Linguagem **FOR**mula **TRAN**slation

Elementos Sintácticos

- Alfabeto

  ```
  A-Z a-z 0-9 _
  + - * /
  ( ) . = , ’ $ : ! " % & ;
  < > ? ␣
  ```

- Palavras
  - identificadores
    - têm de começar por uma letra
    - têm comprimento máximo de 31 caracteres
    - podem conter os caracteres: A-Z,a-z,0-9,_
    - letras minúsculas e maiúsculas são consideradas iguais

    ¡ Usar sempre nomes significativos !
  - Números
    - Inteiros 23, -34, ...
    - Reais 3.4, 0.34E1, ...

- frases - uma "frase" por linha de texto
  - instruções executáveis
  - instruções não executáveis (declarações)
  - comentários

No FORTRAN co-existem dois tipos de escrita possíveis para as frases:

Forma fixa ("fixed form"), era a única forma de escrita possível para o programas em Fortran anteriores à versão Fortran 90.

Cada "frase" corresponde a uma linha (cartão perfurado) com 80 colunas

1 a 5 Etiquetas. Se na primeira coluna estiver um "C" ou um "*" todo a linha é considerada uma *linha de comentário*.

6 Se contiver um caracter (que não o espaço ou o 0 (zero)) é considerada uma *linha de continuação*.

7 a 72 Espaço reservado à escrita das *instruções* e/ou *declarações*, podem começar (e acabar) em qualquer posição.

73-80 São ignoradas, podem ser usadas para identificar as linhas (cartões).

Forma livre ("free form"), é a forma de escrita actualmente adoptada para a escrita de novos programas.

- cada "frase" corresponde a uma linha com 132 colunas;
- a "frase" pode começar e acabar em qualquer posição da linha;
- pode-se prolongar uma "frase" por mais do que uma linha através da utilização do caracter "&", a ser colocado no fim da linha que se pretende ver continuada.

Tipos de Dados e Declarações

Tipos de Dados = Conjunto de Valores + Conjunto de Operações

O FORTRAN é uma linguagem com uma definição estrita de tipos ("strongly typed"), isto é, cada elemento (entidade de informação) tem um e um só tipo bem definido.

- Cada tipo de dados determina um dado conjunto de valores assim como as operações que podem ser aplicadas a esses valores.
- Cada elemento de informação tem um e um só tipo.
- O tipo de cada elemento pode ser deduzido somente do seu formato e do contexto em que está incluído.
- Cada operador requer operandos de um dado tipo e produz resultados de um (eventualmente diferente) dado tipo.

Tipos em FORTRAN

Tipos estáticos

- escalares
  - INTEGER ($\subsetneq \mathbb{Z}$)
  - REAL ($\subsetneq \mathbb{R}$)
  - LOGICAL ($\doteq$ {`.TRUE.`, `.FALSE.`})
- estruturados
  - CHARACTER (palavras, $\subset$ ASCII* ou $\subset$ EBCDIC*)
  - Tabelas (n-dimensionais)
  - Ficheiros
  - Tipos derivados (registos)
  - COMPLEX, ($\subsetneq \mathbb{C}$)

Além destes tipos o FORTRAN 90 introduz os tipos dinâmicos, construídos através da manipulação de ponteiros.

Declaração das variáveis (identificadores) dos vários tipos.

Declaração implícita

- os identificadores cuja primeira letra é `I`, ou `J`, ou `K`, ou `L`, ou `M`, ou `N` são do tipo INTEGER.
- todos os restantes identificadores são do tipo REAL.

Declaração Explícita

<Nome do tipo> `::` *<lista de variáveis>*

Por exemplo `INTEGER :: x,y,z`

A declaração implícita deve ser evitada pois pode facilmente levar a situações de erro.

Para obrigar a declaração explícita de todos os identificadores o Fortran 90 introduziu a declaração:

```
IMPLICIT NONE
```

Esta declaração deve ser usada em todos os novos programas, introduz-se logo a seguir ao começo do programa.

Tipo `INTEGER` ($\subsetneq \mathbb{Z}$)

Conjunto de Valores

Inteiros entre -2^{n-1} e $2^{n-1} - 1$.

Os compiladores de FORTRAN admitem vários valores para o n, o valor por omissão mais usual é de 32 (VAX Fortran). o que corresponde a uma gama de variação de -2147483648 a 2147483647.

