2013

Portugol

Equivalências de estruturas entre Portugol e Fortran(95)

Decode Team
INSTITUTO POLITÉCNICO DE TOMAR

Índice

Nota Geral:

Devido à especificação da linguagem, a tradução só é possível depois de ser executado o fluxograma.

Algumas notas sobre Fortran:

- Não é case sensitive.
- Não necessita de qualquer pontuação para terminar uma linha de codigo.
- As funções podem ser definidas antes ou depois no main.
- O código dever ser guardado num ficheiro com extensão *.f95*.

Estrutura Início

Início:

Em Fortran 95, o inicio de um programa não é definido, logo o programa começa onde a primeira variável é definida(caso sejam usadas variaveis), ou onde o primeiro output ou input apareça.

Apesar disso, podemos começar um programa por :

Program programa

Apesar disto não fazer diferença do inicio do programa, é algo que nos ajuda a identificar o programa.

Nota: O nome do programa (*Programa*), é um nome que queremos dar ao programa.

Nota2: o comando : implicit none , é um comando que impede a possibilidade de haver nomes de variáveis não definidos. Não é obrigatório usar, mas é aconselhável que o seja feito.

Este comando a ser usado, é usado logo no inicio do código

Program programa

implicit none

Estrutura Fim

Fim:

end

Variáveis

Equivalência entre TIPOS de variáveis

TIPO	Portugol	Fortran
Inteiro	Inteiro	integer
Real	Real	real
Texto	Texto	character
Caracter	Caracter	character
Lógico	Logico	logical

Tabela 1 - Tipos de variáveis

Definição e atribuição de variáveis

Nota importante de definição de variaveis

Existem duas formas para definir uma variável :

Integer :: n

Ou

Integer n

Definiçao de variáveis com e sem implicit none

Exemplo 1:

```
Program exemplo
read *,n
print *,n
end
```

O exemplo 1 vai receber um valor do teclado, e escrever para o ecrã, sem que seja necessário definir a variável n

Exemplo 2:

```
Program exemplo
implicit none
integer :: n
read *,n
print *,n
end
```

O exemplo 2, como tem o comando implicit none, é obrigatório definir a variável n.

Se a variável não for definida, o código da erro a compilar.

variavel <- expressao

Se a variável não estiver definida em memória

Passo 1: Avaliar a expressão (VALOR).

Passo 2: Calcular Tipo do VALOR.

Passo 3: Declarar a variável: TIPO variavel = expressao;

Se a variável estiver definida em memória

variavel = expressao

Alguns exemplos de definição e atribuição de variáveis

Existem duas formas de definir variáveis e proceder à sua atribuição.

- integer

  1 – Definir e atribuir variável no mesmo passo:

  Integer :: variavel =valor

  2 – Definir e atribuir variável em passos separados:

  integer :: variavel

  Variável=valor

  Nota 1: Pode ser definido como *integer ou Integer.*

  Nota 2: *valor* é um número inteiro.

- real

  1 – Definir e atribuir variável no mesmo passo:

  Real :: variavel =valor

  2 – Definir e atribuir variável em passos separados:

  Real :: variavel

  variavel=valor

  Nota 1: Pode ser definido como *real* ou *Real*

  Nota 2: *valor* é um número decimal. Ex: 5.3.

- Character

  1 – Definir e atribuir variável no mesmo passo:

  Character(len=comprimento) :: variavel="valor"

  2 – Definir e atribuir variável em passos separados:

  Character(len=comprimento) :: variavel
  variavel=”valor”

Nota 1: comprimento é o numero de caracteres que vamos usar
Nota 2: têm de ser usadas aspas.

- logical

1 – Definir e atribuir variável no mesmo passo:

Logical :: variavel =.false.

2 – Definir e atribuir variável em passos separados:

Logical :: variavel

variavel=.false.

