Linguagens de Programação

Nomes, Vinculações, Verificação de Tipos e Escopos

Baseado em Conceitos de Linguagens de Programação – Robert W. Sebesta

Prof. Lucas Ismaily

Universidade Federal do Ceará
Campus Quixadá

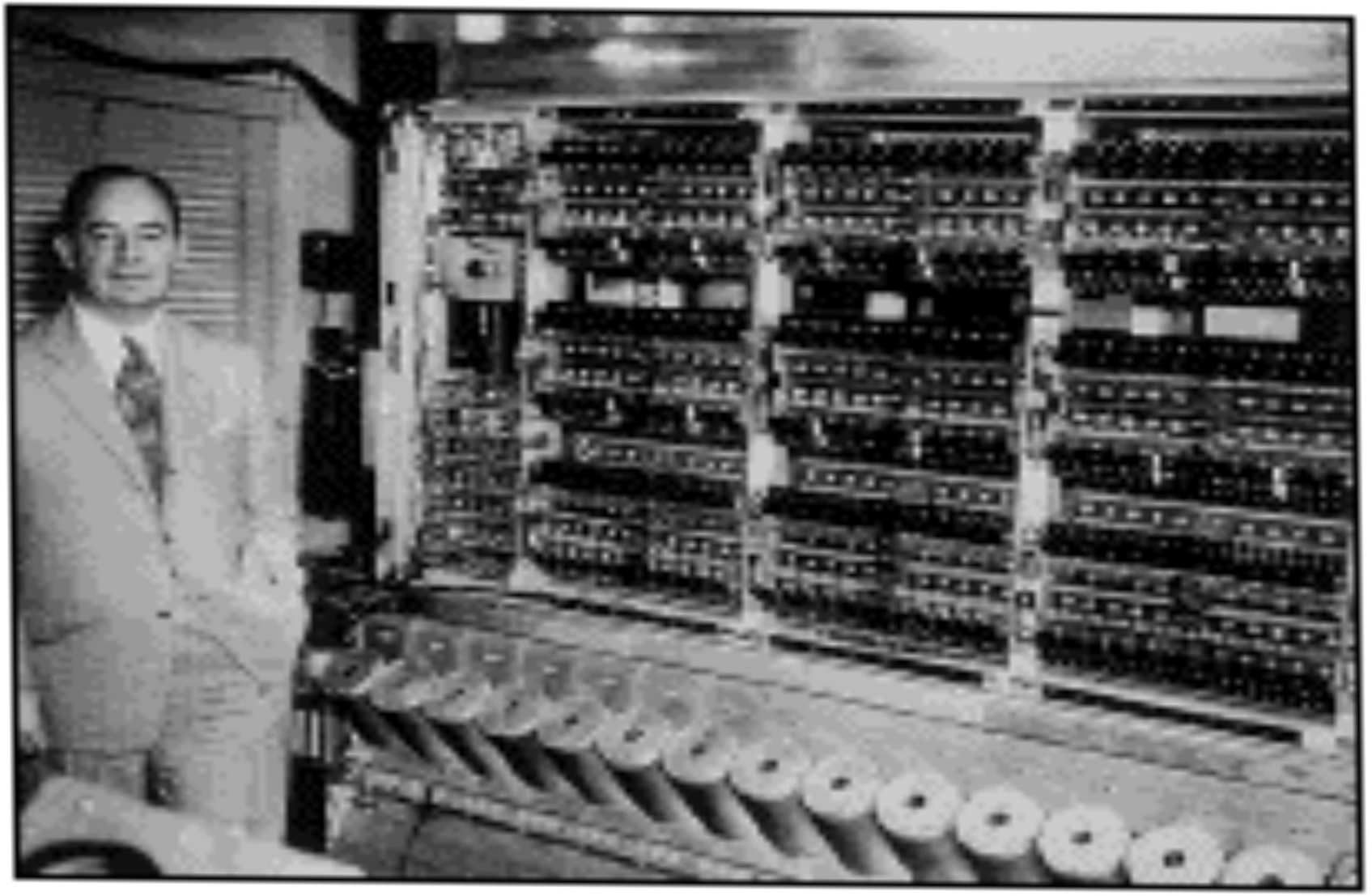
Roteiro

1. Introdução
2. Nomes
3. Variáveis
4. O Conceito de Vinculação
5. Verificação de Tipos
6. Tipificação Forte
7. Compatibilidade de Tipos
8. Escopo
9. Escopo e Tempo de Vida
10. Ambientes de Referenciamento
11. Constantes
12. Inicialização de Variáveis

