Algoritmo: procedimento mecânico (automático) que produz sempre o mesmo resultado ao fim de um número finito de passos.

Programa FORTRAN: texto em que cada linha corresponde a uma instrução, e que é lido de cima para baixo. Implementa um determinado algoritmo fazendo corresponder a uma determinada sequência de passos uma sequência de instruções.

Manipulação de informação:

- Entrada de Informação (leitura de dados).

exterior $\xrightarrow{\text{canal de entrada}}$ memória

- manipulação de informação (cálculo de expressões envolvendo elementos dos tipos básico).

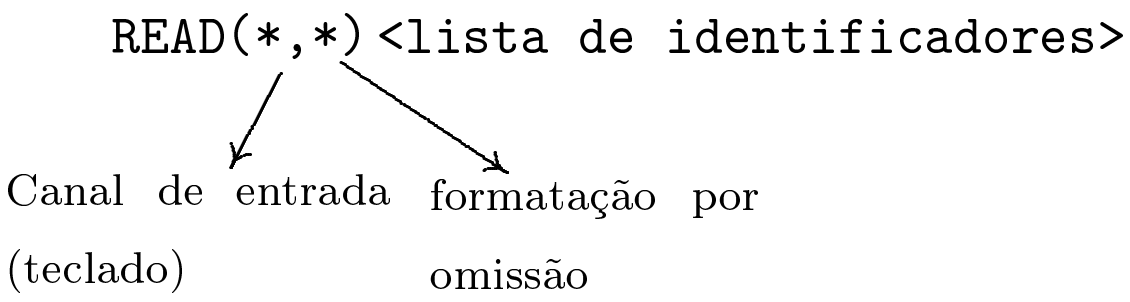
memória $\longrightarrow$ CPU $\longrightarrow$ memória

- saída de informação (escrita de resultados).

memória $\xrightarrow{\text{canal de saída}}$ exterior

Leitura/escrita (simplificada)

Leitura



- Há uma correspondência entre os elementos da lista de identificadores de uma instrução de leitura e os valores entrados (através de um dispositivo de entrada (teclado, ...)), tanto na ordem como no tipo.
- Os elementos a introduzir (teclado, ...) são separados entre si por vírgulas ou por espaços.
- Os elementos do tipo `CHARACTER` têm de ser delimitados por plicas ou aspas.

Exemplo

```
INTEGER :: i,j
REAL :: a
CHARACTER(len=10) :: c

READ (*,*) i,j,c,a
```

Os valores têm de ser introduzidos da seguinte forma (com as variantes já referidas):

```
1,3,'Palavra',7.5
```

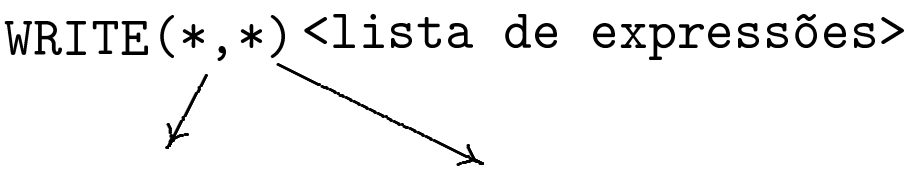
no fim da qual são atribuídos às diferentes variáveis os respectivos valores, ou seja:

```
i ← 1
j ← 3
c ← Palavra␣␣␣
a ← 7.5
```

é de notar que no caso da variável do tipo `CHARACTER` ela é "encostada" à esquerda sendo que as restantes posições são preenchidas com espaços.

Se se tentar introduzir um valor não compatível com o tipo da variável que lhe corresponde na instrução de entrada ocorre uma situação de erro.

Escrita



Canal de saída (ecrã) formatação por omissão

Exemplo

```
WRITE (*,*) 3,mm,15*3
```

O valor das variáveis não é afectado por esta instrução.

