Things you need to know about Fortran

Let's Dive In!

What's Ahead

About Fortran

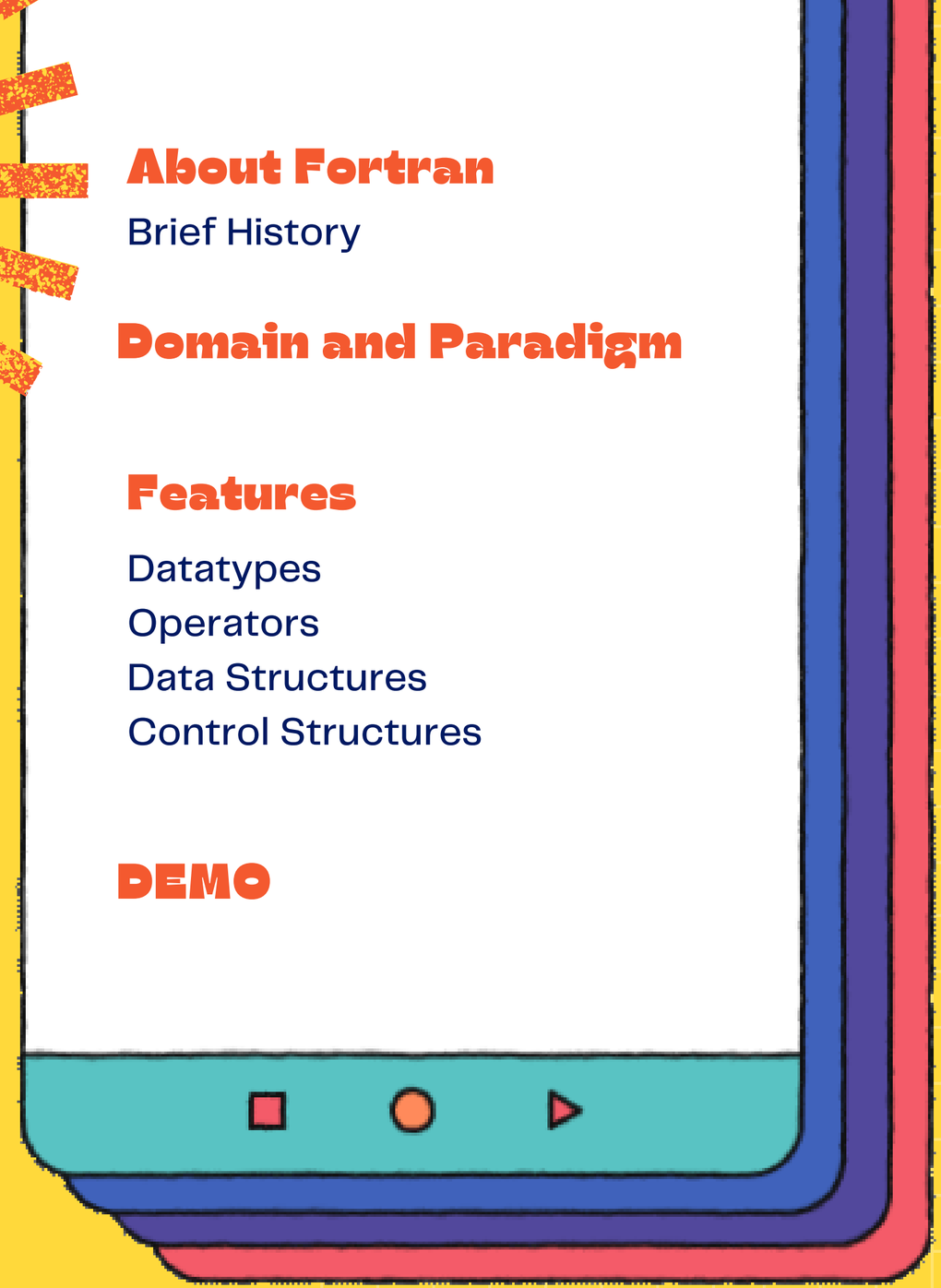
Brief History

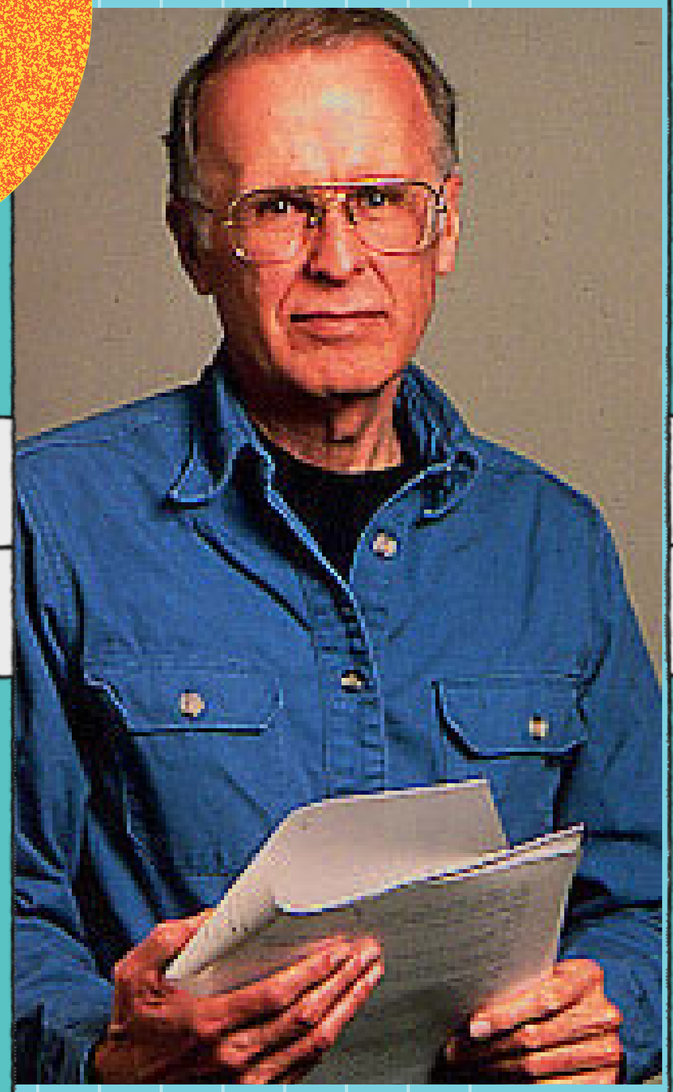
Domain and Paradigm

Features

Datatypes
Operators
Data Structures
Control Structures

DEMO

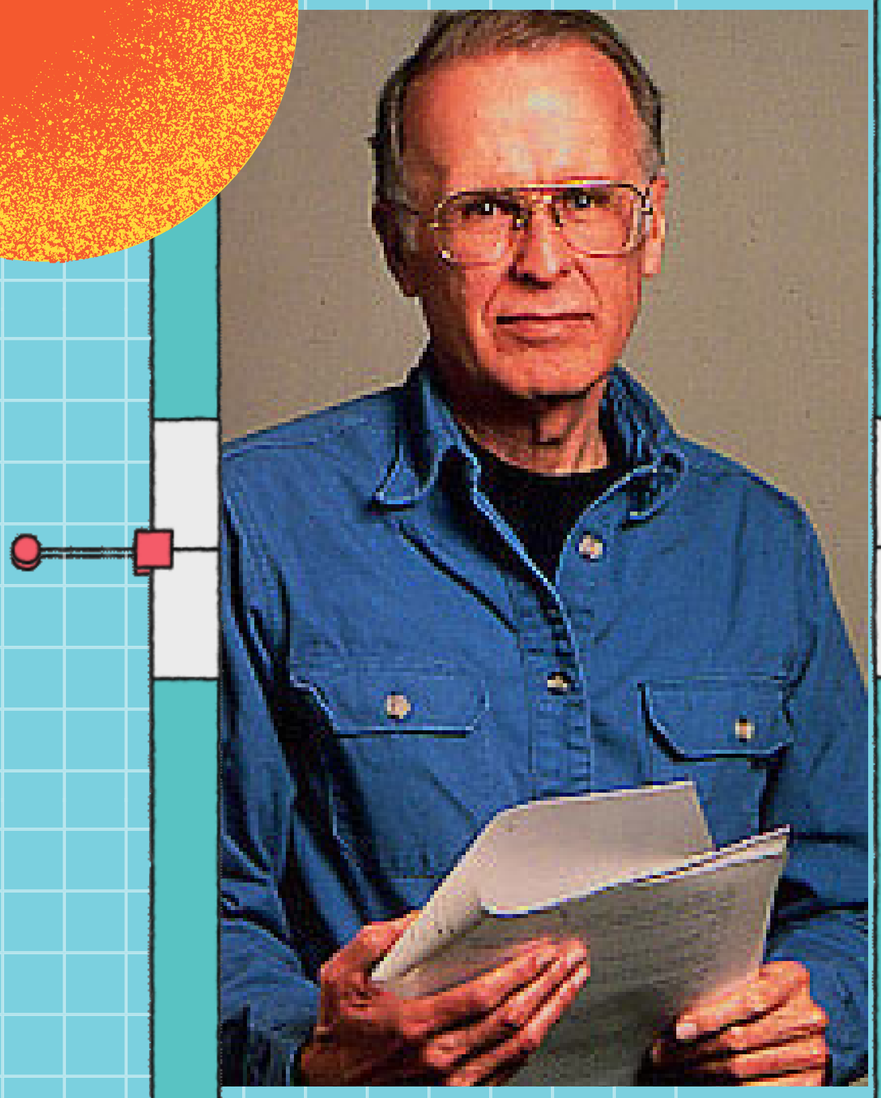




John W. Backus

