Linguagens de Programação

Nomes, Amarrações e Escopo

Andrei Rimsa Álvares
andrei@cefetmg.br

Sumário

- Nomes
- Amarrações
- Escopo

NOMES

Nomes

- Conceito mais amplo que variáveis

- Podem representar
  - Variáveis
  - Labels
  - Sub-programas (funções)
  - Parâmetros formais
  - Outras construções

Decisões de Design

- Qual deve ser o tamanho máximo de um nome?
- Pode-se utilizar caracteres conectores?
- Os nomes são *case-sensitive*?
- Palavras especiais são reservadas?

Formato dos Nomes

- Tamanho fixo ou variável?
  - FORTRAN I e FORTRAN 77 ⇨ 6 caracteres
  - FORTRAN 90 e C ⇨ 31 caracteres
  - C++ ⇨ Sem limites na teoria, mas depende de implementação
  - Java ⇨ Sem limites
- Case-sensitive ou case-insensitive?
  - Case-insensitive diminui a legibilidade do código
    - Ex.: rosa, Rosa, ROSA

Palavras Especiais

- Palavras especiais
  - Nomear ações a serem executadas
  - Separam entidades estáticas

- Palavras reservadas vs palavras-chave
  - Ex.: em FORTRAN

```
REAL laranja
REAL = 3.4
```

```
REAL INTEGER
INTEGER REAL
```

Variáveis

- Variáveis são abstrações de endereços de memória
- Tornam os programas mais fáceis de codificar e entender
- Uma variável pode ser caracterizada pelo seu:
  - Nome
  - Endereço
  - Tipo
  - Valor
  - Escopo
  - Tempo de vida

Variáveis: Nomes

- Identificadores das variáveis

Variáveis: Endereços

- Endereço de memória associado
- Um mesmo nome pode estar associado a diferentes endereços
  - Duas variáveis com o mesmo nome (em funções distintas)
  - Mesma variável, mas em uma função recursiva
- Este endereço é conhecido como *l-value*

Variáveis: Aliases

- Múltiplos identificadores apontando para a mesma área de memória

- Pode haver perda de legibilidade
  - A modificação de uma variável tem efeito em outra variável

Variáveis: Tipo

- Determina faixa de valores que uma variável pode receber
  - Exemplo: short (-32.768 a 32767)

- Conjunto de operações associadas
  - Operações: **+**, **-**, *****, **/**
  - Funções: **abs**, **exp**

Variáveis: Valor

- O conteúdo do endereço de memória associado a variável
  - Abstração da memória com o tamanho da variável

- Também conhecida como *r-value*
  - Para acessar o *r-value*, deve-se conhecer o *l-value*

AMARRAÇÕES

Linguagens de Programação

Conceituação

- **Amarração** (ou *binding*) é uma associação entre entidades de programação
  - Variável/Valor
  - Identificador/Tipo
  - Operador/Símbolo

- O tempo que ocorre a amarração é chamado **tempo de amarração**

- Enfoque na amarração de identificadores a entidades

Tempos de Amarração

- As construções de uma linguagem podem ter os seguintes tempos de amarração

1) Tempo de projeto da LP
2) Tempo de implementação
3) Tempo de compilação
4) Tempo de ligação
5) Tempo de carga
6) Tempo de execução

Por que é importante saber o tempo de amarração?

1) Tempo de Projeto da LP

- Definição dos símbolos da linguagem e seu comportamento
  - Ex: * (multiplicação)

- Definição das palavras reservadas e seu comportamento
  - Ex.: `true`, `if`, `while`

2) Tempo de Implementação

- Implementadas durante a codificação do tradutor (em geral nos compiladores)

- Exemplo: Em C, definição da faixa de valores do tipo
  - `char` ⇨ -128 a 127
  - `int` ⇨ -2,147,483,648 a 2,147,483,647

3) Tempo de Compilação

- Associação da variável com seu tipo
  - Exemplo:

```
int i;
char c;
```

- Associação de expressões com seus operadores
  - Exemplo:

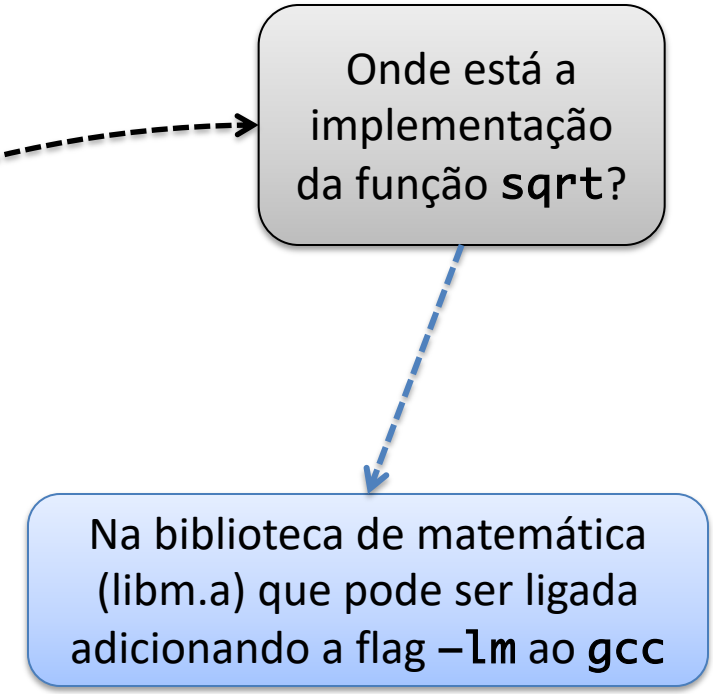
```
3 * 2
(x >> 4) & 0xF
```

4) Tempo de Ligação

- Integração (ligação) de vários módulos compilados previamente
- Exemplo

```
#include <stdio.h>
#include <math.h>

void main()  {
  printf("%f\n", sqrt(25));
}
```



5) Tempo de Carga

- Durante o carregamento do programa
  - Memórias associadas a variáveis globais e constantes
  - Endereços de memória relativos substituídos por endereços de memória absolutos

6) Tempo de Execução

- Ocorridas em tempo de execução

- Exemplo
  - Associação de valor a variável
    - Ex: `int x = 5;`

  - Associação de áreas de memória a variáveis
    - Ex: `int x = new int;`

Exercício

- Considere o seguinte trecho de código em C e preencha a tabela:

```c
int cont;
/* ... */
cont = cont + 5;
```

Exemplo	Tempo de Amarração
Possíveis tipos de *cont*	
Tipo de *cont*	
Possíveis valores de *cont*	
Valor de *cont*	
Possíveis significados do operador +	
Significado do operador + nesta atribuição	
Representação interna do literal 5	

Amarrações

- O momento em que ocorre a ligação pode ser classificado como **cedo** (*early binding*) ou **tardio** (*late binding*)

- Quanto mais cedo ocorre a ligação, maior a eficiência de execução do programa, mas menor a flexibilidade das estruturas disponibilizadas

Amarração de Tipo

- O tipo deve ser amarrado a variável para que esta seja referenciada pelos programas

- Aspectos importantes
  - Como o tipo deve ser definido?
  - Quando a amarração deve ser feita?

- Amarrações de tipos

Amarração Estática	Amarração Dinâmica
• Antes da execução • Permanece inalterada na execução	• Durante a execução • Pode ser criada/alterada na execução

Amarração de Tipo Estática

- Pode ser obtida através de
  - **Declaração explícita**: instrução do programa que lista nomes de variáveis e especifica os tipos.
    - Ex.: em C: int i, char c;
  - **Declaração implícita**: Convenções determinam seu tipo.
    - Exemplo em Fortran: começam com I, J, K, L, M, ou N (**INTEGER**), outras (**REAL**)
    - Exemplo em Perl: começam com $ (**escalar**), @ (**array**), % (**hashtable**)

