

# Controlo em linguagens sequenciais

Linguagens de Programação  
2019.2020

*Teresa Gonçalves*  
[tcg@uevora.pt](mailto:tcg@uevora.pt)

Departamento de Informática, ECT-UÉ

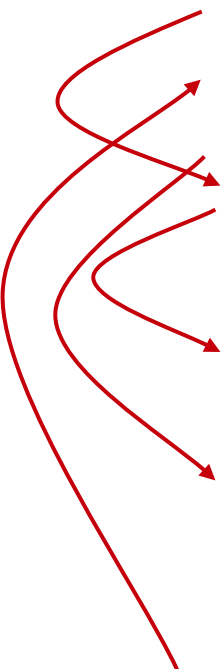
# Sumário

**Controlo estruturado**

**Exceções**

# Controlo estruturado

# Estrutura de controlo no Fortran



```
10 IF (X .GT. 0.000001) GO TO 20
11 X = -X
    IF (X .LT. 0.000001) GO TO 50
20 IF (X*Y .LT. 0.00001) GO TO 30
    X = X-Y-Y
30 X = X+Y
    ...
50 CONTINUE
    X = A
    Y = B-A
    GO TO 11
    ...
```

X = 0.001  
Y = 0.0001

No código assembly pode ocorrer uma estrutura semelhante.

# GO TO considered harmful

## E.W. Dijkstra to Communications of ACM (março 1968)

... the quality of programmers is a decreasing function of the **density of go to statements** in the program they produce.

Later I discovered why the use of the go to statement has such disastrous effects and I did become convinced that the go to statement **should be abolished** from all “higher level” programming languages.

# Controlo de fluxo

## **Legibilidade de um programa**

Linguagem de programação

Estilo de programação

**Uma LP deve fornecer mecanismos que facilitam a organização do controlo de fluxo**

# Progressos na ciência da computação

## Estilo de programação

Agrupar código em blocos lógicos

Evitar saltos explícitos, exceto para retorno da função

Impossibilidade de saltar para o meio de um bloco ou corpo da função

## Construções que estruturam saltos

```
if ... then ... else ... end
```

```
while ... do ... end
```

```
for ... { ... }
```

```
case ...
```

# Exceções



# Exceção

## O que é?

Mecanismo básico para

- Saltar para fora de um bloco ou invocação de uma função

- Passar dados como parte do salto

- Voltar para um ponto definido no programa para continuar o cálculo

## Construções básicas

- Instrução para lançar (dar origem) a exceção

- Mecanismo para capturar (tratar) a exceção

## Linguagens

- Ada, C++, Java, ML, ...

# Funcionalidades

## Abortar chamada de funções

Porque permite saltar fora de uma parte do programa, mas não para uma determinada parte do programa

## Recuperar de erros

Porque permite passar dados juntamente com o salto

## Determinar para onde vai o “salto”

Porque o tratador correto é determinado pelas regras de **âmbito dinâmico**

# Exceção em C++

```
Matriz inverte(Matriz m) {  
    if ... throw Determinante;  
    ...  
};
```

```
try {  
    ...  
    inverte(matriz);  
    ...  
}  
catch (Determinante) {  
    ...  
}
```

Noutro excerto de código o  
tratamento da mesma  
exceção pode ser  
diferente!!!

# Tratador em C++

## Utiliza tipos para distinguir diferentes exceções

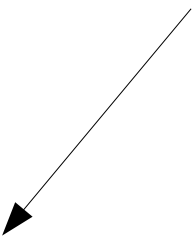
```
throw "Hello World!"  
throw 18;  
throw new String("hello");
```

```
try {  
    ...  
}  
catch( char *message) {  
    ...  
}  
catch ( void *w ){  
    ...  
}
```

# Exceção em ML

```
exception Determinante;  
fun invert (M) =  
  ...  
  if ...  
  then raise Determinante  
  else ...  
  end;  
invert(mat) handle Determinante => ... ;  
  try          catch
```

Valor para a expressão se  
se a exceção for lançada



## Âmbito dinâmico

A chamada da função é o melhor sítio para decidir o que fazer se o determinante for zero

# Exceções em ML

## Declaração

```
exception <nome> of <tipo>
```

Dá um nome à exceção e especifica o tipo de dados passado quando é lançada

## Lançamento

```
raise <nome> <argumentos>
```

Expressão para lançar uma exceção e passar dados

## Captura

```
<expr1> handle <padrão> => <expr2>
```

Avalia a expressão <expr1>. Se for lançada uma exceção correspondente a <padrão>, avalia a expressão <expr2> em substituição

Podem existir **múltiplos padrões**!

