

Interpretação e Compilação (de Linguagens de Programação)

Unidade 4: Ligação e Âmbito

Os identificadores são a primeira ferramenta básica para criar abstrações numa linguagem de programação. Um identificador usado numa expressão (ou programa) representa uma subexpressão cuja definição é feita separadamente. O significado da expressão contendo identificadores deve ser o mesmo da expressão em que os identificadores são substituídos pelas subexpressões que eles representam.

- Literais e identificadores
- Declaração de identificadores
- Âmbito de uma declaração
- Ocorrências de um identificador (livres, ligadas e ligantes)
- Expressões abertas e fechadas
- Construção fundamental **decl id=E in E end**.
- Linguagem com identificadores (declaração e ocorrências ligadas): CALCI.
- Algoritmo interpretador com substituição
- Algoritmo interpretador com ambiente
- Algoritmo compilador com ambiente

Constantes e Identificadores

- Constantes (ou literais)

- Referem entidades ou valores bem determinados em qualquer contexto onde ocorram
- Nas linguagens “naturais” correspondem aos “nome próprios”.
- Linguagem ML: `true, false, []`
- Linguagem C: `1, 1.0, 0xFF, "hello", int`

- Identificadores (ou nomes)

- Referem entidades ou valores que dependem do contexto
- Nas linguagens “naturais” correspondem aos “pronomes”.
- Linguagem Java: `x, Count, System.out`
- Linguagem C: `printf`

Ligação e Âmbito

- Os literais e os identificadores denotam sempre uma entidade bem determinada e inalterável.
- A entidade denotada por um literal (ou o valor de um literal) é determinada pelo próprio literal (23, "hi!", etc).
- A associação entre um identificador e a entidade ou valor denotado chama-se ligação (*binding*)
- Em geral, a ligação entre identificador e entidade denotada estabelece-se num certo contexto sintáctico e é introduzida por uma declaração
- Ao contexto sintáctico onde uma ligação tem efeito chama-se o âmbito (*scope*) da ligação

Ligação e Âmbito

- O identificador **x** denota uma variável de estado (célula de memória)

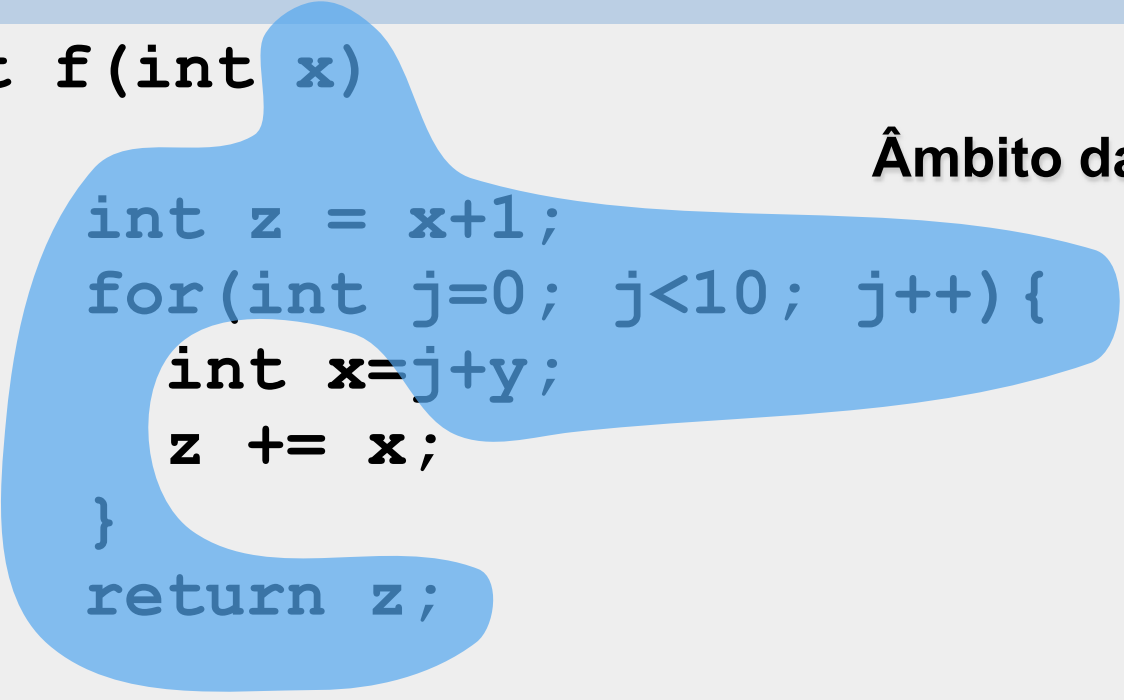
```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ligação e Âmbito

- O identificador **x** denota uma variável de estado (célula de memória)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++){
        int x=j+y;
        z += x;
    }
    return z;
}
```

