

2.5 - Organização de Tabelas de Símbolos

- ♦ A geração de código e a correção semântica de um programa dependem do conhecimento dos atributos dos identificadores usados no programa-fonte.
- ♦ **Exemplo:**
 - a) **análise semântica:** o uso de um nome é consistente com sua declaração?
 - b) **geração de código:** quantas posições de memória devem ser alocadas para um certo nome?
- ♦ Esses atributos são encontrados nas declarações (como em Pascal) ou são implícitos (como no FORTRAN) e são armazenados em uma estrutura de dados denominada tabela de símbolos. As tabelas de símbolos podem ser vistas como:

	identificador	atributos
1		
2		
...		
n		

- ♦ As principais questões no projeto de tabelas de símbolos são:
 - a) o formato das entradas (o que deve ser colocado na tabela?)
 - b) o método de acesso às informações da tabela
 - c) o local onde a tabela deve ser armazenada (memória principal ou secundária?)

Organização de Tabelas de Símbolos

Formato das entradas

- ♦ Que tipo de informação deve ser colocada em atributos?

Variável

- tipo (inteiro, real, lógico, ...)
- forma (variável simples, array, ...)
- precisão (simples, dupla)
- endereço
- se array, número de dimensões
- parâmetro formal?
- qual tipo de correspondência, se for parâmetro formal?
- já foi inicializada?
- já foi referenciada?
- ...

Constantes

- endereço
- tipo
- valor
- ...

Rótulo

- endereço
- já foi referenciado?
- ...

Procedimento

- endereço
- externo?
- é recursivo?
- é função? qual tipo de função?
- quais são os parâmetros formais?
- ...

Palavras-reservadas

- contexto
- ...

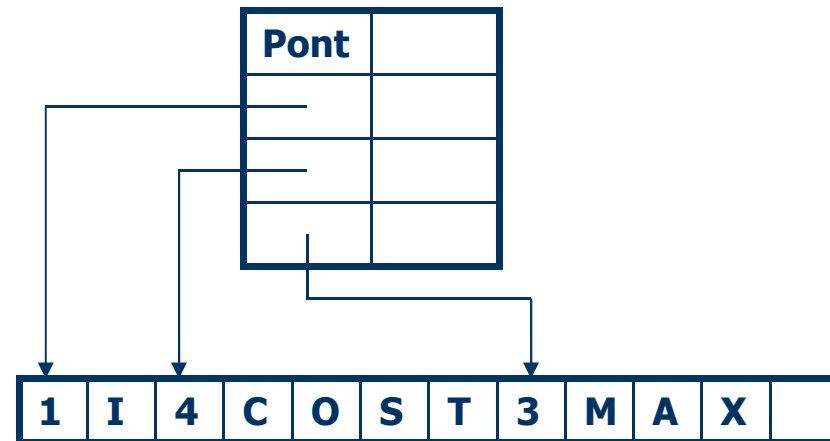
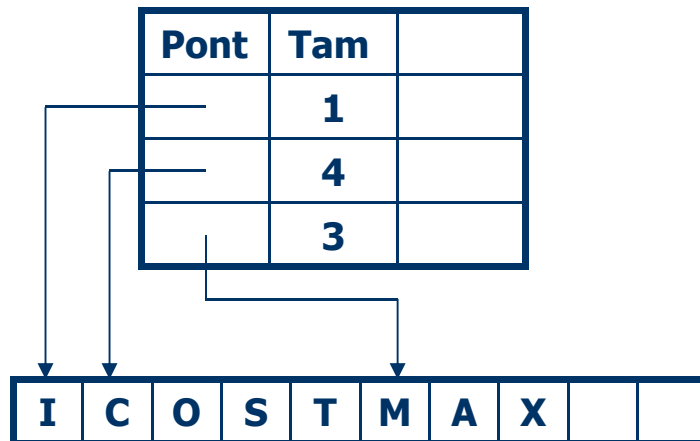
Organização de Tabelas de Símbolos

Organização da tabela

- ♦ O identificador pode ser armazenado no próprio registro (caso a linguagem imponha limites modestos no comprimento dos identificadores):

I	
COST	
MAX	

- ♦ ou, no caso de não existir um limite sobre o comprimento dos identificadores (ou o limite ser grande), ser armazenado por um esquema indireto:



Organização de Tabelas de Símbolos

Implementação da tabela

- ♦ a implementação da tabela de símbolos pode ser feita segundo três alternativas principais:
 - listas lineares (fácil de implementar; acesso lento)
 - tabelas "hash" (difícil de implementar; acesso rápido)
 - árvores (solução de compromisso)