Nota 1: Pode ser definido como *Logical* ou *logical.*

Nota 2: Este tipo de dados pode assumir o valor *true* ou *false.*

Estruturas input/output

Input – Ler

Tipo	fortran
Real	Double
Texto	Line
Lógico	Boolean
INT	Long
Char	Char

Tabela 2 - Tipo de variáveis para leitura

Se a variável não estiver definida em memória

NOTA: apenas é necessário definir a variável caso usemos o comando *implicit none* no inicio do programa

Passo 1: Identificar o tipo (TIPO) de dados que foi lido.

Passo 2: Definir a variável:

TIPO :: variavel

Passo 3: read *, variavel

Se a variável já estiver definida em memória

Passo 1: Realizar apenas o ***Passo 3*** do ponto anterior.

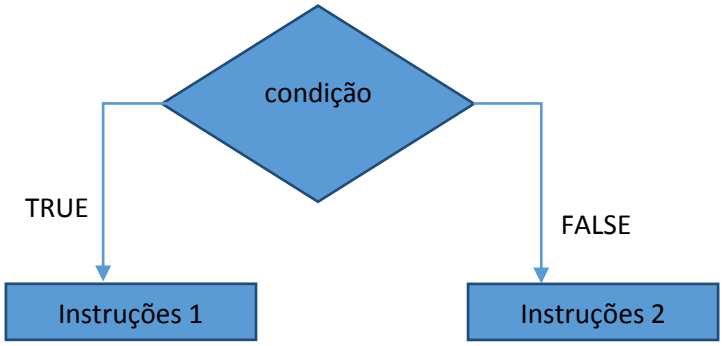
Output – Escrever

Para escrever no ecrã:

Print *, expressao

Estruturas de Decisão

Condição *"if"* e *"if else"*



Para TRUE, escrever:

If (*condição*) then

Instruções 1

Para FALSE:

Se Instruções 2 for igual a ○ (conector) não fazer nada.

Senão, escrever:

else

Instruções 2

Exemplos práticos

Condição "if"

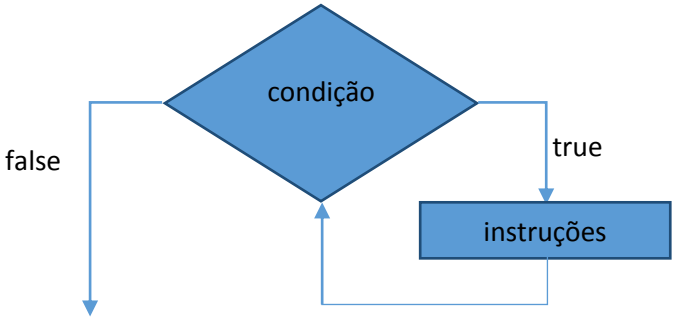
```
if(mod(n,2)==0) then
 print *,"PAR"
endif
```

Condição "if else"

```
if(mod(n,2)==0) then
print *,"PAR"
else
print *,"IMPAR"
endif
```

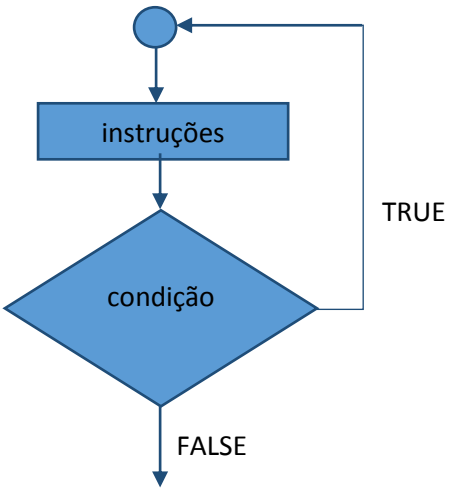
Condição *"while"*

Nota: os ciclos *while* e *do while*, funcionam da mesma forma, não há diferença, logo só vai ser mostrado o ciclo While.




```
While (condição) do
    Instruções
Endwhile
```

Condição *"do while"*



Instruções

```
if (condiçao) exit
enddo
```

Nota : Em fortran no ciclo do while, a condição tem que ser feita como se fosse uma condição de um ciclo IF, ou seja, as instruções vão ser executas se a condição não se verificar, caso se verifique, acaba o ciclo.