Escrita formatada

As leituras/escritas tal como foram descritas atrás são designadas por leituras/escritas em *formato livre (ou pré-definido)*, no FORTRAN é possível explicitar com bastante pormenor a formatação que se deseja. No que se segue vamo-nos concentrar nas escritas, dado que é aí que a formatação nos vai ser mais útil, no entanto o que se vai dizer também se aplica (com as necessárias adaptações) às leituras.

- Formato livre (pré-definido)

  `WRITE (*,*) <lista de expressões>`

- Formatação explícita

  `WRITE (*,<n>) <lista de expressões>`

em que o inteiro n tem de corresponder a uma etiqueta de uma dada instrução de formatação.

As instruções de formatação têm a seguinte forma:

`<n> FORMAT(<c><lista de "palavras" e de especificadores de formato>)`

em que c é um código de controle, vamos simplesmente utilizar o valor `1X`, o qual nos dá uma saída com um espaçamento simples.

Exemplo

```
   WRITE (*,100) x,y,res
100 FORMAT(1X,I3,'+',I3,'=',I3)
```

vai ter como resultado

```
␣␣3+␣␣7=␣10
```

As instruções de formatação podem ser colocadas em qualquer lugar no programa, pode inclusive usar-se um instrução de formatação para várias instruções de escrita, no entanto por questões metodológicas (facilidade de leitura) vamos optar por colocá-las sempre imediatamente após a instrução de escrita a que se refere.

Especificadores de Formato de Saída (entrada)

I - Formatação de saída para inteiros.

`<r>I<w>.<m>`

com:

r, factor de repetição, isto é um especificador de formato para r elementos consecutivos na instrução de escrita (leitura), opcional.

w, comprimento do campo de saída.

m, número mínimo de dígitos a visualizar, opcional.

Se o inteiro é demasiado comprido para o comprimento do campo o espaço referente ao campo de saída é preenchido com asteriscos.

Exemplo

```
   INTEGER :: i=-12,j=4,n=-12345
   WRITE (*,200) i,i+12,j,n
   WRITE (*,210) i,i+12,j,n
   WRITE (*,220) i,i+12,j,n
200 FORMAT(1X,2I5,I6,I10)
210 FORMAT(1X,2I5.0,I6,I10.8)
220 FORMAT(1X,2I5.3,I6,I5)
```

```
␣␣-12␣␣␣␣0␣␣␣␣␣4␣␣␣␣-12345
␣␣-12␣␣␣␣␣␣␣␣␣4␣-00012345
␣-012␣000␣␣␣␣4*****
```

Especificadores de Formato de Saída (entrada)

F - Formatação de saída para reais.

`<r>F<w>.<d>`

com:

r, factor de repetição, isto é um especificador de formato para r elementos consecutivos na instrução de escrita (leitura).

w, comprimento do campo de saída.

d, número de casas decimais.

Se o inteiro é demasiado comprido para o comprimento do campo o espaço referente ao campo de saída é preenchido com asteriscos.

E - Reais, notação científica ($\pm 0.dd\text{E}\pm ee$).

`<r>E<w>.<d>`

ES - Reais, notação científica ($\pm n.dd\text{E}\pm ee$).

`<r>ES<w>.<d>`

Exemplos:

```
    REAL :: x=-12.76,y=4.0,z=-123.5
    WRITE (*,200) x,y,z
    WRITE (*,210) x,y,z
    WRITE (*,220) x,y,z
200 FORMAT(1X,2F5.3,F9.3)
210 FORMAT(1X,2F7.3,E10.2)
220 FORMAT(1X,F7.2,F7.0,ES9.1)
```

```
  *****4.000 -123.500
  -12.760  4.000 -0.12E+03
   -12.76     4.  -1.2E+02
```

Especificadores de Formato de Saída (entrada)

A - Formatação de saída para caracteres.

`<r>A<w>`

com:

r, factor de repetição, isto é um especificador de formato para r elementos consecutivos na instrução de escrita (leitura), opcional.

w, comprimento do campo de saída, opcional. Caso não se especifique o valor de w o tamanho do texto a escrever (ler) é igual ao tamanho do valor da variável do tipo caracter dado.

O texto é encostado à direita.

