unesp

Linguagens de Programação

Aula 11

Celso Olivete Júnior

olivete@fct.unesp.br

unesp

Na aula passada

- Uma definição de subprograma descreve as ações representadas pelo subprograma
- Subprogramas podem ser funções ou procedimentos
- Variáveis locais em subprogramas podem ser dinâmicas da pilha ou estáticas
- Três modelos fundamentais de passagem de parâmetros: modo de entrada, modo de saída e modo de entrada/saída
- Uma corrotina é um subprograma especial que tem múltiplas entradas

Na aula de hoje

Estruturas de controle no nível de unidades

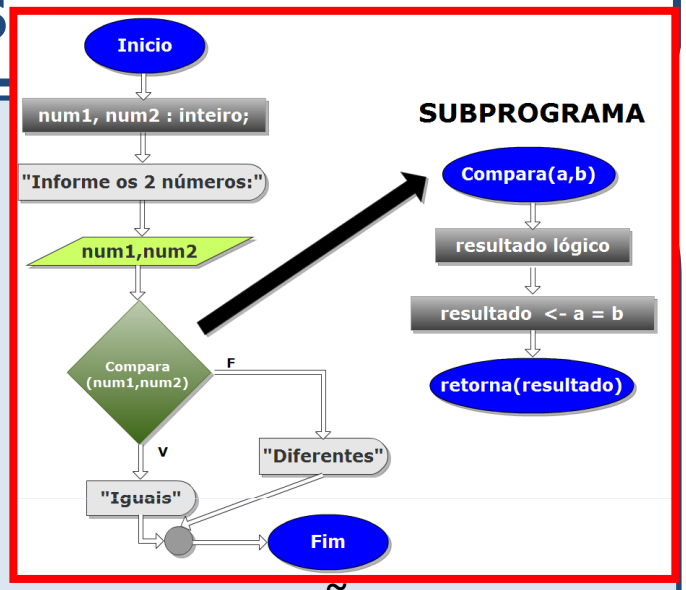
➔ **Implementando subprogramas**

unesp

Roteiro

- A semântica geral de chamadas e retornos
- Implementando subprogramas “simples”
- Implementando subprogramas com variáveis locais dinâmicas da pilha
- Subprogramas aninhados
- Blocos
- Implementando escopo dinâmico

A semântica geral de chamadas e retornos

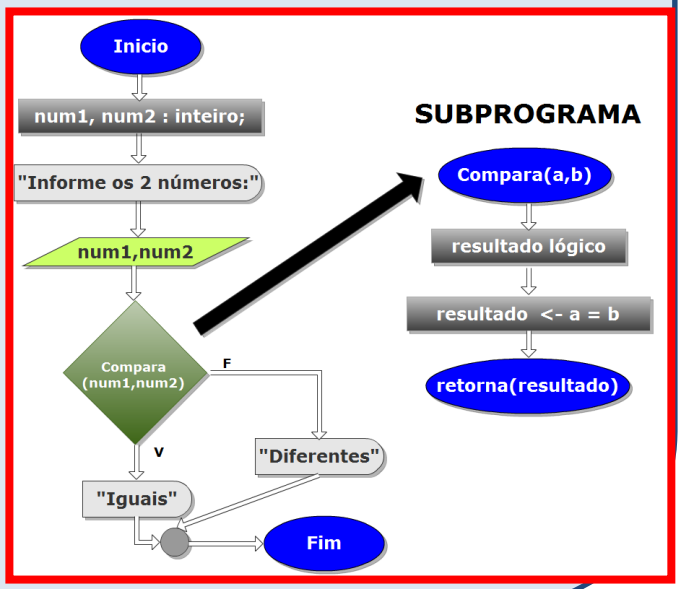


- As operações de chamada e retorno de subprogramas são chamadas de ligação ("*linkagem*") de subprogramas

- Semântica geral das chamadas a subprogramas – ações:
  1. Métodos de passagem de parâmetros (valor, resultado, valor/resultado, referência e nome)
  2. Alocação dinâmica da pilha de variáveis locais
  3. Salvar o estado de execução da unidade de programa chamadora
  4. Transferência de controle e garantia de retorno
  5. Se subprogramas aninhados são suportados, acesso a variáveis não locais deve ser garantido