# Tratador em ML

**ML utiliza *pattern matching* para determinar o tratador apropriado**

```
exception Signal of int;
```

```
fun f(x) =  
  if x=0 then raise Signal(0)  
  else if x=1 then raise Signal(1)  
  else if x=10 then raise Signal(x-8)  
  else (x-2) mod 4
```

```
f(10) handle Signal(0) => 0  
          | Signal(1) => 1  
          | Signal(x) => x+8;
```

# C++ vs. ML

## C++

Tratador escolhido por type matching

Pode lançar qualquer tipo

Stroustrup: “I prefer to define types with no other purpose than exception handling. This minimizes confusion about their purpose. In particular, I never use a built-in type, such as int, as an exception.”

*The C++ Programming Language, 3ª edição*

## ML

Tratador escolhido por pattern matching

Exceção é uma entidade diferente de tipo

Declaração antes da utilização

**O ML obriga o estilo recomendado pelo C++**



# Utilização de exceções

## Condição de erro

Devolver a sub-árvore esquerda de um nó

```
datatype 'a tree = LF of 'a |  
                  ND of ('a tree)*('a tree)
```

```
exception No_Subtree;
```

```
fun lsub (LF x) = raise No_Subtree  
  | lsub (ND(x,y)) = x;
```

```
> val lsub = fn : 'a tree -> 'a tree
```

**A função lança uma exceção quando não existe um valor razoável!**

# Utilização de exceções

## Eficiência

Multiplicar as folhas de uma árvore

```
fun prod(LF x) = x
  | prod(ND(x,y)) = prod(x) * prod(y);
```

## Optimização

```
fun prod(tree) =
  let exception Zero
      fun p(LF x) =
          if x=0 then (raise Zero) else x
        | p(ND(x,y)) = p(x) * p(y)
      in
        p(tree) handle Zero => 0
      end;
```

# Que tratador é utilizado?

```
exception Ovflw;
```

```
fun f(x) =  
  if x<min then raise Ovflw else 1/x;
```

```
(f(x) handle Ovflw=>0) / (f(y) handle Ovflw=>1);
```

## Âmbito dinâmico

Tratador mais recente no stack de execução


A primeira chamada trata a exceção de uma forma; segunda trata de outra

## Porquê dinâmico?

A zona de código que chama a função é o melhor lugar para decidir o que fazer!

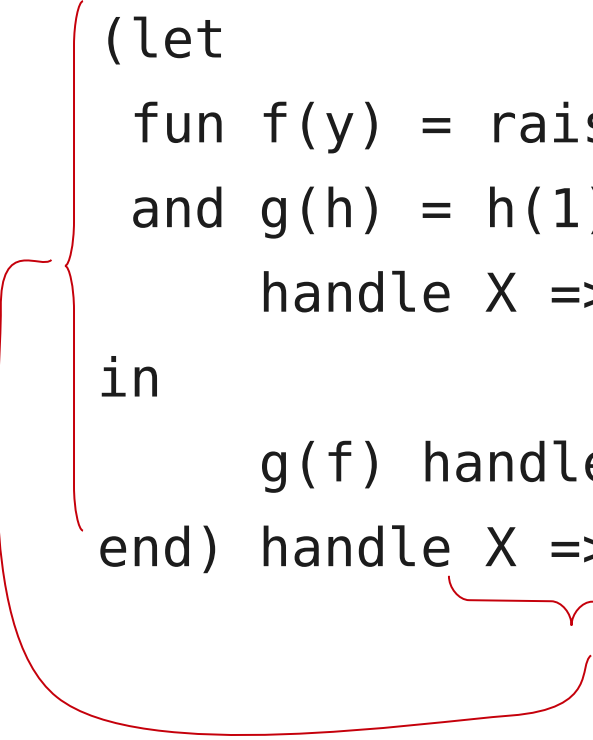
# Comparação de âmbitos

## X: variável



```
val x = 6
let
  fun f(y) = x
  and g(h) = let val x=2
              in h(1) end
in
  let val x = 4 in g(f)
end
end;
```

