Análise semântica

Há algo errado?

```
int main (){

    int x;
    float a,b,c;

    a = 9.3;
    x = a;
    c = b + 1;
    b = ‘8’;
    c = ‘e’ * ‘f’;
    d = 10.5;
    if (a + b) {
        a = x;
    }
}
```

Existem erros sintáticos?

Quais problemas existem neste programa?

Verificações Necessárias

- ✓ **Verificação de declaração dos Identificadores**
  - ✓ Toda variável deve ser declarada;
- ✓ **Verificação de Unicidade de Identificadores**
  - ✓ O nome do identificador não pode se repetir em um mesmo escopo;
- ✓ **Verificação de Compatibilidade de tipos nas Expressões**
  - ✓ Detectar operações impossíveis como multiplicação de strings;
- ✓ **Verificação de Compatibilidade de Tipos nas atribuições**
  - ✓ Detectar tipos incompatível do lado esquerdo e direito das atribuições;
- ✓ **Verificação de Variáveis não Usadas**
  - ✓ Detectar variáveis declaradas e não usadas
- ✓ **Verificação de variáveis não inicializadas**
  - ✓ Detectar variáveis usadas e não inicializadas

Verificações Necessárias

- ✓ **Verificação de declaração dos Identificadores**
  - ✓ Toda variável deve ser declarada;
- ✓ **Verificação de Unicidade de Identificadores**
  - ✓ O nome do identificador não pode se repetir em um mesmo escopo;
- ✓ **Verificação de Compatibilidade de tipos nas Expressões**
  - ✓ Detectar operações impossíveis como multiplicação de strings;
- ✓ **Verificação de Compatibilidade de Tipos nas atribuições**
  - ✓ Detectar tipos incompatível do lado esquerdo e direito das atribuições;
- ✓ **Verificação de Variáveis não Usadas**
  - ✓ Detectar variáveis declaradas e não usadas
- ✓ **Verificação de variáveis não inicializadas**
  - ✓ Detectar variáveis usadas e não inicializadas

Gera erro
Gera aviso

Há algo errado?

```
int main (){

    int x;
    float a,b,c;

    a = 9.3;
    x = a;
    c = b + 1;
    b = '8'; ???? ????
    c = 'e' * 'f'; ???? ????
    d = 10.5;
    if (a + b) { ???? ????
        a = x; ???? ????
    }
}
```

Existem erros sintáticos?

Quais problemas existem neste programa?

Gramática Resumida do Portugol 1 em BNF

<prog> ::= **programa id** <listadecl> <listacmd>
<listadecl> ::= <listadecl> <decl> **|** <decl>
<decl> ::= <tipo> <lista_id>
<tipo> ::= **inteiro | real | caracter | logico**
<listaid> **::= <**listaid> "**,**" **id | id**
<listacmd> **::= <**listacmd> <cmd> **| <**cmd>
<cmd> **: se <**expr> **então <**listacmd> **fimse**
| enquanto <expr> **faca <**listacmd> **fimenquanto**
| id "< -" <expr>
<expr> ::= expr "+" termo
| termo
<termo> ::= termo "*" fator
| fator
<fator> ::= **id**
| num_inteiro
| num_real
| string

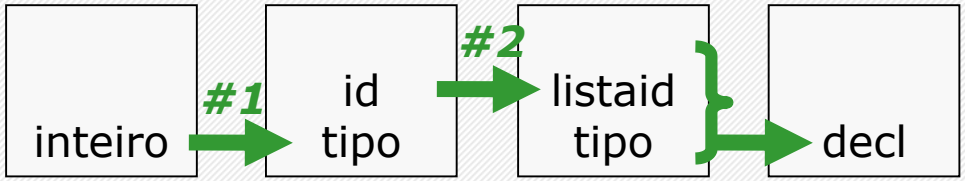
Ações semânticas para Geração da tabela de Símbolos

<decl> ::= <tipo> <listaid>
<tipo> ::= **inteiro** ***#1***
| **real** ***#1***
| **caracter** ***#1***
| **logico** ***#1***
<listaid> **: <**listaid> **, id** ***#2***
| **id** ***#2***

```
#1   tipo = token->getLexeme();
#2   nome = token->getLexeme();
     insere_tabela (tipo, nome);// Verifica a
                                // unicidade
```

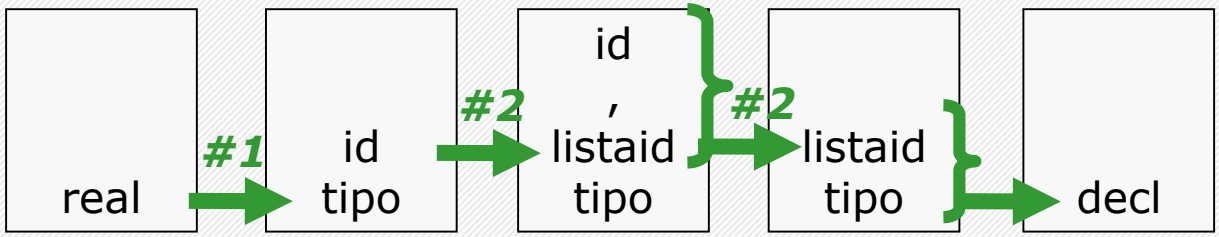
Exemplo 1

inteiro a



Exemplo 2

real b,c



Ações semânticas para Consulta de Variável declarada, usada e inicializada

<cmd> : **se** <expr> **entao** <listacmd> **fimse**
| **enquanto** <expr> **faca** <listacmd> **fimenquanto**
| **id** ***#4*** **"< -"** <expr> ***#5***
<expr> ::= expr "+" termo
| termo
<termo>::= termo "*" fator
| fator
<fator>::= **id** ***#3***
| num_inteiro
| num_real
| string

```
#3 nome = token.getlexeme();
   if (!busca_tabela (nome)) {
      msg("variavel não declarada");
   } else set_usada (nome);

#4 nomeAtrib = token.getlexeme();
   if (!busca_tabela (nomeAtrib)) {
      msg("variavel não declarada");
   }
#5 set_inicializada (nomeAtrib); // OBS: nome foi armazenado antes
```

Exemplo

Inteiro a, b
Real vet [5]
Inteiro funcao coisa (inteiro a, real &x)

Nome	tipo	inic	usada	escopo	param	Pos	Vet	Matriz	Ref	Func
a	inteiro	F	F	global	F	0	F	F	F	F
b	inteiro	F	F	global	F	0	F	F	F	F
vet	real	F	F	global	F	0	T	F	F	F
coisa	inteiro	F	F	global	F	0	F	F	F	T
a	inteiro	F	F	coisa	T	1	F	F	F	F
x	real	F	F	coisa	T	2	F	F	T	F

Exemplo

```
a=1;
b = a + 1;
a = coisa (5, b);
```

Nome	tipo	inic	usada	escopo	param	Pos	Vet	Matriz	Ref	Func
a	inteiro	T	T	global	F	0	F	F	F	F
b	inteiro	T	T	global	F	0	F	F	F	F
vet	real	F	F	global	F	0	T	F	F	F
coisa	inteiro	F	T	global	F	0	F	F	F	T
a	inteiro	F	F	coisa	T	1	F	F	F	F
x	real	F	F	coisa	T	2	F	F	T	F

Exemplo

```
Inteiro funcao coisa (inteiro a, real &x){
   inteiro b,c
   a = c
   b = a
}
```