Introdução

- As linguagens de programação imperativas são baseadas na arquitectura de Von Neumann.
- Os espaços de memória utilizados pelas **variáveis** são diferentes, de acordo com a especificação do **tipo** de variável (propriedade mais importante da variável).

Máquina de von Neumann



Nomes

- Principais considerações acerca de nomes:
  - Qual é o comprimento máximo de um nome?
  - Quais os caracteres de conexão ($? # @ ~; : / \ | ...) que podem ser utilizados em nomes?
  - Os nomes fazem distinção entre maiúsculas e minúsculas?
  - As palavras especiais são:
    - Palavras reservadas? ou
    - Palavras-chave?

Nomes

- Um Nome é uma *palavra* utilizada para identificar uma entidade de um programa.
- As primeiras L.P. utilizavam nomes com apenas 1 carácter, pois eram basicamente matemáticas.
- Exs. de comprimentos máximos de nomes:
  - FORTRAN I: máximo 6 caracteres
  - COBOL: máximo 30 caracteres
  - FORTRAN: 90 e ANSI C: máximo 31 caracteres
  - Ada: não tem limite, e todos são significativos
  - C++: não tem limite, mas as implementações sim.

Nomes

- Caracteres de conexão ($? # ~; : / \ | _ etc.)
  - Pascal, Modula-2, Fortran não permitem
  - As L.P. mais recentes permitem
- Distinção entre letras Maiúsculas e Minúsculas
  - Em C, C++, Java e Modula-2 os nomes são sensíveis às letras maiúsculas e minúsculas;
  - Os nomes em muitas das outras linguagens não são sensíveis.

Nomes

- A sensibilidade a letras maiúsculas e minúsculas tem a seguinte desvantagem:
  - Nomes que parecem iguais são diferentes (problema de legibilidade).

    Ex.: Em C++ os seguintes nomes são distintos: rest, Rest, ResT, REST, ...
  - PIOR problema de legibilidade: Em C++ e Java, os nomes pré-definidos utilizam uma mistura de letras maiúsculas e minúsculas. ex.: IndexOutOfBoundsException()
- Alias: Muitas vezes são utilizados nomes diferentes para referir a mesma entidade
  - Ex. em linguagem C:

typedef int myint; - 'int' e 'myint' são nomes diferentes que referem o mesmo tipo números inteiros 'int'.

Nomes

- Palavras Especiais
  - São utilizadas para tornar programas mais legíveis ao denominar acções.
  - Na maioria das L.P., as palavras especiais são classificadas como reservadas, e em algumas, são somente palavras-chave.
- Palavra reservada: é independente do contexto e não pode ser utilizada como um nome.
  - Exemplo:
    - INTEGER REAL (Não permitido – se REAL for reservada)
    - REAL INTEGER (Não permitido – se INTEGER reservada)

Nomes

- Palavra-chave: é uma palavra que pode tanto ser especial como não, dependendo do contexto onde está a ser utilizada.
  - Exemplo:
    - REAL AUXILIAR
    - REAL REAL (Permitido se REAL for palavra-chave)
    - REAL = 3.4 (Permitido se REAL for palavra-chave)
  - As Palavras-chave criam um problema de legibilidade.

Variáveis

- Uma Variável é uma abstracção de uma ou mais células de memória de um computador.
- Uma variável pode ser caracterizada através dos seguintes seis atributos:
  - Nome;
  - Endereço;
  - Valor;
  - Tipo;
  - Tempo de vida;
  - Escopo.

