FORTRAN 77

V. B. Mendes (1999)

NOÇÕES BÁSICAS DE FORTRAN 77

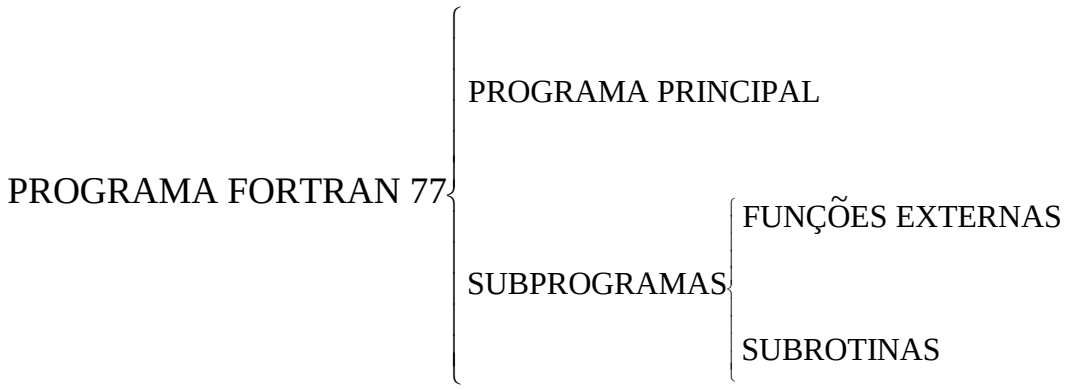
Um programa FORTRAN 77 (F77) é um conjunto de "instruções", que apresenta as seguintes características fundamentais:

- é escrito usando um número máximo de 72 caracteres por linha
- as primeiras seis colunas são reservadas para fins especiais
    - *Exemplos:*

    *a) um c ou * na primeira coluna indica que a linha é um comentário, não sendo por isso executada;*

    b) um caracter na coluna 6 indica que a linha é continuação da linha anterior.
- a primeira instrução é uma declaração PROGRAM
- a última instrução é uma declaração END

A estrutura básica de um programa F77 é a seguinte:



Os subprogramas são de carácter opcional e um programa F77 poderá não apresentar qualquer subprograma. Para além das funções externas, o F77 usa um vasto conjunto de funções (funções intrínsecas), que compreende a generalidade das funções matemáticas. A utilização dos subprogramas torna um programa F77 mais claro e permite uma melhor identificação de erros de programação.

A linguagem F77 utiliza variáveis, cujos nomes consistem de conjuntos de um máximo de 6 caracteres (letras e/ou dígitos), a primeira das quais é obrigatoriamente uma letra (no decorrer deste texto iremos limitar a exposição às versões *standard* de Fortran; certos compiladores aceitam mais do que 6 caracteres). As variáveis podem representar números inteiros, reais, complexos ou caracter.

(NOTA: Por defeito, o F77 assume que as variáveis iniciadas por A-H e O-Z são reais e que as iniciadas por I-N são inteiros; quando não especificado, assumiremos este pressuposto)

1. TIPOS DE VARIÁVEIS

REAL: Uma variável REAL é uma variável que representa um número real, sendo este interpretado pela linguagem Fortran como um número que contém um ponto decimal (ou que está escrita na forma exponencial). Por defeito, as variáveis REAL são de precisão simples (6-7 dígitos decimais) e ocupam 4 *bytes* de memória. A declaração de um variável como REAL faz-se do seguinte modo

REAL*4 nome ou REAL nome

Exemplos de números reais: 1.1534, -0.345E3, 3.1425926

Exemplos de variáveis reais: A1TER, ZFTE, F123,SERTE

Uma variável REAL de dupla precisão (15-16 dígitos decimais) ocupa 8 *bytes* de memória, e a declaração faz-se usando :

REAL*8 nome ou DOUBLE PRECISION nome.