Exemplos práticos

Condição "while"

```
while(l<=10) do
        print *,l
 l=l+1
endwhile
```

Estrutura Conector

Conector

Se for uma condição "*do while*" escrever:

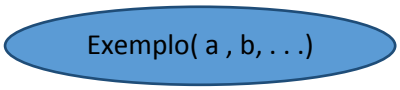
do

Senão, escrever:

Endif

Funções

Definir funções



Nota: Depois da função ser executada pelo menos uma vez o tipo de retorno das função RETURN_TIPO e o TIPOx dos parametros pode ser identificado:

```
RETURN_TIPO function nome( TIPO1 a , TIPO2 b, . . .. )
```

Definir função *Exemplo* sem parâmetros de entrada

```
TIPO function nome ()
```

Definir função *Exemplo* com parâmetros de entrada

```
TIPO function nome(PARAMETRO)
```

TIPO – Executa a função e calcula o tipo de retorno.
Consultar *tabela 1* no ponto *Equivalência entre TIPOS de variáveis*.

NOME – Nome dado à função.

PARAMETRO – Variável utilizada pela função para auxiliar o cálculo.

Chamada de funções

Variavel<-NOME(PARAMETRO)

Variável=NOME(PARAMETRO)

Exemplos do uso de funções

```
 PROGRAM calculo
REAL :: a,b,c
REAL :: av
REAL :: media
a=2.0
b=3.0
c=5.0
AV = media(A,B,C)
PRINT *,”media :”,AV
END

   REAL FUNCTION media(X,Y,Z)
   REAL ::  X,Y,Z,SUM,x
   SUM = X + Y + Z
   media = SUM /3.0

   END
```

Outro tipo de função são as funções recursivas:

Exemplo:

```
recursive integer function fact(k) result(res)
!implicit none
!integer  res,k

if(k>2) then
        res=k*fact(k-1)
 else
       res=k
 end if

 endfunction
```

Para definir funções do tipo recursivo :

recursive TIPO function nome(PARAMETRO) **result(variavel)**

recursive : temos que escrever no inicio da definição da função de forma a que esta funcione de forma recursiva

result(variavel) : a clausula result é obrigatória no tipo de funções recursivas, e o valor da variável é o valor que iremos retornar.

Estrutura de retorno

Return

NOME_DA_FUNÇÂO = expressao

Operadores

Aritméticos

Nome	Portugol	fortran
Adição	a + b	a + b
Subtração	a – b	a - b
Divisão	a / b	a / b
Multiplicação	a * b	a * b
Resto da divisão inteira		Mod(a,2)
Potenciação		Base ** expoente
Concatenação de texto		,

Tabela 3 - Equivalência de operadores aritméticos

Lógicos

Nome	Portugol	fortran
Disjunção	a && b	a .and. b
Conjunção	a || b	a .or. b
Conjunção Exclusiva	a ^ b	
Negação		a.not.b

Tabela 4 - Equivalência de operadores lógicos

Relacionais

Nome	Portugol	Java
Igual	a == b	a == b
Diferente	a != b	a /= b
Maior	a > b	a > b
Maior ou igual	a >= b	a >= b
Menor	a < b	a < b
Menor ou igual	a <= b	a <= b