Variáveis

- Nome: identifica uma entidade.
  - Nem todas as variáveis têm nome.

  ex.: variáveis temporárias show(**x+y**)
- Endereço: é o endereço de memória à qual está associada uma variável.
- Os *Aliases* existem quando duas ou mais variáveis apontam para o mesmo endereço de memória.
  - Os *aliases* são prejudiciais a uma boa legibilidade

Variáveis

- Tipo: determina a faixa de valores que a variável pode conter, e o conjunto de operações definidas para os valores deste tipo.
- Valor: é o conteúdo da célula de memória associada à variável.
  - ex. Considere a seguinte atribuição:

  x := x ;

  x – *Left-value* – endereço da variável

  x – *Right-value* – valor da variável (ou expressão que evolui para um valor)

Vinculação

- Uma vinculação (*binding*) é uma associação entre uma operação e um símbolo, ou entre um atributo e uma entidade.
  - ex. associação entre operação de adição e o símbolo '+'
  - ex. int total;
    - Associação dos atributos (nome, endereço, tipo, valor, etc.) à variável 'total'.
- O momento que é realizada a vinculação é chamado tempo de vinculação (*binding time*).
  - Exemplo:
    - O símbolo * normalmente é vinculado à operação de multiplicação no tempo de projecto da linguagem.

Tempos de Vinculação

- A vinculação pode acontecer nos seguintes tempos:
  - Projecto da linguagem (*Language design time*);
  - Implementação da linguagem (implementação do interpretador/compilador/debuger/etc.) (*Language implementation time*);
  - Durante a compilação (*Compile time*);
  - Durante a ligação do programa às bibliotecas (*Link*);
  - No momento carregamento do programa para execução (*Load time*);
  - No tempo de execução (*Run-time*).

Tempos de Vinculação

Exemplo de tempos de vinculação:

int count = 1;

- Vinculação no tempo de projecto da linguagem:
  - Conjunto de tipos possíveis para 'count';
  - Conjunto de valores possíveis para tipo 'int';
  - Conjunto de significados possíveis para símbolo '='.
- Vinculação no tempo de implementação da linguagem:
  - Representação interna em memória do número '1'.
- Vinculação no tempo de compilação
  - Tipo de 'count';
  - Significado do operador de atribuição '='.
- Vinculação no tempo de execução
  - Associação de 'count' a um endereço de memória;
  - Valor de 'count' - operação de atribuição do número à variável.

Formas de Vinculação de Tipos

A vinculação do tipo de dados de uma variável pode ser especificada de forma estática ou dinâmica.

- Vinculação estática: quando a vinculação ocorrer antes do tempo de execução e permanecer inalterada durante todo o tempo de execução do programa.
- Vinculação dinâmica: quando a vinculação ocorre durante a execução ou é alterada durante a execução do programa.

Vinculação Estática de Tipos

- Antes que uma variável possa ser referenciada, esta deve ser vinculada a um tipo de dados, podendo ser efectuada através de:
  - Declaração explícita: é especificado os nomes da variáveis e os seus tipos.
  - Declaração Implícita: não é especificado o tipo da variável, ele é atribuído através de convenções pré--estabelecidas pela L.P.
    - Exemplo: se o nome iniciar com i,j,k,l,m ou n será implicitamente declarado como integer, caso contrário será do tipo real.

Vinculação Dinâmica de Tipos

- O tipo não é especificado por instrução de declaração, mas sim quando é atribuído um valor para a variável.
  - Vantagem:
    - Flexibilidade de programação, permitindo a criação de subprogramas genéricos (independente do tipo de dados);
  - Desvantagens:
    - A capacidade do compilador detectar erros é diminuída;
    - Custo elevado para implementação (verificação dinâmica de tipo e interpretação do tipo);
  - Ex. JavaScript:

```
list = [12.55, 14.99, 7, 10];
list = 11.65;
```

Vinculação de Memória

- Reserva (Alocação) é o nome dado a vinculação da variável a uma ou mais células de memória disponível (da *pool de memória)*.
- Libertação (Desalocação) é o processo de desvincular e devolver a célula de memória ao *pool* de memória disponível.
- Tempo de vida de uma variável é o tempo durante o qual uma variável está vinculada a uma localização de memória específica.