INTEGER: Uma variável INTEGER é uma variável que representa um número inteiro (número que não contém qualquer ponto decimal). Por defeito, as variáveis INTEGER são de 4 bytes (ou INTEGER*4). As variáveis INTEGER podem ainda ser declaradas como de 1 byte ou de 2 bytes, usando a declaração:

INTEGER*1 nome

INTEGER*2 nome

Exemplos de números inteiros: 1, -546, 31425926

Exemplos de variáveis inteiras: I, J, K234, IJK

CHARACTER: Uma variável CHARACTER representa uma sequência de caracteres. As variáveis CHARACTER ocupam 1 byte de memória por cada caracter e, por defeito, uma variável CHARACTER é uma variável de um byte (CHARACTER = CHARACTER*1). As cadeias de caracteres a atribuir a uma variável CHARACTER são envolvidas por plicas (‘).

Exemplo de uma cadeia de caracteres: ‘Isto serve de exemplo’

COMPLEX: Uma variável COMPLEX é um par ordenado de números reais de precisão simples, representando respectivamente a parte real e a parte imaginária do número complexo. Esse par ordenado é envolvido por parentesis e separados por uma vírgula.

Exemplo de um número complexo: (253.34, 2.534)

Um número complexo de precisão dupla (par ordenado de números reais de precisão dupla), é declarado como

COMPLEX*16 nome

ou

DOUBLE COMPLEX nome

LOGICAL: Uma variável LOGICAL é uma variável com dois valores lógicos possíveis: .TRUE. e .FALSE..

As variáveis são declaradas no início do programa, logo após o nome do programa, segundo a estrutura básica:

```
INTEGER  nomes
REAL  nomes
COMPLEX  nomes
CHARACTER  nomes
LOGICAL nomes
```

No caso das variáveis representarem vectores ou matrizes, as dimensões respectivas são indicadas entre parentesis.

Exemplo um vector de dimensão 50: VEC(50)

Exemplo uma matriz 30x3: MAT(30,3)

2. ATRIBUIÇÃO DE VALORES A VARIÁVEIS

A instrução mais elementar de um programa F77 é a de atribuição, que tem a forma geral:

nome = expressão

ou

nome = constante

(o valor contido em 'expressão' ou o valor numérico 'constante' é atribuído a 'nome')

Exemplo: FROTA = 5

A atribuição de valores iniciais a variáveis que se mantenham inalteráveis durante todo o programa pode ainda ser feito recorrendo às declarações DATA e PARAMETER. Estas declarações são feitas no ínicio do programa ou subprograma F77, logo após a declaração de variáveis. Assim, cada um dos módulos de um programa F77 apresenta a seguinte organização (as atribuições iniciais poderão ser omitidas):

```
PROGRAM     nome (SUBROUTINE nome)
                    Declaração de variáveis (REAL, INTEGER, etc)
                    Atribuições iniciais (PARAMETER, DATA)
                    .
                    .       (Instruções)
                    .
                    END (RETURN, para SUBROTINAS)
```

O formato de uma declaração DATA é o seguinte:

DATA lista1 / constantes1/, lista2 / constantes2/, …

ou

DATA lista1 / constantes1 / lista2 / constantes2/…

Exemplo: DATA A,B,C /1.0,2.0,3.0/,D /4.0/

O formato de uma declaração PARAMETER é o seguinte:

PARAMETER (nome1 = constante1 / expressão1 , nome2 = constante2, …)

Exemplo: PARAMETER (PI=3.1415926536)

3. EXPRESSÕES ARITMÉTICAS

As expressões aritméticas fazem uso dos seguintes operadores aritméticos:

adição	+
subtracção	-
multiplicação	*
divisão	/
exponenciação	**

A exponenciação tem prioridade máxima. A divisão e multiplicação têm prioridade média, a adição e subtracção têm prioridade mínima. As expressões são executadas da esquerda para a direita, para os operadores com idêntica prioridade, à excepção da exponenciação, que é executada da direita para a esquerda. As expressões entre parêntesis são calculadas em primeiro lugar, permitindo, desta forma, quebrar a "regra das prioridades".

Nas operações envolvendo variáveis de diferentes tipos são seguidas as seguintes regras:

- a divisão entre inteiros conduz a um número inteiro, provocando por isso **truncatura** do resultado.

