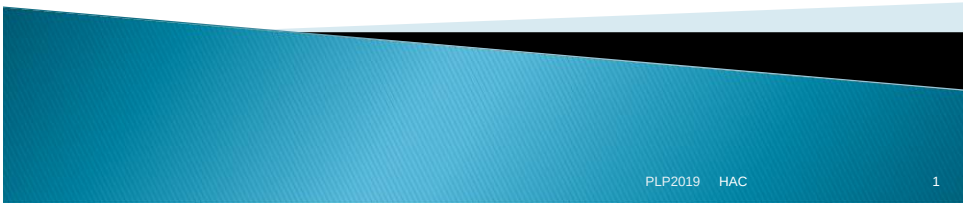
Programação Estruturada

Paradigmas de Linguagens de Programação

Profa. Heloisa – 1º. Sem. 2019



O material apresentado aqui foi extraído, em sua maior parte, de

Sebesta, R.W. *Concepts of Programming Languages*. 9a.edição/ Addison-Weley, 2009. Capítulo 5.

Programação Estruturada

Sebesta, R.W. *Concepts of Programming Languages*. 9a.edição/ Addison-Weley, 2009. Capítulo 5.

- As linguagens desse paradigma são muitas vezes chamadas de:
  - linguagens convencionais,
  - Linguagens de programação estruturada,
  - Linguagens procedurais,
  - linguagens imperativas.

Programação Estruturada

Sebesta, R.W. *Concepts of Programming Languages*. 9a.edição/ Addison-Weley, 2009. Capítulos 5, 9 e 10.

- **Linguagens imperativas:**
- São linguagens que "reconstroem" uma máquina para torná-la mais conveniente para programação.
- Máquinas que influenciaram fortemente a estrutura das linguagens de programação: arquitetura de von Neumann.

- **Características da Arquitetura de Máquina:**
- 1) unidade de processamento
- 2) memória
- 3) registradores que estabelecem comunicação entre 1 e 2

- **Conceitos introduzidos por essa máquina:**
- variável, valor e atribuição
- processamento seqüencial
- programa armazenado

As linguagens imperativas são projetadas de acordo com o princípio do

Modelo de Máquina: uma linguagem deve permitir usar diretamente uma máquina orientada por atribuições

Programação Estruturada

- **Linguagens de Programação Estruturada – ideia básica:**

Controle de Fluxo Estruturado –

um programa é dito estruturado quando o controle de fluxo é evidente da estrutura sintática do texto do programa.

Entidades e atributos de linguagens de programação

- Programas envolvem ***entidades***:
- variáveis, subprogramas, comandos, etc.
- Entidades tem ***atributos***.
  - Variável – nome, tipo, valor
  - Subprograma – nome, parâmetros
- Os atributos das entidades que aparecem em um programa podem ser definidos em diferentes momentos.
- A definição de atributos de entidades é feita por meio de uma operação chamada ***amarração***

Variáveis

- Uma variável de um programa é uma **abstração** de uma célula ou de uma coleção de células de memória de um computador.
- Uma variável pode ser caracterizada por uma sêxtupla de atributos:

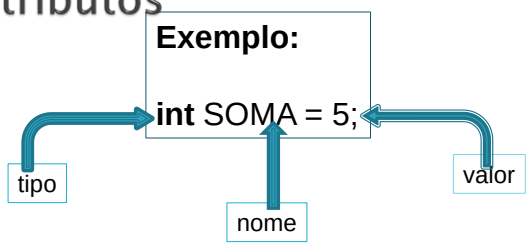
Nome, endereço, valor, tipo, tempo de vida e escopo.

Variáveis

- O estudo dos atributos de uma variável envolve os conceitos:
- sinonímia,
- amarração,
- tempo de amarração,
- declarações,
- regras de escopo e
- ambientes de referência.