Vinculação de Memória

- As vinculações de memória a variáveis, podem ser divididas em quatro categorias de acordo com o seu tempo de vida:
  - Variáveis Estáticas
  - Variáveis Pilhas-Dinâmicas
  - Variáveis Heap-Dinâmicas Explícitas
  - Variáveis Heap-Dinâmicas Implícitas
- Heap: é um conjunto de células de memória, desorganizada devido a imprevisibilidade da sua utilização.

Vinculação de Memória

- Variáveis Estáticas: são vinculadas a células de memória antes do início de execução do programa, e permanecem associadas às mesmas células até o programa terminar.
  - Vantagens:
    - Não existe custo de reservar e libertar memória;
    - Definição de variáveis sensíveis à história.
    - Endereçamento directo de memória (+ rápido).
  - Desvantagens:
    - Reduzida flexibilidade na gestão das variáveis;
    - A existência unicamente de variáveis estáticas não permite recursividade.
  - ex.: C, C++ e Java (usando o especificador static).

Vinculação de Memória

- Variáveis Pilhas-Dinâmicas: a associação é efectuada em tempo de execução, na instrução de declaração, e permanece inalterável até o fim do programa.
  - Vantagens:
    - Permite recursividade.
  - Desvantagens:
    - Custo das operações de reserva e libertação implícita de memória;
  - ex.: C, C++ (variáveis normais):

```
int sum, total;
```

Vinculação de Memória

- Variáveis Heap-Dinâmicas Explícitas: as variáveis são reservadas e libertadas de memória em tempo de execução por declarações explícitas do programador.
  - Vantagens:
    - Gestão dinâmica da memória.
  - Desvantagens:
    - Custo das operações de reserva e libertação de memória;
    - A utilização de apontadores não é segura.
  - Em JAVA todos os objectos são variáveis heap-dinâmicas explicitas.

Vinculação de Memória

- Ex. C++ (Vinculação efectuada através de apontadores):

```
// declaração de um apontador para caracteres
char * buffer;
// ...
//reservar espaço em memória para 500 caracteres
buffer = new char[500];
// ...
// libertar a memória reservada
delete [ ] buffer;
```

Vinculação de Memória

- Variáveis Heap-Dinâmicas Implícitas: são reservadas no momento em que lhe são atribuídas valores e libertadas por meio de instruções explícitas. Por vezes são nomes que se adaptam a diferentes tipos de variáveis.
  - Vantagens:
    - Elevado grau de flexibilidade do tipo de variável, permitindo definir código muito genérico.
  - Desvantagens:
    - Ineficiente porque todos os atributos da variável são dinâmicos;
    - Dificuldade de detecção de erros por parte do compilador.

Verificação de Tipos

- Verificação de tipos: é a actividade que assegura que os operandos de um operador são de tipos compatíveis.
- Um tipo compatível é o mesmo do operador ou com permissão da linguagem, um tipo que pode ser convertido no tipo do operador.
- A operação de conversão automática de um tipo noutro, designa-se por coerção.
- Um erro de tipo, é a aplicação de um operador a um tipo não compatível.

Verificação de Tipos

- Se as vinculações de variáveis de uma linguagem forem estáticas, a verificação de tipos poderá ser efectuada em tempo de compilação.
- A vinculação dinâmica de tipos requer a verificação de tipo em tempo de execução.