Se o comprimento da sequência de caracteres é maior que o comprimento do campo o só os primeiros w caracteres da sequência é que serão escritos (lidos).

Exemplo:

```
   Character(len=7) :: seqcar='exemplo'
   WRITE (*,200) seqcar
   WRITE (*,210) seqcar
   WRITE (*,220) seqcar
200 FORMAT(1X,A)
210 FORMAT(1X,A10)
220 FORMAT(1X,A3)
```

```
exemplo
␣␣␣exemplo
exe
```

Atribuição

É a instrução mais importante dado que é ela quem nos dá acesso à memória permitindo, desse modo, manipular a informação.

`<identificador> = <expressão>`

O tipo da expressão têm de ser de um tipo compatível com o do identificador.

Leitura

identificador toma o valor de *expressão*.

Leitura no tempo

1. É feito o cálculo da *expressão*.
   memória $\longrightarrow$ CPU.
2. O valor calculado é guardado na posição de memória correspondente ao *identificador*.
   CPU $\longrightarrow$ memória.

Para explicitar esta leitura (no tempo) da direita para a esquerda vamos denotar esta instrução da seguinte forma:

`<identificador>` $\longleftarrow$ `<expressão>`

Problema:

Dados dois inteiros efectuar a sua soma.

Especificação:

⟶ x,y

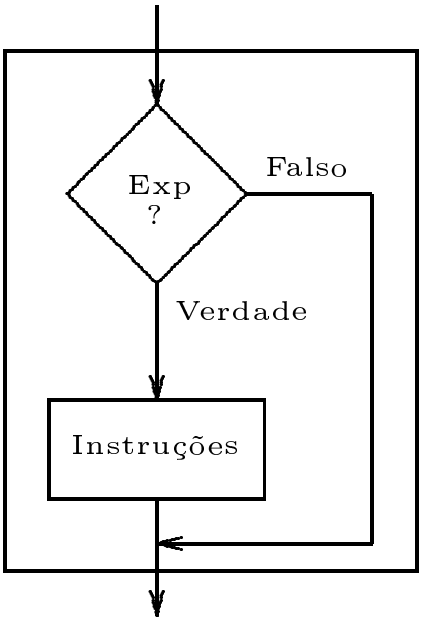
⟵ x+y

Programa:

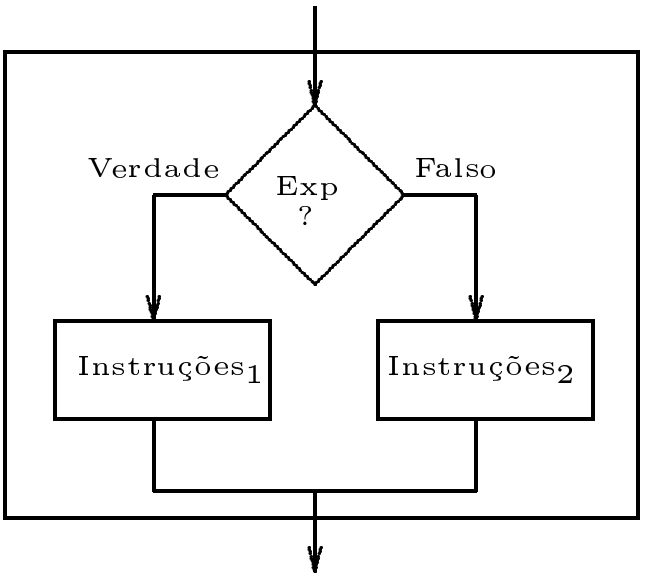
```
PROGRAM somar

 IMPLICIT NONE
                                          x → [XXX]
 INTEGER :: x,y,res                       y → [XXX]
                                        res → [XXX]
 WRITE (*,*) "Introduza dois inteiros"
 READ (*,*) x,y                           x → [4]   ⟵ 4,5
                                          y → [5]
                                        res → [XXX]

 res = x+y                                x → [4]
                                          y → [5]
                                        res → [9]

 WRITE (*,*) x,"+",y,"=",res              x → [4]   ⟶ 9
                                          y → [5]
                                        res → [XXX]
END PROGRAM
```

