

1 The Basics

1.1 Aims

By the end of this worksheet, you will be able to:

- Create and run a FORTRAN 95 program
- Understand basic program structure
- Start to deal with programming errors
- Start to understand **real**, **integer** and **character** variable types.
- Save a copy of your output in Word.

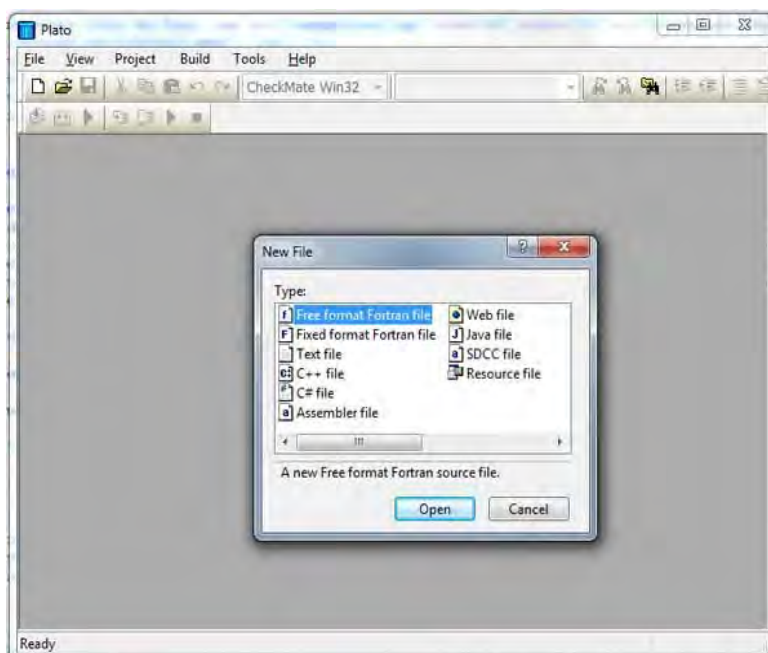
1.2 Install FTN95 Personal Edition

- Search for Silverfrost FTN5 personal edition or click this link http://www.silverfrost.com/32/ftn95/ftn95_personal_edition.aspx.
- Download and install the software accepting all the defaults.

1.3 Your first programming session

- Locate and double click the Plato  icon
- Click **File, New**
- Select **Free Format Fortran File**
- Click **File, Save As**
- Create a directory called fortranprograms and open it
- Type **first.f95**

