



Introduction to Fortran and UNIX

Torunn Lilleeng

• INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

EXAMPLE-
STEP BY STEP

What about FORTRAN

You may think that using a programming language developed in the early fifties is a waste of time, when considering how quickly things evolve in the computing business.

But; Fortran is the dominant programming language used in engineering applications. It is therefore important for engineering graduates to be able to read and modify the Fortran code.

From time to time so-called experts predict that Fortran will rapidly fade in popularity, and soon become extinct. These predictions have always failed.

Fortran is the most enduring computer programming language in history.

Fortran is a general purpose programming language, mainly intended for mathematical computations in e.g. engineering.

Fortran is an acronym for **FOR**mula **TRAN**slation.

Fortran was the first ever high-level programming languages.

The work on Fortran started in the 1950's at IBM, and there has been made many versions since.

We will be using Fortran 77.

Last semester I assume you learned programming in JAVA. The greatest difference between Fortran and JAVA is that JAVA is object related, whilst Fortran is linear related. But still there are similarities between the two languages, so learning JAVA was not a waste of time ;) Neither was JSP.

- **INTRODUCTION**

- UNIX COMMANDS**

- HOW TO GET STARTED?**

- FORTRAN STRUCTURE**

- DECLARATIONS**

- OPENING FILES**

- READING DATA**

- IF STATEMENT**

- ARRAY**

- DO LOOP**

- FORMAT**

- SUBPROGRAMS**

- NAG-ROUTINE**

- COMPILING RUN AND LINKING**

- DEBUGING**

- EXAMPLE-STEP BY STEP**

Can we compare Fortran with JAVA?

Sure we can. Lets take a look....

On the left hand we have the program "Equation" in Fortran, on the right hand we have the same program in JAVA.

```
PROGRAM EQUATION

REAL SOL1,SOL2,D
INTEGER A,B,C
A=2
B=8
C=2
D=B**2-4*A*C

IF(D .LT. 0)THEN
WRITE(*,*)'COMPLEX SOLUTION'

ELSE
      SOL1=(-B+SQRT(D))/(2*A)
      SOL2=(-B-SQRT(D))/(2*A)

      WRITE(*,*)SOL1
      WRITE(*,*)SOL2

END IF

END
```

```
public class Equation{
    public static void main(final String[]args){
        double sol1;
        double sol2;
        int A=2;
        int B=8;
        int C=2;
        double D=B*B-4*A*C;

        if(D<0){
            System.out.println("Complex
                solution");
        }
        else{
            sol1=(-B*Math.sqrt(D))/(2*A);
            sol2=(-B*Math.sqrt(D))/(2*A);
            System.out.println(sol1);
            System.out.println(sol2);
        }
    }
}
```

• INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGGING

EXAMPLE-STEP BY STEP

Introduction to UNIX

During the exercises you'll have to manage both UNIX and Fortran.

Unix is a very powerful and stable operative system, it is fit to run heavy applications.



Operating system

Every computer requires an operating system

An operating system is the program that controls all the other parts of a computer system, both the hardware and the software. It allocates the computer's resources and schedules tasks.

UNIX is a multi-user, multitasking operating system. Multiple users can have multiple tasks running simultaneously. This is very different from PC operating systems. (e.g. Windows)

LINUX

Most of the common Unix tools and programs have been ported to Linux, including almost all GNU software and many X clients from various sources. So if you know Linux, then Unix will be easy to learn.

INTRODUCTION• UNIX COMMANDSHOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP**UNIX Commands.**

Next slide will explain how to get started with UNIX, here are some of the commands that will be useful to know

emacs	- open file
cd	- <u>c</u> hange <u>d</u> irectory
pwd	- displays <u>p</u> resent <u>w</u> orking <u>d</u> irectory
ls	- <u>l</u> ist contents of directory
mkdir	- <u>m</u> ake <u>d</u> irectory
rmdir	- <u>r</u> emove (empty) <u>d</u> irectory
rm	- <u>r</u> emove files
cp	- <u>c</u> opy files
mv	- <u>m</u> ove or rename files
cat	- display file
man	- help on a command
clear	- clear screen
xlf-o prog filename.f	- compiles and links the program
prog	- run the program
xlf -g -o prog exercise1.f	-compilation for the debugger
idebug prog	-start debugger

Examples

mkdir exercise.f	- creates a new directory with name exercise.f
emacs exercise.f	- opens the Fortran-file with name exercise.f
cd exercise.f	-enters the directory of exercise.f
cd ..	-changes directory to the directory above in the hierarchy
cp../exercise.f .	-copies the file exercise.f from the directory above, to the one you are in.
mv exercise.f exercise2.f	-changes file name form exercise.f to exercise2.f
enscript exercise2.f	-prints the file exercise2.f on the standard printer
xlf-o prog exercise2.f	-compiles and links exercise2.f

INTRODUCTIONUNIX COMMANDS• HOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

How to get started?