Conjunto de Operadores

- binários:
  - `+,-,*,/` (divisão inteira)
  - `**` (potência)
- unários: `+,-`
- funções pré-definidas:

`ABS(x)`	$\lvert x \rvert$
`MOD(x,y)`	resto da divisão x/y
`MAX(x,y)`	máximo de x e y
`MIN(x,y)`	mínimo de x e y

Declaração

`INTEGER ::` *<lista de identificadores>*

Representação

2, 344, -5, +234,...

Tipo `REAL` ($\subsetneq \mathbb{R}$)

Conjunto de Valores

Representação com 32 bits (VAX Fortran).

- seis casas decimais de precisão.
- gama de variação entre -10^{37} e 10^{37}.

¡ A representação não é exacta !

¡ As operações entre reais podem introduzir erros de representação !

Conjunto de Operadores

- binários:
  - `+,-,*,/`
  - `**` (exponenciação)
- unários:
  - `+,-`

Conjuntos de Operadores (continuação)

funções pré-definidas:

`SQRT(x)`	$\sqrt{x}$
`ABS(x)`	$\|x\|$
`SIN(x),COS(x),TAN(x)`	funções trigonométricas
`EXP(x), LOG(x), LOG10(x)`	$\epsilon^x, \log_\epsilon(x), \log_{10}(x)$
`MOD(x,y)`	resto da divisão x/y
`MAX(x,y)`	máximo de x e y
`MIN(x,y)`	mínimo de x e y
`ASIN(x),ACOS(x),ATAN(x)`	funções trigonométricas inversas

Declaração

`REAL ::` *<lista de identificadores>*

Representação

7.4, -12.34, 5E+2, -34E-3,...

Expressões (envolvendo inteiros e reais)

- Precedência e associatividade dos operadores.
- Sobrecarga dos Identificadores.
- Expressões envolvendo os dois tipos de dados.

Precedência e associatividade dos operadores

A precedência e a associatividade os operadores em FORTRAN é a usual nas expressões matemáticas.

Da mais alta precedência para a mais baixa:

`**`	associatividade à direita
`*, /`	associatividade à esquerda
`+, -`	associatividade à esquerda

as expressões entre parêntesis são avaliadas em primeiro lugar.

Sobrecarga dos identificadores

As operações `2*4` e `2.0*4.0` são operações (completamente) diferentes, no entanto usam o mesmo símbolo de operador!

Relembrando: em FORTRAN cada elemento tem um e um só tipo; cada operação tem argumentos e resultados de tipos bem definidos!

¿Então?

A forma como o compilador resolve o problema é designada por sobrecarga dos identificadores, temos que:

Duas (ou mais) operações (internamente) diferentes partilham o mesmo identificador (externo), o compilador faz a destrinça automaticamente, e trata de implementar no programa executável a operação apropriada.

Expressões envolvendo os dois tipos de dados

¿ Mas então, o que acontece no caso `2.0*3`?

Temos que:

`3/2` (resp. `3.0/2.0`), dois operandos inteiros, então a operação é a divisão inteira (resp. real), e o resultado é `1` (resp. `1.5`).

No caso `3.0/2`, como um dos argumentos é do tipo `REAL` e outro é do tipo `INTEGER`, considera-se a inclusão `INTEGER` $\subset$ `REAL`, isto é, o compilador procede à conversão (automática) do argumento inteiro para um real, passando-se a ter então dois argumentos reais e, por isso, uma operação real com um resultado real.

Concluindo:

`3.0/2` $\rightarrow$automaticamente$\rightarrow$ `3.0/2.0 = 1.5`

¡ A conversão (automática) de um inteiro para um real pode levar a erros de representação !

Funções de transferência entre INTEGER e REAL

Nome	Tipo do Argumento	Tipo do Resultado	Comentário
`INT(x)`	`REAL`	`INTEGER`	Parte inteira de `x`, com truncatura
`NINT(x)`	`REAL`	`INTEGER`	Arredondamento para o inteiro mais próximo
`REAL(x)`	`INTEGER`	`REAL`	Conversão de um inteiro para um real (representação interna)

Tipo `CHARACTER` (palavras)

Conjunto de Valores Este é um tipo composto, isto é, os elementos deste tipo são sequências (de comprimento fixo) de símbolos do alfabeto FORTRAN.

Temos então que para um dado $n \geq 1$ o tipo `CHARACTER(len=n)`, define o conjunto de todas as palavras constituídas por símbolos do alfabeto FORTRAN de comprimento n.

O alfabeto FORTRAN constitui um conjunto mínimo de símbolos, sendo possível que, dependendo do compilador, outros símbolos possam ser usados. O conjunto de símbolos mais usual é-nos dado pela tabela *ASCII*.