Âmbito da ligação



Ligação e Âmbito

- O identificador `j` denota uma variável de estado (célula de memória)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ligação e Âmbito

- O identificador `j` denota uma variável de estado (célula de memória)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Âmbito da ligação

Elementos de um âmbito

- A **ligação** entre um identificador e a respectiva entidade por este denotada (valor, posição de memória, etc) envolve os seguintes ingredientes:
 - Uma (única!) **ocorrência ligante**:
em geral, corresponde à **declaração** do identificador.
 - O **âmbito** da ligação
A “parte/região/zona/fragmento” do programa onde a ligação em causa tem efeito
 - Várias (zero ou mais) **ocorrências ligadas**
todas as ocorrências do identificador, distintas da ocorrência ligante, que existem dentro do âmbito

Ocorrências ligantes e ligadas

- Ocorrências do identificador **x**

```
int f(int x)  
{  
    int z = x+1;  
    for(int j=0; j<10; j++){  
        int x=j+y;  
        z += x;  
    }  
    return z;  
}
```

Ocorrências ligantes

Ocorrências ligantes e ligadas

- Ocorrências do identificador **x**

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++){
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ocorrências ligadas

Ocorrências ligadas

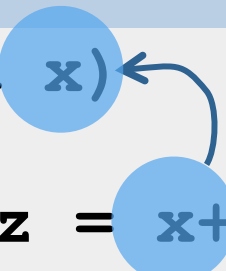
- Para cada ocorrência ligada existe uma e uma só ocorrência ligante (que ocorre na declaração)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ocorrências ligadas

- Para cada ocorrência ligada existe uma e uma só ocorrência ligante (que ocorre na declaração)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```



Ocorrências ligadas

- Para cada ocorrência ligada existe uma e uma só ocorrência ligante (que ocorre na declaração)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ocorrências ligadas

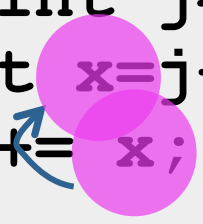
- Para cada ocorrência ligada existe uma e uma só ocorrência ligante (que ocorre na declaração)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Ocorrências ligadas

- Para cada ocorrência ligada existe uma e uma só ocorrência ligante (que ocorre na declaração)

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```



Ocorrências livres

- Uma ocorrência de identificador que não é ligada nem ligante diz-se **livre**

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++) {
        int x=j+y;
        z += x;
    }
    return z;
}
```

Expressões Abertas e Fechadas

- Uma subexpressão diz-se **aberta** se contém ocorrências livres de identificadores
- Uma subexpressão diz-se **fechada** se não contém ocorrências livres de identificadores
- Exemplos de expressões abertas:

```
void f(int x)                                     C
{
    int i;
    for(int i=0;i<TEN;i++) x+=i;
    printf("%d\n",x);
}
```

```
let x=1 in (f x)                                  OCaml
```

Expressões Abertas e Fechadas

- Uma subexpressão diz-se **aberta** se contém ocorrências livres de identificadores
- Uma subexpressão diz-se **fechada** se não contém ocorrências livres de identificadores
- Exemplos de expressões abertas:

```
void f(int x)                                ocorrência livre      C
{
    int i;
    for(int i=0;i<TEN;i++) x+=i;
    printf("%d\n",x);
}
```

```
let x=1 in (f x)                             ocorrência livre      OCaml
```

Semântica de expressões abertas

- A denotação de uma subexpressão de programa só pode ser calculada se se conhecer a denotação de cada identificador que nela ocorra livre.
- A definição de uma semântica composicional para linguagens com declarações de identificadores tem necessariamente que considerar expressões abertas.

Por exemplo, a expressão OCaml