  Exemplo: 4/3 =1

- os valores reais são **truncados** antes de serem atribuídos a variáveis inteiras.

- em operações envolvendo inteiros e reais, os valores inteiros são automaticamente convertidos em reais, pelo que o resultado obtido é sempre um número real.

4. SUBPROGRAMAS: FUNÇÕES E SUBROTINAS

Para além das funções intrínsecas inerentes à linguagem F77 (ver apêndice), o programador poderá crias as suas próprias funções, a que chamaremos funções externas. A estrutura de uma função externa é semelhante à do programa principal, sendo a declaração PROGRAM substituída por uma declaração especial FUNCTION. A forma geral de uma função é a seguinte:

```
(INTEGER, REAL, ..)   FUNCTION  nome (arg1, arg2,...)
Declarações
Instruções
RETURN
END
```

onde arg1, arg2, ... são os argumentos da função. Uma função externa tem obrigatoriamente que conter uma variável com o mesmo nome da função, que representa o valor da função para os argumentos dados. O tipo de função pode ser declarado explicitamente, usando as mesmas regras de declaração de variáveis.

Exemplo: função que calcula a área do círculo

```
REAL FUNCTION AREA (RAIO)
PARAMETER (PI=3.1416)
AREA = PI*RAIO*RAIO
RETURN
END
```

Se pretendermos, no decorrer do programa principal, recorrer à utilização desta função, e atribuir, por exemplo, o resultado da função à variável A, basta fazer:

```
A = AREA (RAIO)
```

Uma subrotina apresenta uma estrutura semelhante ao programa principal, sendo a declaração PROGRAM substituída por uma declaração especial SUBROUTINE. A principal diferença

em relação a uma função externa reside na forma como é referenciada e como os resultados são transferidos para o programa principal. O acesso a uma subrotina faz-se através de uma instrução CALL, seguida do nome da subrotina e da lista dos argumentos que serão usados para transmitir informação entre o programa principal (que poderá ser uma subrotina) e a subrotina. A forma geral da instrução CALL é a seguinte:

```
CALL NOME_ SUBROTINA (lista de argumentos reais)
```

A estrutura geral de uma SUBROUTINE é a seguinte:

```
SUBROUTINE  nome (lista de argumentos mudos)
Declarações
Instruções
RETURN
END
```

A fim de melhor de perceber a diferença entre um função externa e um subrotina, recorremos ao exemplo anterior.

```
PROGRAM CAREA
*       Definir as variáveis como reais
        REAL AREA,R
*       Calcular para um círculo com um raio R de 12 m
        R=12.0
*       Chamar a subrotina que calcula a area
        CALL SAREA (R,AREA)
*       A área é dada pela variável AREA. Escrever no ecrã
        WRITE(*,*)'A area do círculo e: ',AREA, ' metros quadrados'
        END

        SUBROUTINE  SAREA (RAIO,A)
        REAL PI, RAIO,A
        PARAMETER (PI=3.1415926)
        A = PI*RAIO*RAIO
        RETURN

        END
```

Note-se que enquanto numa função só é possível atribuir valores a uma só variável, a subrotina permite passar para o programa principal vários valores para outras tantas variáveis, que são calculados dentro da subrotina.

5. CICLO DO

Um ciclo DO é uma forma de repetir um conjunto de instruções F77. Existem diversas estruturas de ciclos DO. O ciclo DO mais tradicional apresenta a seguinte estrutura geral:

```
      DO label int  =  expr1, expr2 (,expr3)
      Instruções
label CONTINUE
```

onde expr1, expr2 e expr3 são expressões ou valores numéricos inteiros, que representam o valor do início, fim e incremento do ciclo. Este último pode ser omitido, se o incremento a usar for a unidade. O *label* (etiqueta) é um número inteiro, a escrever nas colunas 1 a 5, que indica o fim do ciclo DO. Poderá haver vários ciclos DO num ciclo DO.