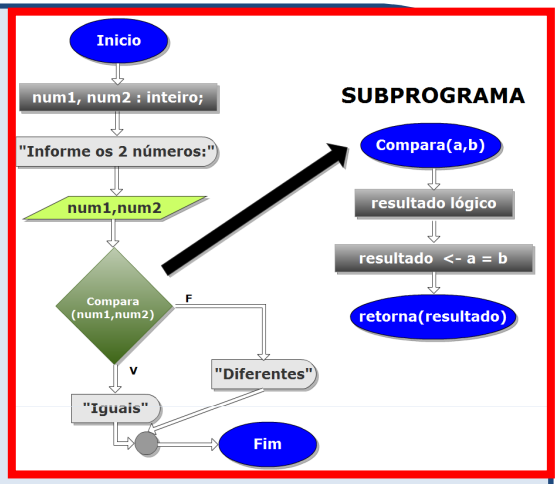
A semântica geral de chamadas e retornos

- Semântica geral de retornos de subprograma:
  1. Parâmetros do modo de saída ou do modo de entrada/saída devem ter seus valores retornados
  2. Liberação de parâmetros locais dinâmicas da pilha
  3. Retomar o estado de execução
  4. Retornar o controle ao chamador



Implementando subprogramas "simples": semântica de chamada

- Subprogramas "simples": não podem ser aninhados e todas as variáveis locais são estáticas.
- Semântica de chamada. Ações:
  1. Salvar o estado da execução da unidade de programa atual
  2. Calcular e passar os parâmetros (se houver)
  3. Passar o endereço de retorno para o subprograma chamado
  4. Transferir o controle para o subprograma chamado




```
Program procedure;

Uses crt;

Definição do Procedimento
Procedure linha;
Var
   I : integer;  { variável local }
Begin
      For i := 1 to 20 do
         Write ('* ');
      Writeln;
End;

Programa Principal
Begin
   Clrscr;
   Linha; { chamada do procedimento }
End.
```

Implementando subprogramas "simples": semântica de retorno

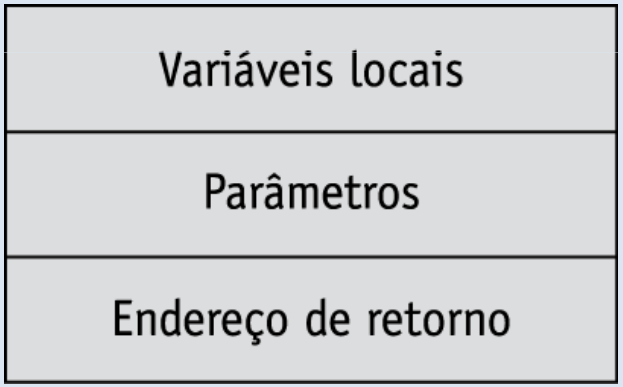
- Semântica de retorno. Ações:
  1. Se existirem parâmetros com passagem por valor-resultado ou parâmetros no modo de saída, os valores atuais desses parâmetros são movidos para os parâmetros reais (usados na chamada) correspondentes
  2. Se o subprograma é uma função, o valor funcional é movido para um local acessível ao chamador
  3. O estado da execução do chamador é restaurado
  4. O controle é transferido de volta para o chamador
  5. Armazenamento requerido:
     - ✓ Informações de estado sobre o chamador, parâmetros, endereço de retorno, valor de retorno para funções e variáveis temporárias usadas pelo subprograma

Implementando subprogramas "simples": partes

- Um subprograma simples consiste em duas partes separadas: o código real e a parte não código (variáveis locais e dados listados, que podem mudar)
- O formato, ou *layout*, da parte que não é código de um subprograma é chamado de *registro de ativação - RA* (guarda o estado e as variáveis locais durante a execução)
- Uma *instância de registro de ativação* - IRA é um exemplo concreto de um RA → criado no momento da invocação do subprograma

Um registro de ativação para subprogramas "simples"

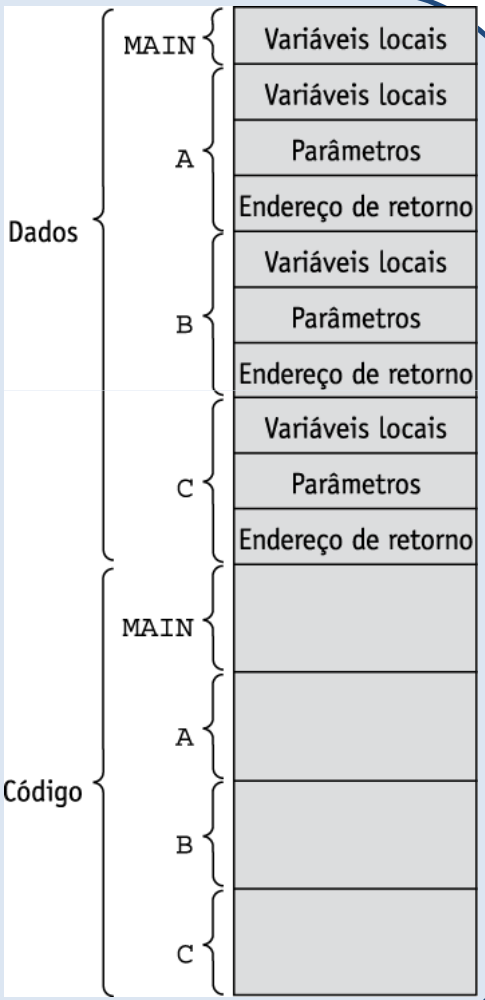
- Pode haver apenas um RA.
- Formato *(layout)* de um RA