¡ As maiúsculas são diferentes das minúsculas !

Antes de falar sobre as operações convêm referir a forma de representar elementos deste tipo, assim como declarar variáveis.

Declaração

Palavra de comprimento $n \geq 1$

`CHARACTER(len=n) ::` < lista de identificadores >

Palavras de comprimento 1 isto é, caracteres.

`CHARACTER` < lista de identificadores >

Representação

```
"palavra", "3.14", 'x < y <= z'
```

Conjunto de Operações

- especificação de sub-palavras
- concatenação (junção de duas palavras numa só palavra)

Especificação de sub-palavras ($n \leq m$)

<identificador>(< n >:< m >)

representa a sub-palavra formada pelos caracteres da palavra entre as posições (inclusive) n e m, se $n > m$ temos a palavra nula.

Concatenação de palavras

<identificador1> `//` <identificador2>

Exemplos

- considerando a seguinte declaração:

  `CHARACTER(len=8) :: a`

  e para um valor de `a` igual a `'12345678'` temos então que:

  `a(2:4)` = `'234'`, `a(3:7)` = `'34567'`

- considerando as seguintes declarações:

  ```
  CHARACTER(len=8) :: a
  CHARACTER(len=4) :: c,d
  ```

  e para um valor de `a` igual a `'12345678'`, `c` igual a `'1234'`, e `d` igual a `'5678'` temos então que:

  ```
  a(1:4) // a(5:8) = '12345678'
  c // d = '12345678'
  ```

¡ Uma palavra tem sempre um comprimento fixo, imposto pela sua declaração !

Considerando a seguinte declaração:

```
CHARACTER(len=8) ::  a,b,c
```

então temos que:

"atribuição"	valor
`a = '12345678'`	12345678
`b = a(5:8)`	5678␣␣␣␣
`c = '12345678910'`	12345678

Funções de transferência

nome da função	tipo do argumento	tipo do resultado
ACHAR	INTEGER	CHARACTER
IACHAR	CHARACTER	INTEGER

Tipo `LOGICAL` (valores lógicos)

Conjunto de Valores

{.TRUE., .FALSE.}

Conjunto de Operações

{.AND., .OR., .EQV., .NEQV., .NOT.}

Tabela de Verdade

e_1	e_2	$e_1 \wedge e_2$	$e_1 \vee e_2$	$e_1 \Leftrightarrow e_2$	$e_1 \not\Leftrightarrow e_2$	$\neg e_1$
V	V	V	V	V	F	F
V	F	F	V	F	V	F
F	V	F	V	F	V	V
F	F	F	F	V	F	V

Predicados pré-definidos.

Operador	Sintaxe antiga	Significado
==	.EQ.	igualdade
/=	.NEQ.	desigualdade
>	.GT.	maior
>=	.GE.	maior ou igual
<	.LT.	menor
<=	.LE.	menor ou igual

- Comparação entre inteiros

  `2 < 3` $\mapsto$ `.TRUE.`

  ¡ Sem problemas !

- Comparação entre reais

  `2.0 < 3.0` $\mapsto$ `.TRUE.`

  mas

  `(x/y)*y == x` $\mapsto$ `?`

  Por erros de representação e no efectuar da operação esta comparação pode não dar verdadeira.

  ¡ A comparação (em termos de igualdade) entre reais é de evitar !

  Se é necessário comparar reais em termos da igualdade devemos recorrer ao seguinte:

  `ABS(x-y) <= epsilon`

  Isto é $|x - y| \leq \varepsilon$, com epsilon um valor ditado pela representação usada ($\varepsilon = 0.5 \times 10^{-6}$).

- Comparação entre inteiros e reais

  Conversão automática, logo trata-se de comparação entre reais.

- Comparação entre caracteres (`CHARACTER`)

  A comparação é feita tendo em conta a codificação de caracteres usada, normalmente a tabela ASCII.

  `'A' < 'B'` $\mapsto$ `.TRUE.`

  `'a' < 'B'` $\mapsto$ `.FALSE.`

  ¡ Quando se comparam caracteres as maiúsculas e as minúsculas são consideradas diferentes !

- Comparação de palavras (`CHARACTER(len=<n>)`)

  A comparação é feita letra a letra

  `'AAAB' > 'AAAA'` $\mapsto$ `.TRUE.`

  Quando as palavras são de comprimento desigual temos:

  Compara-se normalmente até se atingir o fim da palavra mais curta, se as palavras são iguais (até o atingir o fim da mais curta), a palavra mais comprida é considerada maior.