Exemplo: ciclo para ler 50 valores que constituem um dado vector X

```
      DO  5 I = 1, 50
      READ (LU,*) X(I)
5     CONTINUE
```

6. DECISÕES LÓGICAS

Para além das expressões numéricas, o F77 permite a avaliação de expressões lógicas, que podem ter o valor verdadeiro ou falso, e que são muito úteis na construção das instruções IF. As expressões lógicas são formadas usando **operadores relacionais**, que estabelecem a relação entre dois valores aritméticos, e **operadores lógicos**, que combinam dois ou mais valores lógicos. Os operadores relacionais usados em F77 são:

operador relacional F77	operador matemático equivalente
.LT.	<
.LE.	≤
.EQ.	=

.GE.	≥
.GT.	>
.NE.	≠

Os operadores lógicos são os seguintes:

.NOT.	(prioridade máxima)
.AND.	
.OR.	(prioridade mínima)

O resultado dos operadores lógicos .AND. e .OR. entre duas expressões lógicas, L1 e L2, é o seguinte:

L1	L2	L1**.OR.**L2	L1**.AND.**L2
V	V	V	V
V	F	V	F
F	V	V	F
F	F	F	F

A instrução IF faz utilização de expressões lógicas e é usada no teste de condições. Existem basicamente 2 formas principais de IF: o IF simples e o bloco IF.

O **IF simples** tem a seguinte estrutura:

```
IF (condição) THEN
  Instruções
END IF
```

ou seja, se a condição a testar for verdadeira (e só nestas condições) o programa executa o conjunto de instruções.

Exemplo: Testar se um número é maior do que 100. Caso afirmativo, adicione 50 e escreva no ecrã o resultado dessa operação:

```
IF (I.GT.100) THEN
I = I + 50
WRITE (6,*) I
END IF
```

Se a acção a executar é uma acção simples, este IF pode ser escrito sob a forma:

```
IF (condição) acção
```

Exemplo: Testar se um número é maior do que 100. Caso afirmativo, adicione 50.

```
IF (I.GT.100) I=I+50
```

O **bloco IF** tem a seguinte estrutura:

```
IF (condição1) THEN
  Instruções
  ELSE IF (condição2) THEN
    Instruções
    ELSE IF (condição3) THEN
      Instruções
      ⋮
    ELSE
    Instruções
END IF
```

A instrução IF pode ser usada em conjunção com os ciclos DO, permitindo criar **ciclos DO condicionais**. Para sair de um ciclo DO condicional, é útil a instrução GOTO:

```
GOTO label
```

Exemplo: Testar se um número inteiro é negativo, zero ou positivo. Note-se que um teste semelhante para um número real necessita de uma definição "formal" do valor zero.

```
*       ler o número a testar
        WRITE (6,*) 'Introduza o número a testar...'
        READ (5,*) N
*       testar
        IF (N.GT.0) THEN
              WRITE (6,*) 'Numero positivo'
        ELSE IF (N.LT.0) THEN
              WRITE (6,*) 'Numero negativo
        ELSE
              WRITE (6,*) 'Zero'
        END IF
```

7. FORMATOS

Existem dois comandos básicos que permitem a entrada e saída de dados num programa F77:

READ *, lista_de_valores

para entrada de dados a partir do teclado, e

PRINT *, lista_de_valores

para saída de dados no ecrã.

Estes dois tipos de leitura são muito restritivos; para permitir maior flexibilização na entrada e saída de dados, usa-se a instrução FORMAT. Para permitir a leitura e escrita de dados de e para ficheiros, é ainda possível especificar a unidade lógica a utilizar. As instruções READ e WRITE permitem a utilização de uma lista de informação de controlo que define as unidades e formatos. Na sua forma geral, tem-se:

READ (lista de informação de controlo) valores de entrada
WRITE (lista de informação de controlo) valores de saída

A especificação da unidade de leitura na lista de informação de controlo é feita usando UNIT = u, onde *u* é o número da unidade de leitura. (Em Unix, a unidade 5 corresponde à leitura de dados a partir do teclado e 6 a escrita de valores no ecrã.). A especificação do formato é feita usando FMT = *label*, onde *label* é uma instrução FORMAT.

NOTA: A indicação UNIT = e FMT = pode ser omitida.