Tabela 5 - Equivalência de operadores relacionais

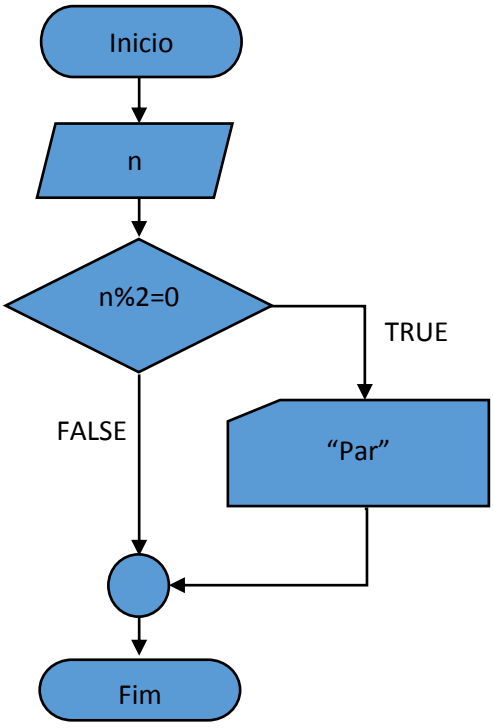
ANEXO

Para uma compreensão mais abrangente do uso das estruturas, ficam alguns exemplos mais extensivos, com o uso de várias estruturas em algoritmos completos.

Algoritmo com o uso da condição "if"

Problema: Verificar se um número introduzido pelo utilizador é par.

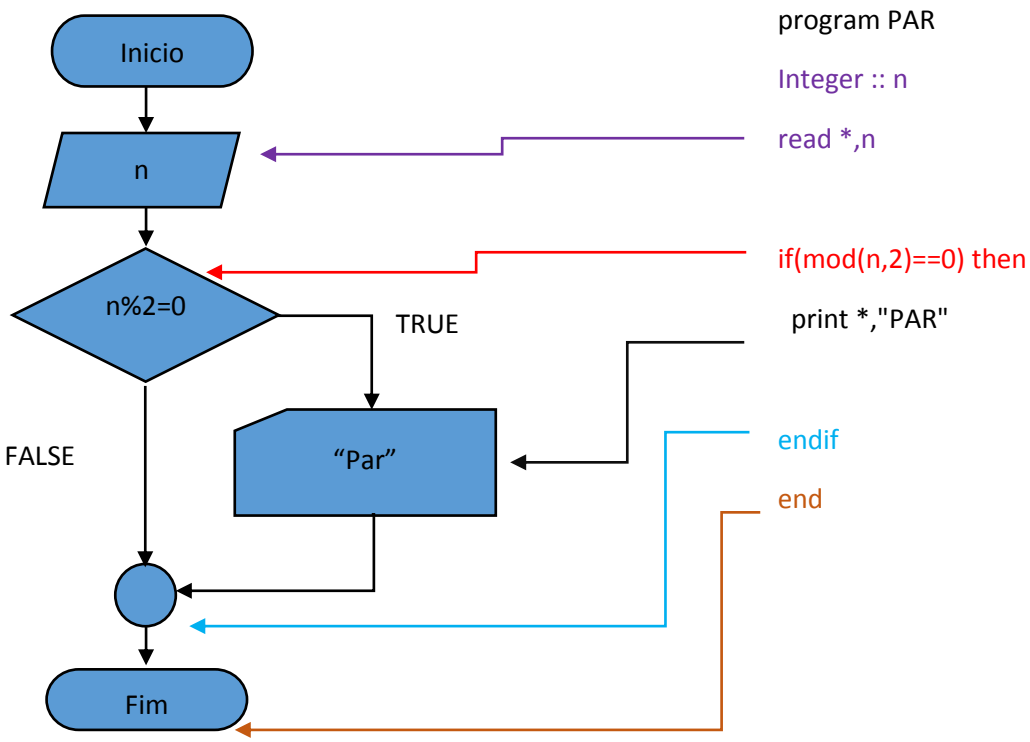
Fluxograma



Código

```
program PAR

Integer :: n

read *,n

if(mod(n,2)==0) then

  print *,"PAR"

endif

end
```