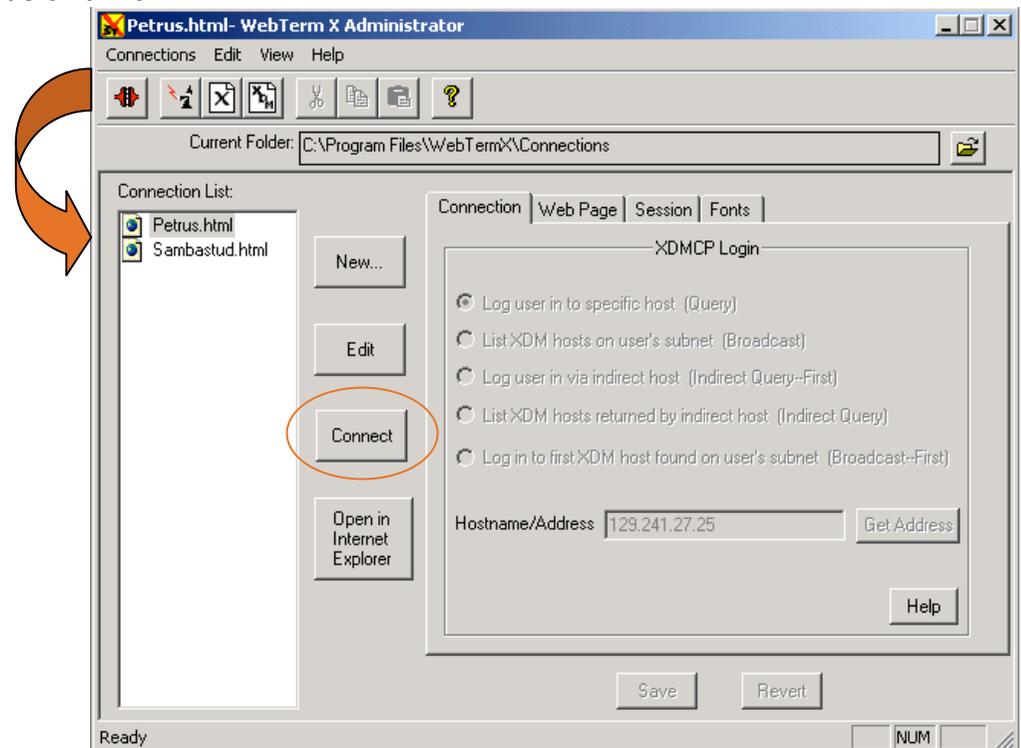
By entering the start menu you'll find a program with the name **WebTerm X Administrator**.

When you open this program your screen will look like the illustration.

To the left you can see the **Connection List**. We will use the server with the name **Petrus**, so highlight Petrus and hit **Connect**.

Log on as you would in Windows, the **login name** and the **password** are the same. Then choose; **Go!**

In some occasions you might need the **server name**, `petra1.petrus.unit.no`, but usually there is no need for applying that.



INTRODUCTIONUNIX COMMANDS• HOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

How to get started?

Now I assume you have entered the program. In the upper left corner you will find an icon named **xterm**. Enter this icon and you'll be in the terminal window. The window will show a line of command `bash-2.05a$`, this means you are in your home directory.

Since UNIX has an hierarchic directory system it might be useful to create the system at once....

Making the catalogue structure:

- bash-2.05a\$ **mkdir** Fag
- bash-2.05a\$ **cd** Fag
- bash-2.05a\$ **mkdir** AnvendtData
- bash-2.05a\$ **cd** AnvendtData
- bash-2.05a\$ **mkdir** Exercise1
- bash-2.05a\$ **mkdir** Exercise2
-
-
- bash-2.05a\$ **mkdir** ExerciseN



INTRODUCTIONUNIX COMMANDS• HOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

If you have followed this recipe your catalogue structure should be as illustrated. To make sure that you have a correct structure, type,

•bash-2.05a\$pwd

which explains that you are now in the directory AnvendtData, a sub directory of Fag etc.
/home/petra1b/<students name>/Fag/AnvendtData

To make sure that you have made the directories for exercises, which are sub directories of AnvendtData, type;

•bash-2.05a\$ ls

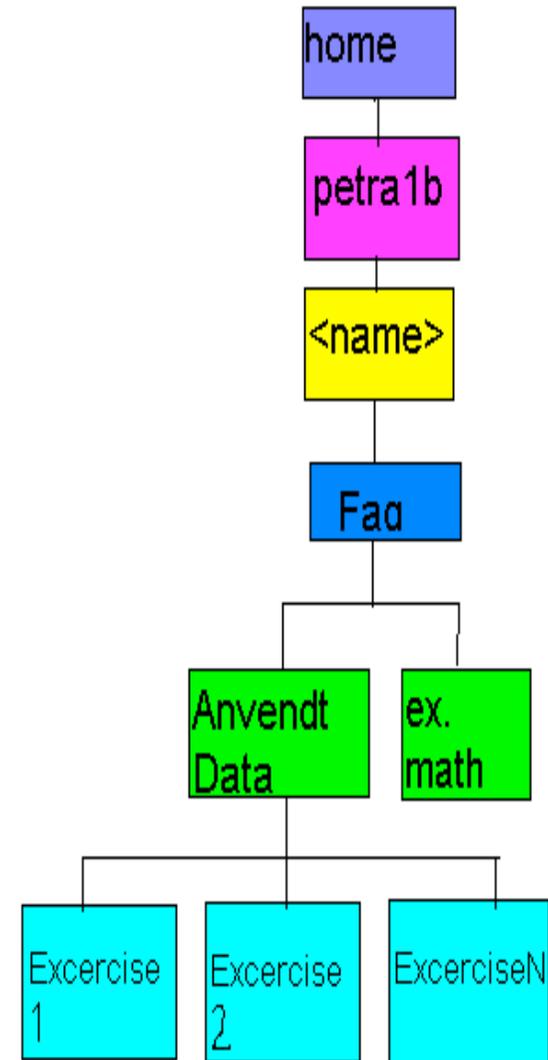
and the different exercises that you have created will show like this:

Exercies1 Exercies2

Go into your exercise directory with **cd** Exercise1, then open the directory with **emacs** Exercise1 to start programming.

The next time you enter the program you don't have to make directories, instead you can use the **cd** and **emacs** commands to enter the directory in the order of the catalogue structure:

Since you already have entered home, petra1b and <student name>, you start by changing directory form <student name> to Fag.



INTRODUCTIONUNIX COMMANDS• HOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

- bash-2.05a\$ **cd** Fag
- bash-2.05a\$ **cd** AnvendtData
- bash-2.05a\$ **emacs** ExerciseN

In the emacs editor you may use the toolbar like in Windows, or you can use the ctrl-commands you learned it in JAVA. If you take a look at the toolbar, you'll see that the ctrl-commands are defined for every option. The first time you save a file you must save it as Fortran file. You can either hit file in the tool bar, choose **Save Buffer as** and write the name of the file like this name.f on the bottom of the page(the marker will be there). To enter the file after closing it, you'll have to enter the directory above and then type **emacs name.f**

**Useful to know**

When using UNIX, things will go much faster if you learn, and use, the following:
To switch between previously used commands, use the **up** and **down** arrows

If you, for example, are changing to a directory named documents, type cd do, and hit TAB-then UNIX will fill out the rest of the name itself. If you have several directories that begins with do, UNIX lists all of them if you hit TAB one more time.