Nome	tipo	inic	usada	escopo	param	Pos	Vet	Matriz	Ref	Func
a	inteiro	T	T	global	F	0	F	F	F	F
b	inteiro	T	T	global	F	0	F	F	F	F
vet	real	F	F	global	F	0	T	F	F	F
coisa	inteiro	F	T	global	F	0	F	F	F	T
a	inteiro	T	T	coisa	T	1	F	F	F	F
x	real	F	F	coisa	T	2	F	F	T	F
b	inteiro	T	F	coisa	F	0	F	F	F	F
c	inteiro	F	T	coisa	F	0	F	F	F	F

Escopos

Escopos

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```

Escopos

```
funcao inicio () {                    // começa escopo inicio

  para (inteiro x; x<10; x++){        // começa escopo inicio01
    escreva(x)
  }                                   // fim escopo inicio01
  x= 1                    // não declarada no escopo inicio

  inteiro x               // declarada no escopo inicio
  se (x==0){                          // começa escopo inicio02
    x = 2                 // declarada no escopo inicio
    inteiro x=1           // declarada no escopo inicio02
  }                                   // fim do escopo inico02
}
```

Escopos

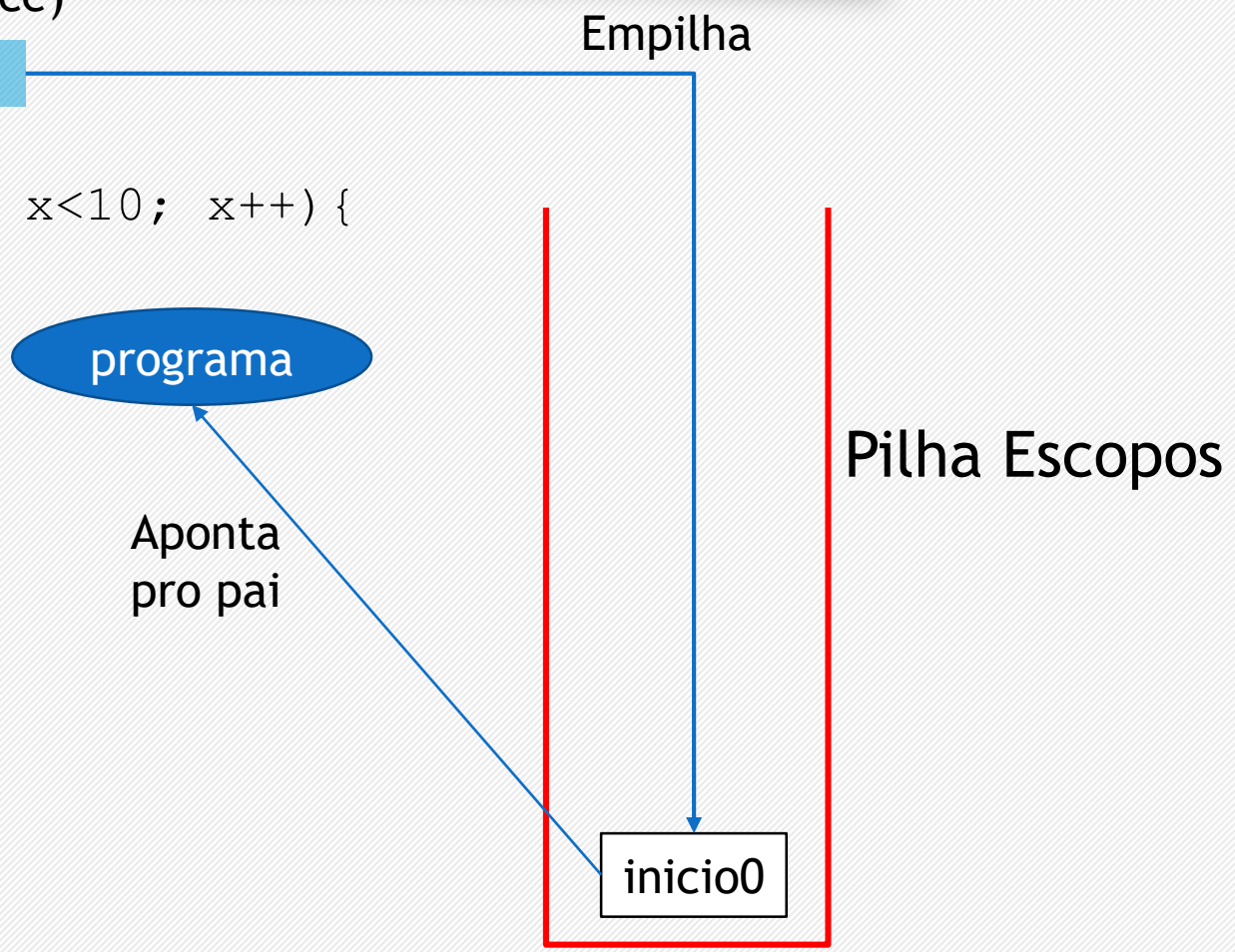
id	tipo	escopo

geraEscopo(label,indice)

```
funcao inicio () {

   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```

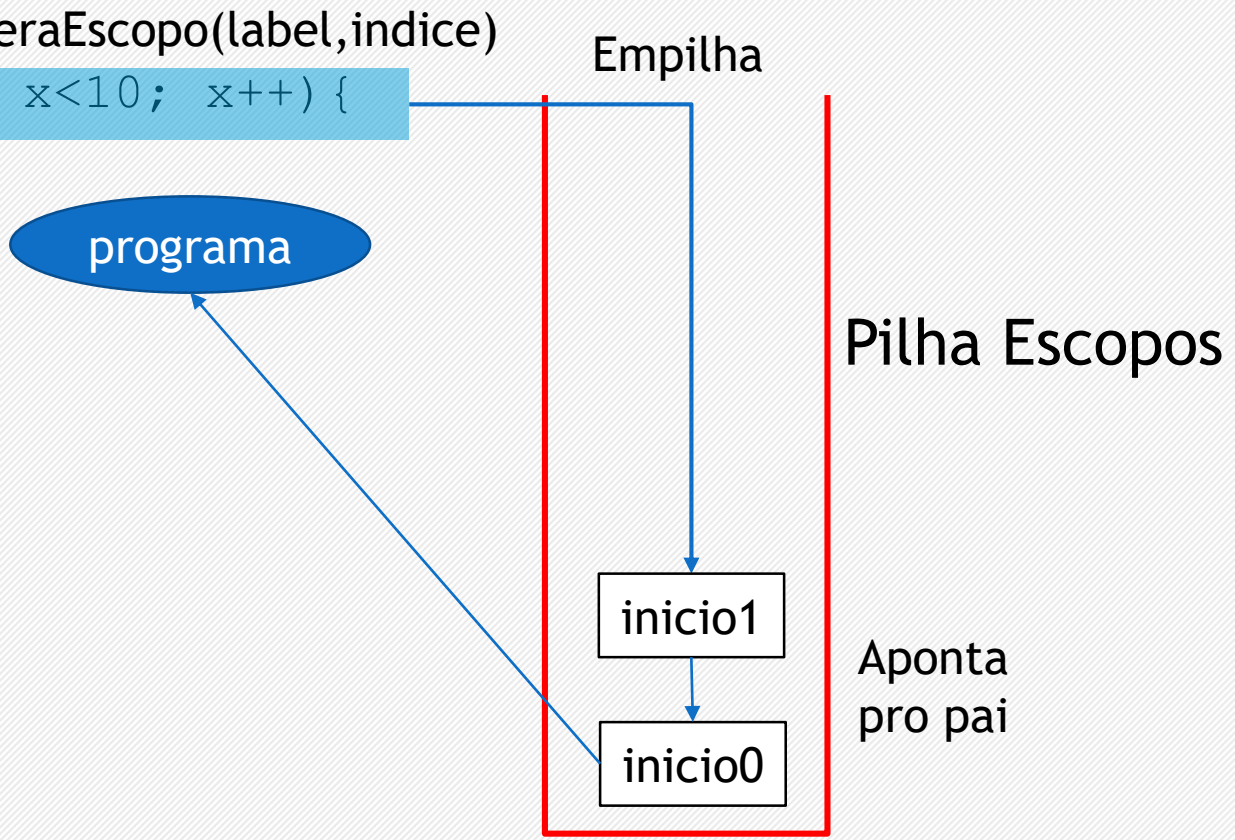


Escopos

id	tipo	escopo
x	inteiro	inicio1