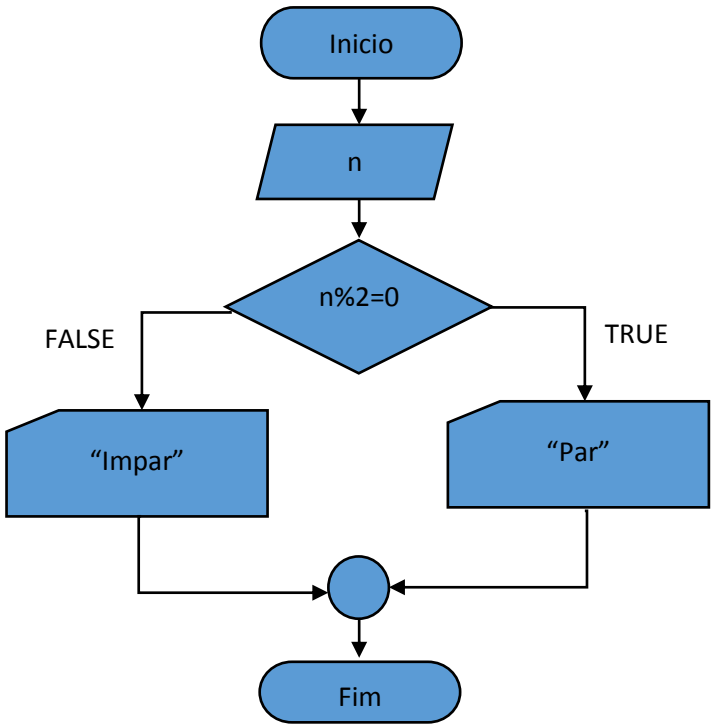
Esquema detalhado



Algoritmo com o uso da condição “if else”

Problema: Verificar se um número introduzido pelo utilizador é par ou ímpar.

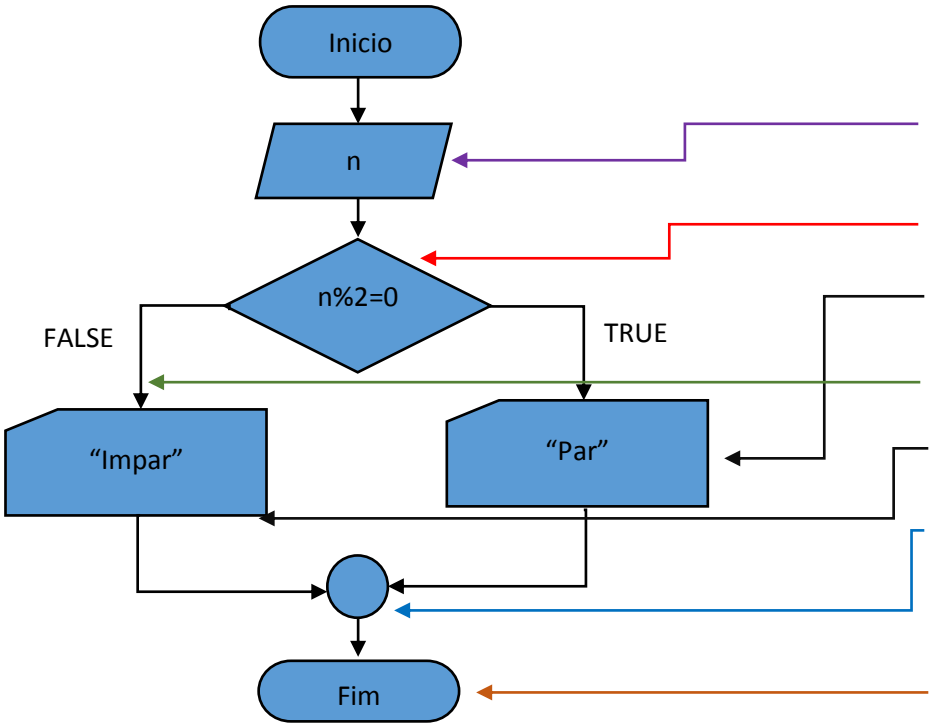
Fluxograma



Código:

```
program ParOuImpar
Integer :: n
read *,n
if(mod(n,2)==0) then
  print *,"PAR"
else
  print *,"IMPAR"
endif
end
```