```
let x = 2 in (x+x)
```

é fechada mas contém uma subexpressão aberta.

- Um programa (fragmento fechado) pode conter no seu interior expressões abertas.

[Dê exemplos de linguagens de programação onde seja possível compilar um programa aberto]

Ambiente

- Um programa fechado fornece necessariamente ligações para todas as ocorrências livres de identificadores que ocorram nas suas subexpressões (através de declarações).

Para cada subexpressão E de um programa P , ao conjunto de todas as ligações no âmbito das quais E ocorre chama-se o **ambiente** de E em P .

Ambiente (Quiz)

- Qual o ambiente da subexpressão “x+1”?

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++){
        int x=j;
        z+=x;
    }
    return z;
}
```

Ambiente (Quiz)

- Qual o ambiente da subexpressão “z+=x”?

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++){
        int x=j;
        z+=x;
    }
    return z;
}
```

Ambiente (Quiz)

- Qual o ambiente da subexpressão “return z”?

```
int f(int x)
{
    int z = x+1;
    for(int j=0; j<10; j++){
        int x=j;
        z+=x;
    }
    return z;
}
```


Exemplo: A Linguagem CALCI

- A linguagem CALCI estende a linguagem CALC com a possibilidade de se poderem introduzir e usar identificadores usando a construção **declare**:

```
decl Id = Expressão1 in Expressão2 end
```

Numa expressão **decl**, a primeira ocorrência de *Id* é **ligante**, no âmbito definido pela *Expressão2*

- Definimos os programas CALCI como sendo as **expressões fechadas** da linguagem CALCI.

Por exemplo:

```
decl x=2 in decl y=x+2 in (x+y) end end
```

A Linguagem CALCI (como tipo indutivo)

- Tipo de dados CALCI com os construtores: num, add, mul, div, sub, id, decl

num:	$\text{Integer} \rightarrow \text{CALCI}$
id:	$\text{String} \rightarrow \text{CALCI}$
add:	$\text{CALCI} \times \text{CALCI} \rightarrow \text{CALCI}$
mul:	$\text{CALCI} \times \text{CALCI} \rightarrow \text{CALCI}$
div:	$\text{CALCI} \times \text{CALCI} \rightarrow \text{CALCI}$
sub:	$\text{CALCI} \times \text{CALCI} \rightarrow \text{CALCI}$
decl:	$\text{String} \times \text{CALCI} \times \text{CALCI} \rightarrow \text{CALCI}$

A Linguagem CALCI (como tipo indutivo)

- Tipo de dados CALCI com os construtores: num, add, mul, div, sub, id, decl

num: Integer \rightarrow CALCI

id: String \rightarrow CALCI

add: CALCI \times CALCI \rightarrow CALCI

mul: CALCI \times CALCI \rightarrow CALCI

div: CALCI \times CALCI \rightarrow CALCI

sub: CALCI \times CALCI \rightarrow CALCI

decl: String \times CALCI \times CALCI \rightarrow CALCI

Semântica de CALCI (I)

- A função semântica I de CALCI pode ser definida por um **algoritmo** que “sabe como interpretar” **todas** as expressões de CALCI, determinando o seu **valor ou efeito**.

$$I : \text{CALCI} \rightarrow \text{Integer}$$

CALCI = conjunto das expressões fechadas

Integer = conjunto dos significados (denotações)

Interpretador de CALCI

- Algoritmo $\text{eval}(E)$ para calcular o valor de uma expressão fechada qualquer E de CALCI:

$\text{eval} : \text{CALCI} \rightarrow \text{Integer}$

$\text{eval}(\text{num}(n))$	$\triangleq n$
$\text{eval}(\text{add}(E1, E2))$	$\triangleq \text{eval}(E1) + \text{eval}(E2)$
...	
$\text{eval}(\text{decl}(s, E1, E2))$	$\triangleq [G = \text{subst}(s, E2, E1);$ $\text{eval}(G);]$

A Função Subst

$\text{subst}(s, E, F)$

Calcula a expressão que resulta de substituir todas as ocorrências livres do identificador s pela expressão F na expressão E .

$\text{subst}(s, s+s+2, y+z) = (y+z)+(y+z)+2$

$\text{subst}(y, \text{decl } x=y \text{ in decl } y=2 \text{ in } x+y, u) = \text{decl } x=u \text{ in decl } y=2 \text{ in } x+y$

Definição da Função Subst

$$\text{subst}(s, \text{num}(n), F) \triangleq \text{num}(n);$$
$$\text{subst}(s, \text{id}(s), F) \triangleq F;$$
$$\text{subst}(s, \text{add}(E1, E2), F) \triangleq \text{add}(\text{subst}(s, E1, F), \text{subst}(s, E2, F));$$

■ ■ ■

$$\mathbf{subst}(s, \mathbf{decl}(s, E1, E2), F) \triangleq [\text{/* caso } s = s' \text{ */ } G = \mathbf{subst}(s, E1, F); \mathbf{decl}(s, G, E2);]$$

