Trabalho de Linguagens de Programação: Resumo do Capítulo 10

Implementando Subprogramas

Patrick Anderson Matias de Araújo e Gabriel Araújo

Universidade Federal do Tocantins

Tópicos do Capítulo 10

1. A semântica geral de chamadas e retornos
2. Implementando subprogramas “simples”
3. Implementando subprogramas com variáveis locais dinâmicas da pilha
4. Subprogramas aninhados
5. Blocos
6. Implementando escopo dinâmico

A semântica geral de chamadas e retornos

A semântica geral de chamadas e retornos

- As operações de chamada e retorno de subprogramas são juntas chamadas de *ligação de subprogramas*
- Semântica geral das chamadas a subprogramas
  - Métodos de passagem de parâmetros
  - Alocação dinâmica da pilha de variáveis locais
  - Salvar o estado de execução da unidade de programa chamadora
  - Transferência de controle e garantia de retorno
  - Se subprogram as aninhados são suportados, acesso a variáveis não locais deve ser garantido
- Semântica geral de retornos de subprograma:
  - Parâmetros do modo de saída ou do modo de entrada devem ter seus valores retornados
  - Liberação de locais dinâmicas da pilha
  - Retomar o estado de execução
  - Retornar o controle ao chamador

Implementando subprogramas “simples”

Implementando subprogramas ”simples”: semântica de chamada

- Semântica de chamada:
  - Salvar o estado da execução da unidade de programa atual
  - Calcular e passar os parâmetros
  - Passar o endereço de retorno para o subprograma chamado
  - Transferir o controle para o subprograma chamado

Implementando subprogramas ”simples”: semântica de retorno

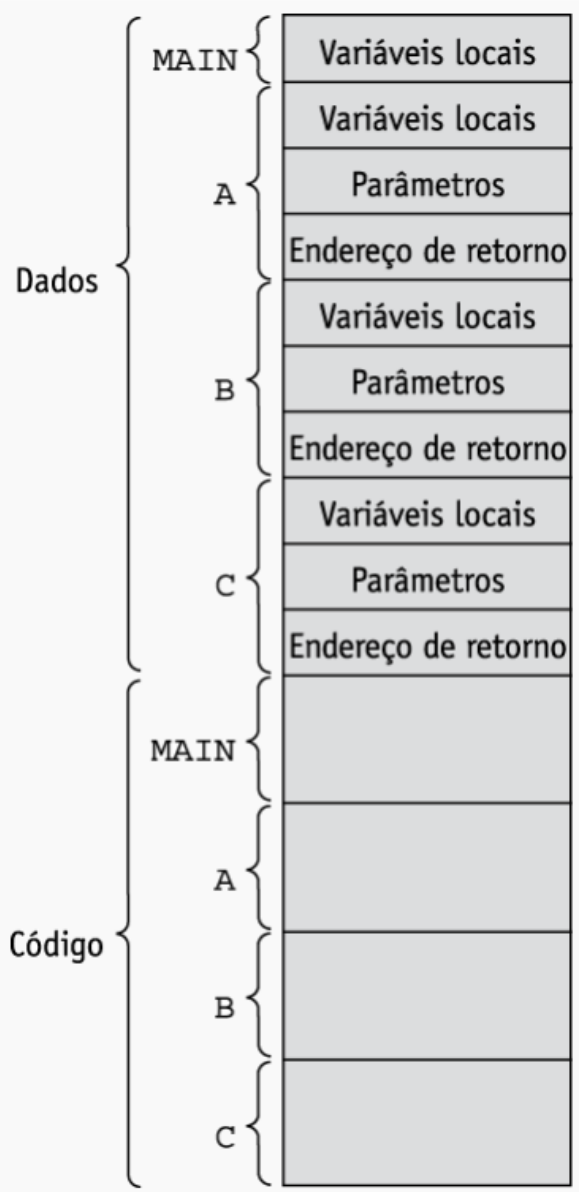
- Semântica de retorno:
  - Se existirem parâmetros com passagem por valor-resultado ou parâmetros no modo de saída, os valores atuais desses parâmetros são movidos para os parâmetros reais correspondentes
  - Se o subprograma é uma função, o valor funcional é movido para um local acessível ao chamador
- O estado da execução do chamador é restaurado
- O controle é transferido de volta para o chamador
- O controle é transferido de volta para o chamador
- Armazenamento requerido:
- Informações de estado, parâmetros, endereço de retorno, valor de retorno para funções