Selecção

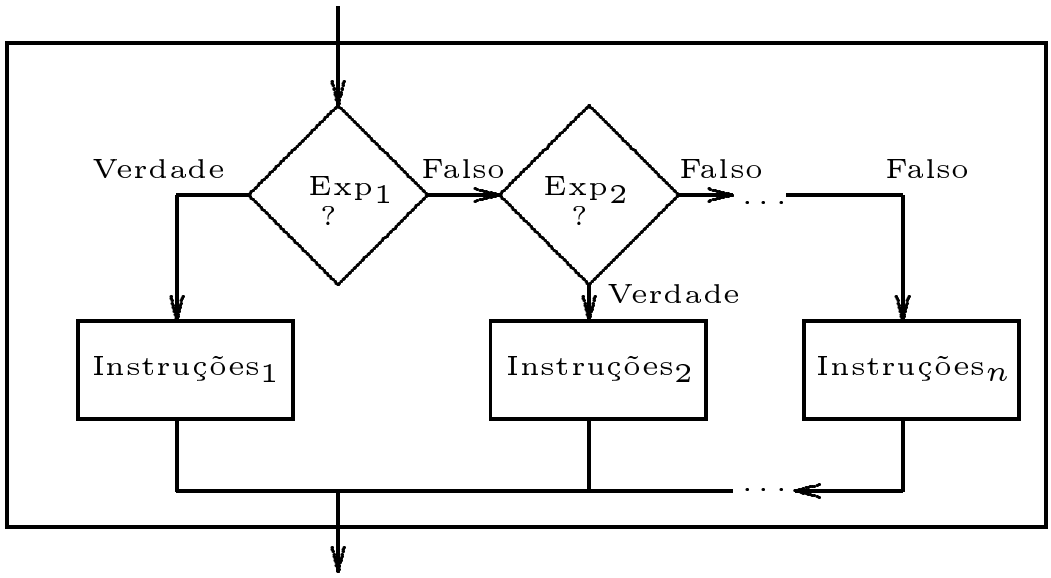
Alternativa dicotómica (verdade/falso):




```
IF (<exp?>) THEN
   <instrução1>
   .
   .
   .
   <instruçãoN>
END IF
```




```
IF (<exp?>) THEN
   <instrução1.1>
   .
   .
   .
   <instrução1.N>
ELSE
   <instrução2.1>
   .
   .
   .
   <instrução2.N>
END IF
```



Para o caso de um só ramo "ELSE IF".

```
IF (<exp1?>) THEN
   <instrução1.1>
   :
   <instrução1.N>
ELSE IF (<exp2?>) THEN
        <instrução2.1>
        :
        <instrução2.N>
     ELSE
        <instrução3.1>
        :
        <instrução3.N>
END IF
```

É de notar que esta última variante é equivalente à forma (preferível):

```
IF (<exp1?>) THEN
   <instrução1.1>
   :
   <instrução1.N>
ELSE
   IF (<exp2?>) THEN
       <instrução2.1>
       :
       <instrução2.N>
   ELSE
       <instrução3.1>
       :
       <instrução3.N>
   END IF
END IF
```

no qual se coloca um "IF" dentro de um outro "IF".

Problema:

Calcular as raízes reais de uma equação do segundo grau.

Especificação:

$\longrightarrow a, b, c$

$\longleftarrow r_1, r_2$

Algoritmo:

```
alg
  binómio ← b^2 − 4ac
  se  binómio < 0 então
       raízes imaginárias (1)
  senão
       se  binómio = 0 então
            raiz real dupla (2)
       senão
           raízes reais (duas) simples (3)
       fimse
  fimse
fimalg
```

O "aninhar" de condicionais dentro de outros condicionais tem o efeito de uma conjunção, ou seja nos vários ramos temos:

(1) binómio < 0

(2) $\neg$ (binómio < 0) $\wedge$ (binómio $= 0$) $\Leftrightarrow$ binómio $= 0$