"I'm so lazy writing programs, so instead I will make my own programming system that would make it faster and easier to write programs."

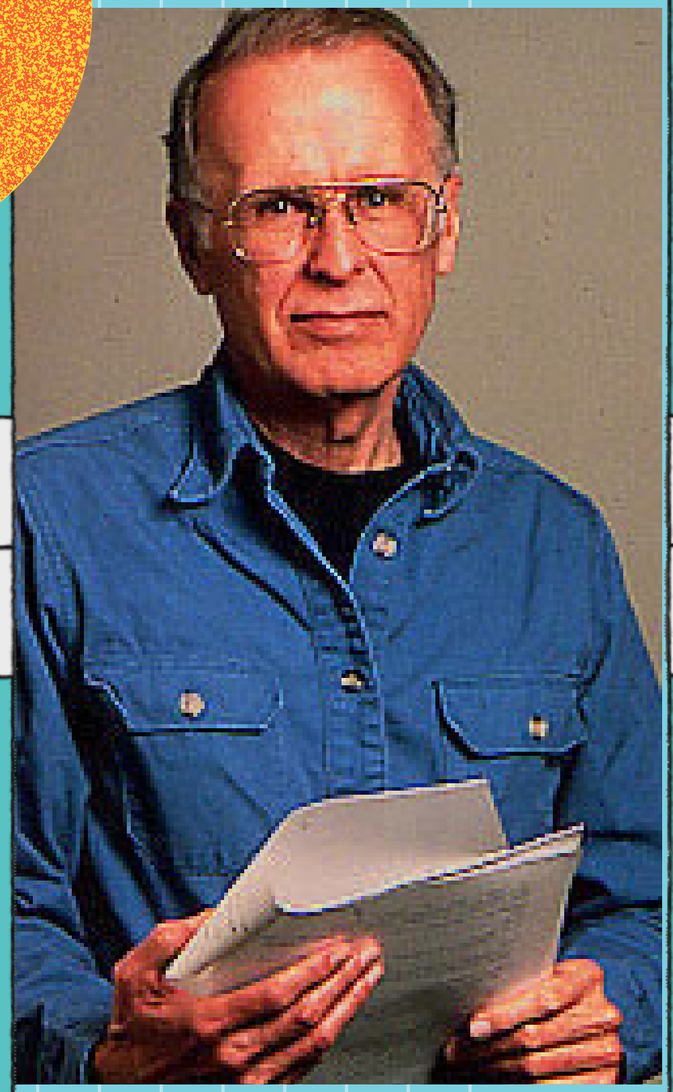
CALLED SPEEDCODING



John W. Backus

Principal Author of Fortran

Submitted a proposal to his superiors at IBM to develop a more practical alternative to assembly language for programming their IBM 704 mainframe computer in the late 50's



John W. Backus

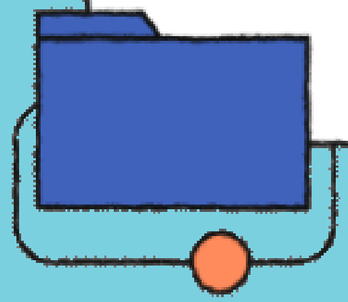
Design not only a better language but also one that would be easier and faster for programmers to use when working on the machine

Preliminary Report, Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN.