Variáveis e seus atributos

Nome



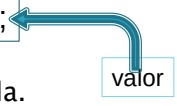
- Nome (ou identificador) é uma cadeia de caracteres usada para identificar alguma entidade em um programa (variável, subprograma, parâmetro,...).
- Cada linguagem de programação define uma regra de formação para nomes, mas a mais comum é uma letra seguida de uma cadeia de letras, dígitos e underscore.

Variáveis e seus atributos

Valor

Exemplo:

int SOMA = 5;



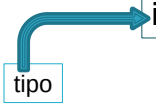
- É o conteúdo da célula de memória associada a ela.
- A célula de memória associada a uma variável é uma célula abstrata e não uma célula física.
- Uma célula física geralmente tem 1 byte
- Uma célula abstrata tem o tamanho necessário para o tipo da variável associada.

Variáveis e seus atributos

Tipo

Exemplo:

int SOMA = 5;



- Determina o intervalo de valores que uma variável pode armazenar e o conjunto de operações que são definidas para os valores desse tipo.
- Exemplo, tipo int especifica um intervalo de valores de -2147483648 a 2147483647 e as operações aritméticas.

Variáveis e seus atributos

Endereço

- É o endereço da posição de memória associada com a variável.
- Uma variável pode ter endereços diferentes em momentos diferentes (diferentes chamadas para um subprograma que tem uma variável local), que corresponde a instanciações da variável.
- Várias variáveis podem ter o mesmo endereço, o que é chamado de aliases

Variáveis e seus atributos – Endereço

- Exemplos (variáveis que tem o mesmo endereço):
  - Union em C e C++

```
struct item {
        char nome[50];
        float preco;
        union {
             float volume;
             unsigned peso;
        }
}
```

  - Volume e peso ocupam a mesma posição de memória (têm o mesmo endereço)
  - Apenas o primeiro da union campo pode ser inicializado
  - Apenas um campo da union pode ser acessado por vez

Variáveis e seus atributos – Endereço

- Exemplos (variáveis que tem o mesmo endereço):
  - Dois ponteiros apontando para a mesma posição de memória
  - Duas variáveis de referência apontando para a mesma posição de memória

Variáveis e seus atributos

Tempo de vida

- Intervalo de tempo durante o qual uma área de memória está amarrada a uma variável

Escopo

- Trecho do programa onde uma variável é conhecida.
- **(Antes de detalhar conceitos relacionados a tempo de vida e escopo, vamos ver os conceitos de amarração)**

Amarração

- ***Amarração*** : Associação entre uma entidade e seus atributos. Especifica a natureza exata dos atributos de uma entidade.
- ***Tempo de amarração***: momento em que a amarração ocorre – importante na diferenciação de linguagens.
- A amarração pode ser:
  - **estática** – ocorre antes da execução do programa e não pode ser mudada
  - **dinâmica** – ocorre em tempo de execução e pode ser mudada, respeitando as regras da linguagem.

Amarração e tempo de amarração de atributos da variável

- Valor
- Tipo
- Tempo de vida
- Escopo

Amarração de Valor

- Representado de forma codificada na área de memória amarrada a variável. Pode ser uma referência (ponteiro).
- Amarração dinâmica (mais comum)
  - através do comando de atribuição.
  - **Soma = 50;**
- Amarração estática
  - definição de constantes
  - PASCAL: **const n = 50**

Amarração de Tipo Estática

- Especificação da classe de valores que podem ser associados à variável, bem como das operações que podem ser usadas sobre eles.
- **Amarração de tipo estática**:
  - definida através da declaração de variáveis, que pode ser:
  - Declaração Explícita – comando que lista as variáveis e seus tipos;
  - **float x, y;**
  - **int i, j;**
  - Declaração Implícita – feita normalmente por convenções definidas para os nomes das variáveis.
  - Exemplo: no FORTRAN, variáveis que começam com I, J, K, L, M ou N são inteiras.