Amarração de Tipo Dinâmica

- A variável é vinculada a um tipo quando lhe é atribuída um valor em uma instrução de atribuição
- A variável que está sendo atribuída é vinculada ao tipo do valor, variável ou expressão do lado direito da atribuição
- Vantagens:
  - Flexibilidade e generalidade
- Desvantagens:
  - Redução da capacidade de detecção de erro pelo compilador
  - Custo de implementação da vinculação dinâmica

Amarração de Tipo Dinâmica

- Exemplo de alteração de tipo dinamicamente em C#

```
dynamic variavel = new Int32();
variavel = 10;   // Int32
variavel = 2.5;  // Double
variavel = ”25/02/2011"; //String
variavel = DateTime.Parse(variavel); // Data
```

- Exemplo de operadores que alteram o tipo
  - PHP

```
<?php
  $x = 20;
  var_dump($x);
  $x .= 'Texto...';
  var_dump($x);
?>
```

  - JavaScript

```
<script>
   var x = 20;
   alert(typeof(x));
   x += "Texto...";
   alert(typeof(x));
</script>
```

Amarrações de Armazenamento

- **Alocação** ⇨ Amarração feita em uma célula de memória disponível obtida de um pool de memória

- **Desalocação** ⇨ Devolver a célula de memória desvinculada à variável de volta ao pool de memória

- **Tempo de vida** ⇨ O tempo no qual aquela variável está vinculada à célula de memória alocada
  - Começa quando a variável é alocada e termina quando ela é desalocada

Amarrações de Armazenamento

- Classificação das variáveis quanto à amarração de armazenamento
  - Variáveis estáticas
  - Variáveis stack-dinâmicas (pilha)
  - Variáveis heap-dinâmicas
    - Explícitas
    - Implícitas

Variáveis Estáticas

- Vinculadas à memória antes da execução e permanecem até o programa terminar

- Vantagens
  - Variáveis em subprogramas sensíveis à história (retêm valores entre chamadas)
  - Eficiência: endereçamento direto; inexistência de *overhead* de alocação e desalocação

- Desvantagens
  - Flexibilidade reduzida (recursão)
  - Impossibilidade de compartilhamento da memória

Variáveis Estáticas

- Exemplo em C

```c
#include <stdio.h>

static char* var = "global";

void main() {
   printf("%s\n", var);
}
```

Em C é alocada em tempo de carga

- Exemplo em Java

```java
class Foo {
   public static String bar = "foobar";
}
```

Em Java é alocada em tempo de carregamento da classe

Variáveis Stack-Dinâmicas

- Vinculadas à células de memória no momento da declaração, mas o tipo é definido estaticamente

- Vantagens
  - Permite recursividade
  - Permite o compartilhamento de memória

- Desvantagens
  - Overhead de alocação e desalocação em tempo de execução
  - Não são sensíveis à história

Variáveis Stack-Dinâmicas

- Exemplo em C

```
void funct(int size) {
  char buffer[size];
  /* ... */
}
```

Variáveis Heap-Dinâmicas Explícitas

- Células (abstratas) de memória alocadas e desalocadas pelo programador durante a execução do programa
- A heap mantém uma coleção de células alocadas dinamicamente (altamente desorganizada)
- Células somente podem ser referenciadas através de ponteiros ou referências
- A vinculação de tipo é feita estaticamente, mas o armazenamento é feito dinamicamente

Variáveis Heap-Dinâmicas Explícitas

- Vantagens
  - Gestão dinâmica de memória
- Desvantagens
  - Custo associado a manutenção da heap (alocação/desalocação)
  - A utilização de ponteiros não é segura

Variáveis Heap-Dinâmicas Explícitas

- Exemplo em C

```c
int* node = malloc(sizeof(int));
/* ... */
free(node);
```

- Exemplo em C++

```cpp
int* node = new int;
/* ... */
delete node;
```

- Exemplo em Java

```java
Integer node = new Integer(10);
/* ... */
```

Onde é feita a liberação da memória em Java?