## X: exceção



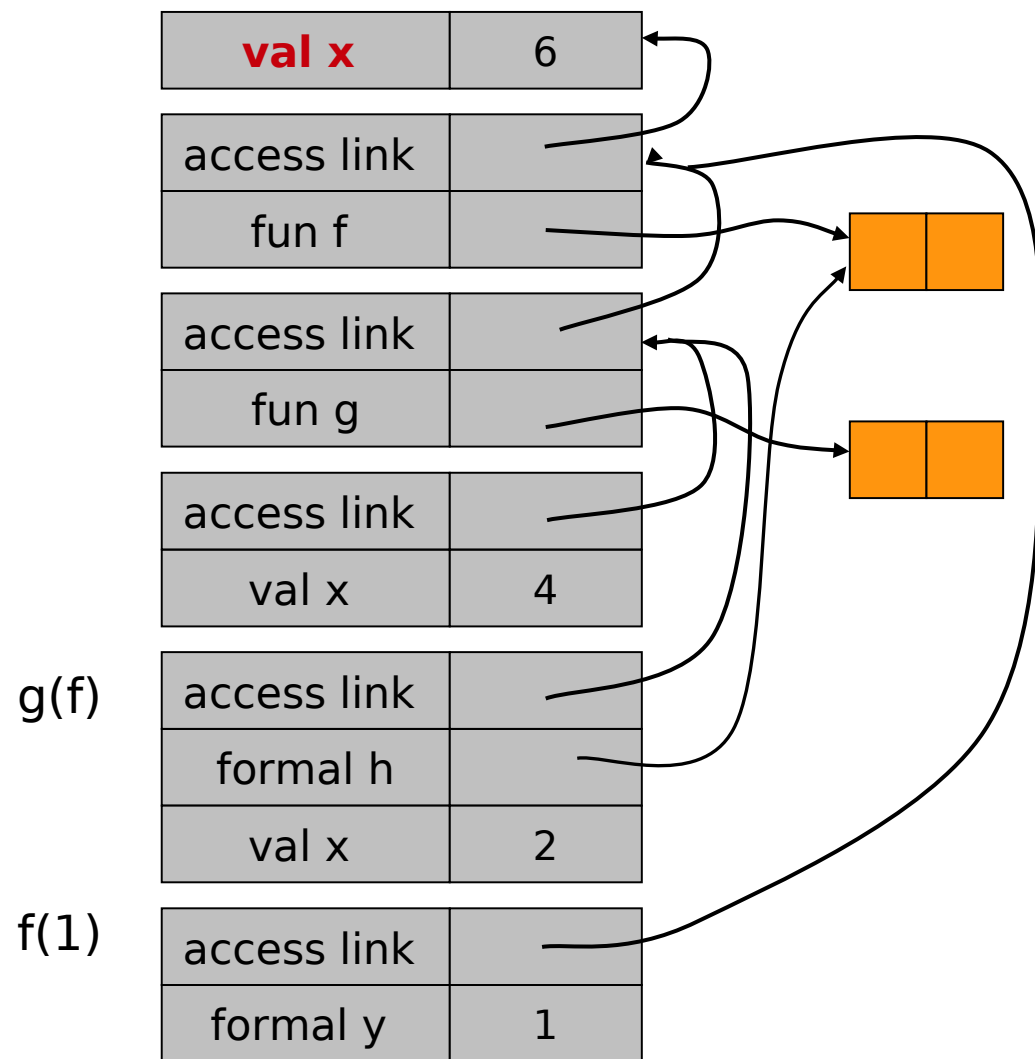
```
exception X;
(let
  fun f(y) = raise X
  and g(h) = h(1)
  handle X => 2
in
  g(f) handle X => 4
end) handle X => 6;
```

# Âmbito estático das declarações

```
val x = 6
let
  fun f(y) = x
  and g(h) =
    let val x=2 in h(1) end
in
  let val x = 4 in g(f) end
end;
```

## Âmbito estático

Procurar o primeiro x, seguindo os access links



# Âmbito dinâmico do tratador de exceções

```
exception X;  
(let fun f(y) = raise X  
  and g(h) = h(1) handle X => 2  
in  
  g(f) handle X => 4  
end) handle X => 6;
```