Amarração de Tipo Dinâmica

- **Amarração de tipo dinâmica**:
- variáveis não são declaradas, e a mesma variável pode conter valores de tipos diferentes durante a execução do programa.
- O tipo da variável é definido quando um valor é atribuído a essa variável. A variável sendo atribuída é amarrada ao tipo da expressão do lado direito da atribuição
- **Vantagem**: flexibilidade de programação
  - Exemplo: Um programa que trabalha com dados numéricos pode ser escrito de maneira genérica, tratando qualquer tipo de dado numérico

Amarração de Tipo Dinâmica

- Exemplo em JavaScript:

list = [34.5 4.8 5.1];
O tipo de list passa a ser um array unidimensional de tamanho 3

list = 47;
Neste ponto, a mesma variável list se torna um tipo escalar.

Amarração de Tipo Dinâmica

- **Desvantagens**:
  - Diminui a capacidade de detecção de erros pelo compilador
    - Qualquer variável pode receber valores de qualquer tipo;
    - Ao receber um valor de tipo diferente, a variável tem seu tipo automaticamente alterado.
    - **Exemplo em Javascript**: i e x tem valores escalares;
    - y é array
    - Supondo que a intenção é copiar x para i **i=x;**
    - Mas foi digitado (com erro): **i=y;**
    - **O erro não é detectado, i é convertido para o tipo array**

Amarração de Tipo Dinâmica

- **Desvantagens**:
  - Custo maior:
    - Verificação de tipo em tempo de execução;
    - Toda variável deve ter um descritor em tempo de execução associado para manter o tipo atual;
    - Memória usada pelo valor da variável deve ser de tamanho variável.
    - São linguagens usualmente interpretadas. Um compilador não pode construir código de máquina para um comando
      A+B
    - Sem saber o tipo das variáveis A e B

Amarração de memória e Tempo de vida de variável

- A amarração de memória a variáveis é uma característica importante das linguagens imperativas
- Tempo de vida é o intervalo de tempo durante o qual uma área de memória está amarrada a uma variável
- ***Alocação***: ação que adquire áreas de memória para variáveis de um *pool* de memória disponível.
- ***Desalocação:*** processo de devolver uma célula de memória que foi desligada de uma variável, ao pool de memória disponível.
- O tempo de vida de uma variável começa quando ela é associada a uma célula de memória específica e termina quando é desassociada dessa célula.

Amarração de memória e Tempo de vida de variável

- Do ponto de vista de amarração de memória as variáveis escalares (não estruturadas) podem ser:
- Estáticas
- Dinâmicas de pilha
- Dinâmicas de heap explícitas
- Dinâmicas de heap implícitas

Variáveis estáticas

- A alocação de memória ocorre antes da execução do programa, nos casos de:
- variáveis globais – devem ser acessíveis a todo o programa
- variáveis locais estáticas – são declaradas dentro de um subprograma mas devem reter valores entre execuções separadas do subprograma (sensíveis à história).
- **Vantagens**:
- eficiência – endereçamento direto (depende da implementação)
- não exige custo adicional para alocação e liberação de memória

Variáveis estáticas

- **Desvantagem**:
- pouca flexibilidade – linguagens que usam apenas alocação estática não dão suporte à:
  - subprogramas recursivos
  - nem a compartilhamento de memória (Subprogramas que executam em momentos diferentes e tem variáveis de grande dimensão).

Variáveis estáticas

- Exemplos:
  - FORTRAN I, II, IV – todas as variáveis eram estáticas.
  - C e C++ – permitem definir uma variável em uma função com o modificador ***static***, fazendo com que ela se torne estática.
  - O modificador ***static*** usado em uma definição de classe em C++, Java ou C# não tem o mesmo significado quanto à tempo de vida; nesse caso, signifca que a variável é uma variável de classe e não de instância.
  - Pascal – não possui variáveis estáticas

Variáveis dinâmicas de pilha

- **Variáveis dinâmicas de pilha são alocadas da pilha de execução**
- A alocação de memória da variável é feita quando a declaração dessa variável é ***elaborada***, mas o tipo da variável é amarrado estaticamente.
- ***Elaboração:*** processo de alocação que acontece no momento em que é iniciada a execução do código onde aparece a declaração (ativação).