Do not include special symbols or characters in file-and directory names. Such can be #, å, !, @.

Do not type file- and directory names with a space (instead of test file, use testfile or test_file)

INTRODUCTIONUNIX COMMANDSHOW TO GET STARTED?• FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

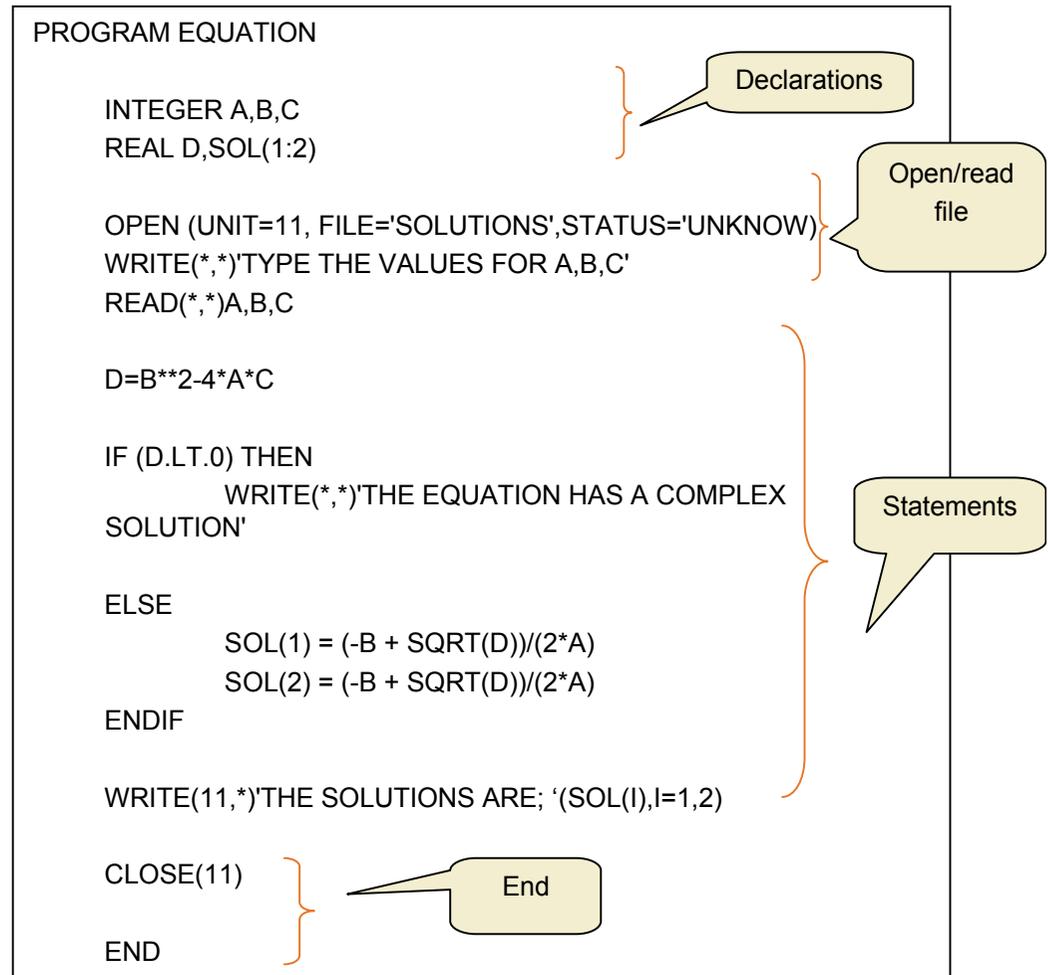
Fortran

Now that you have entered the program, and made the directory structure, you can start programming.

Fortran usually has a **structure** like this;

- Program name
- Declarations
- Opening/reading files
- Statements
- End

As I explain the structure, piece by piece, I will build up a complete program sequence.



INTRODUCTIONUNIX COMMANDSHOW TO GET STARTED?● FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP**Arithmetic operators**

The **arithmetic operators** are:

+	addition
-	Subtraction
/	division
*	Multiplication
**	exponentiation

The priority rule is: ** has highest priority followed by / and * followed by + and -

Within any priority level, evaluation is carried out from left to right

In general, any expression enclosed in parentheses is evaluated first.

The use of parenthesis is highly recommended

Typical mathematical formula:

$$D = B^2 - 4AC$$

Equivalent FORTRAN statement:

$$D = B**2 - 4*A*C$$

Fortran statement

Fortran77 programs are typed in lines of up to 72 characters, with the first six columns of each line reserved for special purposes.

As a result, Fortran77 statements always begin before, or after, **COLUMN 7**. In emacs, the TAB key will bring you to column 7.

If the **first column** contains "c" or a "**", the entire line would be treated as a comment.

A line can only contain 72 characters, but you will often be in need of more space.

A "*" in the **6th column** specifies that this line is a continuation of the previous.

Both Fortran and UNIX are very sensible about spellings. If you name the directory AnvendtData, then you have to type the name exactly like that when you want to enter the directory. If you type anvendtdata without the capital letters, the program will not find the directory.

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

• DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGING

EXAMPLE-STEP BY STEP

Declarations

A Fortran program always start with declaring the variables you will be using. A variable consists of 1-6 characters chosen from the letters a-z and the digits 0-9.

List of Fortran data types:

- Integer
- Real
- Character
- Logic
- Complex

INTEGER

An **Integer** data type is an exact number. Often used as a numerator in loops.

Example:

INTEGER MONTHS

Declares the variable MONTHS as an Integer. In this example we wish to use the variable MONTHS as an exact number eg. 1, 5 or 8 months, not 1.5 or 6.4 months

REAL (known as double in JAVA)

The **real** data type stores numbers using a floating-point representation. It handles numbers with a fractional part as well as round numbers.

