentation. You should be able to locate (after some rummaging around)

- Sun Studio 12: Fortran Programming Guide - 174 pages
- Sun Studio 12: Fortran User's Guide - 216 pages
- Sun Studio 12: Fortran Library Reference - 144 pages
- Fortran 95 Interval Arithmetic Programming Reference - 166 pages

Happy reading :-)

4.12 Program Development

A number of ways of developing programs have been used, including:

- Using an integrated development environment, including
  - NAG Fortran Builder under Windows.
  - Microsoft Visual Studio with the Intel compiler under Windows.
  - Oracle Sunstudio under SuSe Linux.
- Using a DOS box and simple command line prompt under Windows.
- Using ssh to log in to the Archer service.
- Using a VPN, and SSH to log in to the IBM Power 7 system at Slovak Hydrometeorological Institute Jeseniova 17.
- Using a console or terminal window under SuSe Linux.
- Using X-Windows software to log into the SUN Ultra Sparc systems.
- Using terminal emulation software to log into the SUN Ultra Sparc.
- Using DEC terminals to log into the DEC VAX and DEC Alpha systems.
- Using PCs running terminal emulation software to log into the DEC VAX and DEC Alpha systems.

It is likely that you will end up doing at least one of the above and probably more. The key stages involved are:

- Creating and making changes to the Fortran program source.
- Saving the file.
- Compiling the program:
- If there are errors you must go back to the Fortran source and make the changes indicated by the compiler error messages.
- Linking if successful to generate an executable:
- Automatic link. This happens behind the scenes and the executable is generated for you immediately.
- Manual link. You explicitly invoke the linker to generate the executable.
- Running the program.

- Determining whether the program actually works and gives the results expected.

These steps must be taken regardless of the hardware platform, operating system and compiler you use. Some people like working at the operating system prompt (e.g., DOS, Linux and UNIX), and others prefer working within a development environment. Both have their strengths and weaknesses.

4.13 Problems

4.1 Compile and run Example 1 in this chapter. Experiment with the following types of input.

Ian
Ian Chivers
"Jane Margaret Sleightholme"

4.2 Compile and run Example 2 in this chapter.
Think about the following points:

- Is there a difference between separating the input by spaces or commas?
- Do you need the decimal point?
- What happens when you type in too many data?
- What happens when you type in too few data?

If you have access to more than one compiler repeat the above and compare the results.

4.3 Write a program that will read in your name and address and print them out in reverse order.
Think about the following points:

- How many lines are there in your name and address?
- What is the maximum number of characters in the longest line in your name and address?
- What happens at the first blank character of each input line?
- Which characters can be used in Fortran to enclose each line of text typed in and hence not stop at the first blank character?
- If you use one of the two special characters to enclose text what happens if you start on one line and then press the return key before terminating the text?

The action here will vary between Fortran implementations.