Implementando subprogramas "simples": partes

- Duas partes separadas: o código real e a parte não código (variáveis locais e dados listados, que podem mudar)
- Duas partes separadas: o código real e a parte não código (variáveis locais e dados listados, que podem mudar) formato, ou layout, da parte que não é código de um subprograma é chamado de *registro de ativação*
- Uma *instância de registro de ativação* é um exemplo concreto de um registro de ativação (uma coleção de dados na forma de um registro de ativação)

UM REGISTRO DE ATIVAÇÃO PARA SUBPROGRAMAS "SIMPLES"

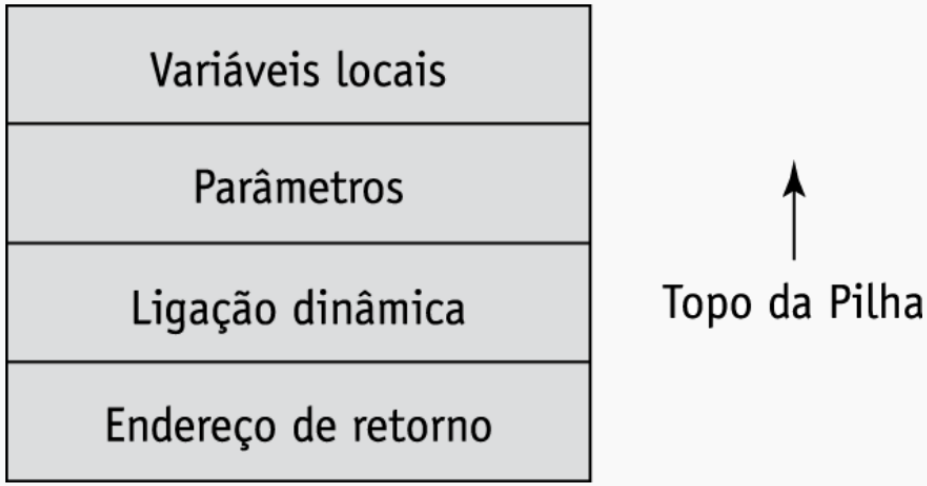
Variáveis locais
Parâmetros
Endereço de retorno



Implementando subprogramas com variáveis locais dinâmicas da pilha

Implementando subprogramas com variáveis locais dinâmicas da pilha

- Registros de ativação mais complexos
  - O compilador deve gerar código que faça alocação e liberação implícitas de variáveis locais
  - Recursão deve ser suportada (adiciona a possibilidade de múltiplas ativações simultâneas de um subprograma)

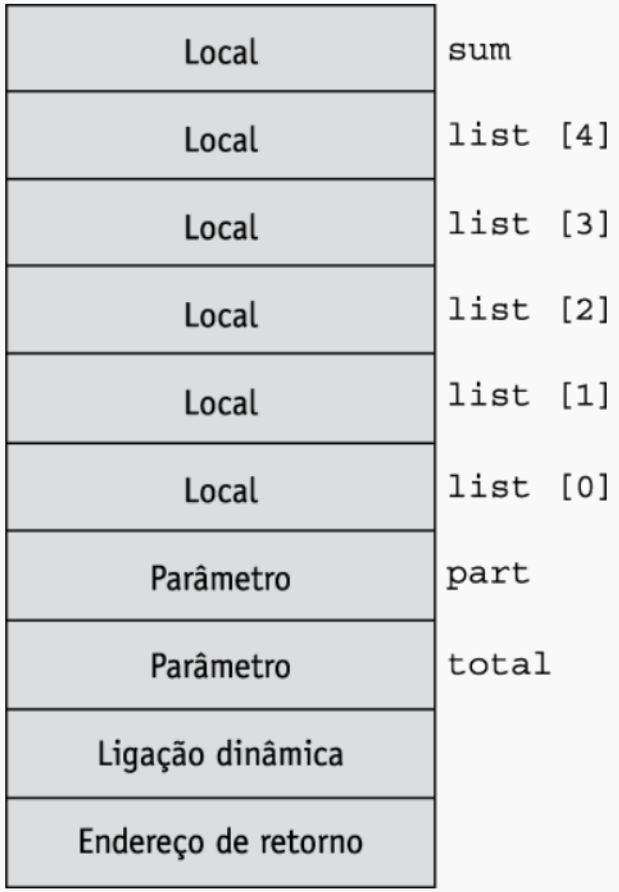


Implementando subprogramas com variáveis locais dinâmicas da pilha: registro de ativação