(3) $\neg$ (binómio < 0) $\wedge \neg$ (binómio $= 0$) $\Leftrightarrow$ binómio > 0

O Programa:

```
PROGRAM equacao2grau

 IMPLICIT NONE

 REAL :: a,b,c
 REAL :: x1,x2,binomio

 WRITE (*,*) "Introduza a,b,c: "
 READ (*,*) a,b,c

 binomio = b**2-4*a*c
 IF (binomio < 0) THEN
    WRITE (*,*) "Raizes Imaginarias"
 ELSE
    IF (binomio == 0) THEN
       x1 = -b/(2*a)
       WRITE (*,*) "Uma raiz real dupla, ",x1
    ELSE
       x1 = (-b+SQRT(binomio))/(2*a)
       x2 = (-b-SQRT(binomio))/(2*a)
       WRITE (*,*) "Duas raizes reais, ",x1,x2
    END IF
 END IF
END PROGRAM
```

Problema:

Dado três números inteiros positivos determinar se poderão ser considerados como os lados de um triângulo, Em caso afirmativo indicar o tipo e a área.

Notas:

- num triângulo a soma de dois lados é sempre maior do que o outro.
- três lados iguais, é um triângulo equilátero.
- só dois lados iguais, é um triângulo isósceles.
- todos os lados diferentes, é um triângulo escaleno.
- área $= \sqrt{(s(s-a)(s-b)(s-c)}$, com $s = (a+b+c)/2$

Especificação:

$\longrightarrow l1, l2, l3$

$\longleftarrow$ classificação do triângulo e cálculo da área (se possível).

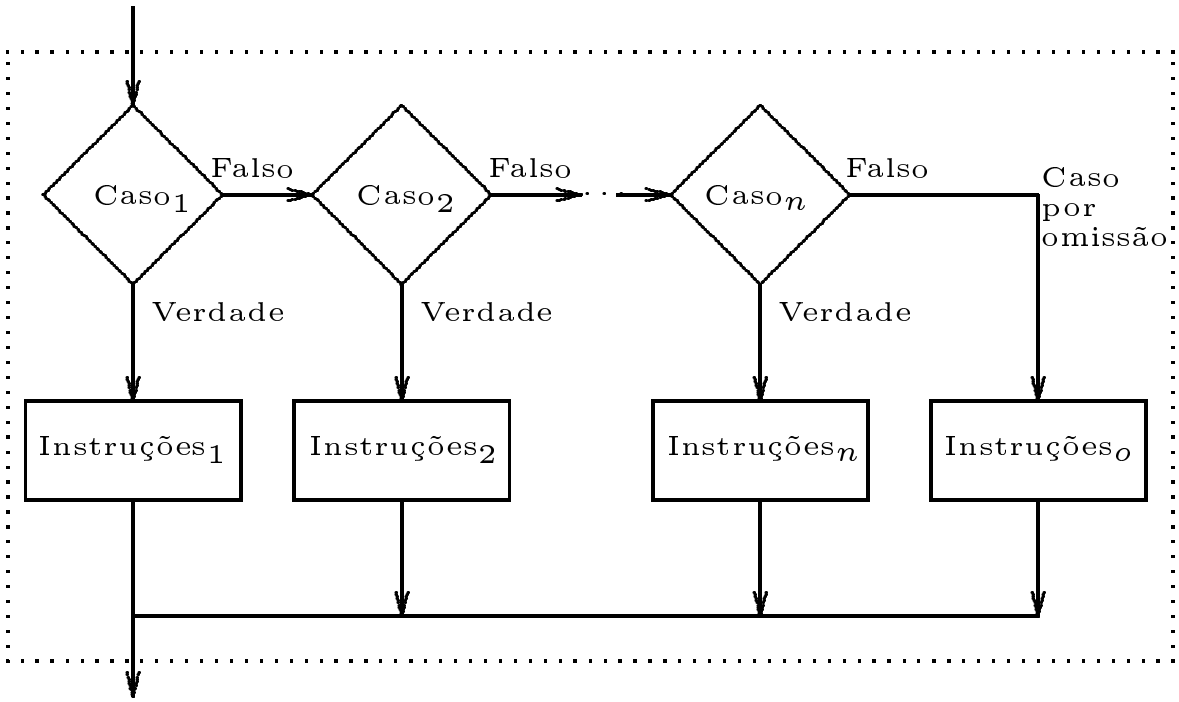
Algoritmo:

```
alg
 ler(l1,l2,l3)
 triângulo ← verdadeiro
 se  (l1+l2>l3)∧(l1+l3>l2)∧(l2+l3>l1) então
    s ← (l1+l2+l3)/2
    área ← √(s(s−l1)(s−l2)(s−l3))
    se  (l1=l2)∨(l1=l3)∨(l2=l3) então
       se  (l1=l2)∧(l2=l3) então
          classifica ← "equilátero"
       senão
          classifica ← "isósceles"
       fimse
    senão
       classifica ← "escaleno"
    fimse
  senão
       triângulo ← falso
  fimse
  se  triângulo então
       escrever(classifica,área)
  senão
       escrever("Não é um triângulo")
  fimse
fimalg
```