```
funcao inicio () {
   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```



Escopos

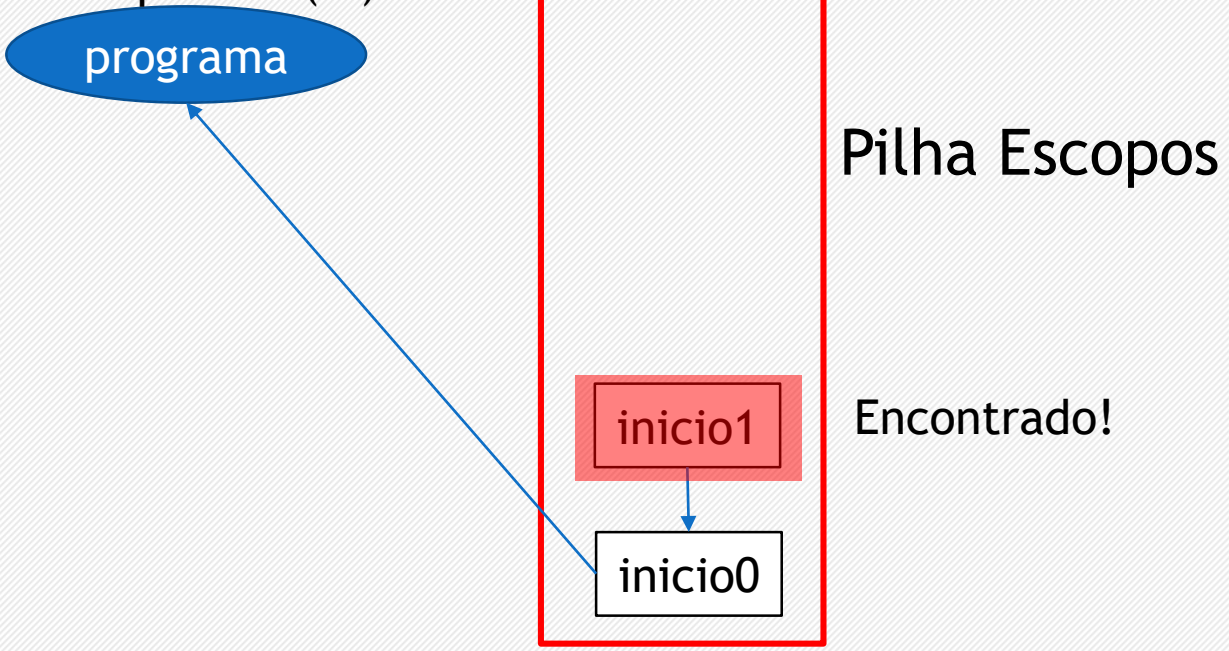
id	tipo	escopo
x	inteiro	inicio1

```
funcao inicio () {

   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```

buscaIdEscopoAtual(id)



Escopos

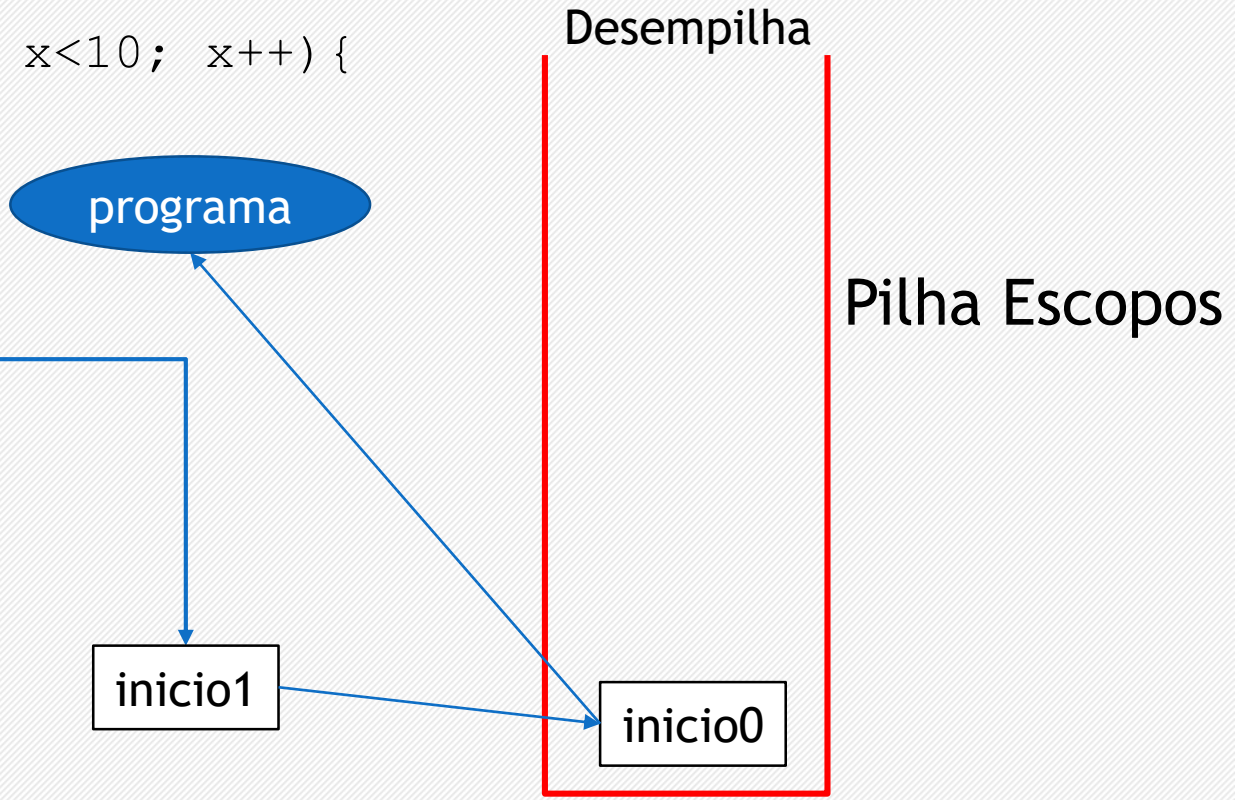
id	tipo	escopo
x	inteiro	inicio1

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```

fimEscopo()

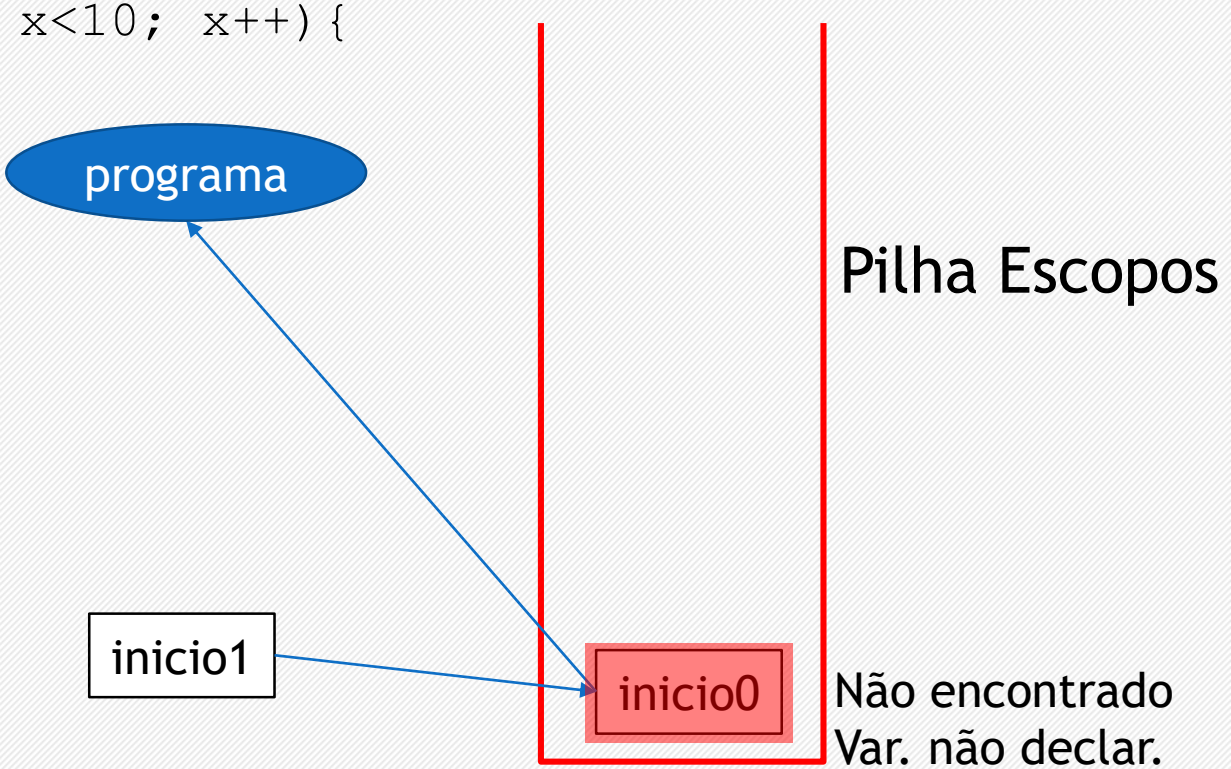