- O formato de um registro de ativação é estático, mas o tamanho pode ser dinâmico
- A *ligação dinâmica* é um ponteiro para a base da instância de registro de ativação do chamador
- Uma instância de registro de ativação é criada dinamicamente quando um subprograma é chamado
- Instâncias de registro de ativação residem na pilha de tempo de execução
- O PE (*Environment Pointer*, em inglês) é mantido pelo sistema de tempo de execução. Ele sempre aponta para a base da instância do registro de ativação da unidade de programa que está sendo executada

Um exemplo: função C

```
void sub(float total, int
part){
        int list[5];
        float sum;
        ...
}
```

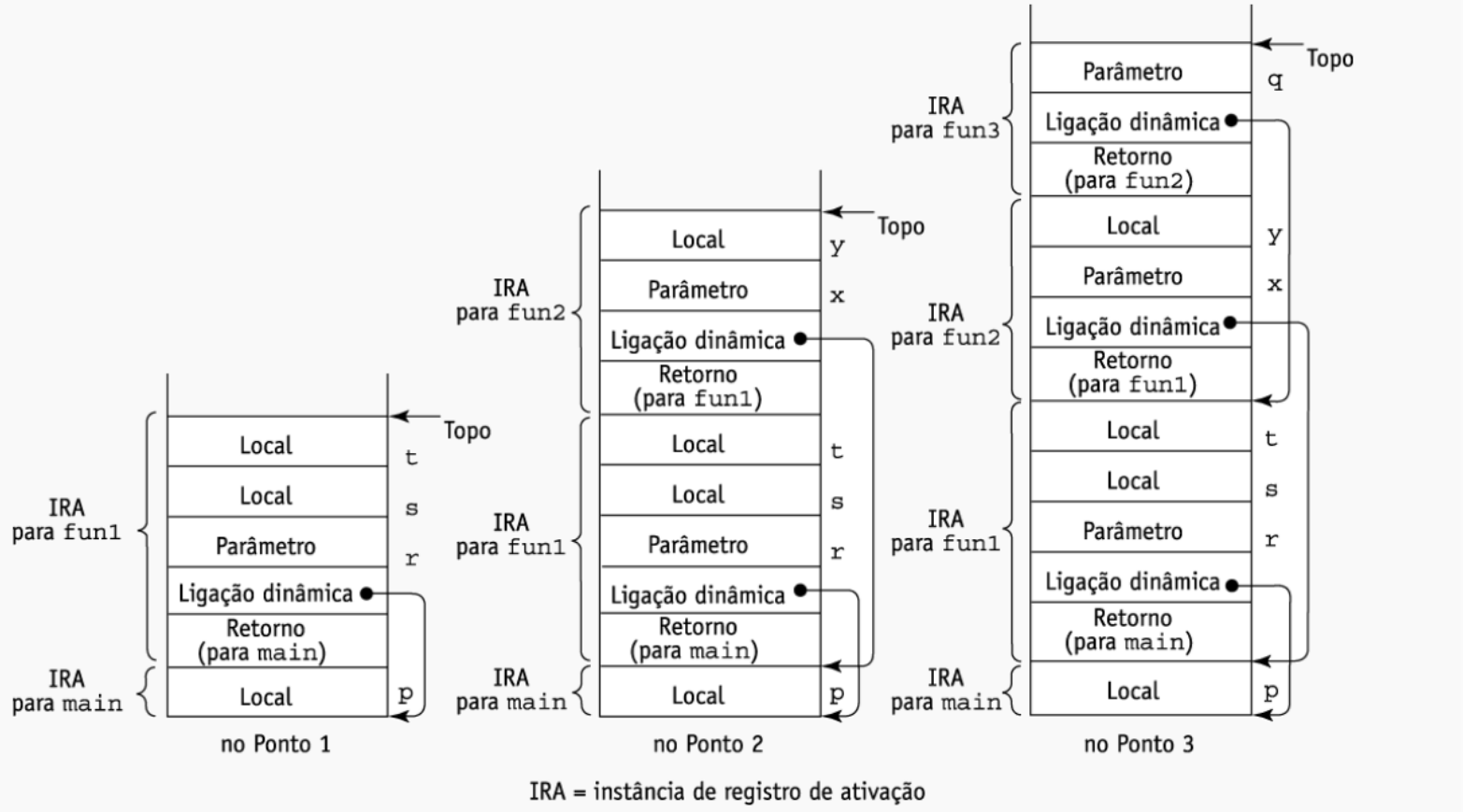


Um exemplo sem recursão

```
void A(int x){
        int y;
        ...
        C(y);
        ...
}
void B(float r){
        int s, t;
        ...
        A(s);
        ...
}
void C(int q){
        ...
}
```

```
void main(){
        float p;
        ...
        B(p);
        ...
}
/*
        main chama B
        B chama A
        A chama C
*/
```

Cadeia dinâmica e deslocamento local



Cadeia dinâmica e deslocamento local

- A coleção de ligações dinâmicas presentes na pilha em um dado momento é chamada de *cadeia dinâmica* ou *cadeia de chamadas*
- Referências a variáveis locais podem ser representadas no código como deslocamentos a partir do início do registro de ativação do escopo local, cujo endereço é armazenado no PE. Tal deslocamento é chamado de *deslocamento local (local_offset)*