Um registro de ativação para subprogramas "simples"

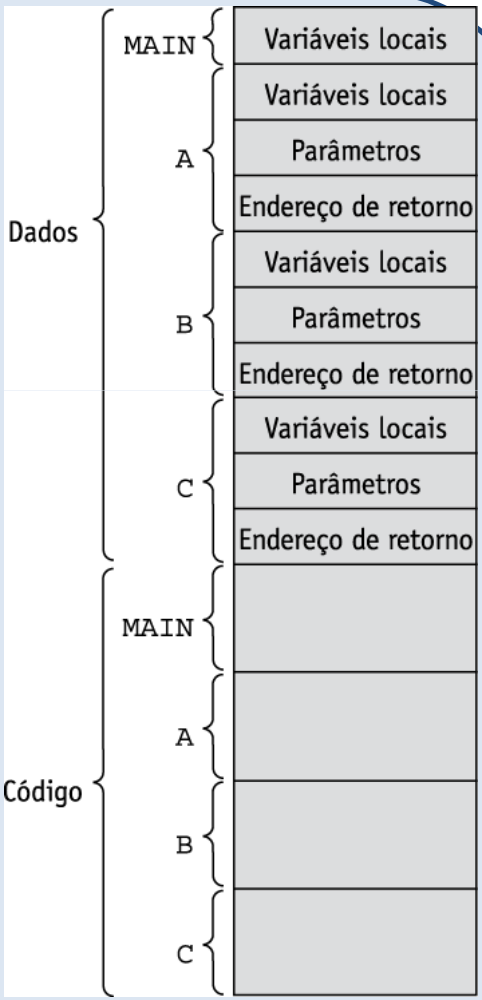
- Ex: programa formado por um programa principal MAIN e três subprogramas (A, B e C)

duas partes separadas: o código real e a parte não código (variáveis locais)



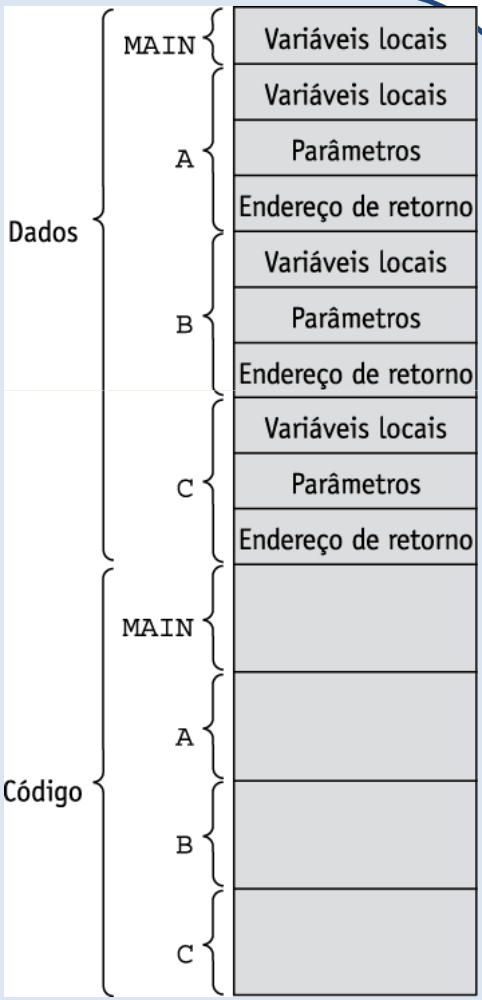
Um registro de ativação para subprogramas "simples"

- As 4 unidades do programa (MAIN, A, B, C) podem ter sido compiladas em períodos diferentes.
- O programa executável é unido pelo **ligador** → parte do SO – chamado de carregadores, ligadores, editores de ligação...