The first formal proposal for the language – published in November of 1954

FORTRAN 1957

3 years after – the language is comercially released



6 Decades of FORTRAN

FORTRAN 1957

Initial release of the langugae

FORTRAN 66

became the first industry-standard version

FORTRAN 77

ANSI standard in 1977

Fortran 90

ANSI standard in 1992

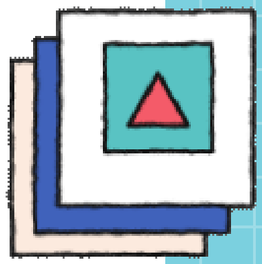
Fortran 95

ANSI standard in 1997

Fortran 2003

added support for object oriented programming

a compiled structured imperative programming language that is heavily used and is designed for numeric and scientific computing



What is Fortran?

enabled the rapid writing of computer programs that ran nearly as efficiently as programs that had been laboriously hand coded in machine language

Fortran Applications

design of bridges

factory automation control

aeroplane structures

storm drainage design

astrophysical modeling of stars and galaxies

analysis of scientific data

Datatypes

INTEGER whole number (positive, negative, zero)

E.g. -13 12345 +5

E.g. integer(kind=4) : : value

REAL decimal numbers or exponential

E.g. -13 .0 123.45 +0.005
.123E-3 3E5 3D5 .123D-3

E.g. real(kind=8) : : value

COMPLEX a pair of values that represent a complex number

E.g. (3.0, -5.0) => 3.0 -5.0i

Datatypes

LOGICAL — boolean data types

E.g. .TRUE. .FALSE.

CHARACTER — a sequence of one or more characters enclosed in a single quotation

E.g. 'Zara' 'uy'

E.g.
```
character (len = 40) :: name
name = 'Zara Ali'
```

Implicit Typing

DEFAULT

- **A-H, O-Z define REAL variables**
- **I-N define INTEGER variables**

E.g. **IMPLICIT INTEGER (A-Z)**
IMPLICIT COMPLEX (U,V,W)
IMPLICIT CHARACTER*2 (P)

OPERATORS

Arithmetic

+ – / * **

Relational

== or .EQ.

/= or .NE.

> or .GT.

< or .LT.

>= or .GE.

<= or .LE.

Logical

.AND.

.OR.

.NEQV.

.XOR.

.EQV.

.NOT.