```
subst(s, decl(s', E1, E2), F)  $\triangleq$  [/* caso s  $\neq$  s' */ G = subst(s, E1, F);  
decl(s', G, subst(s, E2, F)); ]
```

Interpretador de CALCI

- Algoritmo $\text{eval}(E)$ para calcular o valor de uma expressão fechada qualquer E de CALCI:

$\text{eval} : \text{CALCI} \rightarrow \text{integer}$

$\text{eval}(\text{num}(n))$	$\triangleq n$
$\text{eval}(\text{add}(E1, E2))$	$\triangleq \text{eval}(E1) + \text{eval}(E2)$
...	
$\text{eval}(\text{decl}(s, E1, E2))$	$\triangleq [G = \text{subst}(s, E2, \text{eval}(E1));$ $\text{eval}(G);]$

Interpretador de CALCI

- Algoritmo $\text{eval}(E)$ para calcular o valor de uma expressão fechada qualquer E de CALCI:

$\text{eval} : \text{CALCI} \rightarrow \text{integer}$

$\text{eval}(\text{ num}(n))$	$\triangleq n$
$\text{eval}(\text{ add}(E1,E2))$	$\triangleq \text{eval}(E1) + \text{eval}(E2)$
...	
$\text{eval}(\text{ decl}(s, E1, E2))$	$\triangleq [G = \text{subst}(s, E2, \text{eval}(E1));$ $\text{eval}(G);]$

Semântica de CALCI (2)

- A semântica da linguagem CALCI baseada em substituições é muito conveniente do ponto de vista da especificação pois é muito simples.

$\text{eval} : \text{CALCI} \rightarrow \text{Integer}$

- Já do ponto de vista operacional, é conveniente definir uma semântica mais concreta, recorrendo à manipulação de ambientes.
- A manipulação de ambientes também é mais conveniente como técnica de implementação de interpretadores.

Semântica de CALCI (2)

A função semântica I de CALCI pode ser definida por um algoritmo interpretador para expressões de CALCI, determinando o seu **valor ou efeito**, dado um ambiente contendo os valores dos seus identificadores livres.

$$I : \text{CALCI} \times \text{ENV} \rightarrow \text{Integer}$$

CALCI = programas abertos

ENV = ambientes

Integer = significados (denotações)

Semântica de CALCI (2)

A função semântica I de CALCI pode ser definida por um algoritmo interpretador para expressões de CALCI, determinando o seu **valor ou efeito**, dado um ambiente contendo os valores dos seus identificadores livres.

$$I : \text{CALCI} \times \text{ENV} \rightarrow \text{integer}$$

CALCI = programas abertos

ENV = ambientes

Integer = significados (denotações)

Semântica de CALCI (2)