Variáveis dinâmicas de pilha

- Para variáveis declaradas no início de um procedimento (função, método):
- Na chamada do procedimento acontece ⇒
  - ativação
  - **elaboração (alocação de memória)**
  - execução
- No término da execução do procedimento acontece ⇒
  - retorno do controle à unidade chamadora
  - **desalocação da memória**

Variáveis dinâmicas de pilha

```
// Exemplo em C
int soma(int P1, int P2)
                    //soma recebe P1,P2 e retorna um int
{
     int res;
     res = P1 + P2;
     return(res); //retornando o valor para a unidade que chamou
}
```

A variável **res** e os parâmetros **P1** e **P2** terão memória alocada na pilha de execução após a chamada da função

Variáveis dinâmicas de pilha

- Variáveis declaradas no início de um comando composto

```
main ( )  {
  int i = 0, x = 10;
  while (i++ < 100)  {
        float z = 3.34;
        .....
  }
}
```

A elaboração (alocação de memória) acontece no início da execução do comando ou no início da execução da unidade.

Variáveis dinâmicas de pilha

- Algumas linguagens (C++, JAVA) permitem que variáveis sejam declaradas em **qualquer lugar** do programa.
- Em algumas implementações, todas as variáveis declaradas em uma função ou método são amarradas a memória no **início da execução da função**, mesmo que sua declaração não apareçam no começo.
- A variável se torna visível a partir da sua declaração

Variáveis dinâmicas de pilha

Vantagens:

- implementação de recursão
- subprogramas usam a mesma memória

```
// Exemplo
#include <stdio.h>
int soma(int P1, int P2)
 {                  //soma recebe P1,P2 e retorna um int
     int res;
     res = P1 + P2;
     return(res); //retornando o valor para a unidade que chamou
}
void main()
{
      SOMA(2,10); //memória para parâmetros e res é alocada na pilha de execução
     SOMA(10,20);  //memória para parâmetros e res pode ser alocada
                        // na mesma posição que a chamada anterior
}
```


Variáveis dinâmicas de pilha

Desvantagem:

- custo adicional de alocação, que não é significativo, pois as variáveis declaradas no início de um subprograma são alocadas juntas.
- FORTRAN 77 e FORTRAN 90 – permitem o uso de variáveis dinâmicas de pilha
- C, C++, JAVA, C# – variáveis locais são dinâmicas de pilha por default
- Pascal e ADA – todas as variáveis (não-heap) de subprogramas são dinâmicas de pilha

Variáveis Dinâmicas de Heap – explícitas

- São alocadas na heap – coleção de células de memória de uso desorganizado, pois é imprevisível
- Células de memória sem nome são alocadas explicitamente por instruções do programador, por meio de :
  - Um operador (C++ ou Ada)
  - Chamada para um subprograma (C)
- Algumas linguagens tem também um operador para liberar a memória
- Só podem ser referenciadas por ponteiros ou variáveis de referência.
- São usadas frequentemente para implementar estruturas dinâmicas que crescem e diminuem durante a execução.

Variáveis Dinâmicas de Heap – explícitas

Exemplo: C

Funções malloc, calloc e realloc: alocam memória
Função free: libera a memória reservada

```
void * malloc(int num_bytes);

void free(void * p);
```

```
// Exemplo  em C
char *ptr;
ptr = malloc (10);
        // aloca memória na heap com o número de bytes
        // especificado – retorna ponteiro para primeira posição
.....
free(ptr)
```

Variáveis Dinâmicas de Heap – explícitas

Exemplo: C++

Operador **new** –
- Seu operando é um tipo
- Aloca uma posição de memória na heap e retorna um ponteiro para essa posição

```
int *intnode;     // Cria um ponteiro
intnode = new int;   // aloca uma posição de memória na
                     // heap do tipo int
....
delete intnode;      // desaloca a posição de memória
                     // para a qual intnode aponta
```

C++ tem o operador delete porque não tem liberação de memória implícita (garbage collection)

Variáveis Dinâmicas de Heap – explícitas

Outro exemplo – C++

```
int * iPtr = new int[numTests];
          // aloca memória na heap com o número de bytes
        // especificado – retorna ponteiro para primeira posição

.....
  delete [] iPtr;
```

Variáveis Dinâmicas de Heap – explícitas

JAVA:

Todos os dados, exceto primitivos escalares são objetos portanto são dinâmicos de heap e acessados por variáveis de referência.