Example:

REAL SALARY, OVTIME

Declares the variables SALARY and OVTIME as floating-point numbers. These may be of the form: 1.75E+5, 12345.678, 2000.0 If you need double precision, you should declare the variable as;

REAL*8 (REAL is by default REAL*4)

PROGRAM EQUATION

```
INTEGER A,B,C  
REAL D  
REAL SOL (1:2)
```

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

• DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGING

EXAMPLE-STEP BY STEP

CHARACTER

The **character** data type stores a character or a text string. If you wish to have input from the user of a program, for e.g. if he wishes to continue or not, you may use the data type character.

Example:

CHARACTER ANSWER, INPUT*5

Declares the variables ANSWER and INPUT. ANSWER may contain one or many characters, while INPUT has a maximum of five characters. If both variables declared as CHARACTER should have a maximum of five letters, you simply write:

CHARACTER*5 ANSWER, INPUT

COMPLEX

The **complex** data type stores two real values as a single entity. Complex numbers arise naturally when extracting the roots of negative numbers, and they are used in many branches of mathematics, physics, and engineering.

A complex number is often represented as $(A + iB)$, where A and B are the real and imaginary parts respectively, and $i^2 = -1$.

LOGIC (works like boolean in JAVA)

The **logical** data type is mainly used in conjunction with IF statements which select a course of action according to whether some condition is true or false. A logical variable (or array element) may be used to store such a condition value for future use.

Logical variables and arrays are also useful when dealing with two-valued data such as whether a person is male or female, a file open or closed, power on or off, etc.

PARAMETER

The **parameter** statement is used to assign names to constants. A parameter will not be changed during the programming.

PARAMETER (name1=expression, name2=expression,...)

Example:

PARAMETER (PI = 3,1415, N=20)

Declares the variable PI to the value 3,1415 and N to value 20.

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONS● OPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGGINGEXAMPLE-
STEP BY STEP**Opening Files**

Very often you need a large amount of input data and you wish to use your output data to make a chart or a graph etc. It is then very useful to have external output and input files. Syntax used can be viewed in example under. It is customary to open the output file at the same time as you open the input file, even though you may not need it in the beginning.

OPEN (UNIT=<integer expression>, FILE="<filename>",STATUS="literal")

UNIT assigns a unique number to this file which will have to be used every time this file is referred to.

FILE is a character string denoting the file.

STATUS is by default "NEW", "OLD" or "UNKNOWN".

For an output file, STATUS should be "NEW" or "UNKNOWN" (because it has not been created yet), whereas an input file should either read "OLD" or "UNKNOWN" (because this file must be created before you run the program).

```
PROGRAM EQUATION
```

```
INTEGER A,B,C
```

```
REAL D
```

```
REAL SOL(1:2)
```

```
OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')
```

```
OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
```

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

DECLARATIONS

OPENING FILES

• READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGING

EXAMPLE-STEP BY STEP

Reading data

As in JAVA we have the opportunity to operate with different files, but we'll concentrate on using one file for the programming part, a second file which will contain input data and a third file which will contain results from the program.

Two ways of reading input data:

- From **file**
- From **screen**



From file

Reading input data from file:
First, input file must be open. Then, the command is:

READ(<unit number>, <format>)

list of variables. Unit number is the same one that was assigned to the file when opening. Format is also discussed later in his module, but it is common to use *, which means free format.

After the bracket, the variables listed in the input file should be listed, in the proper order.

From screen

Read data from screen

In order to read data from screen, you must first make the program ask for input data. Such a command may be:

PRINT(*,*) '<request for input>'

The PRINT command prints to screen, * is still free format followed by a statement

When this has been done, the program must read the input

READ (*,*)list of variables

The READ command allocates the input data to the pre-declared **variable**.

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

● IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGGING

EXAMPLE-STEP BY STEP

PROGRAM EQUATION

```
INTEGER A,B,C  
REAL D  
REAL SOL(1:2)
```

```
OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')  
OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
```

```
WRITE(*,*)'TYPE AN INTEGER FOR EACH OF THE VALUES A AND B'
```

```
READ(*,*)A, B  
READ(10,*)C
```

The IF statement

The **conditional statements** are an important part of any programming language. The most common statement in Fortran is the IF statement, which has several forms.

The simplest one is the logical IF statement:

IF (logical expression) executable statement

It says that if something is true, do something. If you wish to include several statements, the general form is:

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

● IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

EXAMPLE-
STEP BY STEP

IF (logical expression) **THEN**
statements

ELSEIF (logical expression) **THEN**
statements

:

ELSE statements

ENDIF

EXAMPLE OF IF STATEMENT

```
PRINT *, 'TYPE IN THE TIME, IN MILITARY TIME E.G. 1000:'  
READ *, TIME
```

```
IF (TIME .LT. 1130) THEN  
PRINT *, 'NOT LUNCH YET'
```

```
ELSE IF (TIME .EQ. 1130) THEN  
PRINT *, 'GO AND HAVE LUNCH'
```

```
ELSE IF (TIME .GT. 1200) THEN  
PRINT *, 'SORRY, YOUR  
LUNCH HOUR HAS PASSED'
```

```
ENDIF
```

This is a simple example that says if it's not 11.30 it is not lunch, if it is 11.30 you can have lunch and if it's more than 12.00 it is too late.