O Programa

```
PROGRAM triangulos

 IMPLICIT NONE

 INTEGER :: l1,l2,l3
 LOGICAL :: triangulo
 REAL :: s,area
 CHARACTER(len=10) :: classifica

 WRITE (*,*) 'Introduza tres inteiros '
 READ (*,*) l1,l2,l3

 triangulo = .TRUE.
 IF ((l1+l2>l3) .AND. (l1+l3>l2) .AND. (l2+l3>l1)) THEN
    s = (l1+l2+l3)/2.0
    area = SQRT(s*(s-l1)*(s-l2)*(s-l3))
    IF ((l1==l2) .OR. (l1==l3) .OR. (l2==l3)) THEN
       IF ((l1==l1) .AND. (l2==l3)) THEN
          classifica = 'Equilatero'
       ELSE
          classifica = 'Isosceles'
       END IF
    ELSE
       classifica = 'Escaleno'
    END IF
 ELSE
    triangulo = .FALSE.
 END IF
 IF (triangulo) THEN
    WRITE (*,110) classifica,area
110 FORMAT(1X,"E' um triangulo ",A10," com area ",F10.4)
 ELSE
    WRITE (*,*) "Nao e' um triangulo"
 END IF
END PROGRAM triangulos
```

Selecção Múltipla




```
SELECT CASE (<expressão>)
CASE (<selector1>)
   <instrução1.1>
   .
   .
   .
   <instrução1.M>
CASE (<selector2>)
   <instrução2.1>
   .
   .
   .
   <instrução2.M>
...
CASE (<selectorN>)
   <instruçãoN.1>
   .
   .
   .
   <instruçãoN.M>
CASE DEFAULT
   <instruçãoO.1>
   .
   .
   .
   <instruçãoO.M>
END SELECT
```

Selecção de um entre vários casos

Notas:

- A *expressão* define uma gama de valores dos tipos `INTEGER`, `CHARACTER`, ou `LOGICAL`.
- Os diferentes caso têm de ser mutuamente exclusivos (disjuntos dois a dois).
- A *expressão* e os *selectores* têm de ser do mesmo tipo.
- A forma de os diferentes casos definirem sub-gamas de variação da *expressão* é dada por:
  - `valor` $\mapsto \{x \in expressão \mid x = \text{valor}\}$
  - `:valor` $\mapsto \{x \in expressão \mid x \leq \text{valor}\}$
  - `valor:` $\mapsto \{x \in expressão \mid x \geq \text{valor}\}$
  - `valor1:valor2` $\mapsto \{x \in expressão \mid \text{valor1} \leq x \leq \text{valor2}\}$
- O `CASE DEFAULT` define o caso por omissão o qual se verifica quando todos os outros falham. No caso de não estar presente (este caso é opcional) então no caso de um valor não coberto pelos selectores a instrução `SELECT CASE` não é executada.