  `'AAABCZ' > 'AAB'` $\mapsto$ `.FALSE.`

  `'AAABCZ' > 'AAA'` $\mapsto$ `.TRUE.`

Precedência e Associatividade

Da mais alta para a mais baixa (precedência).

`<,<=,>,>=,==,/=`	associatividade à esquerda
`.NOT.`	
`.AND.`	associatividade à esquerda
`.OR.`	associatividade à esquerda
`.EQV., .NEQV.`	associatividade à esquerda

As leis da lógica proposicional são válidas, no caso dos quantificadores temos de ter em conta que um algoritmo é sempre (por definição) um processo finito. Temos então:

$$\forall_{x \in X} P(x) \underset{X \text{ finito}}{\Leftrightarrow} P(x_1) \wedge P(x_2) \wedge \ldots \wedge P(x_n)$$

$$\exists_{x \in X} P(x) \underset{X \text{ finito}}{\Leftrightarrow} P(x_1) \vee P(x_2) \vee \ldots \vee P(x_n)$$

$$\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$$

$$\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$$

$$A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$$

$$A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$$

Declaração e Inicialização

Quando se pretende introduzir uma variável (informação a manipular) num programa temos que efectuar uma declaração, na qual especificamos o seu tipo.

Vejamos por exemplo de um programa para somar dois inteiros:

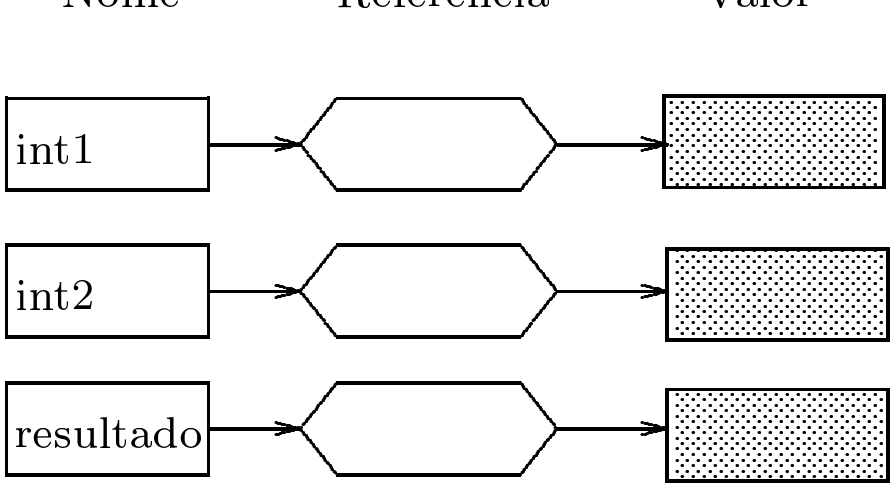
$\longrightarrow$ dois inteiros

$\longleftarrow$ o resultado da soma dos dois inteiros

Necessitamos de três inteiros, temos então a seguinte declaração:

```
INTEGER :: int1, int2, resultado
```

Com esta declaração definem-se três identificadores, os quais ficam associados a três posições de memória capazes de guardar valores inteiros. Graficamente vamos representar essa situação da seguinte forma:



o valor inicial das variáveis é indefinido.

¿ Qual o valor após a declaração ?

¡ Indeterminado !

Embora haja compiladores que, por omissão, inicializam as variáveis isso não está estabelecido na linguagem FORTRAN. Como tal a utilização de uma variável admitindo a sua inicialização por parte do compilador é considerado um erro de programação.

¡ É necessário inicializar (sempre) uma variável antes de a usar !

- através de uma instrução de atribuição (ver mais à frente).
- através de uma declaração com inicialização explícita.

```
<tipo> ::  <lista de <identificador>=<valor>>
```

Exemplo

```
INTEGER ::  hh=0,mm=0,ss=0
```

Constantes

Uma situação distinta desta última é quando se quer declarar um identificador com um determinado valor inicial e se pretende, ao mesmo tempo, que esse valor não seja alterado no decorrer do programa. Por exemplo a declaração do valor de π através de um identificador, `pi=3.141593`.

Nestes casos temos uma declaração de constante:

```
<tipo>,PARAMETER::<lista de <identificador>=<valor>>
```

Exemplo

```
REAL, PARAMETER ::  pi=3.141593
```

O interesse deste tipo de declaração é de:

- simplificação de escrita;
- facilidade na alteração de um dado valor que é utilizado em muitos pontos de um dado programa.