This is not a complete program, declarations have not been made

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

● IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

EXAMPLE-
STEP BY STEP

PROGRAM EQUATION

```
INTEGER A,B,C  
REAL D  
REAL SOL(1:2)
```

```
OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')  
OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
```

```
WRITE(*,*)'TYPE AN INTEGER FOR EACH OF THE VALUES A AND B'
```

```
DO 15 I=1,5  
READ(*,*)A, B  
READ(10,*)C  
D=B**2-4*A*C
```

```
IF (D.LT.0) THEN  
WRITE(*,*)'THE EQUATION HAS A COMPLEX SOLUTION'
```

```
ELSE
```

```
SOL(1) = (-B + SQRT(D))/(2*A)  
SOL(2) = (-B - SQRT(D))/(2*A)
```

```
WRITE(11,*)'THE SOLUTIONS ARE; '(SOL(I),I=1,2)
```

```
END IF  
15 CONTINUE
```

[INTRODUCTION](#)[UNIX COMMANDS](#)[HOW TO GET
STARTED?](#)[FORTRAN
STRUCTURE](#)[DECLARATIONS](#)[OPENING FILES](#)[READING DATA](#)[● IF STATEMENT](#)[ARRAY](#)[DO LOOP](#)[FORMAT](#)[SUBPROGRAMS](#)[NAG-ROUTINE](#)[COMPILING RUN
AND LINKING](#)[DEBUGGING](#)[EXAMPLE-
STEP BY STEP](#)

Logical Expression

Fortran Statement	Meaning	Analogue Symbol
.GT.	Greater Than	>
.LT.	Less Than	<
.GE.	Greater or Equal	≥
.LE.	Less or Equal	≤
.EQ.	Equal	=
.NE.	Not Equal	≠

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

● ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGGING

EXAMPLE-
STEP BY STEP

Array

An array is a group of storage locations that have the same name. Individual members of an array are called elements, and their distinguishing feature is the common name followed by a subscript or an index in parentheses.

REAL POPULATION (2000:2004)

Whose elements are;
POPULATION(2000), POPULATION(2001), POPULATION(2002),
POPULATION(2003), POPULATION(2004)

If you want to refer to the population for 2003, the reference is
POPULATION(2003)

Values are assigned to array elements in the same way that the values are assigned to regular variables.

Example;

POPULATION(2000)= 1500
POPULATION(2001)=POPULATION(2000)*1.2

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENT● ARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGINGEXAMPLE-
STEP BY STEP

Array

It is also helpful to use variables and expressions as subscripts. Take a look at this;

```
DO 15 I=1500,1505
  POPULATION(I)=I
15 CONTINUE
```

This will give an array like this;

150	151	152	153	154	155
POP (150)	POP (151)	POP (152)	POP (153)	POP (154)	POP (155)

It is clever to store values in arrays when you are doing many similar calculations repeatedly. That way your program will look better organized.

Two-dimensional arrays are divided into rows and columns, just like matrixes.

To specify a two dimensional array with the name DATA of type integer, you should write;

```
INTEGER DATA(3,3)
```

This array will look like this;

	1	2	3
1	(1,1)	(1,2)	(1,3)
2	(2,1)	(2,2)	(2,3)
2	(3,1)	(3,2)	(3,3)

To fill the array with values you may use a Do-loop, but you can still write down the values like you've done before.

Example;

```
INTEGER DATA(3,3)
...
DO 10 I=1,3
  DATA(I,1)=1
  DATA(I,2)=4
  DATA(I,3)=3
10 CONTINUE
```

The result of this will be;

	1	2	3
1	1	4	3
2	1	4	3
3	1	4	3

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

● DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGGING

EXAMPLE-
STEP BY STEP

The Do loop

The repetition of a number of statements for a predetermined number of times, is so important that Fortran contains a special construction which allows this to be done.

In general, a "**DO loop**" may contain any Fortran statement, including another do statement, known as a "nested DO loops".

The syntax is:

```
DO 100 INDEX= initial, limit, increment  
                  'statements'  
100 CONTINUE
```

The number 100 is a statement label

The INDEX is a variable, but it may be either real or integer. It starts at the initial, ends at the limit and increases with the increment. Increments are normally not included, as you most of the time wish to run the loop for every step in the interval.

The CONTINUE statement closes the DO loop

For example, DO 100 MONTHS=1,N,6 would go through all N months in steps of six.

Typically, there will be many loops and other statements in a single program that requires a statement label. The programmer is responsible for assigning a unique number to each label in each program (or subprogram).

Recall that column positions 2-5 are reserved for statement labels. The numerical value of statement labels have no significance, so any integer numbers can be used.

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAY● DO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGGINGEXAMPLE-
STEP BY STEP

The Do loop

Example:

This example will sum all the numbers from 1 to 10, while it counts how many steps it takes to sum up the numbers.

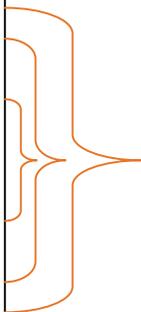
```
SUM=0
COUNT=1
DO 10 = NUMBER,1,10
    SUM=SUM+NUMBER
    COUNT= COUNT + 1
10 CONTINUE
```

Nested DO loops

DO loops can be nested within other DO loops, just as you can use IF structures within other IF structures. A nested DO loop cannot use the same index as an outer DO loop. A nested DO loop and its CONTINUE statement must be completely within the outer DO loop.

Example;

```
DO 5 I=1,5
    DO 10 J=1,10
        DO 15 K=1,25,5
            PRINT(*,*)I,J,K
        15 CONTINUE
    10 CONTINUE
5 CONTINUE
```



INTRODUCTION

UNIX COMMANDS

EXAMPLES
HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

• DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

EXAMPLE-
STEP BY STEP

PROGRAM EQUATION

```
INTEGER A,B,C  
REAL D  
REAL SOL(1:2)
```

THE
SOLUTIONS ARE
STORED IN AN
ARRAY

```
OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')  
OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
```

```
WRITE(*,*)'TYPE AN INTEGER FOR EACH OF THE VALUES A AND B'
```

```
DO 15 I=1,5  
READ(*,*)A, B  
READ(10,*)C
```

The do-loop
contains the
if statement

```
D=B**2-4*A*C
```

```
IF (D.LT.0) THEN  
WRITE(*,*)'THE EQUATION HAS A COMPLEX SOLUTION'
```

```
ELSE
```

```
SOL(1) = (-B + SQRT(D))/(2*A)  
SOL(2) = (-B - SQRT(D))/(2*A)
```

```
WRITE(11,12)'THE SOLUTIONS ARE: '(SOL(I),I=1,2)
```

```
END IF  
15 CONTINUE
```