O controlo de erros no processo de leitura pode ser feito usando especificações ERR = s, END = s, ou IOSTAT = *ios*, onde s é o *label* da instrução a executar se ocorrer um erro no processo de leitura (ERR), ou fim de ficheiro (END). O especificador *ios* pode apresentar os seguintes valores:

- 0 – processo de leitura normal;
- positivo – ocorreu um erro no processo de leitura;
- negativo, se o fim do ficheiro foi atingido sem erro.

Uma declaração FORMAT tem a forma geral:

```
label   FORMAT (especificações de formato)
```

Os especificadores de formatos mais importantes são:

Iw leitura/escrita de um inteiro com w dígitos (incluindo o sinal)

Fw.d leitura/escrita de número real com *w* dígitos, dos quais *d* constituem a parte decimal (incluindo o sinal)

Ew.d leitura/escrita de número real com *w* dígitos, dos quais *d* constituem a parte decimal, mas com notação científica (incluindo o sinal)

nX leitura/escrita de n espaços em branco

Tc leitura/escrita da variável na coluna c

Aw leitura/escrita de variável CHARACTER com *w* caracteres

/ mudança de linha

‘cadeia de caracteres’ escrita do conteúdo entre plicas (cadeia de caracteres)

O formato pode também ser incluído na instrução READ/WRITE (formato embebido):

READ(u,'(especificadores de formato)') lista de variáveis

WRITE(u,'(especificadores de formato)') lista de variáveis

Exemplo: Os conjuntos de instruções F77 seguintes são equivalentes:

```
        READ (8, 1200, END=10) X, Y, Z
          bloco de instruções
10      CONTINUE
          bloco de instruções
1200    FORMAT (F5.3, 2x, F14.6, 2x, I4)
        END
```

```
        READ (8, '(F5.3, 2x, F14.6, 2x, I4)', END=10) X, Y, Z
          bloco de instruções
10      CONTINUE
          bloco de instruções
        END
```

8. FICHEIROS

Um ficheiro é um conjunto de registos, ou seja, uma sequência (formatada ou não) de caracteres e/ou valores. Para que um ficheiro possa ser usado como unidade de leitura ou escrita, este terá que ser "ligado" ao programa, de forma a que este reconheça a sua existência ou proceda à sua criação. A "abertura" ou "fecho" de um ficheiro num programa F77 é feita usando as instruções OPEN e CLOSE, respectivamente. Estas instruções têm a forma geral:

```
OPEN (lista de especificadores)
CLOSE (lista de especificadores)
```

Da lista de especificadores, são de realçar :

1) UNIT = u

2) FILE = fn

   fn é o nome do ficheiro (variável caracter).

3) STATUS = st

- st pode assumir a designação 'NEW', 'OLD', 'UNKNOWN', 'SCRATCH'. Um ficheiro 'NEW' não poderá existir. Um ficheiro 'OLD' tem que existir. Um ficheiro 'UNKNOWN' pode ou não existir. Um ficheiro 'SCRATCH' é um ficheiro temporário a usar pelo programa, que será apagado automaticamente após a finalização do programa. No caso da instrução CLOSE, o STATUS pode ainda incluir 'DELETE', para apagar o ficheiro.

4) ERR = s

5) IOSTAT = ios

Os especificadores UNIT, ERR e IOSTAT são idênticos aos usados na instrução READ.

Para além das instruções de OPEN e CLOSE, existem ainda duas instruções que permitem leituras especiais num ficheiro:

```
REWIND (u)
BACKSPACE (u)
```

REWIND conduz ao início do ficheiro, enquanto que BACKSPACE repete a leitura do último registo.

9. ORGANIZAÇÃO DE UM PROGRAMA FORTRAN

- Declarações iniciais

PROGRAM, FUNCTION ou SUBROUTINE

- LINHAS DE COMENTÁRIO (podem ser incluídas em qualquer ponto do programa)
- IMPLICIT
- PARAMETER
- Declarações explícitas

INTEGER

REAL

DOUBLE PRECISION

COMPLEX

CHARACTER

LOGICAL

- DATA
- instruções
- FORMAT (podem ser incluídas em qualquer ponto do programa)
- RETURN (se SUBROUTINE)
- END