Listas lineares

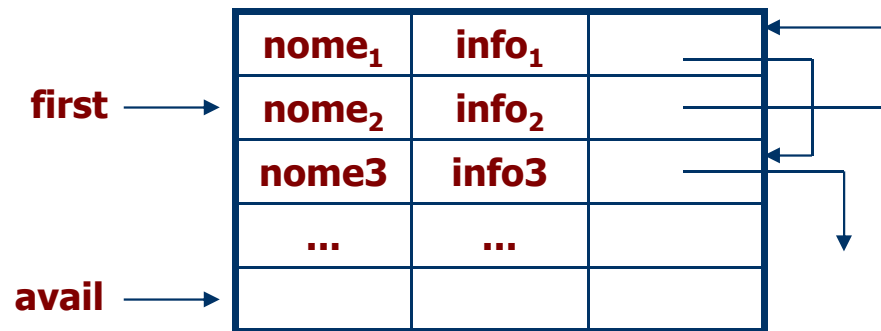
- ♦ Novos nomes são adicionados à lista a medida que vão sendo encontrados.

nome₁	info₁
nome₂	info₂
...	...
nome_n	info_n
avail →	

- ♦ vantagem: simples de implementar e ocupa o mínimo possível de espaço.
- ♦ desvantagem: acesso proporcional a n.

- ♦ Listas de auto-organização: Ao custo de um pequeno espaço adicional, podemos diminuir (em geral) o tempo gasto na procura, adicionando um campo LINK.

Organização de Tabelas de Símbolos



- ♦ Quando um nome é referenciado ou quando seu registro está sendo criado, move-se o registro desse nome para o início da lista (apontado por first). Isso faz com que os nomes referenciados mais frequentemente ocupem o início da lista, onde são encontrados rapidamente. Além disso, se um pequeno conjunto de nomes é muito usado numa pequena seção do programa, tais nomes irão tender para o início da lista enquanto essa seção estiver sendo processada pelo compilador.
- ♦ Motivação para esse tipo de estrutura: os programas não referenciam nomes aleatoriamente.

Árvores binárias

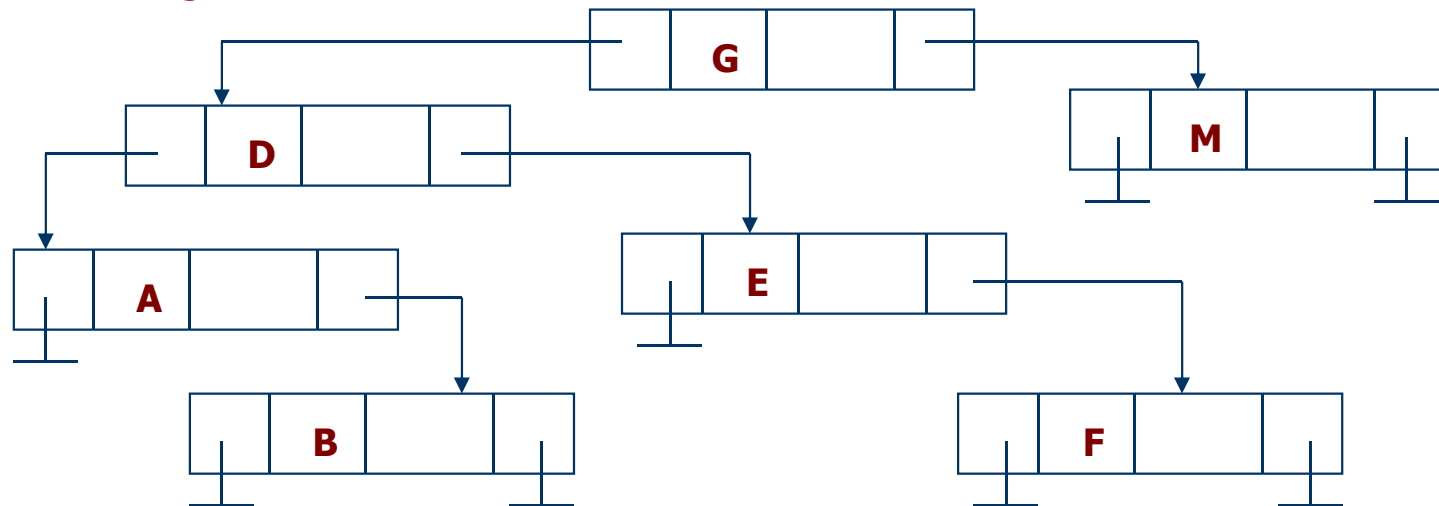
- ♦ Cada nó da árvore representa uma entrada na tabela de símbolos e tem a forma:

LLINK	NOME	INFO	RLINK
-------	------	------	-------

Organização de Tabelas de Símbolos

onde:

- a) todos os nomes $NOME_j$ acessíveis de $NOME_i$ a partir de $LLINK_i$ são tais que $NOME_j < NOME_i$ (em ordem alfabética);
 - b) todos os nomes $NOME_j$ acessíveis de $NOME_i$ a partir de $RLink_i$ (e então seguindo qualquer sequência de links) são tais que $NOME_i < NOME_j$
- ♦ Exemplo: Se os nomes são encontrados na ordem G, D, M, E, A, B e F teremos a seguinte estrutura:



Rotina de procura:

P aponta para a raiz (inicialmente)

N é o nome procurado

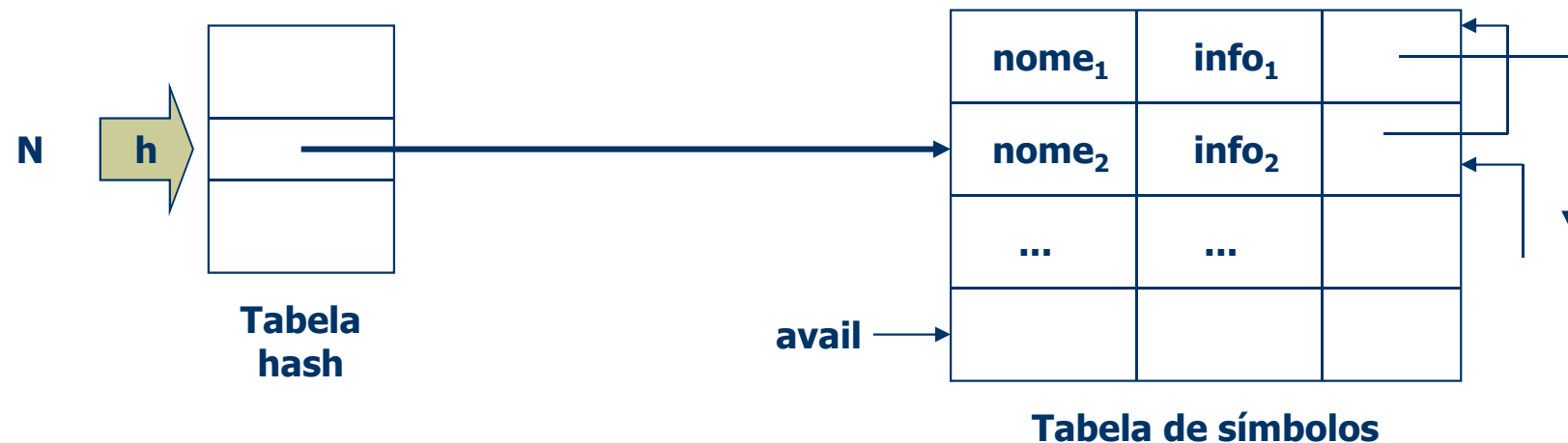
(tempo de procura = $\mathcal{O}(n \times \log_2 n)$!)

```
while P ≠ nil do
  if N = Nome[P] then ...
  else if N < Nome[P] then P := LLink[P]
  else P := RLink[P];
```

Organização de Tabelas de Símbolos

Tabelas hash

- Existem muitas variações dessa técnica de procura. Vamos apresentar uma forma bem simples:



- A tabela hash consiste de k posições numeradas $0, 1, \dots, k-1$. Essas posições contêm ponteiros para o início de k listas separadas (algumas podem ser vazias) na tabela de símbolos. Cada registro da tabela de símbolos pertence a exatamente uma dessas listas. Para determinar se o nome N está na tabela de símbolos, aplicamos a N uma função h (função hash) tal que $h(N)$ é um inteiro entre 0 e $k-1$. O registro para N na tabela de símbolos está na lista apontada por $h(N)$.
- para incluir um nome N na tabela de símbolos, criamos um registro para ele na primeira posição disponível da tabela de símbolos e ligamos esse registro ao início da lista de número $h(N)$.