Variáveis Heap-Dinâmicas Implícitas

- Armazenamento (célula de memória) somente é associado quando um valor é atribuído

- Vantagens
  - Elevado grau de flexibilidade do tipo de variável, permitindo definir códigos genéricos

- Desvantagens
  - Ineficiente porque todos os atributos da variável são dinâmicos
  - Dificuldade de detecção de erros pelo compilador

Variáveis Heap-Dinâmicas Implícitas

- Exemplo em JavaScript

```
<script>
    var list = [ 5, "texto" ];
    alert(typeof(list));
</script>
```

- Exemplo em Perl

```
#!/usr/bin/perl

my @array = ( "1", "2", "3" );

for $e (@array) {
    print $e . "\n";
}
```

ESCOPO

Escopo

- Em quais partes do programa (instruções) uma variável é visível (visibilidade)
  - Ou seja, se pode ser referenciada naquele ponto
- Variáveis não-locais de uma unidade do programa são aquelas visíveis, mas não declaradas lá
- As regras de escopo determinam como os nomes são associados com as variáveis

Escopo

- Exemplo

```
01: int a = 13;
02: void f() {
03:    int b = a;    // Referência à variável do ponto 1
04:    int a = 2;
05:    b = b + a;// Referência à variável do ponto 4
06: }
```

Escopos Aninhados

- Definição de escopos dentro de outros, formando uma sequência de escopos com sucessores e antecessores

- A redefinição de uma variável com o mesmo nome (num escopo interior) de uma já existente num escopo exterior, permite "esconder" (*shadow*) a definição exterior
  - C++, Pascal, ADA permitem o acesso a estas variáveis escondidas em escopos exteriores
  - Já Java não aceita *shadow* dentro de procedimentos

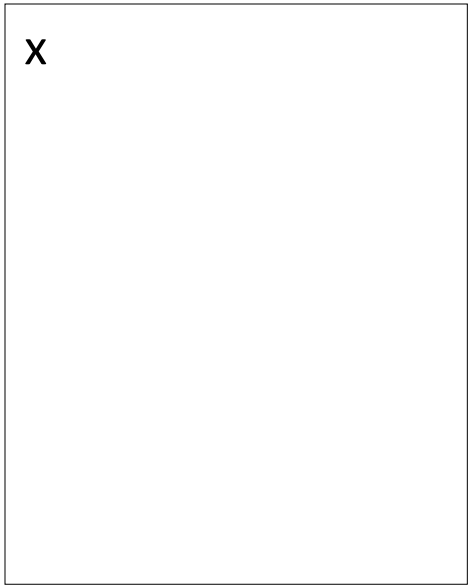
Tipos de Escopo

- Estático
  - Baseado na descrição textual do programa
  - Amarrações feitas em tempo de compilação
    - Ex: Algol, Pascal, C, C++, Java, C#, Ada…
- Dinâmico
  - Baseado na sequência de ativação (chamadas) dos módulos do programa
  - Amarrações feitas em tempo de execução
    - Ex: Lisp, APL, SNOBOL, Smalltalk…

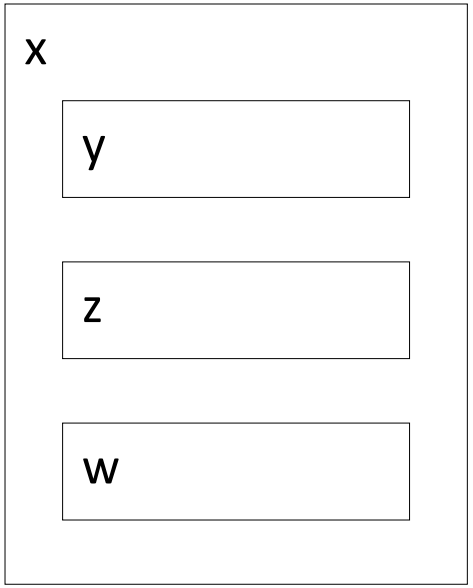
Escopo Estático

- Blocos
  - Permitem que uma seção de código possua suas próprias variáveis locais com escopo reduzido
  - Trecho de código delimitado por marcadores
    - Pascal: **begin** e **end**
    - C, C++: **{** e **}**