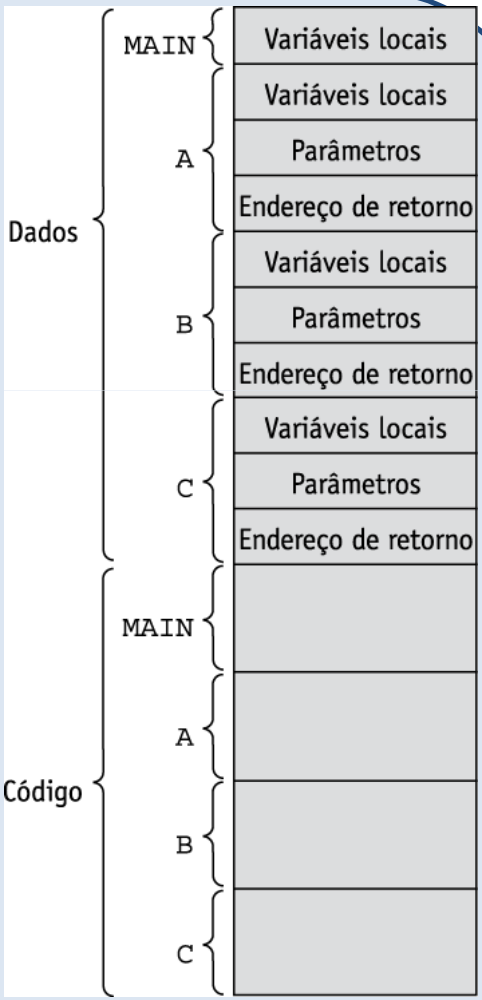
Um registro de ativação para subprogramas "simples"

- Quando o **ligador** é chamado para o prog. principal, sua primeira tarefa é "encontrar" os demais arquivos e carregá-los na memória principal



Um registro de ativação para subprogramas "simples"

- Execução:
  1. Ligador (chamado por MAIN) obtêm o código de máquina para os subprogramas A, B e C juntamente com suas IRA e carrega-os na memória
  2. Ligador corrige os endereços de destino para todas as chamadas (A,B,C)

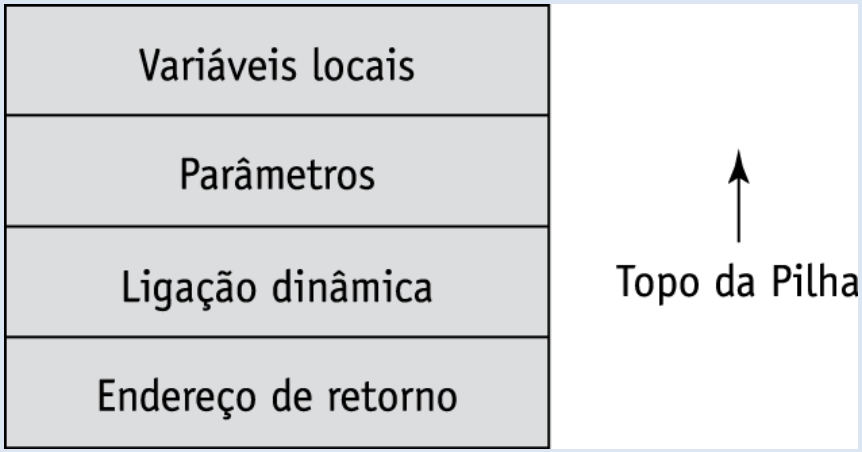


Implementando subprogramas com variáveis locais dinâmicas da pilha

- Requer **RA** 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)
    - ✓ Recursão requer múltiplas IRA
  - O formato de um RA é conhecido em tempo de compilação

Um registro de ativação típico para uma linguagem com variáveis locais dinâmicas da pilha

- Exemplo de um **RA**
  - Como o endereço de retorno, a ligação dinâmica e os parâmetros são colocadas na IRA pelo chamador, essas entradas devem aparecer primeiro
  - O **endereço de retorno** é um ponteiro para a instrução seguinte à chamada no segmento de código da unidade de programa chamadora



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 IRA é criada dinamicamente quando um subprograma é chamado
- IRA residem na pilha de tempo de execução
- O PE (*Environment Pointer*) é mantido pelo sistema (SO) em tempo de execução. Ele sempre aponta para a base da IRA da unidade de programa que está sendo executada

Um exemplo: função C

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

- Ativar um subprograma requer a criação dinâmica de um RA para o subprograma
- O PE (ponteiro de ambiente) controla a execução do subprograma.
  - Inicialmente aponta para a base (RA do programa principal)
  - Posteriormente aponta para a base do RA em execução.
  - Após retorno do subprograma, o topo da pilha é configurado para o valor de PE menos um