Tipificação Forte

- Linguagem fortemente tipificada: é uma linguagem na qual é possível detectar todos os erros de tipos durante o processo de compilação.
- Vantagens:
  - Detecção da utilização indevida de variáveis que resultam em erros de tipo.
- Linguagens fortemente tipificadas:
  - ADA, ML e Java

Compatibilidade de Tipos

- Quando duas variáveis são de tipos compatíveis, qualquer uma delas pode ter seu valor atribuído à outra.
- Existem dois métodos diferentes de compatibilidade de tipos:
  - Compatibilidade de Nome
  - Compatibilidade de Estrutura

Compatibilidade de Nome

- Compatibilidade de Nome: Variáveis possuem tipos compatíveis se estiverem na mesma declaração ou em declarações que usem o mesmo nome de tipo.
- Vantagens:
  - mais fácil de implementar.
- Desvantagens:
  - é altamente restritiva:
    - Ex. em Pascal: Sub-faixas do tipo inteiro não são compatíveis com o tipo inteiro.

Compatibilidade de Nome

- Ex. em C++:

```
// todas as variáveis são de tipos compatíveis:
int x, y, z;  //  (mesma declaração)
int w; // (mesmo tipo de nome das variáveis x, y, z)
```

- Ex. em Pascal:

```
// variáveis de tipos não compatíveis:
type classification= 0...20;
var  first : integer;
     last : classification;
first := last;  -> erro semântico
```

Compatibilidade de Estrutura

- Compatibilidade de Estrutura: variáveis têm tipos compatíveis se os seus tipos tiverem estruturas idênticas.
- Vantagens:
  - mais flexibilidade.
- Desvantagens:
  - mais difícil de implementar.
- Ex. em C (compatibilidade de estrutura):

```
char letraA = 'A';
int ordemA = 65;
letraA = ordemA; // ou ordemA = letraA; é válido
```

Compatibilidade de Estrutura

- Em C++ e Java, a compatibilidade de objectos está relacionado com a hierarquia de herança (ver capítulo 11).
- A linguagem C utiliza equivalência de tipos por estrutura, enquanto o C++ utiliza equivalência de nomes.
- Ex. em Pascal "Padrão":

```
// Não existe compatibilidade por estrutura:
type type1 = array[1..50];
     type2 = array[1..50];
     type3 = real;
     type4 = real;
type1 e type2 -> tipos incompatíveis (não compatib. por estrut.)
type3 e type4 -> tipos compatíveis (compatibilidade de nome)
```

Inferência de Tipo

- Inferência de tipo: capacidade que a linguagem tem para determinar os tipos das variáveis tendo em consideração o contexto em que encontram-se. Desta forma não é necessário especificar todos os tipos das variáveis.

- **Exemplo em ML:**

```
fun area(l:int, w:int): int = l*w
```

Funções equivalentes (por inferência de tipo):

```
fun area(l, w): int = l*w
fun area(l:int, w) = l*w
```

Situação de erro (impossível inferir):

```
fun area(l, w) = l*w
```

Escopo

- O escopo (*scope*) de uma variável, representa a área do programa onde esta é visível.
- Ex. em C e C++ (escopo definido por bloco):

```
void foo (int x) {  // início do escopo de x
   int y;        // início do escopo de y
   x = y = 0;
  }              // fim do escopo of x and y
```

- As regras de escopo de uma L.P. determinam a forma como as referências e nomes estão associadas a variáveis.

Escopo

- Algumas regras de escopo:
  1. Procurar variáveis localmente
  2. Procurar em ordem crescente do escopo até encontrar uma declaração para o nome da variável.
- 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" a definição exterior.
  - C++, Pascal, ADA, etc. permitem o acesso a estas variáveis escondidas em escopos exteriores.

Escopo

- O escopo de uma variável pode ser:
  - Estático;
  - Dinâmico.
- Escopo Estático
  - Baseado no texto estático do programa;
  - Para associar uma referência a uma variável é necessário procurar a sua declaração;
  - O escopo estático pode ser determinado estaticamente antes da execução (*Load Time*).

Escopo Estático

- Considere o seguinte programa em linguagem C.