- O deslocamento local de uma variável em um registro de ativação pode ser determinado em tempo de compilação

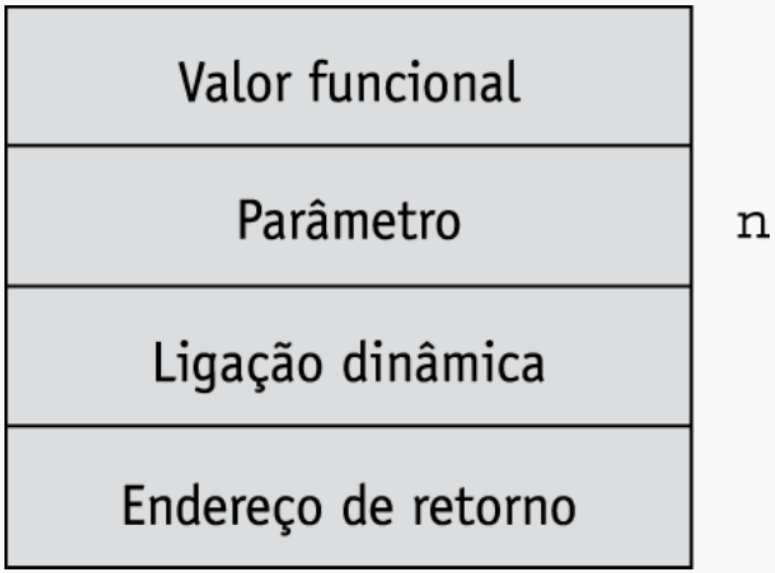
Um exemplo com recursão

- Considere o seguinte exemplo de programa em C, que usa recursão para calcular a função fatorial

```
int factorial(int n){
        <———————————— 1
        if (n <= 1)
                return 1;
        else
                return (n * factorial(n - 1));
       <———————————— 2
}
```

```
void main() {
        int value;
        value = factorial(3);
       <———————————— 3
}
```

O registro da ativação para factorial



Subprogramas aninhados

Subprogramas aninhados

- Algumas das linguagens de programação de escopo estático não baseadas em C (Fortran 95, Ada, Python, JavaScript e Lua) usam variáveis locais dinâmicas da pilha e permitem que os subprogramas sejam aninhados
- Todas as variáveis não estáticas que podem ser acessadas não localmente estão em instâncias de registro de ativação existentes e, logo, estão em algum lugar na pilha
- O processo de referência para uma variável não local:
  1. Encontrar a instância de registro de ativação na pilha na qual a variável foi alocada
  2. Usar o deslocamento local da variável (dentro da instância de registro de ativação) para acessá-la

Localizar uma referência não local

- Encontrar o deslocamento é fácil
- Encontrar a instância de registro de ativação correta
  - Regras de semântica estática garantem que todas as variáveis não locais que podem ser referenciadas foram alocadas em alguma instância de registro de ativação que está na pilha quando a referência é feita