Escopos

id	tipo	escopo
x	inteiro	inicio1

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1
buscaIdEscopoAtual(id)
  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```



Escopos

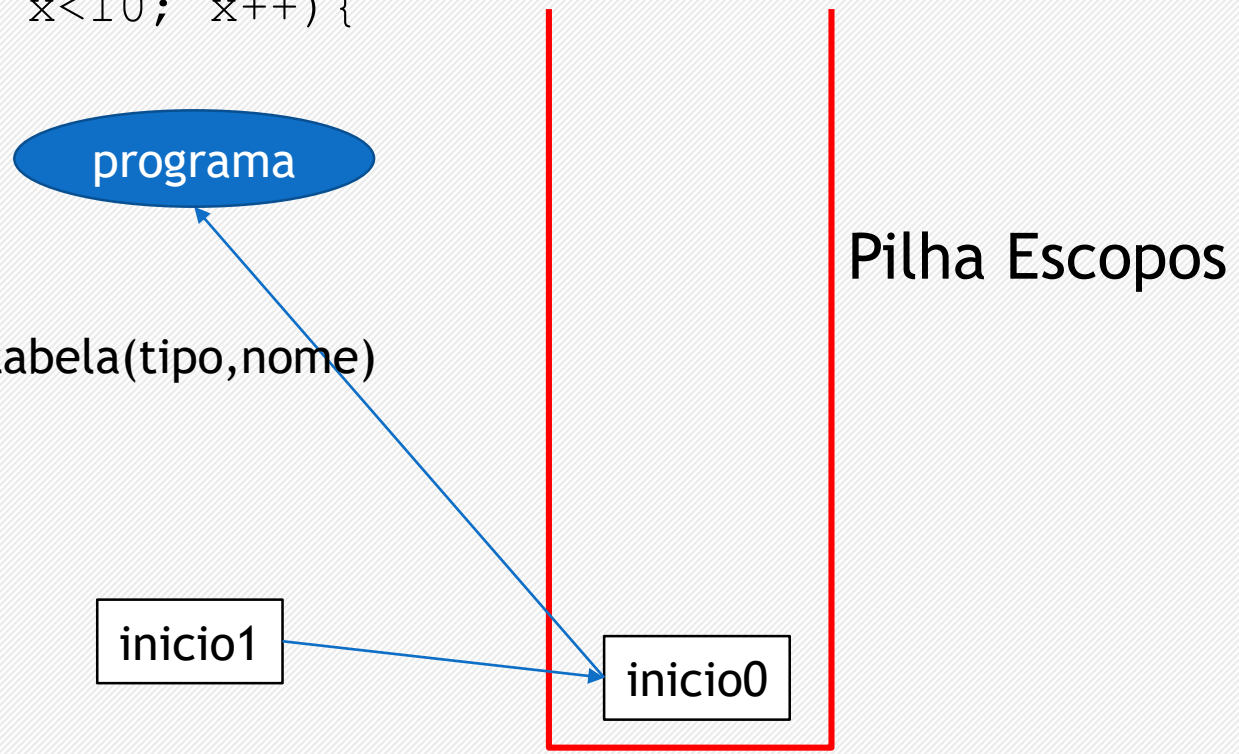
id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```

insere_tabela(tipo,nome)



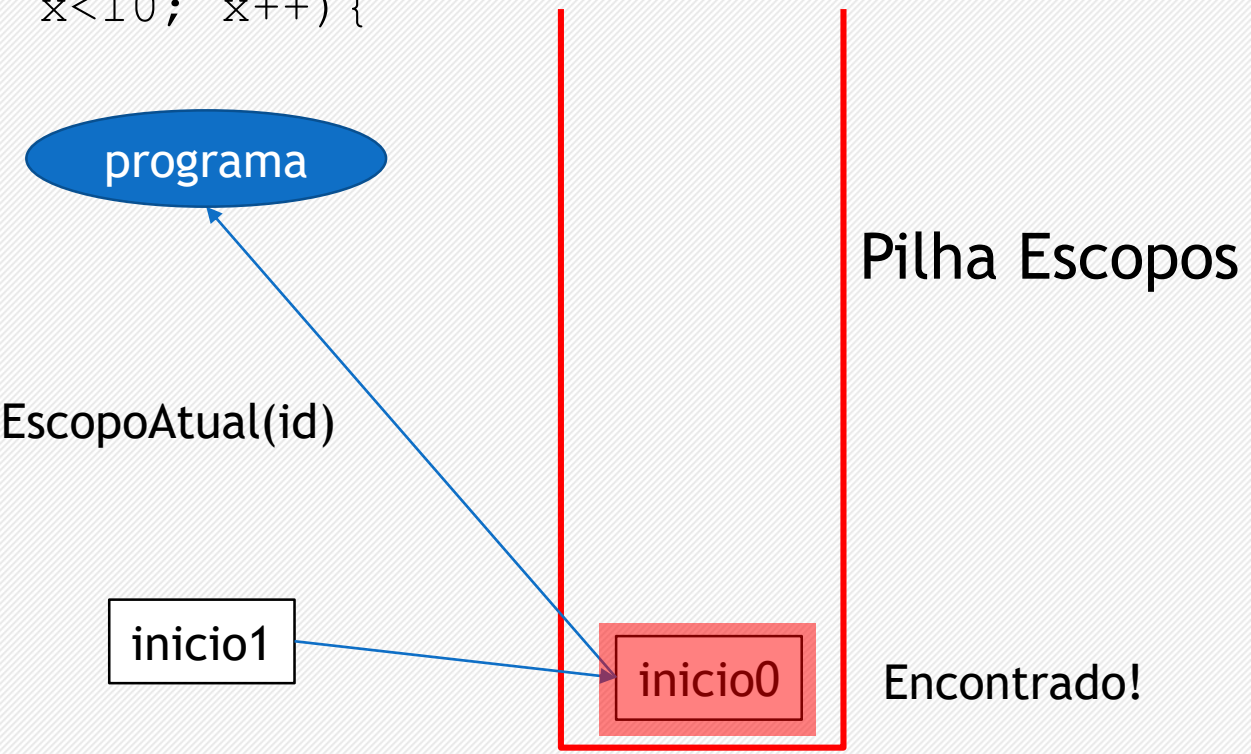
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){ buscaIdEscopoAtual(id)
    x = 2
    inteiro x=1
  }
}
```