Problema: Escreva um programa que simule uma máquina de calcular capaz de aceitar as operações aritméticas elementares, e que execute um só operação de cada vez.

Especificação:

⟶ operando1, operador, operando2 (um inteiro, um caracter '+', ou '-', ou 'x', ou '/', e um inteiro).

⟵ operando1 operador operando2 (isto é o resultado da aplicação do operador aos dois operandos).

Algoritmo

```
alg
 ler(x1,op,x2)
 erro ← falso
 caso (op) seja
  ('+')
     res ← x1+x2
  ('-')
     res ← x1-x2
  ('x')
     res ← x1*x2
  ('/')
     res ← x1/x2
  por omissão
     erro ← verdade
 fimcaso
 se  erro então
    escrever("Operador não considerado")
 senão
    escrever(res)
 fimse
fimalg
```

O Programa:

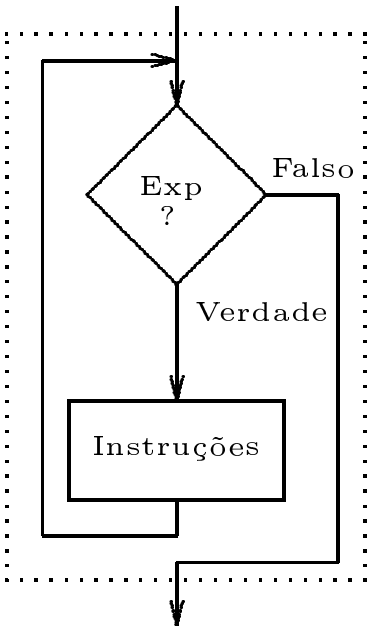
```
PROGRAM maqCalcV1

  IMPLICIT NONE

  INTEGER :: x1,x2,res
  CHARACTER :: op
  LOGICAL :: erro

  WRITE (*,*) "Introduza um inteiro, um sinal de &
       operacao e um outro inteiro"
  READ (*,*) x1,op,x2

  SELECT CASE (op)
     CASE ('+')
        res = x1+x2
     CASE ('-')
        res = x1-x2
     CASE ('x')
        res = x1*x2
     CASE ('/')
        res = x1/x2
     CASE default
        erro = .TRUE.
  END SELECT
  IF (erro) THEN
     WRITE (*,*) "Operador nao considerado"
  ELSE
     WRITE (*,100) x1,op,x2,res
100  FORMAT(1X,I3,A1,I3,"=",I5)
  END IF
END PROGRAM maqCalcV1
```

Repetição (ciclos)

Repetição de um grupo de instruções sujeitas a uma condição de paragem.

- Explorar a velocidade da máquina para fazer um cálculo repetitivo.
- É necessário ter em conta que uma condição de paragem mal definida pode dar origem a um ciclo infinito.

Ciclo **Enquanto** (‘‘`DO WHILE`’’)



enquanto a *expressão* lógica é verdadeira repetem-se as *instruções* do corpo do ciclo.

Em FORTRAN

```
DO WHILE (<expressão>)
   <instruções>
END DO
```

Um ciclo é uma instrução composta na qual se pretende efectuar um determinado cálculo, temos então de analisar com cuidado:

- **a condição inicial**, isto é, se as variáveis que vão entrar no cálculo repetitivo têm os seus valores iniciais correctos;
- **o invariante do ciclo**, isto é, a expressão matemática cuja validade se mantém inalterada durante o decorrer do funcionamento do ciclo, e que expressa o que se pretende calcular;
- **a condição de paragem**, isto é, a expressão lógica que estabelece a condição de paragem do ciclo. De igual modo a condição de paragem estabelece as condições finais, isto é, o valor final que as variáveis que vão entrar no cálculo repetitivo vão ter no fim do mesmo.