A desalocação não é feita explicitamente, mas implicitamente pelas rotinas de coleta de lixo (garbage collection)

Desvantagens:

- Dificuldade de usar corretamente (ponteiros e variáveis de referência)
- Custo adicional de referência, alocação e desalocação.

Variáveis Dinâmicas de Heap – implícitas

- São alocadas na heap apenas quando valores são atribuídos
- Todos os atributos são definidos cada vez que os valores são atribuídos
- Exemplo em JavaScript:
- Lista = [56.1 7.8 43.5 5.6]
- **Vantagem**: alta flexibilidade
- **Desvantagens**:
  - custo elevado, todos os atributos são dinâmicos
  - Não permite verificação de erros pelo compilador

Escopo

- Trecho do programa onde uma variável é conhecida (visível).
- Uma variável é ***visível*** dentro do seu escopo e ***invisível*** fora dele.
- As **regras de escopo** de uma linguagem definem como uma ocorrência de um nome é associada a uma variável.
- Uma variável é **local** a uma unidade de programa se é declarada nessa unidade
- Uma variável é **não local** a uma unidade de programa se é visível mas não é declarada nessa unidade.

Escopo

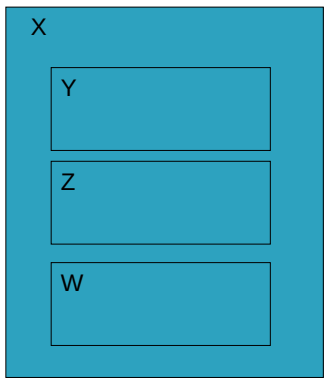
Escopo de variáveis

```
#include <stdio.h>
#include <conio.h>
//declaração de variáveis globais
// ---------------- Funçao main()-------------------
 int main(void)
 {
   //declaração das variáveis locais da main()
    return(0);
 }
// ------------------------------------------------
void funcao1(variáveis locais de parâmetros)
{      // declaração das variáveis locais da função1
      return;
 }
```

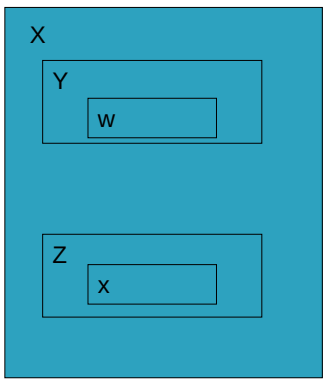
Escopo estático (léxico)

- O escopo das variáveis pode ser determinado estaticamente – antes da execução.
- Há duas categorias de linguagens com escopo estático:
  - Linguagens em que subprogramas podem ser aninhados, criando escopos estáticos aninhados
  - Linguagens em que subprogramas não podem ser aninhados e os escopos aninhados são criados por classes aninhadas e blocos
- Linguagens que permitem subprogramas aninhados: Ada, Pascal, JavaScript, Fortran 2003, PHP

Escopo estático (léxico)