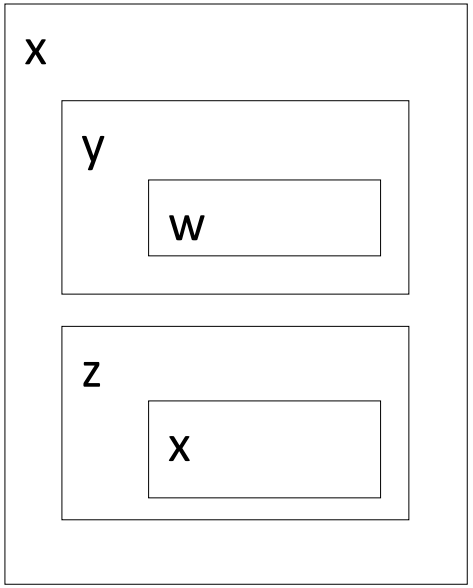
Escopo Estático



Bloco Monolítico

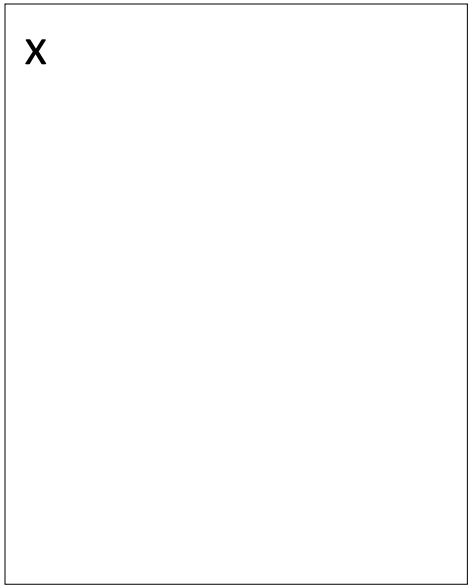


Blocos Não Aninhados



Blocos Aninhados

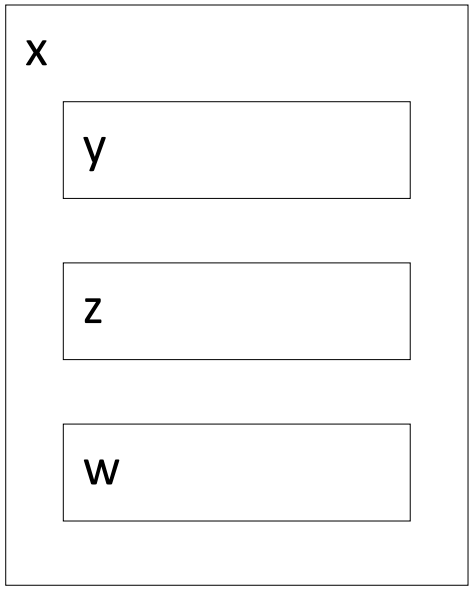
Escopo Estático



Bloco Monolítico

- Programa com um único bloco
  - Amarrações com visibilidade global
- Estrutura muito elementar
  - Não apropriada para grandes programas
  - Dificulta a codificação por grandes equipes de programadores
- Ex: BASIC e COBOL

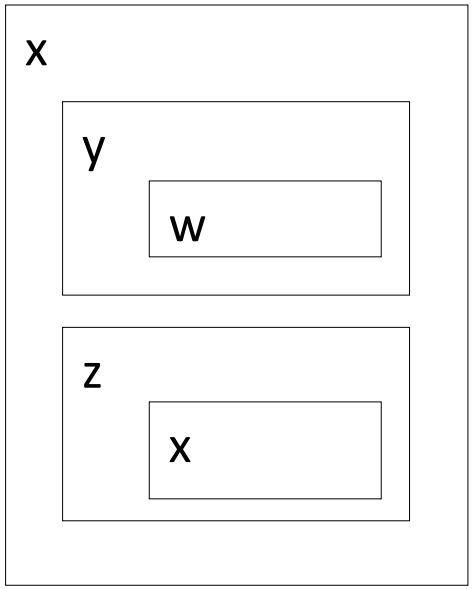
Escopo Estático



Blocos Não Aninhados

- Programa dividido em blocos
  - Somente locais ou globais
- Qualquer variável não-local deve ser global
- Exemplo
  - FORTRAN: Todos os subprogramas são separados e cada um atua como um bloco