Esquema detalhado

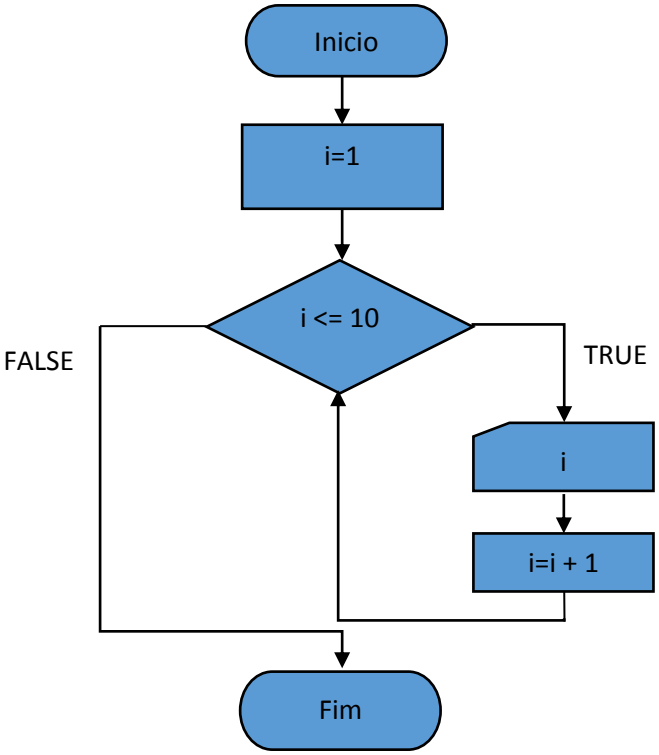



```
program ParOuImpar

Integer :: n
read *,n

if(mod(n,2)==0) then

  print *,"PAR"

else

  print *,"IMPAR"

endif


end
```

Algoritmo com o uso da condição “while”

Problema: Escrever um número de 1 a 10.