Data Structures

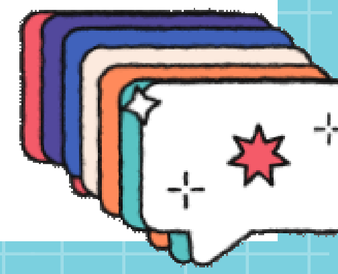
STRUCTURES

ARRAYS

ARRAYS

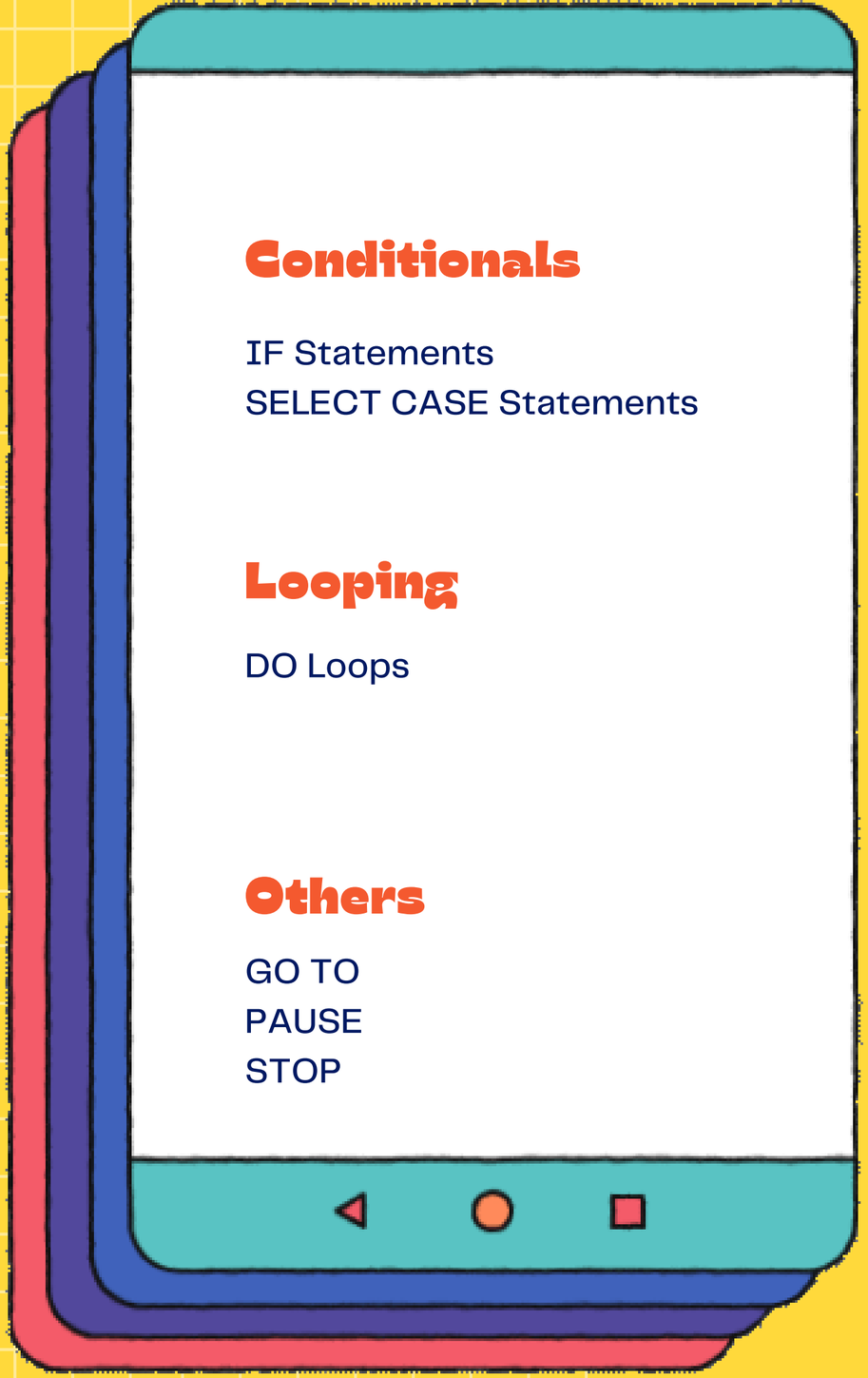
```
real, dimension(5) :: numbers
real, dimension(5, 3) :: matrix

number(1) = 2
number(5) = 6
matrix(1,1) = 2
```



STRUCTURES

```
type fruit
    real :: diameter
    real :: length
    character :: colour
end type
```



Conditionals

IF Statements
SELECT CASE Statements

Looping

DO Loops

Others

GO TO
PAUSE
STOP

CONTROL STRUCTURES

LOGICAL IF

SYNTAX

IF (logical-expression) statement

E.G.

```
Y = 0.0

IF (X .NE. 0.0) Y = 1.0/X

WRITE(*,*)X,Y
```

If the logical-expression evaluates to .TRUE., then the statement is executed and control passes on to the next executable statement in the program.

Otherwise, the statement is ignored and control passes on to the next executable statement in the program

BLOCK IF

IF-THEN-END IF

SYNTAX

```
IF (logical-expression) THEN
    statement1
    statement2
    ...
    statementn
  END IF
```

E.G.

```
          Y = 0.0
          Z = -1.0
  IF (X .NE. 0.0) THEN
      Y = 1.0/X
      Z = X**2
  END IF
          WRITE(*,*)X,Y,Z
```

This is the simplest form of the block IF.

Instead of a single executable statement, the word THEN follows the logical-expression.

Then one or more executable statements are listed in the order they should be executed and the block IF statement concludes with an END IF statement.

BLOCK IF

IF-THEN-ELSE-END IF

SYNTAX

```
IF (logical-expression) THEN
  statement-block1
ELSE
  statement-block2
END IF
```

E.G.

```
IF (TRNSCT .GE. 0.0) THEN
      CREDIT = CREDIT + 1
 ELSE
      DEBIT  = DEBIT + 1
END IF
```

This slightly more complicated case allows for one statement block to be executed if the logical-expression evaluates to .TRUE. and an alternative statement block to be executed if the logical-expression is .FALSE.