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAY● DO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGINGEXAMPLE-
STEP BY STEP

When you run the program the screen will look like this. The program asks if I can type values for A and B, I have typed 2 for A and 8 for B.

```
bash-2.05a$ emacs equation.f
bash-2.05a$ xlf -o prog equation.f
** equation    === End of Compilation 1 ===
1501-510  Compilation successful for file equation.f.
bash-2.05a$ prog
  TYPE A AND B
2 8
-0.2679491937
-3.732050896
bash-2.05a$ █
```

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

• FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGGING

EXAMPLE-
STEP BY STEP

Format

When the program has calculated whatever it was supposed to calculate, it would be of no use if you don't get the results in a readable and understandable manner, and preferably in a external file so you can use your calculations in a graphical presentation etc

For this we will use the WRITE and FORMAT statements

Syntax:

WRITE(*, label) list-of-variables

FORMAT(format-code)

The wildcard * writes the result to screen, whereas a unit number would write to an external file assigned to this number.

A wide variety of format combinations exist.

A - text string

D - double precision numbers, exponent notation

E - real numbers, exponent notation

F - real numbers, fixed point format

I - integer

X- horizontal skip (space)

/ - vertical skip (new line)

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOP● FORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGINGEXAMPLE-
STEP BY STEP**Format examples**

The format code F11.3 would make 1 million look like:

1000000.000 -a total of 11 spaces where 3 has been assigned to the decimal part. Notice that period, plus and minus will take up one space each.

For large numbers it is better to use the exponent notation:

E7.2 would produce 1.00E+6 out of 1million

If your declaration is REAL your format should be E, but if you have an double precision you'd better use D.

```
WRITE(*,100) P,T,Z
100 FORMAT(2F8.4,F7.6)
```

should look something like this:

```
P          T          Z
303.4058   451.6251   0.98654
```

```
SUM=125.5
PRINT(*,5)'THE NUMBER IS;'SUM'
5  FORMAT(A15,X,F5.2)
```

THE NUMBER IS; 125.50

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOP● FORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGINGEXAMPLE-
STEP BY STEP

PROGRAM EQUATION

```
INTEGER A,B,C
REAL D,SOL(1,2)
```

```
OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')
OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
```

```
WRITE(*,*)'TYPE AN INTEGER FOR EACH OF THE VALUES A AND B'
```

```
DO 15 I=1,5
READ(*,*)A, B
READ(10,*)C
```

```
D=B**2-4*A*C
```

```
IF (D.LT.0) THEN
WRITE(*,*)'THE EQUATION HAS A COMPLEX SOLUTION'
```

```
ELSE
```

```
SOL(1) = (-B + SQRT(D))/(2*A)
SOL(2) = (-B - SQRT(D))/(2*A)
```

```
WRITE(11,12)'THE SOLUTIONS ARE; ' (SOL(I),I=1,2)
```

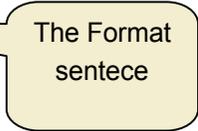
```
END IF
```

```
12 FORMAT(A30,F10.2)
```

```
15 CONTINUE
```

```
CLOSE(10)
CLOSE(11)
```

```
END
```



The Format
sentece

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

• SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

EXAMPLE-
STEP BY STEP

Subprograms

As the program becomes larger and more complicated, it is harder to keep the program well arranged. Some times we need to perform the same set of operations at more than one location in the program. In these situations it will be useful to use a subprogram. Subprograms are statements that are defined separately, and referred to when needed.

We have two types of subprograms; function and subroutine.

Function

We also have different types of functions. Fortran has some library functions like square root, cosine and so on.

Examples

SQRT(X) = square root of X

ABS(X) = absolute value of X

MAX(A,B,C...) = finds max of A,B,C...

SIN(X) = sinus of X

You can also use statement functions; a computation written in one single assignment statement. Define the statement function at the beginning of your program.

function name (argument list)=expression

example:

FAHRENHEIT(TEMP)=1.8*TEMP + 32

In this example we use the variable TEMP and calculates the degrees in Fahrenheit.

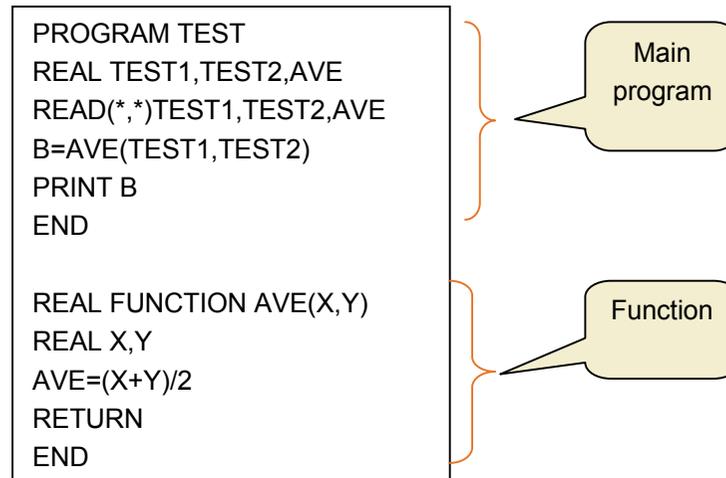
INTRODUCTIONUNIX COMMANDSHOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMAT● SUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKINGDEBUGINGEXAMPLE-STEP BY STEP

If the computation cannot be written in one statement, you must use the function subprogram, which is a program itself. The function sub program is separate from the main program. It begins with a nonexecutable statement;

FUNCTION name (argument list)

because the function is separate from the main program, it must end with END. The function is called implicitly by setting the variable parameter equal to the function name. The function must also contain RETURN, if not the main program will lose its control.

Example;

**Subroutine**

Whereas a function is restricted to represent a single value, subroutines can compute many. If several values need to be returned from a module, the subroutine takes in different variables, does the calculations and sends the results back to the main program.