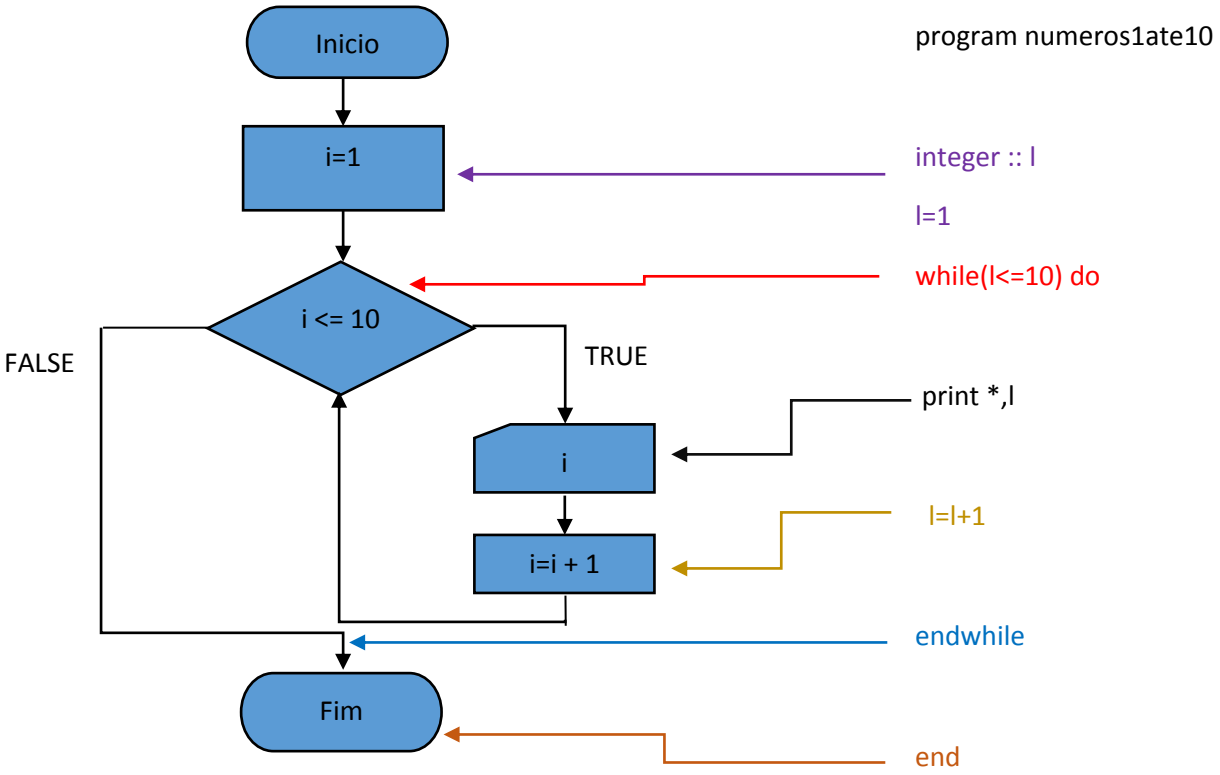
Fluxograma



Código

```
program numeros1ate10

integer :: l

l=1

while(l<=10) do

  print *,l

   l=l+1

endwhile

end
```

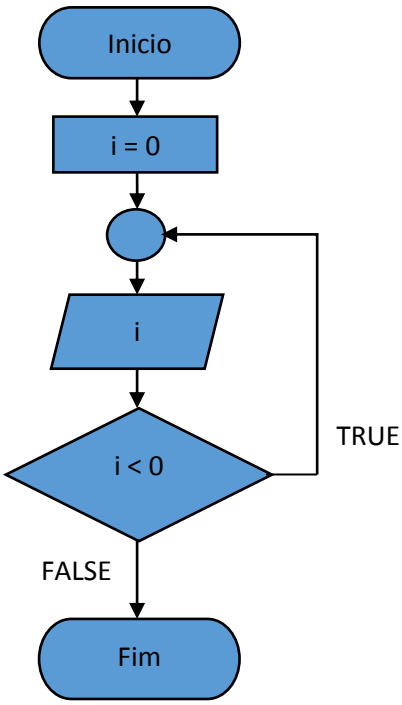
Esquema detalhado



Algoritmo com o uso da condição " do while"

Problema: Pedir um número positivo.

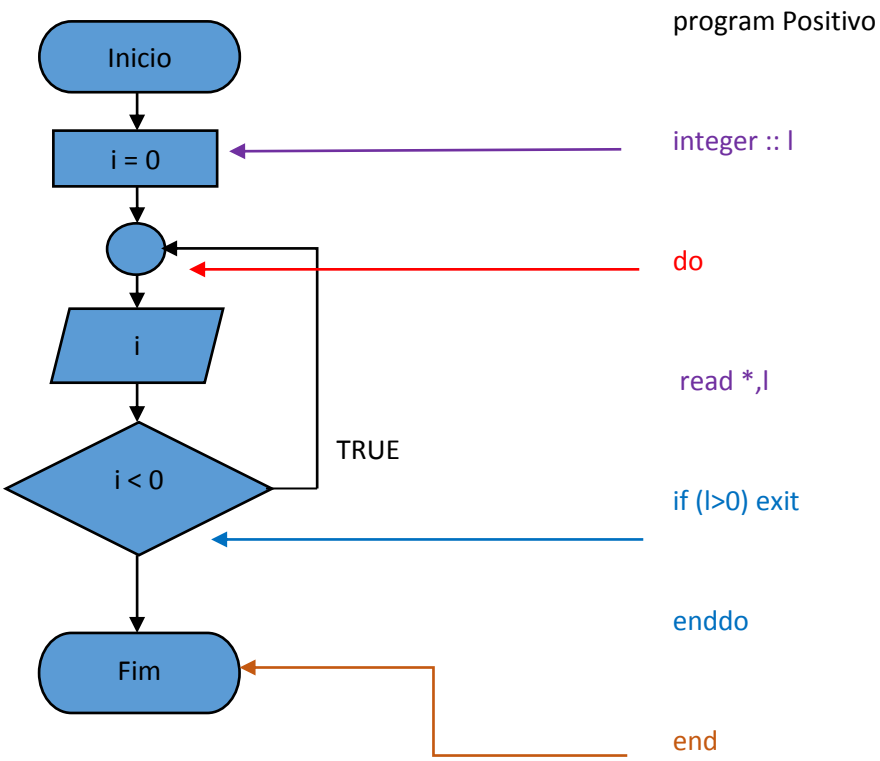
Fluxograma



Código

```
program Positivo
integer :: I
do
  read *,I
if (I>0) exit
enddo
end
```