A função semântica I de CALCI pode ser definida por um algoritmo interpretador para expressões de CALCI, determinando o seu valor ou efeito, dado um ambiente contendo os valores dos seus identificadores livres.

$$I : \text{CALCI} \times \text{ENV} \rightarrow \text{integer}$$

CALCI = programas abertos

ENV = ambientes

Integer = significados (denotações)

Ambiente “mutável”

- Na prática, é conveniente implementar ambientes usando uma estrutura de dados mutável ao estilo Object-Oriented.
- Numa linguagem estruturada, com âmbitos encaixados hierarquicamente, a adição e remoção de ligações entre identificadores e valores segue uma disciplina LIFO.
- Um ambiente guarda as associações correspondentes a um determinado âmbito (e todos os âmbitos envolventes). A partir de um ambiente pode criar-se um novo nível, correspondendo a um âmbito encaixado.
- Os ambientes têm duas operações fundamentais:

Environ BeginScope()

- que cria um novo nível local vazio, onde serão colocadas as novas ligações.
- Não pode existir mais que uma ligação para um mesmo identificador no mesmo nível (Porquê?)

Environ EndScope()

- que coloca o ambiente no estado anterior à última operação BeginScope().

Ambiente “mutável”

- Na prática, é conveniente implementar ambientes usando uma estrutura de dados mutável ao estilo Object-Oriented.
- Outras operações fundamentais:

void Assoc(String id, Value val)

- Adiciona uma nova ligação que associa ao identificador **id** o valor **val** indicado.
- A ligação é adicionada ao último nível (mais recente) do ambiente.

- Devolve o valor associado ao identificador **id** no ambiente.

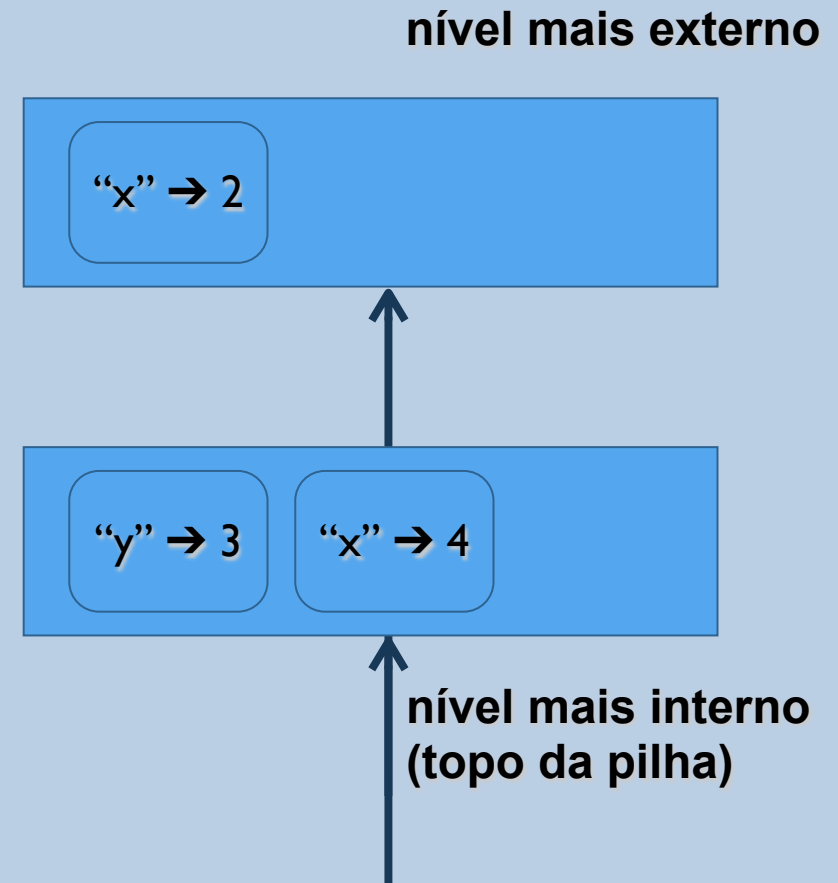
Value Find(String id)

- A pesquisa é efectuada do nível mais “recente” para o mais “antigo”, de modo a respeitar o encaixe dos âmbitos das declarações.

A “interface” Ambiente

- **Simule mentalmente:**

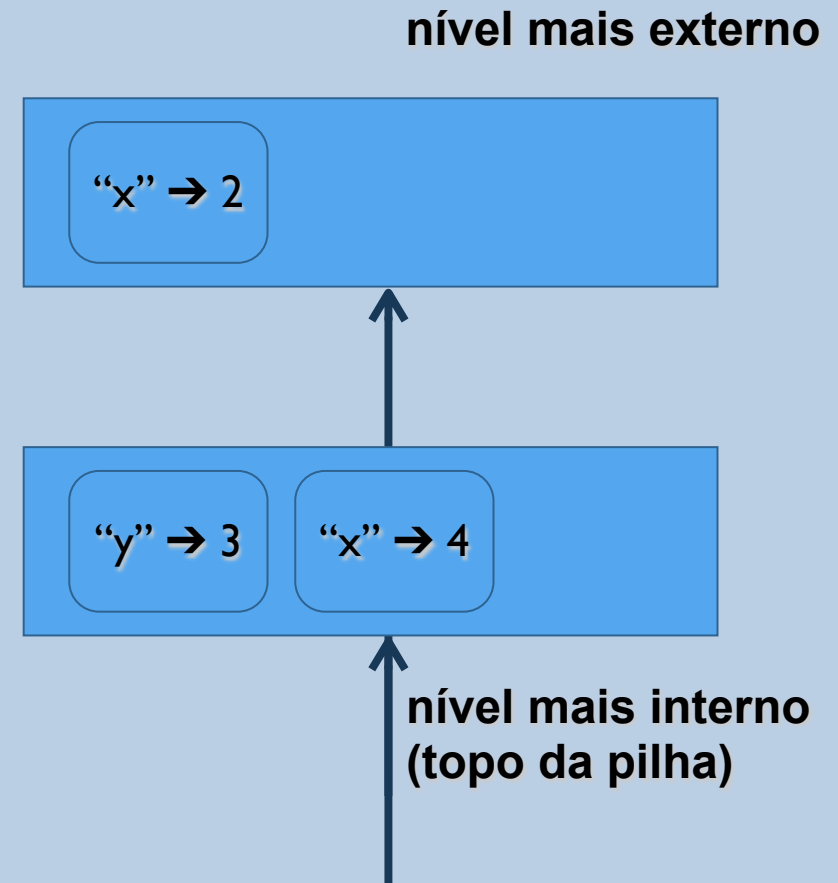
```
env = new Environment();  
env.Assoc("x", 2);  
val = env.Find("x");           // devolve 2  
env = env.BeginScope();  
env.Assoc("y", 3);  
env.Assoc("x", 4);  
val = env.Find("y");           // devolve 3  
val = env.Find("x");           // devolve 4  
env=env.EndScope()  
val = env.Find("x")            // devolve 2
```



A “interface” Ambiente

- Implementado como pilha de dicionários ...