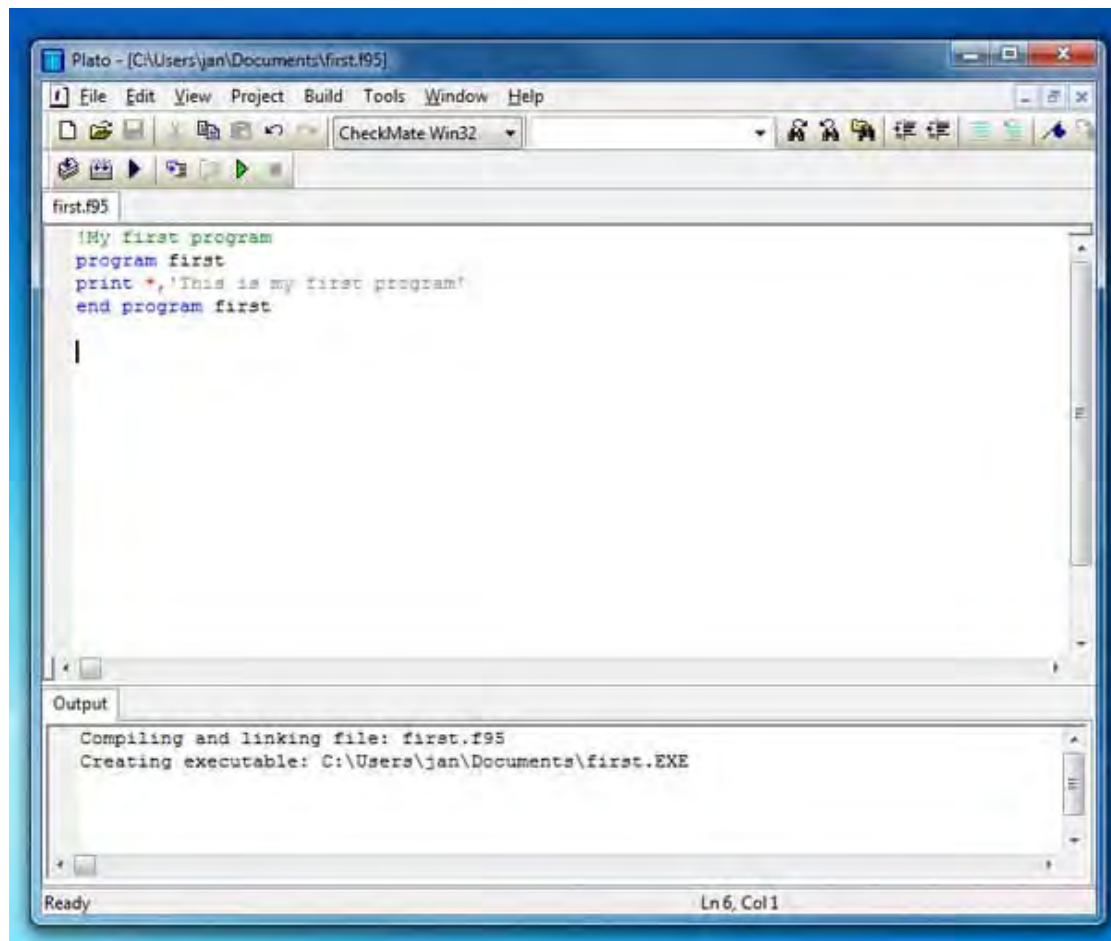
1.4 Plato - a programming environment



Plato is a "programming environment". Within Plato, you can create and edit programs and get them to run. Plato's editor is special – it understands the syntax of various programming languages. We tell Plato which language we are using when we create our empty file and save it with a **.f95** (FORTRAN 95) extension. Provided you have given your file the appropriate extension, Plato's editor will be able to check the syntax of the program, highlighting the various keywords that it knows about using a colour code to distinguish between the various elements of the language.

Always ensure that your program files have a .f95 extension

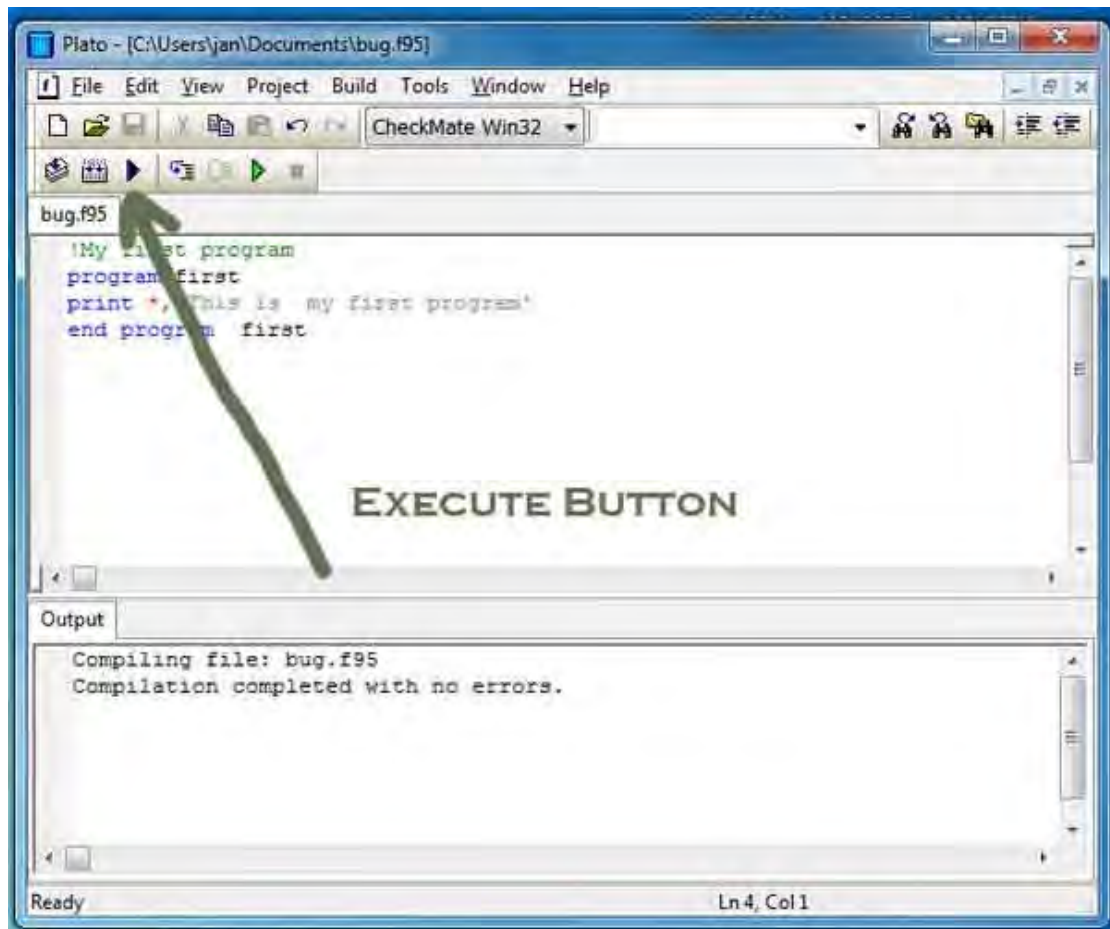
1.5 Running your first FORTRAN 95 Program