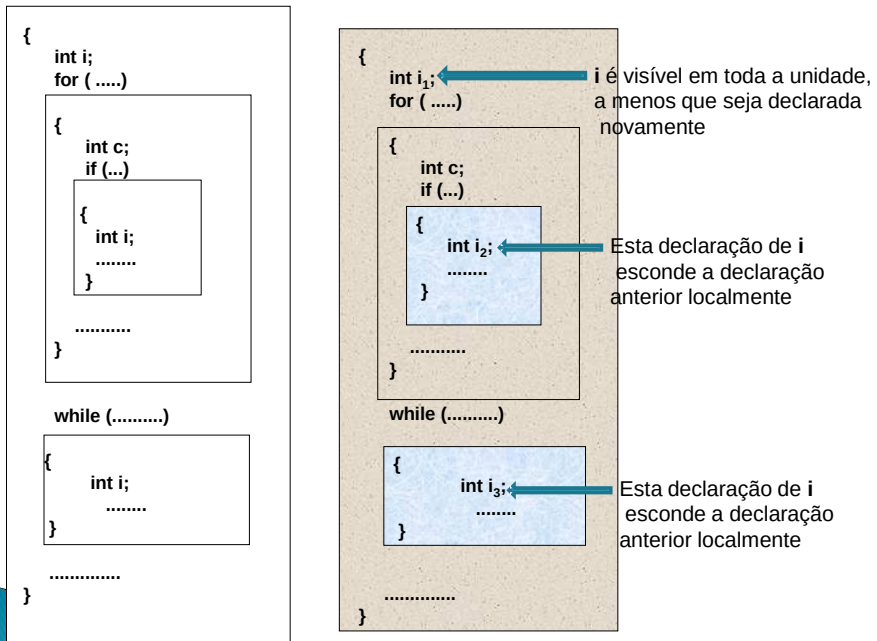
Subprogramas não aninhados



Subprogramas aninhados

Escopo estático (léxico)

- Quando uma variável é referenciada, seus atributos são determinados pelo comando em que ela é declarada
- Quando uma variável é referenciada em um subprograma, sua declaração é procurada nesse subprograma.
- Se não for encontrada, a busca continua no subprograma onde esse mais interno foi definido (pai estático), e assim por diante.



Escopo estático (léxico)

- Regras para definição do escopo:
  - Uma variável é visível na unidade em que foi declarada e nas internas, a menos que tenha sido redefinida.
  - Uma nova declaração da mesma variável em uma unidade mais interna esconde a definição anterior
  - Em algumas linguagens como ADA, variáveis escondidas, de escopos mais gerais, podem ser acessadas com referências seletivas:

P1.x

Blocos

- Muitas linguagens permitem que seções de código (blocos) tenham suas próprias variáveis locais.
- São variáveis dinâmicas de pilha
- Linguagens baseadas em C permitem que comandos compostos (sequência de comandos delimitados por chaves) tenham declarações iniciando um novo escopo

```
if (list[i] < list[j]) {
  int temp;
  temp = list[i];
  list[i] = list [j];
  list[j] = temp;
}
```

- Escopos criados por blocos, que podem estar aninhados, são **tratados exatamente da mesma maneira que aqueles criados por subprogramas**.

- Algumas linguagens (C++, JAVA) permitem que variáveis sejam definidas em qualquer lugar do bloco

```
void f( ) {
    int a = 1;
    a = a + 1;
    int b = 1;
    b = b + a;
}
```

- O escopo dessas variáveis é do ponto em que são definidas até o final do bloco (ou função) em que aparecem.

Escopo Global

- Definições de variáveis fora das funções (ou subprogramas em geral) cria variáveis globais, que podem ser visíveis em todas as funções.
- Linguagens em que o programa é uma sequência de definições de funções e permitem variáveis globais: C, C++, PHP, Python
- Uma variável global em C é implicitamente visível em todas as funções subsequentes do arquivo, exceto naquelas que tem uma definição local para uma variável com mesmo nome

Amarração Dinâmica a escopo

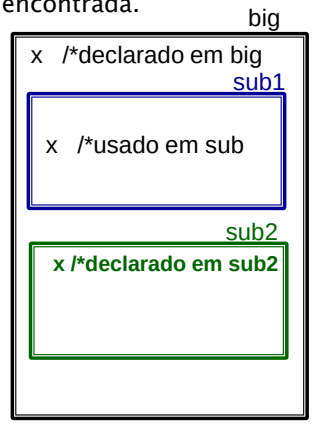
- o escopo é definido em função da execução do programa. O efeito de uma declaração se estende até que uma nova declaração com o mesmo nome seja encontrada.

```
procedure big;
  var x : integer;
  procedure sub1;
    begin { sub1 }
    ..... x .....
    end; { sub1 }
  procedure sub2;
    var x : integer;
    begin { sub2 }
    .....
    end; { sub2 }
  begin { big }
  .....
  end; { big }
```


Amarração Dinâmica a escopo

- o escopo é definido em função da execução do programa. O efeito de uma declaração se estende até que uma nova declaração com o mesmo nome seja encontrada.