Escopo Estático



Blocos Aninhados

- Blocos podem ser aninhados dentro de outros blocos
- Identificadores podem ser amarrados em cada bloco
  - Identificadores são procurados de "dentro para fora"
- Vantagem: programas + legíveis
- Desvantagem: ocultamento de variáveis

Escopo Estático

- Exemplo de ocultamento em C

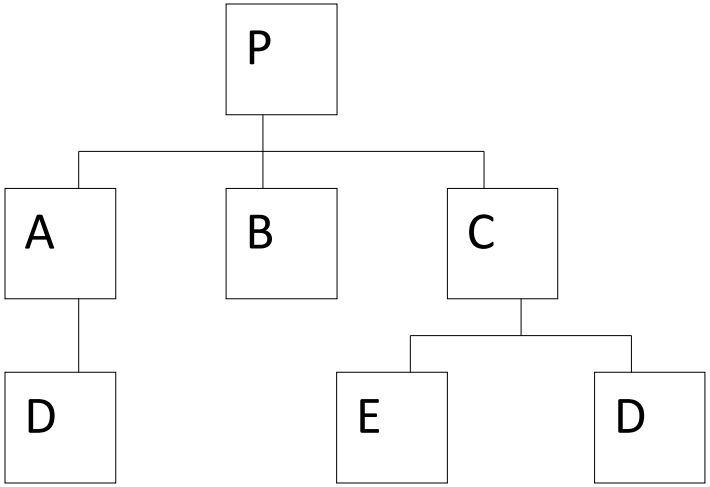
```c
#include <stdio.h>

void main() {
    int i = 0;
    int x = 10;
    while (i++ < 100) {
        float x = 3.231;
        printf("x = %f\r\n", x * i);
    }
}
```

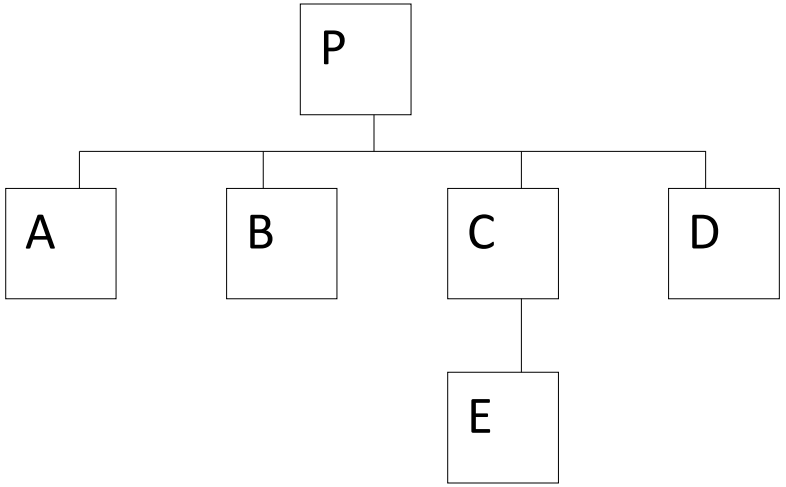
Java não permite tal ocultamento!