Example (the same program as earlier, but here with subroutine)

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMAT● SUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGINGEXAMPLE-
STEP BY STEP

```

PROGRAM EQUATION
  INTEGER A,B,C
  REAL SOL(1:2)

  OPEN (UNIT=10, FILE='IN.DAT', STATUS='OLD')
  OPEN (UNIT=11, FILE='SOLUTION',STATUS='UNKNOWN')
  WRITE(*,*)'TYPE AN INTEGER FOR EACH OF THE VALUES A AND B'

  DO 15 I=1,5
  READ(*,*)A, B
  READ(10,*)C
  CALL CALCULATION(A,B,C, SOL)
  WRITE(11,12)'THE SOLUTIONS ARE: '(SOL(I), I=1,2)
12  FORMAT(A30,F10.2)
15  CONTINUE
  END

  SUBROUTINE CALCULATION (A,B,C, SOL)

  INTEGER A,B,C
  REAL D, SOL(1:2)
  D=B**2-4*A*C

  IF (D.LT.0) THEN
  WRITE(*,*)'THE EQUATION HAS A COMPLEX SOLUTION'

  ELSE
          SOL(1) = (-B + SQRT(D))/(2*A)
          SOL(2) = (-B - SQRT(D))/(2*A)

  END IF
  RETURN
  END

```

Main program

Subroutine

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

• NAG-ROUTINE

COMPILING RUN AND LINKING

DEBUGGING

EXAMPLE-STEP BY STEP

Subroutine

A subroutine is referenced with an executable statement whose general form is;

CALL subroutine name (argument list)

The first line in a subroutine identifies it as a subroutine;

SUBROUTINE name (argument list)

A subroutine uses the argument list not only for inputs to the subroutine, but also for all the values that has returned to the calling program. The arguments in the CALL statement must match in type, number, and order with those used in the subroutine definition.

The subroutine is a separate program, the arguments are the only link between the main program and the subroutine. The values used in the subroutine whom are not subroutine arguments, are local variables (like D in the square root example). Their values are not accessible from the main program.

The subroutine, like the function, requires a return statement to return control to the main program. It also requires an END statement because it is a complete program module.

A subroutine may referre to other functions or call other subroutines, but it cannot call itself.

NAG-routine

The Petrus-server includes a NAG-library of scientific subroutines. Note that all real variables should be declared as **REAL*8** (double precision) since the NAG-routines require this. The NAG-routine may be linked in by the command ;

xlf -o prog fil.f -L/localiptibm3/lib -l nag

You can take a look at the different routines at www.nag.com. Edit 'explore how nag can help you engineering', under Numerical Software you can edit NAG's libraries. NAG offers libraries in both Fortran77 and Fortran90, here you'll find callable routines for many mathematical and statistical areas.

INTRODUCTION

UNIX COMMANDS

HOW TO GET STARTED?

FORTRAN STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

• COMPILING RUN AND LINKING

DEBUGGING

EXAMPLE-STEP BY STEP

Compiling run and linking

When you are done writing your program, it's time to compile your program.

We will use the following compiler

xl f -o prog fort.f

Simply write this in the UNIX terminal window. Fort.f is the name of the Fortran file. If your file is called Exercise1.f, you should write `xl f -o prog Exercise1.f`

Hopefully the program will compile successfully, but most likely a list of errors is going to show. No programmer gets everything right the first time.

The compiler will, when it detects an error, let you know in which line the program the error occurred. The most frequent errors are the simplest ones; the programmer forgot a parenthesis, a comma, used too many columns (remember that you are only allowed to use 72 columns) or just wrote the same word in two different ways. Debug your errors and compile over again.

Run Program

When the compilation is complete it is time to run your program. Type "prog" in your terminal window, and the program should run.

If your program writes the results to an output file, you may open the file and view the results. Sometimes the output files give no result or show strange numbers. If this happens, you can use the debug operator.

INTRODUCTIONUNIX COMMANDSHOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKING● DEBUGGINGEXAMPLE-STEP BY STEP**Debug**

To run the debug operator the program must have compiled perfectly. Do like this to enter;

```
bash-2.05a$ xlf -g -o prog Exercise1.f
```

followed by; bash-2.05a\$ ideo debug prog

The screenshot shows the IBM Distributed Debugger window. The main pane displays the source code of a Fortran program named 'PROGRAM EQUATION'. The code is as follows:

```

1      PROGRAM EQUATION
2
3      REAL SOL1, SOL2, D
4      INTEGER A, B, C
5
6
7      A=0
8      B=8
9      C=2
10     D=B**2-4*A*C
11
12     IF(D .LT. 0)THEN
13         WRITE(*,*) 'COMPLEX SOLUTION'
14
15     ELSE
16         SOL1=(-B+SQRT(D))/(2*A)
17         SOL2=(-B-SQRT(D))/(2*A)
18
19         WRITE(*,*) SOL1
20         WRITE(*,*) SOL2
21
22     END IF
23
24     END

```

The debugger interface includes several panels:

- Stacks:** Shows 'Thread 1 [1392703] (Runnable)'.
- Breakpoints:** Empty.
- Monitors:** Shows 'Expression Monitors' and 'Value Panes'.
- Locals:** Empty.

Callout boxes provide the following instructions:

- "If you press Debug, your opportunities will show" (pointing to the Debug button in the toolbar).
- "If your program is large, a breakpoint would be helpful" (pointing to the Breakpoint button in the toolbar).
- "Your main program" (pointing to the source code window).
- "Locals will show the results" (pointing to the Locals panel).

The status bar at the bottom indicates 'Debugger ready...'.

INTRODUCTIONUNIX COMMANDSHOW TO GET STARTED?FORTRAN STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN AND LINKING• DEBUGGINGEXAMPLE-STEP BY STEP

The debugger doesn't help you find errors like a missing comma, typing error etc, but it is very useful in finding sequential errors in the program.