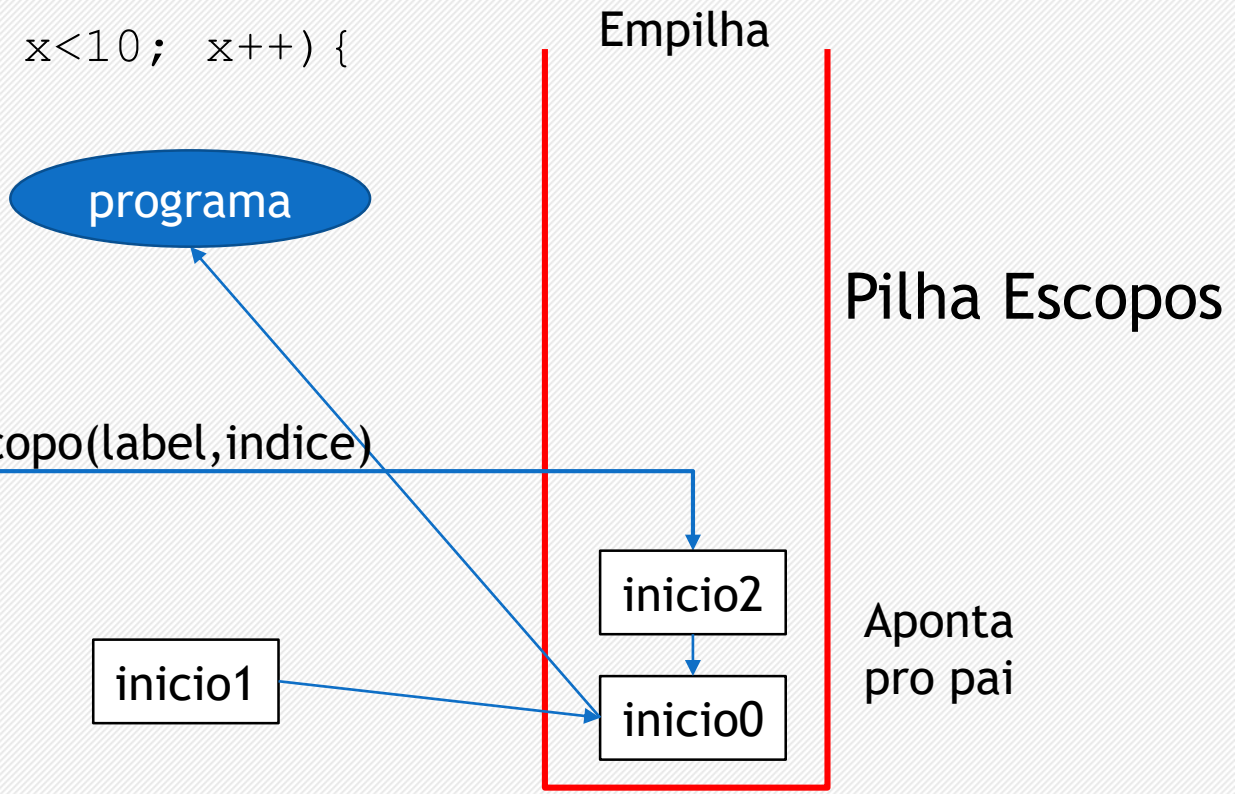
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```



Escopos

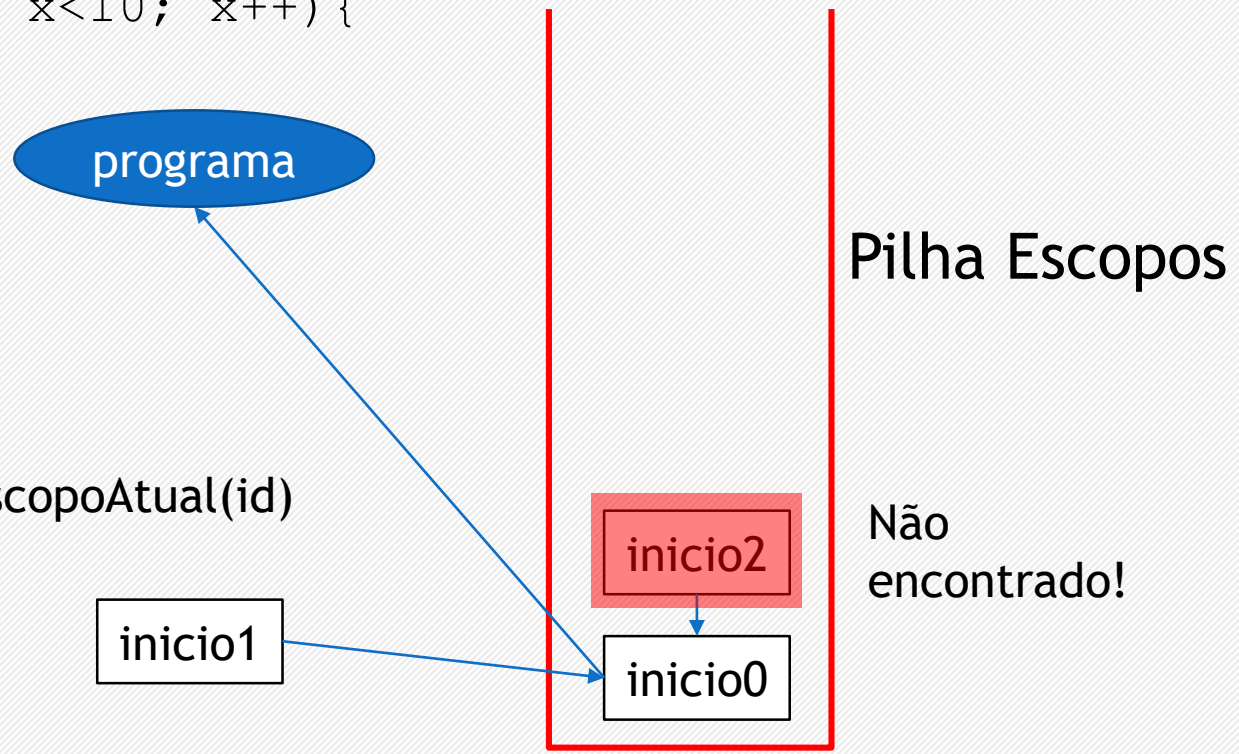
id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```

buscaIdEscopoAtual(id)