Encadeamentos estáticos

- Um *encadeamento estático* é uma cadeia de ligações estáticas que conectam certas instâncias de registro de ativação na pilha
- A *ligação estática* aponta para o final da instância de registro de ativação de uma ativação do ancestral estático
- A cadeia estática de uma instância de registro de ativação conecta a todos os seus ancestrais estáticos
- *Profundidade estática* é um inteiro associado com um escopo estático que indica o quão profundamente ele está aninhado no escopo mais externo
- O deslocamento de encadeamento *(chain_offset)* ou profundidade de aninhamento *(nesting_depth)* de uma referência não local é a diferença entre a profundidade estática do procedimento que contém a referência a x e a profundidade estática do procedimento contendo a declaração de x
- A referência à variável pode ser representada por: (chain_offset, local_offset),

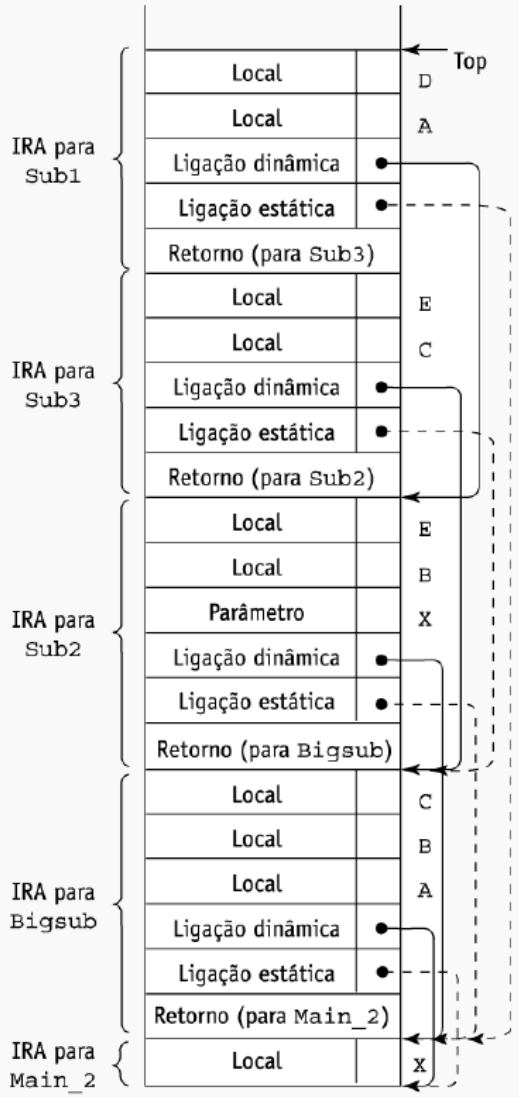
Exemplo de Programa em Ada

```
procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
    procedure Sub1 is
      A, D : Integer;
      begin -- of Sub1
      A := B + C;  <----------------------1
    end;  -- of Sub1
    procedure Sub2(X : Integer) is
      B, E : Integer;
      procedure Sub3 is
        C, E : Integer;
        begin -- of Sub3
        Sub1;
        E := B + A:   <--------------------2
        end; -- of Sub3
      begin -- of Sub2
      Sub3;
      A := D + E;  <----------------------3
      end; -- of Sub2 }
    begin -- of Bigsub
    Sub2(7);
    end; -- of Bigsub
  begin
  Bigsub;
end; of Main_2 }
```

- A sequencia de chamada a procedimentos é:

  Main_2 **chama** Bigsub
  Bigsub **chama** Sub2
  Sub2 **chama** Sub 3
  Sub3 **chama** Sub1

Conteúdo da pilha na posição 1 do programa



IRA = instância de registro de ativação

Manutenção de cadeias estáticas

- No momento da chamada,
  - A instância de registro de ativação deve ser encontrada
  - The dynamic link is just the old stack top pointer
  - The static link must point to the most recent ari of the static parent
    - Dois métodos

    1. Busca a cadeia dinâmica
    2. Trata chamadas a subprogramas e definições como referências variáveis e definições

Avaliação de cadeias estáticas

- Problemas:
  1. Uma referência não local é lenta se a profundidade de aninhamento é grande
  2. Código com tempo limitado é difícil:
     - Custos de referências não locais são difíceis de determinar
     - Mudanças de código podem mudar a profundidade de aninhamento