Escopo Estático

- Problemas com estrutura aninhada



Bloco D duplicado



Bloco D acessível por mais blocos que intencionado

Escopo Estático

- C utiliza uma abordagem mista
  - Definidos por funções: **Blocos não-aninhados**
    - Funções podem ser reusadas e variáveis globais podem ser compartilhadas (visibilidade global)
  - Definidos internamente: **Blocos aninhados**
    - Não pode ser reaproveitado

Escopo Estático

- Exemplo em C

```
01: #include <stdio.h>
02:
03: int x = 10;
04: int y = 15;
05:
06: void f() {
07:   if (y – x)
08:     int z = x + y;
09: }
10:
```

```
11: void g() {
12:   int w;
13:   w = x;
14: }
15:
16: void main() {
17:   f();
18:   x = x + 3;
19:   g();
20: }
```

Escopo Dinâmico

- São amarradas de acordo com o fluxo de programa
- Identificador é associado à declaração mais próxima na pilha de chamada
- Exemplo em linguagem hipotética

```
01: program p  {
02:     integer x = 1;
03:     procedure sub1() {
04:         print(x);
05:     }
06:     procedure sub2() {
07:         integer x = 3;
08:         sub1();
09:     }
10:     sub2();
11:     sub1();
12: }
```

Escopo Dinâmico

- Problemas
  - **Perda de eficiência** ⇨ verificação de tipos em tempo de execução
  - **Redução na legibilidade** ⇨ difícil determinar a sequência de chamadas
  - **Acesso lento a variáveis não-locais** ⇨ difícil identificar os identificadores da sequência de chamadas
  - **Redução da confiabilidade** ⇨ variáveis locais acessadas por quaisquer funções chamadas

Tempo de Vida

- Escopo e tempo de vida possuem uma relação próxima, mas são conceitos **diferentes**
  - Escopo estático é um conceito **textual**
  - Tempo de vida é um conceito **temporal**

Tempo de Vida

```
program P
var m: integer;
   procedure R (n: integer);
   begin
      if n > 0 then R (n-1)
   end;
begin
   R(2)
end.
```

Escopo de n

Escopo de m

Tempo de Vida

- Exemplo em C

```
void foo() {
    static int y;
    /* ... */
}

int main() {
    int x;
    foo();
}
```

- Variável x
  - **Escopo:** não estende foo
  - **Tempo de vida:** estende foo
- Variável **y**
  - **Escopo:** visível em foo
  - **Tempo de vida:** estende por todo o programa

Ambientes de Referenciamento

- Conjunto de todos os nomes visíveis em uma determinada instrução ou ponto de programa

- No escopo estático, o ambiente de referenciamento é constituído de todas as variáveis declaradas em seu escopo local, com o conjunto de todas as variáveis de seus escopos ascendentes visíveis

Ambientes de Referenciamento

```
program exemplo;
    var a,b: integer;
    procedure sub1;
        var x,y: integer;
        begin (1) end;
    procedure sub2;
        var x: integer;
        procedure sub3;
            var x: integer;
            begin (2) end;
        begin (3) end;
    begin (4) end;
```

- Ponto (1)
  - x, y (???)
  - a, b (???)
- Ponto (2)
  - x (???)
  - a, b (???)
- Ponto (3)
  - x (???)
  - a, b (???)
- Ponto (4)
  - a, b (???)

Ambientes de Referenciamento

```
program exemplo;
    var a,b: integer;
    procedure sub1;
        var x,y: integer;
        begin (1) end;
    procedure sub2;
        var x: integer;
        procedure sub3;
            var x: integer;
            begin (2) end;
        begin (3) end;
    begin (4) end;
```

- Ponto (1)
  - x, y (sub1)
  - a, b (exemplo)
- Ponto (2)
  - x (sub3)
  - a, b (exemplo)
- Ponto (3)
  - x (sub2)
  - a, b (exemplo)
- Ponto (4)
  - a, b (exemplo)

Constantes

- Uma constante é uma variável vinculada a um valor no momento em que é vinculada a uma célula de memória
- O valor das constantes não pode ser alterado
- Ajudam na legibilidade e confiabilidade do programa

Inicialização de Variáveis

- A vinculação de uma variável a um valor, no momento em que é vinculada a uma célula de memória, é designada por inicialização

- Exemplos
  - Fortran: REAL PI INTEGER SOMA
    DATA SOMA /0/, PI /3,14159/
  - Ada: SOMA: INTEGER : = 0;

- Pascal não oferece meios de se inicializar variáveis, exceto na atribuição

ISSO É TUDO, PESSOAL!