Exercise 1.1

- Type in the following *exactly* as shown:

```
!My first program
program first
print *, 'This is my first program'
end program first
```



- Click the **black ►**, (the **Execute** button).
- Plato will get FTN95 to check your program for errors. If it finds any problems, it will give you the details. If you have typed in the program *exactly* as shown above, an executable file will be generated (first.exe). Plato will then automatically get the program to start executing.
- A banner will appear for a couple of seconds and will then disappear (that's the price we have to pay for using the free software)
- A black console window will appear.
- Press **Return** to close the window. **Do not** click the X at the top right of the window.

Plato can get upset if you do not press **Return** to close the window, try this...

- Save your program first!
- Run the program again (click ►)
- This time click the X at the top right of the window to close it.
- Make up your own mind about which is the better way to close this window in future!

1.6 Program Structure

Examine the following short program:

program sum	!a: name of program
!an example of program structure	!b: a comment
real :: answer,x,y	!c: declarations
print *, 'Enter two numbers'	!d: output
read *, x	!e: input
read *, y	!e: input
answer=x+y	!f: arithmetic
print *, 'The total is ', answer	!g: output
end program sum	!h: end of program

There are a number of general points here:

- The program is made up of a number of lines. Each line is called a **statement**.
- Each **statement** is made up of
 - **variable** names e.g. **answer, x, y**
 - **operators** e.g. **+, -** etc
 - **keywords** e.g. **read, print**
- The **statements** are executed sequentially.

Let's break the program down, line by line:

- a) The name of the program. Keep it reasonably short and meaningful.
- b) A comment explaining the purpose of the program. Comments are indicated by an exclamation mark. All text to the right of an exclamation mark is ignored by the compiler. Programmers use comments to help them remember how a program works. **Use of appropriate comments in programs aids understanding and is good practice.**
- c) **Variables** - **answer, x** and **y** are used to store floating point numbers – we indicate this by **declaring** them as **real**.
- d) **print ***, outputs to the screen – the asterisk means use the default number of decimal places when the number is written to the screen.
- e) We **read** information from the keyboard and store the values in **x** and **y**.
- f) Do some arithmetic and store the answer in **answer**.
- g) Output the result to the screen
- h) Conclude the program

1.7 More on Input and Output

Exercise 1.2

- Open a new file and call it io.f95.
- Type in the following program:

```
program io
real :: x,y,z
print *, 'enter the values x,y and z'
read *, x,y,z
print *, 'the values you typed are for z,y,x are: ',z,y,x
end program io
```
- Execute it by pressing ►
- You can enter the numbers one at a time and press the **Enter** key each time.
- Execute the program again
- This time type all three numbers on one line separated by commas.

Look at the **print** statement

```
print *, 'the values you typed are for z,y,x are: ',z,y,x
```

In this statement, we are outputting four separate things, a literal string of characters,

```
'the values you typed are for z,y,x are: '
```

and the **variables** z, y, and x. We may output several items at one time, provided they are separated by commas.