```
env = new Environment();  
env.Assoc("x", 2);  
val = env.Find("x");      // devolve 2  
env = env.BeginScope();  
env.Assoc("y", 3);  
env.Assoc("x", 4);  
val = env.Find("y");      // devolve 3  
val = env.Find("x");      // devolve 4  
env=env.EndScope()  
val = env.Find("x")      // devolve 2
```



Interpretador de CALCI

- Algoritmo $\text{eval}(E, \text{env})$ para calcular o valor de uma expressão E da linguagem CALCI:

$\text{eval} : \text{CALCI} \times \text{ENV} \rightarrow \text{Integer}$

$\text{eval}(\text{num}(n), \text{env}) \triangleq n$

$\text{eval}(\text{id}(s), \text{env}) \triangleq \text{env.Find}(s)$

$\text{eval}(\text{add}(E1, E2), \text{env}) \triangleq \text{eval}(E1, \text{env}) + \text{eval}(E2, \text{env})$

...

$\text{eval}(\text{decl}(s, E1, E2), \text{env}) \triangleq [\text{v1} = \text{eval}(E1, \text{env});$
 $\text{env} = \text{env.BeginScope}();$
 $\text{env.Assoc}(s, \text{v1});$
 $\text{val} = \text{eval}(E2, \text{env});$
 $\text{env} = \text{env.EndScope}();$
 $\text{val}]$

Erros de execução

- O que é que acontece se não for encontrado o identificador no ambiente?
A que corresponde essa falha?
A um erro de execução do tipo “identificador não declarado”.
- A função não está definida para os programas em que há ocorrências de identificadores sem declaração.
- Que outros tipos de erros de execução podem ocorrer?
- São o mesmo tipo de erros? É possível evitar uns e outros não?