BLOCK IF

IF-THEN-ELSE IF-THEN-ELSE-END IF

SYNTAX

```
IIF (logical-expression1) THEN
  statement-block1
ELSE IF (logical-expression2) THEN
  statement-block2
ELSE IF (logical-expression3) THEN
  statement-block3
...
ELSE
  statement-blockn
END IF
```

E.G.

```
IF (MOD(YEAR,400) .EQ. 0) THEN
      WRITE(*,*)YEAR,' is a leap year'
    ELSE IF (MOD(YEAR,100) .EQ. 0)
THEN
      WRITE(*,*)YEAR,' is NOT a leap year'
    ELSE IF (MOD(YEAR,4) .EQ. 0) THEN
      WRITE(*,*)YEAR,' is a leap year'
    ELSE
      WRITE(*,*)YEAR,' is NOT a leap year'
    END IF
```

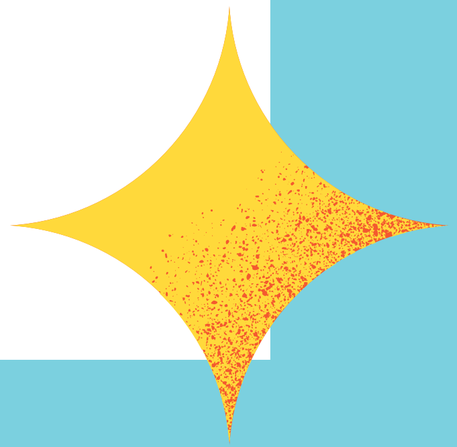
SELECT CASE STATEMENT

E.G.

```
select case( hour )
       case( 1:8 )
           activity = 'sleep'
       case( 9:11, 13, 14 )
           activity = 'class'
       case( 12, 17 )
           activity = 'meal'
       case default
           activity = 'copious free time'
end select
```

SYNTAX

```
select case( expr )
    case( values )
        ...
    case default
        ...
end select
```



DO LOOP

SYNTAX

```
DO loop_control
    do block
END DO
```

E.G.

```
DO K=2, 10 ,2
    PRINT*, K
END DO
```

- Loop_control can include a third parameter to specify increments
- If loop_control is omitted, loop will execute indefinitely or until some statement in do_block transfers out

GENERAL DO LOOP

SYNTAX

```
DO
    statement_sequence1
    IF(logical_exp) EXIT
    statement_sequence2
END DO
```

E.G.

```
DO
    READ*, R
    IF(R .LT. 0) EXIT
    PRINT*, R
END DO
```

- Execute do_block including terminating statement and loop back continually
- Must include some means to terminate loop
- If EXIT is omitted, loop will execute indefinitely or until some statement in do_block transfers out

DO WHILE LOOP

SYNTAX

```
DO WHILE( logical_expr )
   ...
END DO
```

- Execute code while logical expression is true and terminate otherwise
- Loop will not execute if the logical expression is false from the start

E.G.

```
X = 1
        DO WHILE(X<=10)
            WRITE*, X
            IF(R .LT. 0) EXIT
            X  = X + 2
        END DO
```

UNCONDITIONAL GO TO

SYNTAX

GO TO label

E.G.

```
10 CONTINUE
      WRITE(*,*)'Enter a positive value'
      READ(*,*)A
   IF (A .LE. 0.0) GO TO 10
```

- When an unconditional GO TO is encountered, control is immediately transferred to the executable statement labelled with the label.
- may be used to transfer control out of a block IF or DO loop but not in to them
- often used in conjunction with logical or block IF statements to construct loops when a simple DO loop is not appropriate

PAUSE

SYNTAX

PAUSE 'string'

E.G.

IF (DEBUG) PAUSE 'Entering main DO loop'

- halts the program in such a way that execution can be resumed in some manner by the user

NOTE: Implementation of the PAUSE statement is system-dependent and its use is strongly discouraged.

STOP

SYNTAX

STOP 'string'

E.G.

```
IF (X .EQ. 0.0)
STOP 'Emergency stop – Denominator is zero'
```

- stops the program and returns control to the computer's operating system
- This string must be a constant, not a variable, or it can be an integer up to five digits long

DEMO TIME!

Thank you for listening!