Exercise 1.3

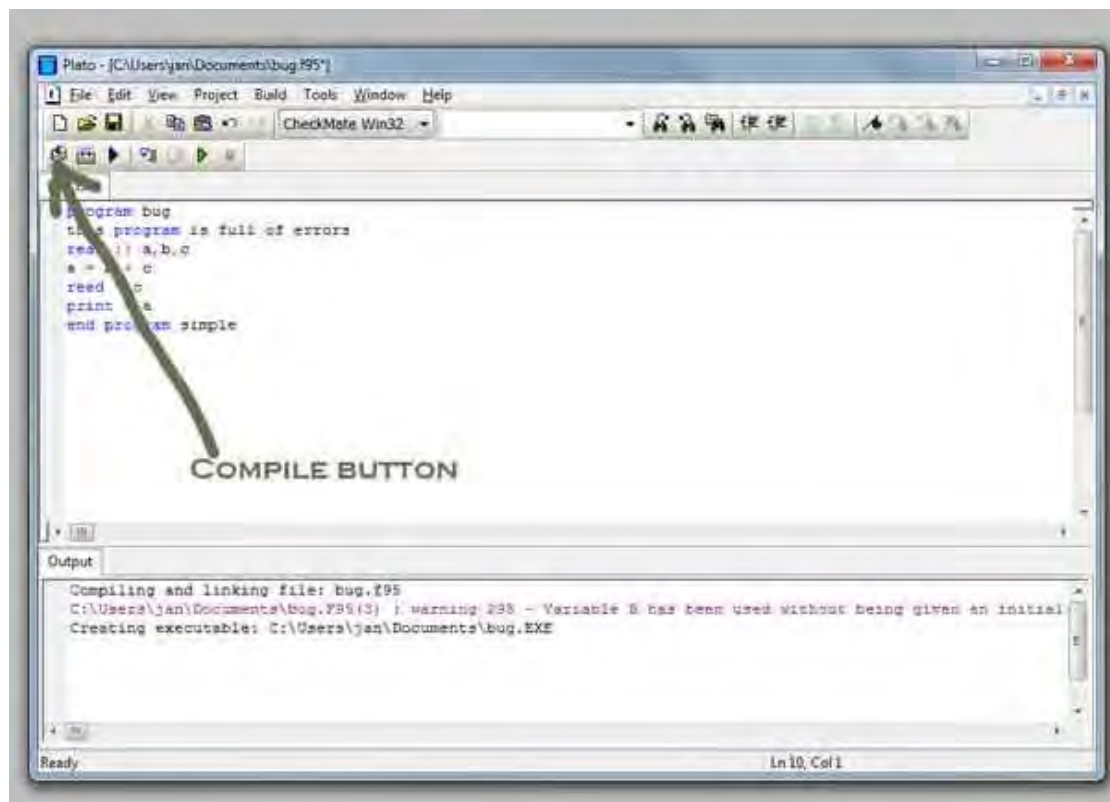
The following program has a number of errors.


- ❑ Create a new file called **bug.f95** and then type in the following program exactly as shown.
- ❑ You can also download this file from <http://fortrantutorial.com/fortrantutorial-example-programs/index.php>

```
program bug
this program is full of errors
real :: a,b,c
a = b + c
read *,c
print *,a
end program simple
```

The compiler will report two error messages when it attempts to compile. Click on the **details** button. Each error generates a message.

**Double clicking on the message will
take you to the line in the program where the fault occurs.**



- ❑ Correct the two errors.
- ❑ Click Execute 
- ❑ There is now one further error, Plato will provide a yellow warning alert. **Watch the screen carefully! The window will close and then the program will start to execute. Something is not correct however... the program will "hang".** It is actually waiting for you to input a value, because of the line `read *,c`. To the user of the program, this is not at all obvious – they may have thought that the program has crashed!
- ❑ Type in a number then press **enter**
- ❑ The program returns an strange value. This is an "**execution time**" error.
- ❑ We need to find out what the warning message was. Click the "compile" button (to the right of the binoculars). Then click the "details" button. Plato will tell you that the variable **b** has not been given a value.
- ❑ Correct the program to give **b** a value, and then execute the program again.
- ❑ There is **still** a problem. This time, it is a problem with the program's **logic**.

Need a Hint? The program statements are executed **sequentially**.

```
a=b+c
read *, c
print *, a
```

The statement `a=b+c` doesn't make sense, as at this stage of the program, we haven't yet given a value to `c`.

Important points to note

- ❑ There are two types of errors associated with this program: **compiler** errors and **run-time** errors.
- ❑ The program is also **user-unfriendly**. The program waits for input without telling the user what is needed.