For instance, here you can see that I have an error in my equation program. I have opened it in the debugger, and I want help to find the error. You will step into the current source line in the program, and by each step you can see how the variable values change. This is done by choosing Locals, as shown in previous slide. By the time you have reached the source line which causes the error, changes will arise. Until you reach line 17 sol1 and sol2 would be zero, but when you pass line 17 the solutions change. That means that these lines have an error. As you see, the reason for the error is that $A=0$ and you cannot divide anything with zero. You should probably write a comment in your program, so that the variable A is not allowed to be equal to zero.

Monitors Locals Storage Registers

Locals Value Panes

Thread 1

- sol1 = NaNQ
- c = 2
- b = 8
- a = 0
- d = 64.00000000
- sol2 = -INF

the lines that contains errors

sol1 and sol2 have odd results

```

14
15     ELSE
16         SOL1=(-B+SQRT(D))/(2*A)
17         SOL2=(-B-SQRT(D))/(2*A)
18
19     WRITE(*,*)SOL1
20     WRITE(*,*)SOL2
21
22     END IF
23
24     END
  
```

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGGING● EXAMPLE-
STEP BY STEP**EXAMPLE- STEP BY STEP**

The text written in capital, bold letters is the program. The regular text is the comments.

PROGRAM EQUATION

**INTEGER A,B,C
REAL D,SOL(1:2)**

*(The first thing to do is to declare the different variables that will be included in the program. In this program we will try to find the solutions of a second degree equation, the in-variables can then be chosen to be integers, while the other variables must be of the category REAL. A REAL variable may contain a floating number. I choose to store the solutions in one array, but you may also store them as two different variables.)

**OPEN (UNIT=11, FILE='SOLUTIONS', STATUS='UNKNOWN')
OPEN (UNIT=10, FILE='INDATA', STATUS='OLD')**

*(The next step will be to open the files that we are getting the information from, or writing to. The solutions we get from calculating this equation will be written in the file SOLUTIONS. Just to show different ways of reading I have stored the values for C in the file INDATA. The files unit has to be a unique number, and the status is set to be UNKNOWN, NEW is also a possibility since this is a new file that will be made now.)

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGING

● EXAMPLE-
STEP BY STEP

WRITE(*,*)'TYPE THE VALUES FOR A and B'

*(We are going to make a program that can dissolve the equation for the different numbers you type currently, and not for some numbers from an already existing file. Therefore we don't need to open a file that consist any data, instead we make the program ask for the numbers; A and on the screen.)

READ(*,*)A, B

*(Now the program will have to read the values for the variables; A and B from the screen)

READ(10,*)C

*(Reads the C-value from the file, INDATA, with the unit 10)

D=B2-4*A*C**

*(We all know the equation of second degree $(x = (-b \pm \sqrt{b^2-4ac})/2a)$
We also agree to that if the square root are less than zero, it will fail. If the square root fails, the whole equation will fail. Instead of trying to dissolve the whole equation at once, we first take a look at the square root. We now declare a new variable D, who includes the values in the parenthesis of the square root.

INTRODUCTIONUNIX COMMANDSHOW TO GET
STARTED?FORTRAN
STRUCTUREDECLARATIONSOPENING FILESREADING DATAIF STATEMENTARRAYDO LOOPFORMATSUBPROGRAMSNAG-ROUTINECOMPILING RUN
AND LINKINGDEBUGING● EXAMPLE-
STEP BY STEP

```

IF (D.LT.0) THEN
WRITE(*,*)'THE EQUATION HAS A COMPLEX SOLUTION'
ELSE
SOL(1) = (-B + SQRT(D))/(2*A)
SOL(2) = (-B - SQRT(D))/(2*A)
ENDIF

```

*(This IF-statement decides whether the variable D is less than zero. If that's true, the program will write THE EQUATION HAS A COMPLEX SOLUTION to the screen. If the if-statement fails, then we know that the equation will not fail because of the square root statement, and the program will calculate the different solutions of the whole equation, storing the values in the array SOL.)

```

WRITE(11,12)'THE SOLUTION ARE:' (SOL(I),I=1,2)

```

*(After discovering the solutions, we write them to the file SOLUTIONS with UNIT=11 and the format 12. The solutions are written to an array SOL. To store the values in the array we use an implicit DO-loop, SOL(I),I=1,2, which distribute the values in the array. If we had N possible solutions, we should have N units in our array and the DO-loop would look like this; SOL(I),I=1,N)

```

12  FORMAT(A,F10.2)

```

*(The format stores space for a sting text and variable real values with ten numbers whereas two of them are decimals.)

```

CLOSE(11)

```

*(This sentence closes the file that stores the solutions)

```

END

```

*(Brings the program to close)

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGGING

EXAMPLE-
STEP BY STEP

References

Etter, D.M. 1993. Structured FORTRAN 77 For Engineers and Scientists – 4th edition. The Benjamin/Cummings Publishings Company, Inc., Redwood City, California

Preuss, H. 1992. Numerical Recipes in Fortran - 2nd edition. Cambridge University Press

Page, Clive G. 2001 Professional Programmer's Guide to Fortran77.
University of Leicester, UK

<http://www.library.cornell.edu/nr/bookfpdf.html>

<http://www.itea.ntnu.no/~kandal/unixkurs/>

<http://obelix.dawsoncollege.qc.ca/~dhackett/442/commands.html>

[Introduction to Fortran Programming](#)

INTRODUCTION

UNIX COMMANDS

HOW TO GET
STARTED?

FORTRAN
STRUCTURE

DECLARATIONS

OPENING FILES

READING DATA

IF STATEMENT

ARRAY

DO LOOP

FORMAT

SUBPROGRAMS

NAG-ROUTINE

COMPILING RUN
AND LINKING

DEBUGGING

EXAMPLE-
STEP BY STEP

About this module

Title: Introduction to Fortran

Author: Jon Kleppe

Assistant producer: Torunn Lilleeng

Size: 1.3 mb

Publication date: 1. August 2004

Abstract: An introductory module for Anvendt Datateknikk

Software required: PowerPoint XP/XP Viewer