Exemplo:

Problema: calcular $1 + 2 + 3 + 4 + \cdots + 100$

Especificação:

$\longrightarrow$

$\longleftarrow \sum_{i=1}^{100} i$

- condição inicial, $k = 1 \wedge s = \sum_{i=1}^{k} i$
- invariante do ciclo $s = \sum_{i=1}^{k} i$, com $1 \leq k \leq 100$, isto é a cada passagem (repetição) no ciclo vai-se adicionar mais uma parcela ao somatório.
- condição de paragem, $k = 100$

Se "somarmos" a condição de paragem ao invariante do ciclo (e à condição inicial) obtemos:

$$s = \sum_{i=1}^{k} i, \text{com } 1 \leq k \leq 100 \wedge k = 100 \Leftrightarrow s = \sum_{i=1}^{100} i$$

em FORTRAN temos

```
i=1
somatorio=0
DO WHILE (i<=100)
  somatorio=somatorio+i
  i=i+1
END DO
```

Outro Exemplo:

Problema: Dado uma lista de inteiros positivos encontrar o elemento da lista de maior valor.

Especificação:

$\longrightarrow$ lista de inteiros positivos (a serem lidos do teclado). $\longleftarrow$ max (o elemento da lista de maior valor).

¿ Quantos elementos têm a lista ?

Na definição do comprimento de uma lista temos de encarar duas soluções:

- número pré-definido, por exemplo: **uma lista com n elementos**;
- existência de uma *sentinela*, isto é, de um elemento pertencente à lista de elementos a serem lidos e que determina o fim da leitura, consoante os casos esse elemento pode, ou não, pertencer à lista de elementos. Por exemplo **uma lista de inteiros positivos**, a sentinela pode ser um qualquer número negativo o qual não vai fazer parte da lista de elementos a considerar.

Algoritmo

Condição Inicial O primeiro elemento a ser lido pode ser considerado o maior (a lista tem de conter pelo menos um elemento).

Invariante: `max` é o maior dos elementos lidos até ao momento.

Condição de paragem: prossegue-se a leitura até se ler um número inteiro negativo.

Temos então:

```
alg
  ler(elem)
  max<-elem
  enquanto elem < 0 faz
     se  elem > max então
        max <-elem
     fimse
     ler(elem)
  fimenquanto
  escrever(max)
fimalg
```

Repetição (ciclos)

Repetição de um grupo de instruções um número pré-determinado de vezes.

$$\sum_{i=0}^{n} P_i(x) = \underbrace{P_0(x) + P_1(x) + \cdots + P_n(x)}_{n+1 \text{ parcelas}}$$

início $i = 0$, fim $i = n$, incremento (implícito) $+1$.

isto é temos de fazer o cálculo

```
soma←soma+parcela
```

$n + 1$ vezes.

```
para i←0 até n incremento 1
  soma←soma+parcela_i
fimpara
```

Em FORTRAN

```
DO <variável controlo>=<inicio>,<fim>[,<incremento>]
  <instruções>
END DO
```

Notas (para o ciclo `DO`)

- O parâmetro opcional `incremento` é opcional, o seu valor por omissão é 1.
- Os parâmetros (`início,fim,incremento`) podem ser constantes, ou variáveis, ou expressões. O número de iterações que vão ser efectuadas é calculada no inicio do ciclo e não volta a ser recalculado.
- Se `início*incremento <= fim*incremento` o ciclo é executado, caso contrário o ciclo é equivalente à instrução nula.

Exemplos

$$\prod_{i=1}^{n} x = x^n = \underbrace{x \times x \times \cdots \times x}_{n \text{ vezes}}$$

```
potencia=1
DO i=1,n
  potencia=potencia*x
END DO
```

$$\prod_{i=1}^{n} i = n! = 1 \times 2 \times \cdots \times n$$

```
factorial=1
DO i=1,n
  factorial=factorial*i
END DO
```

(mais) Notas:

- A variável de controlo do ciclo nunca deve ser modificada no interior do ciclo.

  ```
  DO i=1,n
    i=2  XXX ERRO XXX
  END DO
  ```

  Os compiladores (em geral) detectam estas situações assinalando-as.

- A variável de controlo tem de ser do tipo `INTEGER`.

- Após o terminar (normal) de um ciclo `DO` incremental, o valor da variável de controlo não está definido.