Mostradores (*displays*)

- Uma alternativa ao encadeamento estático que resolve os problemas com essa abordagem
- Ligações estáticas são armazenadas em uma única matriz chamada mostrador (display)
- O conteúdo do mostrador em um determinado momento é uma lista de endereços das instâncias de registro de ativação

Blocos

Blocos

- Blocos são escopos locais especificados pelo usuário para variáveis
- Um exemplo em C

```
{
int temp;
temp = list [upper];
list [upper] = list [lower];
list [lower] = temp
}
```

- O tempo de vida de *temp* no exemplo começa quando o controle entra no bloco
- A vantagem de usar tal variável local é que ela não pode interferir com outras variáveis com o mesmo nome que são declaradas em outros lugares do programa

Implementando blocos

- Dois métodos
  1. Blocos são tratados com o subprogramas sem parâmetros que são sempre chamados a partir do mesmo local do programa
     - Cada bloco tem um registro de ativação; uma instância é criada a cada vez que o bloco é executado
  2. Já que o máximo de armazenamento necessário para um bloco pode ser determinado, esse espaço pode ser alocado depois das variáveis locais no registro de ativação

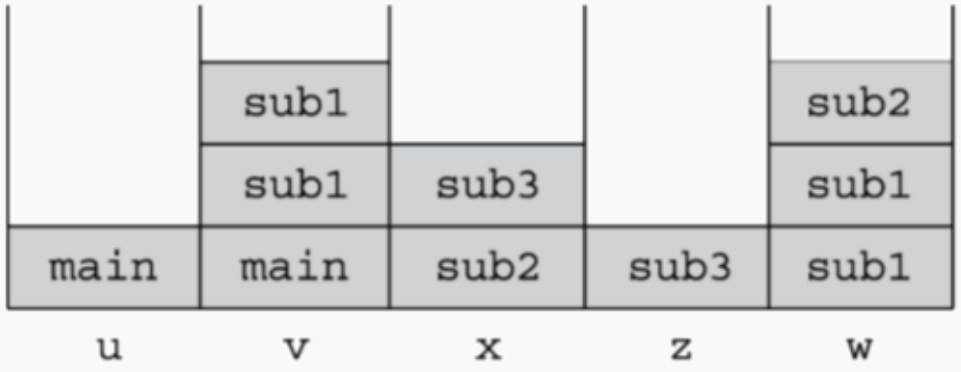
Implementando escopo dinâmico

Implementando escopo dinâmico

- *Acesso profundo*: as referências a variáveis não locais podem ser resolvidas com buscas por meio das instâncias de registros de ativação dos subprogramas ativos
  - O tamanho da cadeia não pode ser estaticamente determinado
  - Os registros de ativação devem armazenar os nomes das variáveis
- *Acesso raso*: coloca as variáveis locais em uma tabela central
  - Uma pilha separada para cada nome de variável
  - Tabela central com entrada para cada nome de variável

Usando acesso raso para implementar escopo dinâmico

```
void sub3() {
  int x, z;
  x = u + v;
  …
}
void sub2() {
  int w, x;
  …
}
void sub1() {
  int v, w;
  …
}
void main() {
  int v, u;
  …
}
```



(Os nomes nas células da pilha indicam as unidades de programa da declaração da variável.)

Resumo

- A semântica de ligação de subprogramas requer muitas ações por parte da implementação
- No caso de subprogramas ”simples”, essas ações são relativamente básicas
- Linguagens dinâmicas da pilha são mais complexas
- Subprogramas em linguagens com variáveis locais dinâmicas da pilha e subprogramas aninhados têm dois componentes
  - código real
  - registro de ativação
- *Acesso raso*: coloca as variáveis locais em uma tabela central
  - Uma pilha separada para cada nome de variável
  - Tabela central com entrada para cada nome de variável

Resumo (continuação)

- Instâncias de registro de ativação contêm os parâmetros formais e as variáveis locais, dentre outras coisas
- A ligação estática é usada para permitir referências para variáveis não locais em linguagens de escopo estático
- O acesso às variáveis não locais em uma linguagem de escopo estático pode ser implementado pelo uso de encadeamento dinâmico ou por meio de algum método de tabela variável central