Fix the run time error by:

- read in a value for `b`
- correct the order of the statements
- make the program more user-friendly,

then compare your program with the one called **bugfixed.f95** at

<http://fortrantutorial.com/fortrantutorial-example-programs/>

1.8 More Data types – integer and character

So far, we have only used **real** (floating point numbers) in our programs. We can also specify that numbers are **integer** and **character**. Program **convert**, below, demonstrates their use.

Within a given range, **integers** are always represented *exactly* whereas the precision of real numbers is limited by the architecture of the machine. The **real** variable type gives us 6 figure decimal precision. (If this doesn't seem enough – don't worry we'll come back later on when we examine how to increase the number of digits of precision in Section 4).

Character variables hold strings of characters like

```
'A happy day was had by all'
'Yes'
'N'
'3 + 4 equals 7'
```

When the **character variable** is declared, we show the maximum length that the string can occupy by following the name by a * then its maximum length. The example below has a maximum length of 10 characters allowed for a person's name – this might not always be enough! You have to make a judgement here.

```
program convert
!This example shows the use of integer and character variables.
  implicit none
  integer    :: pounds,pence,total
  character  :: name*10
  print *, 'What is your name?'
  read *, name
  print *, 'Hi ', name, '! Enter number of pounds and pence'
  read *, pounds,pence
  total =100 * pounds + pence
  print *, 'the total money in pence is ',total
end program convert
```

NOTE the inclusion of the line

implicit none

By including it in your program, FORTRAN will check that you have properly declared all your variable types. In the bad old days of programming, declaration of variables was thought to be unnecessary and the old FORTRAN compilers used an implicit convention that integers have names starting with the letters in the range i – n, all the others being real. FORTRAN still allows you to do this if we don't include the line, **implicit none**. Time has shown that **one of the commonest reasons for error in a program** is the incorrect use of variables.

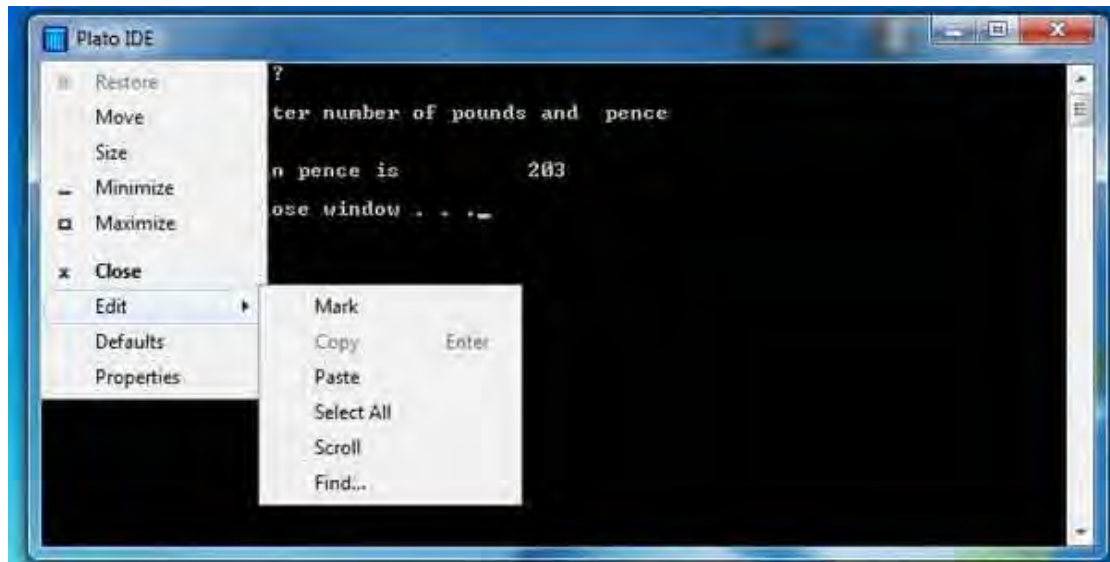
Always use *implicit none* at the start of every program.

Exercise 1.4

With the program **convert** in section 1.5 as a guide, write a program to test out everything you've learned so far. You might include different types of variables, for example **real**, **integer**, and **character**. Include input and output using **read** and **print**. An example might be a program that asks people questions, including things like their age and name and so on. It could, for example, print out their year of birth with a suitable message. It's up to you, just use your imagination.

1.9 Saving the contents of Output Window

Run your last program again. When the black output window opens right click on the Plato icon in the top left corner



- Click on edit
- Click Select all
- Click copy
- Open a new document in Word or Notepad and click paste.