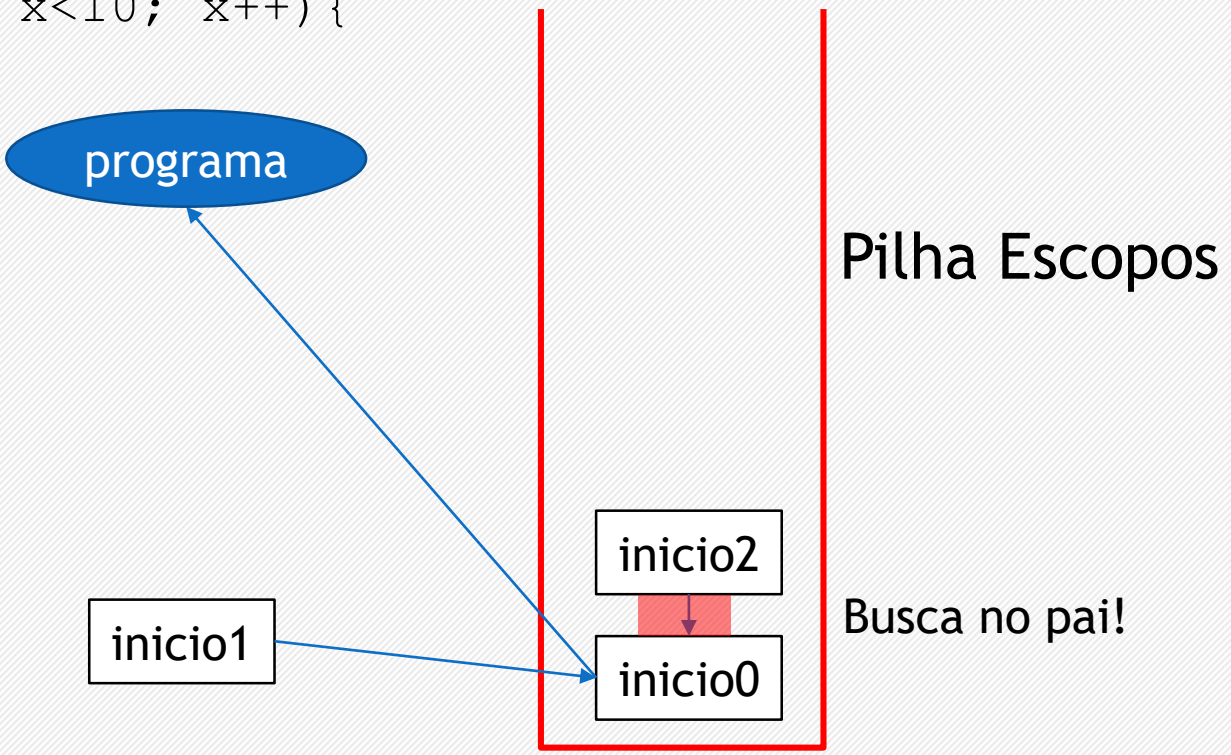
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```



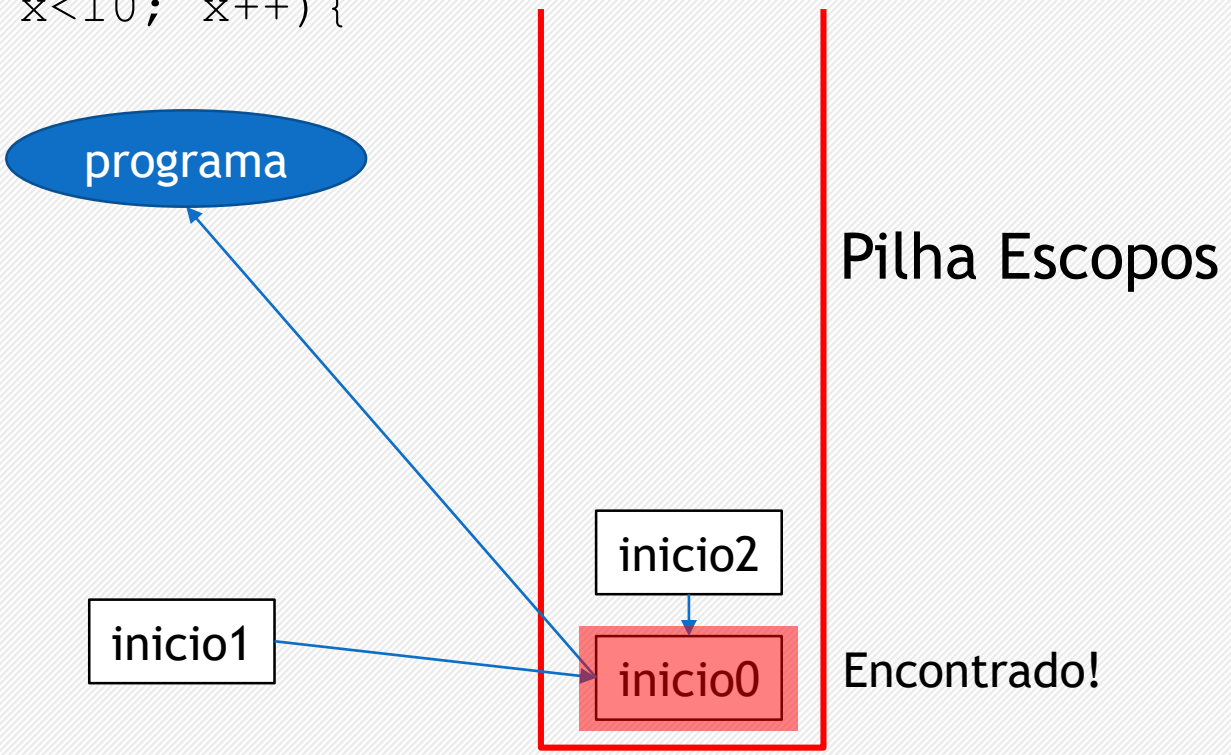
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```



Escopos

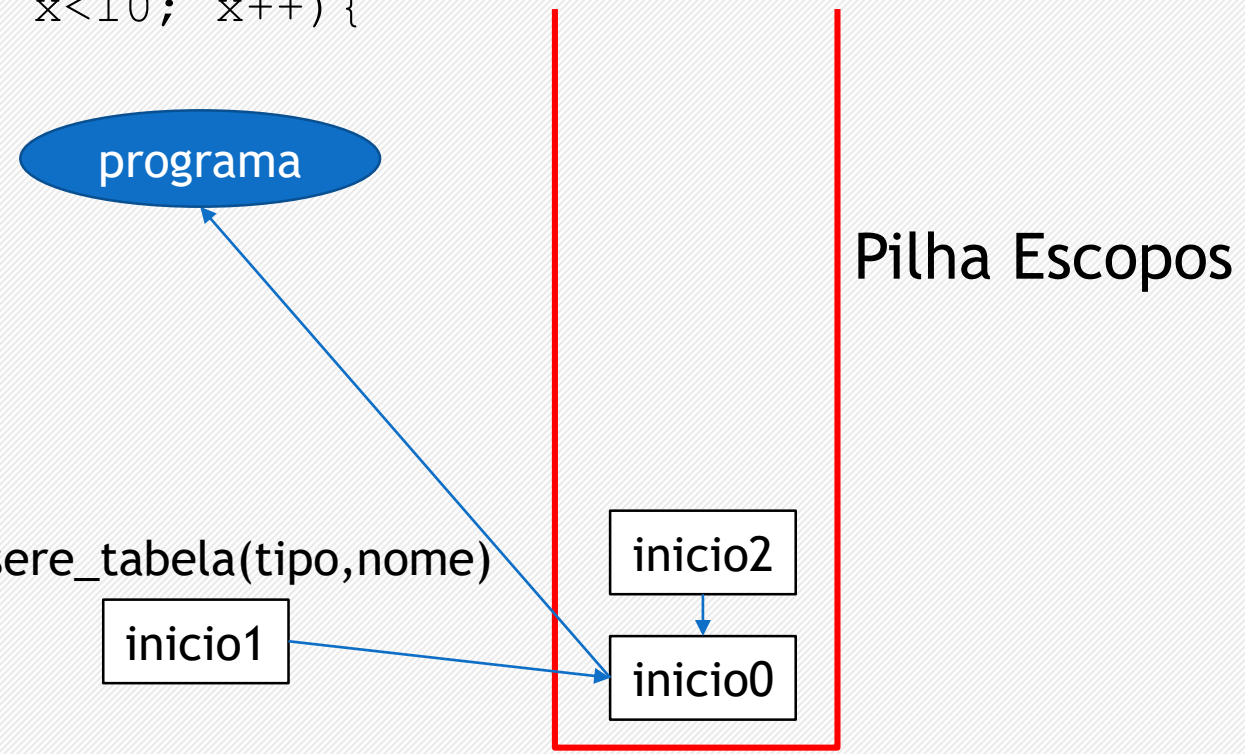
id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0
x	inteiro	inicio2