âmbito

âmbito

tratador

tratador

Notação posfixa

*O operador (tratador) aparece  
depois do operando (expressão)*

# Âmbito dinâmico do tratador de exceções

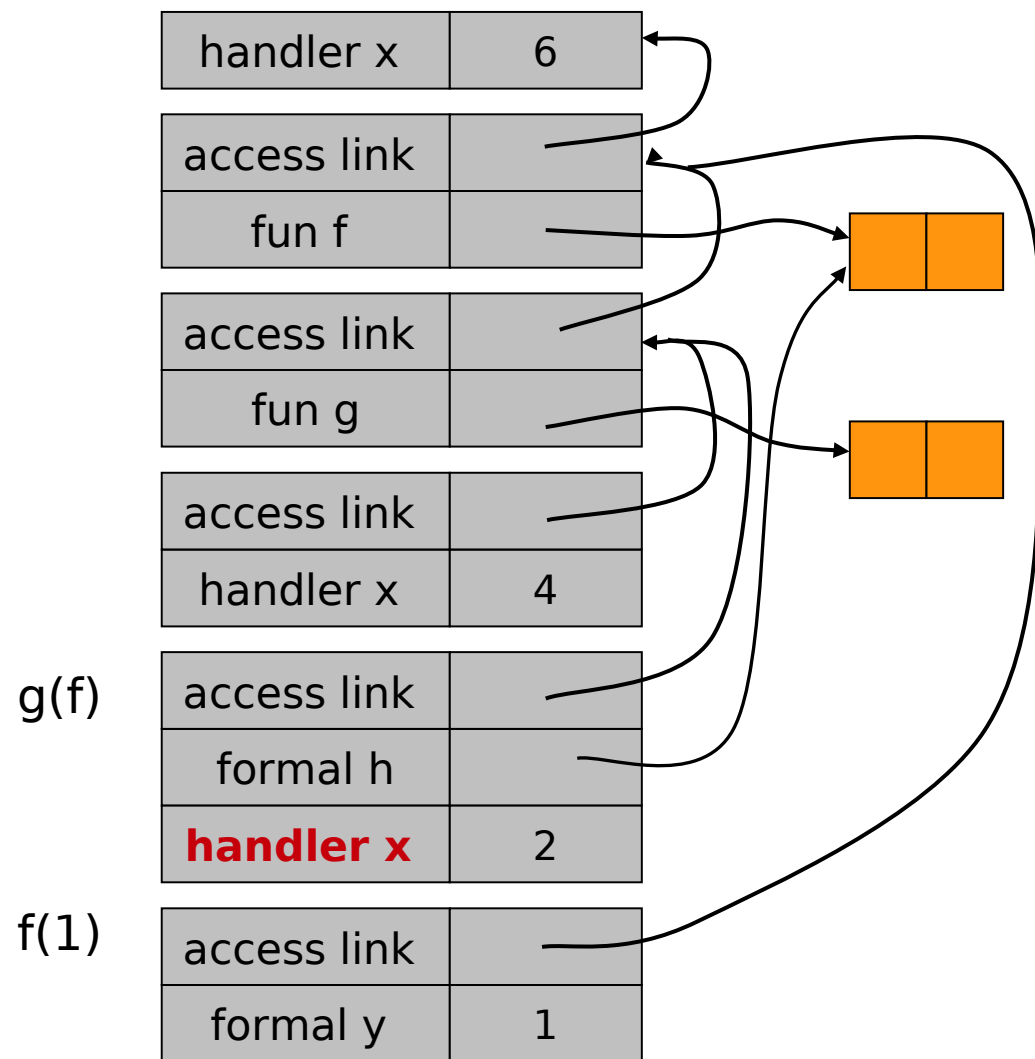
```
exception X;  
(let  
  fun f(y) = raise X  
  and g(h) = h(1) handle X => 2  
in  
  g(f) handle X => 4  
end) handle X => 6;
```

## Âmbito dinâmico

Procurar o handler de X, subindo na cadeia dinâmica de chamadas que conduziu ao levantamento de X

handler X => 2

São libertados os RA até ao handler!



# Exceções e tipos

## Tratador

`e1 handle A => e2`

É necessária a conformidade de tipos

`e1` do mesmo tipo que `e2`

Exemplo

`1 + (e1 handle X => e2)`

## Lançador

`raise <excp>`

A expressão não tem valor

Tipo genérico 'a para permitir a inferência de tipos

Exemplo

`1 + raise X`



# Exceções e alocação de recursos

## Gestão de recursos

Ao ser lançada uma exceção, os recursos (ficheiros, locks, ...) alocados entre o tratador e o lançador de exceções deixam de estar acessíveis

```
exception X;  
(let  
    val y = ref [1,2,3]  
    in  
        ... raise X  
    end) handle X => ...
```

## Solução?

Não existe uma solução **sistemática, efectiva e “limpa”** para tratamento destas situações

### ML

Os dados da heap são reclamados pelo “garbage collector”

### C++

Os destrutores dos objetos existentes no stack de execução são chamados na libertação de um RA