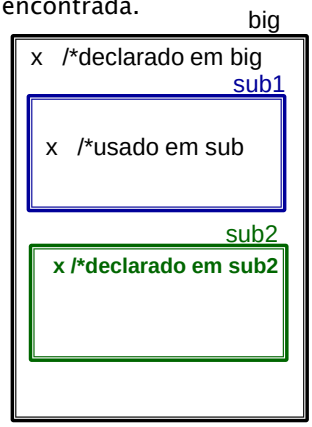
```
procedure big;
  var x : integer;
  procedure sub1;
    begin { sub1 }
    ..... x .....
    end; { sub1 }
  procedure sub2;
    var x : integer;
    begin { sub2 }
    .....
    end; { sub2 }
  begin { big }
  .....
  end; { big }
```



Amarração Dinâmica a escopo

- o escopo é definido em função da execução do programa. O efeito de uma declaração se estende até que uma nova declaração com o mesmo nome seja encontrada.

```
procedure big;
  var x : integer;
  procedure sub1;
    begin { sub1 }
    ..... x .....
    end; { sub1 }
  procedure sub2;
    var x : integer;
    begin { sub2 }
    .....
    end; { sub2 }
  begin { big }
  .....
  end; { big }
```



A qual declaração se refere esse uso da variável x?

Amarração Dinâmica a escopo

- As referências a um identificador não podem ser identificadas na compilação
- Para a sequência de chamadas: big - sub2 - sub1
  - a referência a x em sub1 é ao x declarado em sub2
- Para a sequência de chamadas: big - sub1
  - a referência a x em sub1 é ao x declarado em big

Ambiente de referência

- Ambiente de referência de um comando é a coleção de todas as variáveis que são visíveis no comando.
- O ambiente de referência de um comando em uma linguagem de escopo estático é formado pelas variáveis declaradas no seu escopo local mais as declaradas nos escopos ancestrais que são visíveis

Exemplo em ADA

```
procedure Example is
  A, B, : Integer;
  ......
  procedure Sub1 is
      X, Y, : Integer;
      begin { sub1 }
      ......  <----------------- 1
      end; { sub1 }
  procedure Sub2 is;
       X : Integer;
       ......
       procedure Sub3 is;
          X : Integer;
          begin { Sub3}
           ......  <-------------2
           end; { Sub3 }
        begin { Sub2}
           ......  <-------------3
        end; { Sub2 }
 begin { Example}
  ......  <------------------4
 end; { Example }
```

Ambientes de referência

Ponto 1:
X e Y de Sub1, A e B de Example

Ponto 2:
X de Sub3, (X de Sub2 escondido), A e B de Example

Ponto 3:
X de Sub2, A e B de Example

Ponto 4:
A e B de Example

Exemplo em linguagem semelhante a C com escopo dinâmico

```
void  sub1 ( )   {
      int   a, b;
      ......  <----------------1
}   /* end of sub1   */
void  sub2 ( )   {
      int   b, c;
      ......  <----------------2
      sub1;
}   /* end of sub2   */
void main ( ) {
    int   c, d;
      ......  <----------------3
      sub2( );
}   /* end of main   */
```

Ambientes de referência:

Ponto 1:
a e b de sub1, c de sub2, d de main
(c de main e b de sub2 estão escondidas)

Ponto 2:
b e c de sub2, d de main
(c de main está escondida)

Ponto 3:
c e d de main