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```

insere_tabela(tipo,nome)



Escopos

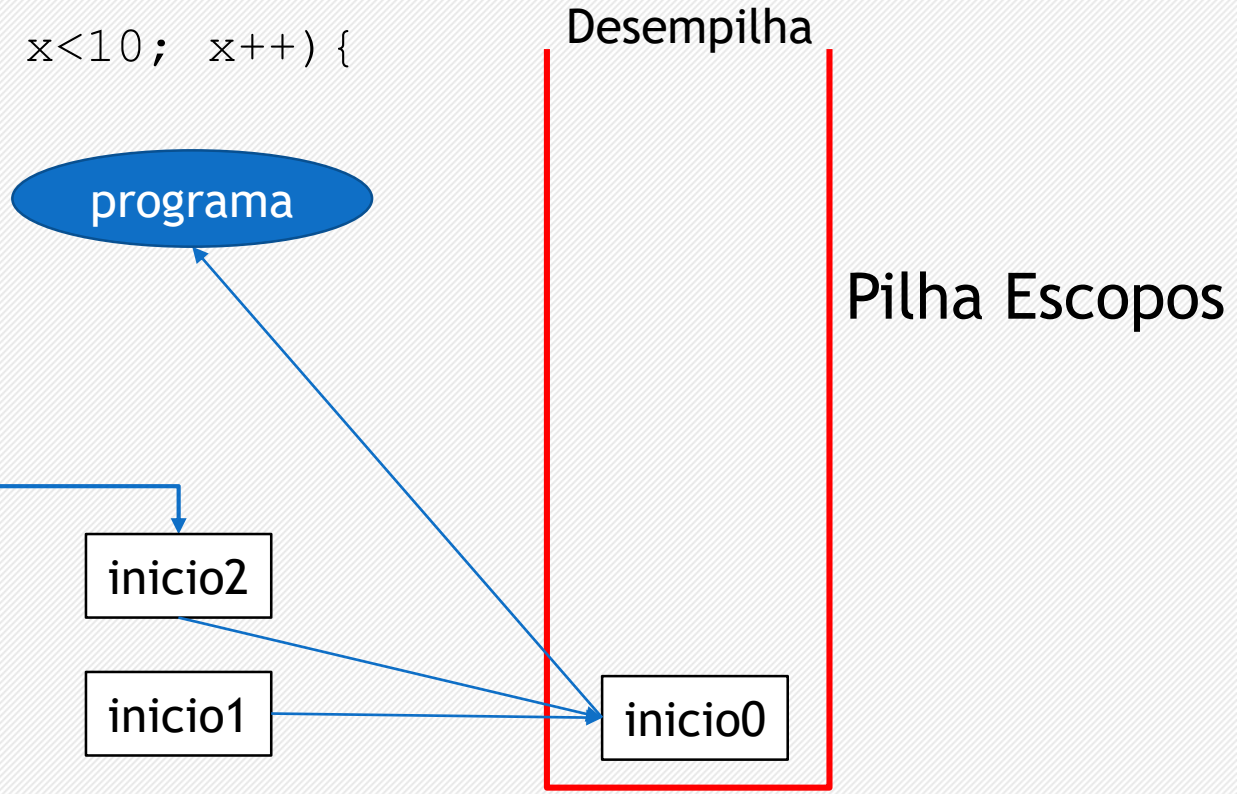
id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0
x	inteiro	inicio2

```
funcao inicio () {

   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```

fimEscopo()



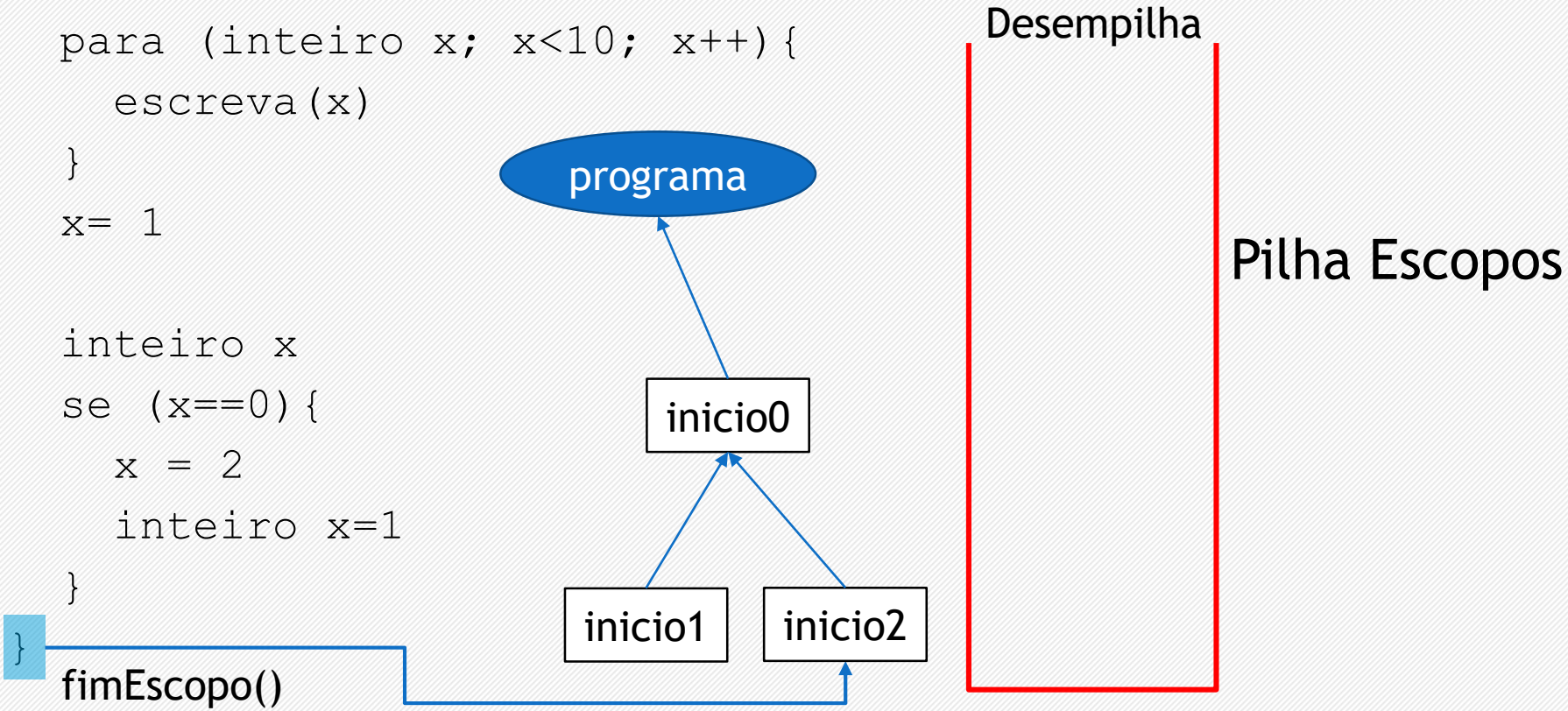
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0
x	inteiro	inicio2