Organização de Tabelas de Símbolos

- ♦ A função hash h deve ser tal que:
 - a) distribui os nomes uniformemente nas k listas;
 - b) é fácil de ser calculada para cadeias de caracteres.
- ♦ Se a propriedade (a) for obtida e se n é o número de nomes na tabela de símbolos, o comprimento médio de cada lista será n/k , o que diminuirá o tempo gasto em procuras por um fator de k .

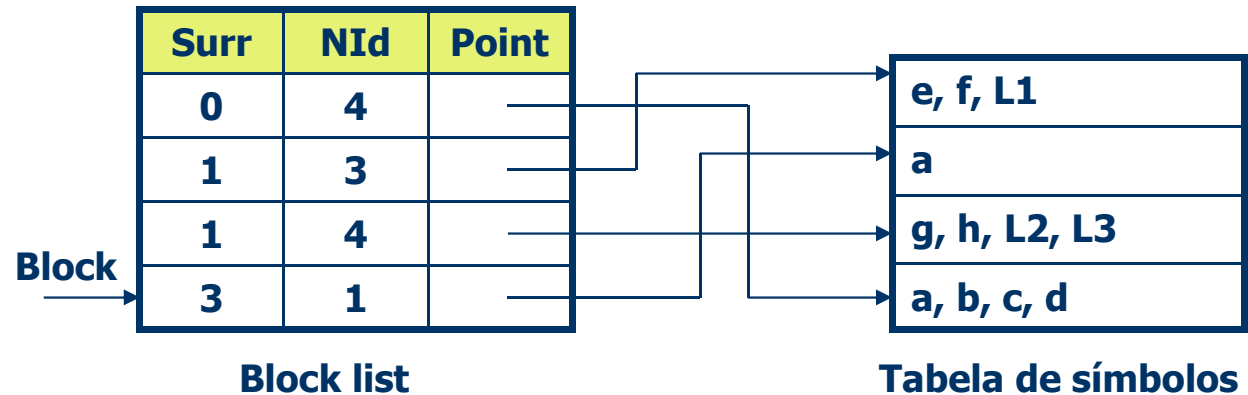
Tabelas de símbolos estruturadas em blocos

- ♦ A estrutura de bloco torna os nomes locais ao bloco ou procedimento no qual esses nomes são declarados. Quando o compilador encontra o uso de um identificador é necessário descobrir qual o bloco envolvente mais próximo que contém uma declaração desse identificador. Logo, é preciso ter uma estrutura de dados que torne ativos os nomes disponíveis para referência, à medida que o programa-fonte vai sendo lido. Por outro lado, nomes que não são mais ativos não podem ser simplesmente descartados porque informações sobre eles poderão ser necessárias em passos subsequentes. A idéia é então dividir a tabela de símbolos em duas partes: a área ativa e a área inativa.
- ♦ O acesso correto aos identificadores pode ser implementado juntando todos os identificadores do mesmo bloco em registros da tabela de símbolos e construir uma outra tabela (denominada "block list") contendo ponteiros para esses registros.

Organização de Tabelas de Símbolos

Exemplo:

```
begin
  real a,b,c,d;
  ...
  begin
    real e,f;
    ...
    L1:
    ...
  end;
  begin
    real g,h;
    ...
    L2:
    begin
      real a;
      ...
    end;
    L3:
    ...
  end;
end;
```



onde: Block - bloco atual
 Surr - bloco envolvente
 NId - número de identificadores

Algoritmo de procura:

```
B := Block;
while B ≠ 0 do
begin
  n := BlockList[B].NId;
  p := BlockList[B].Point;
  for i := p until p+n-1 do
    compare(Simb, Tab[i]);
  if não_achou then
    B := BlockList[B].Surr;
end;
```

Organização de Tabelas de Símbolos

- ♦ A montagem da tabela de símbolos é feita quando o bloco termina.

```
begin
  real a,b,c,d;
  ...
  begin
    real e,f;
    ...
    L1:
    ...
  end;
  begin
    real g,h;
    ...
    L2:
    begin
      real a;
      ...
    end;
    L3:
    ...
  end;
end;
```

Abertura de bloco:

```
LastB := LastB + 1;
BlockList[LastB] := (CurrB,0,Topel);
CurrB := LastB;
```

Fechamento de bloco:

```
BlockList[CurrB].Point := Lastel + 1;
for i := 1 to BlockList[CurrB].NId do
begin
  Lastel := Lastel + 1;
  S[Lastel] := S[Topel];
  Topel := Topel + 1;
end;
CurrB := BlockList[CurrB].Surr;
```