- Qual o resultado do programa '0' ou '1'?

```
int x = 0;
void foo ()
{
  x++;
}
void bar ()
{
  int x = 0;
  foo ();
}
int main()
{
    bar ();
    printf("%d",x);
}
```

Escopo Estático

- Blocos – método para criar novos escopos dentre de um programa.

Considere o seguinte programa em C++.

- Qual o resultado da aplicação?

```
int i = 0; // variável global ao modulo
int main()
{   int i = 5;
    { int j = 10;
        { int k = 15;
          cout << "i = " << i << endl;
          cout << "j = " << j << endl;
          cout << "k = " << k << endl;
        } // k termina aqui
    } // j termina aqui
    cout << "i = " << i << endl;
    cout << "j = " << j << endl;
    cout << "k = " << k << endl;
    return 0;
}
```

Escopo Estático

Considere o seguinte programa em Pascal Padrão (escopo estático).

- Qual o resultado do programa '1' ou '2' ou '3'?

(1) – Variável 'x' global
(2) – Variável 'x' de sub2
(3) – Variável 'x' global

```
program exemplo;
var x: integer;
    procedure sub1;
    begin
            x := 1;  (1)
    end;
    procedure sub2;
    var x: integer;
    begin
            x := 2; (2)
    end;
begin
    x := 3; (3)
    sub1();
    sub2();
    writeln(x);
end.
```

Escopo Estático - Avaliação

- As variáveis globais são visíveis a todos os procedimentos/funções/métodos;
  - Dificulta futuras modificações ao programa;
- Solução para o escopo estático: encapsulamento;
- A maioria das linguagens imperativas utilizam escopo estático: Pascal, C, C++, Java, Ada, etc..

Escopo Dinâmico

- O escopo dinâmico baseia-se na sequência de chamadas de subprogramas e pode ser determinado somente em tempo de execução.
- Para associar uma referência a uma variável é necessário procurar a sequencia de invocação até encontrar a sua declaração.

Escopo Dinâmico

Considere o seguinte programa em Pascal (escopo dinâmico).

- Qual o resultado do programa '1' ou '2' ou '3'?

(1) – Variável 'x' global
(2) – Variável 'x' de sub2

```
program exemplo;
var x: integer;
    procedure sub1;
    begin
        x := x + 1;
    end;
    procedure sub2;
    var x: integer;
    begin
        sub1();
    end;
begin
    x := 1; (1)
    sub1(); (1)
    sub2(); (2)
    writeln(x);
end.
```

Escopo Dinâmico - Avaliação

- Programas menos confiáveis que os de escopos estáticos;
- Problemas com a legibilidade do programa;
- Execução mais lenta;
- Menos utilizado que o escopo estático;
- Linguagens que utilizam escopo dinâmico: Smalltalk, APL, SNOBOL.

Escopo e Tempo de Vida

- Escopo e tempo de vida não estão directamente relacionados.
- No exemplo:
  - Escopo da variável 'x' não se estende para a função foo();
  - Tempo de vida da variável 'x', estende-se para a função foo().
  - Escopo de 'y' válido na função foo();
  - Tempo de vida de 'y' estende-se por todo o programa.

Exemplo em linguagem C:

```
void foo ()
{ static int y;
    ...
}
int main()
{
    int x;
    foo ();
}
```

Ambiente de Referenciamento

- O ambiente de referenciamento de uma instrução é o conjunto de todos os nomes visíveis na instrução.
- No escopo estático, o ambiente de referenciamento é constituído de todas as variáveis declaradas em seu escopo local, conjuntamente com o conjunto de todas as variáveis de seus escopos ascendentes visíveis.

Ambiente de Referenciamento

- No exemplo ao lado, os ambientes de referenciamento em cada ponto do programa:

(1): 'x' e 'y' de sub1 e 'a' e 'b' de exemplo.

(2): 'x' de sub3 e 'a' e 'b' de exemplo.

(3): 'x' de sub2 e 'a' e 'b' de exemplo.

(4): 'a' e 'b' de exemplo

```
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;
```

Constante

- 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 designado 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, excepto durante a execução das instruções de atribuição.