Esquema detalhado

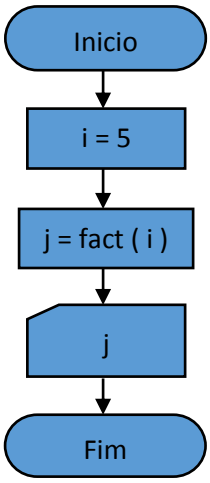


Algoritmo com o uso de uma função

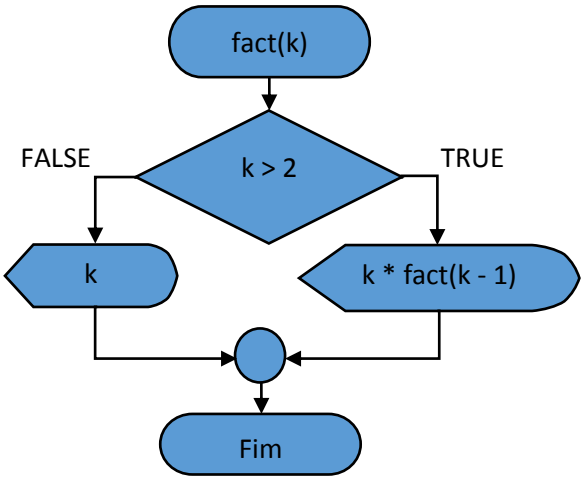
Problema: Factorial de um número.

Fluxogramas

Código principal



Função fact(k)



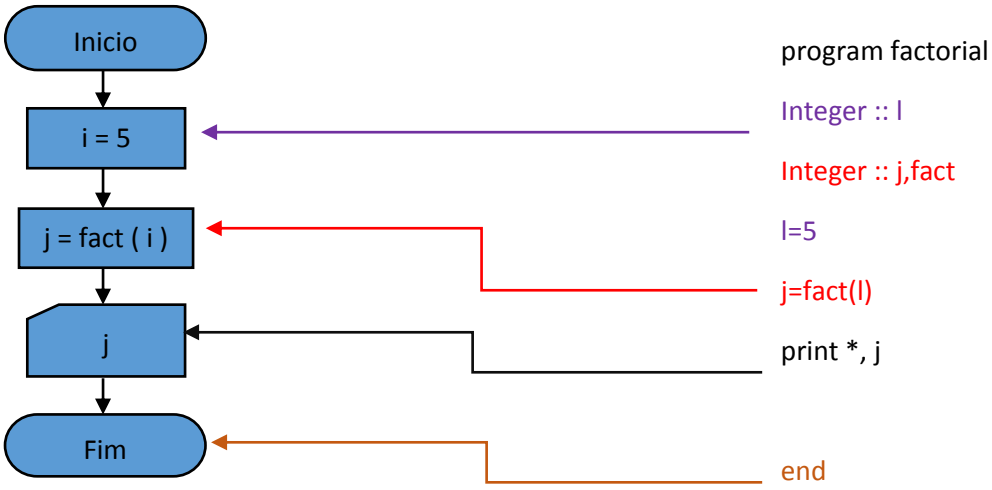
Código

Codigo Principal	**Função**

program factorial
 	
Integer :: l,j,fact
l=5
 	
j=fact(l)
print *, j
 	
end
	Recursive Integer function fact(k) result(res)
	
	if(k>2) then
	res=k*fact(k-1)
	else
	res=k
	end if
	
	endfunction

Esquema detalhado

MAIN



FUNÇÂO