```
funcao inicio () {

  para (inteiro x; x<10; x++){
    escreva(x)
  }
  x= 1

  inteiro x
  se (x==0){
    x = 2
    inteiro x=1
  }
}
```



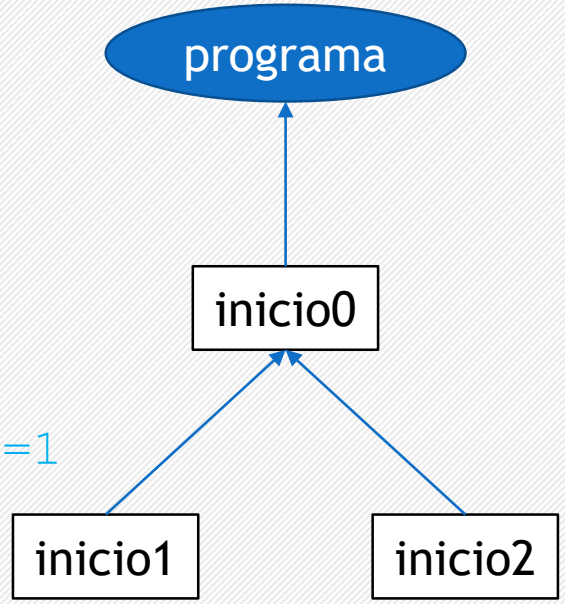
Escopos

id	tipo	escopo
x	inteiro	inicio1
x	inteiro	inicio0
x	inteiro	inicio2

```
funcao inicio () {

   para (inteiro x; x<10; x++){
      escreva(x)
   }
   x= 1

   inteiro x
   se (x==0){
      x = 2
      inteiro x=1
   }
}
```



Verificação de Tipos

Verificação de Tipos

```
int main (){

    int a,b,x;
    float c=0.5;
    char d='1';
    bool h=true;

    x = 1;
    a = 1;
    x = x + a + 1 + 1 + 1 + 1 / 2.0;
    a = 1 + 2 + 3 + c;
    b = 1 + 1 * d;
    if ( c > 0 ) {
       c = 1;
    }
}
```

Qual o tipo resultante das operações acima?

As atribuições são compatíveis?

Ações semânticas para atribuição com tipos incompatíveis

<atrib> ::= **id** ***#5*** **"< -"** **<**expr**>** ***#11***
<expr> ::= expr "*" termo ***#10***
| termo
<termo>::= termo "+" fator ***#10***
| fator

<fator>::= **id** ***#9***
| num_inteiro ***#8***
| num_real ***#7***
| string ***#6***

```
#5  tp_id = busca_tipo (token.getlexema());
#6  push ("string");
#7  push ("real");
#8  push ("int");
#9  tipo_id = busca_tipo (token.getlexema());
    push(tipo_id)
#10 tipo1 = pop();
    tipo2 = pop();
    if (tipo1 == tipo2) // tipos compatíveis
      push (tipo1);
    else
      push ("erro");
#11 tp_exp = pop();
    if (tp_id != tp_exp || tp_id == "erro" || tp_exp == "erro")
      msg ("erro tipos incompatíveis")
```

Ações semânticas para atribuição com tipos incompatíveis

<atrib> ::= **id #5 "< -" <expr> #11**
<expr> ::= expr "*" termo **#10**
| termo
<termo>::= termo "+" fator ***#10***
| fator

<fator>::= **id *#9***
| num_inteiro ***#8***
| num_real ***#7***
| string ***#6***

inteiro a
a <- 10 + a

```
#5  tp_id = busca_tipo (token.getlexema());
#6  push ("string");
#7  push ("real");
#8  push ("int");
#9  tipo_id = busca_tipo (token.getlexema());
    push(tipo_id)
#10 tipo1 = pop();
    tipo2 = pop();
    if (tipo1 == tipo2) // tipos compatíveis
      push (tipo1);
    else
      push ("erro");
#11 tp_exp = pop();
    if (tp_id != tp_exp || tp_id == "erro" || tp_exp == "erro")
      msg ("erro tipos incompatíveis")
```

a #5 <- 10 #8 + a#9
termo "+" fator #10
id <- <expr> #11

Tabela de Compatibilidade

Para expressões

ExpTable[OP1][OP2][OP]

SemanticTable.resultType(OP1,OP2,OP)

Retorna o tipo resultante ou erro!

Atribuições

AtribTable[ESQ][DIR]

SemanticTable.atribType(ESQ,DIR)

Retorna Ok, Warning ou Error!

Modelo de tabelas de compatibilidade

Exceções:

INT / INT = FLOAT

INT (+-*/) FLOAT = ???

FLOAT (+-*/) INT = ???

CHAR + CHAR = ???

STRING + CHAR = ????

CHAR + STRING = ???

Compatibilidade de operações

	int	char	float	string
int	+ - / *	erro	+ - / *	erro
char				
float				
string				

Compatibilidade de atribuições

	int	char	float	string
int	ok	erro	aviso	erro
char				
float				
string				

Ações semânticas para atribuição com tipos incompatíveis

<atrib> ::= **id** ***#5*** **"<-"** <expr> ***#11***
<expr> ::= expr "*" termo ***#10***
| termo
<termo>::= termo "+" fator ***#10***
| fator

<fator>::= **id** ***#9***
| num_inteiro ***#8***
| num_real ***#7***
| string ***#6***

```
#5  tp_id = busca_tipo (token.getlexema());
#6  push ("string");
#7  push ("real");
#8  push ("int");
#9  tipo_id = busca_tipo (token.getlexema());
    push(tipo_id)
#10 tipo1 = pop();
    tipo2 = pop();
    if (tipo1 == tipo2) // tipos compatíveis
      push (tipo1);
    else
      push ("erro");
#11 tp_exp = pop();
    if (tp_id != tp_exp || tp_id == "erro" || tp_exp == "erro")
      msg ("erro tipos incompatíveis")
```

Ações semânticas para atribuição com tipos incompatíveis

<atrib> ::= **id** ***#5*** **"<-"** <expr> ***#13***
<expr> ::= expr "*"***#10*** termo ***#12***
| termo
<termo>::= termo "+"***#11*** fator ***#12***
| fator

<fator>::= **id** ***#9***
| num_inteiro ***#8***
| num_real ***#7***
| string ***#6***

```
#5  tp_id = busca_tipo (token.getlexema())
#6  push (SemanticTable.STR) // Tipo do operando
#7  push (SemanticTable.FLO) // Tipo do operando
#8  push (SemanticTable.INT) // Tipo do operando
#9  tipo_id = busca_tipo (token.getlexema())
    push(tipo_id)             // Tipo do operando
#10 push(SemanticTable.MUL) // Operacao
#11 push(SemanticTable.SUM) // Operacao
#12 tipo2 = pop();
    operacao = pop();
    tipo1 = pop();
    resultExp = SemanticTable.resultType(tipo1,tipo2,operacao)
    if ( resultExp != SemanticTable.ERR ) // Tipos Compatíveis
    push(resultExp)
#13 resultExp = pop();
    resultAtrib=SemanticTable.atribType(tp_id,resultExp)
    if( resultAtrib != SemanticTable.ERR) show("Atribuicao Ok")
```