Pilha de tempo de execução
RA para sub

Local	sum
Local	list [4]
Local	list [3]
Local	list [2]
Local	list [1]
Local	list [0]
Parâmetro	part
Parâmetro	total
Ligação dinâmica	
Endereço de retorno	

Um exemplo sem recursão

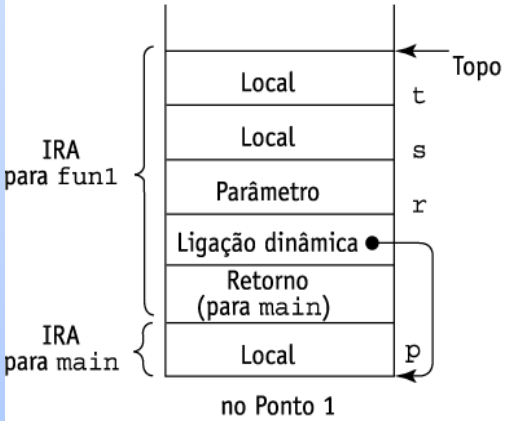
```
void fun1(float r) {
   int s,t;
   ...<----------1
   fun2(s);
   ...
}
void fun2(int x) {
   int y;
   ... <----------2
   fun3(y);
   ...
}
void fun3(int q) {
   ... <----------3
}
void main() {
   float p;
   ...
   fun1(p);
   ...
}
```

main chama fun1
fun1 chama fun2
fun2 chama fun3

Um exemplo sem recursão
pilha

```
void fun1(float r) {
   int s,t;
   ...<----------1
   fun2(s);
   ...
}
void fun2(int x) {
   int y;
   ... <----------2
   fun3(y);
   ...
}
void fun3(int q) {
   ... <----------3
}
void main() {
   float p;
   ...
   fun1(p);
   ...
}
```

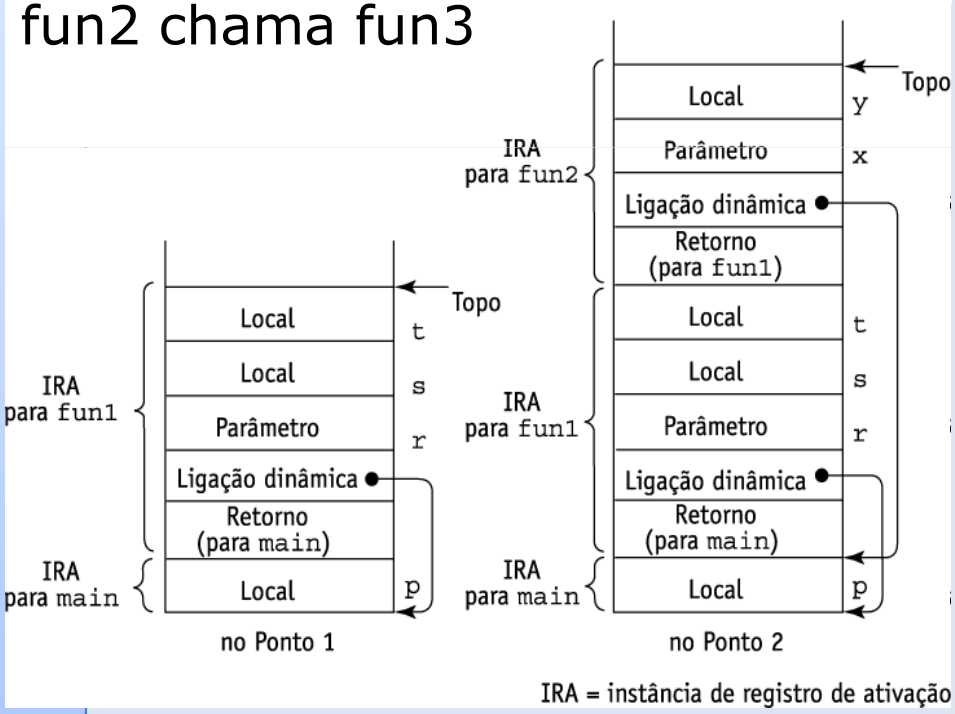
main chama fun1
fun1 chama fun2
fun2 chama fun3



Um exemplo sem recursão
pilha

```
void fun1(float r) {
   int s,t;
   ...<----------1
   fun2(s);
   ...
}
void fun2(int x) {
   int y;
   ... <----------2
   fun3(y);
   ...
}
void fun3(int q) {
   ... <----------3
}
void main() {
   float p;
   ...
   fun1(p);
   ...
}
```

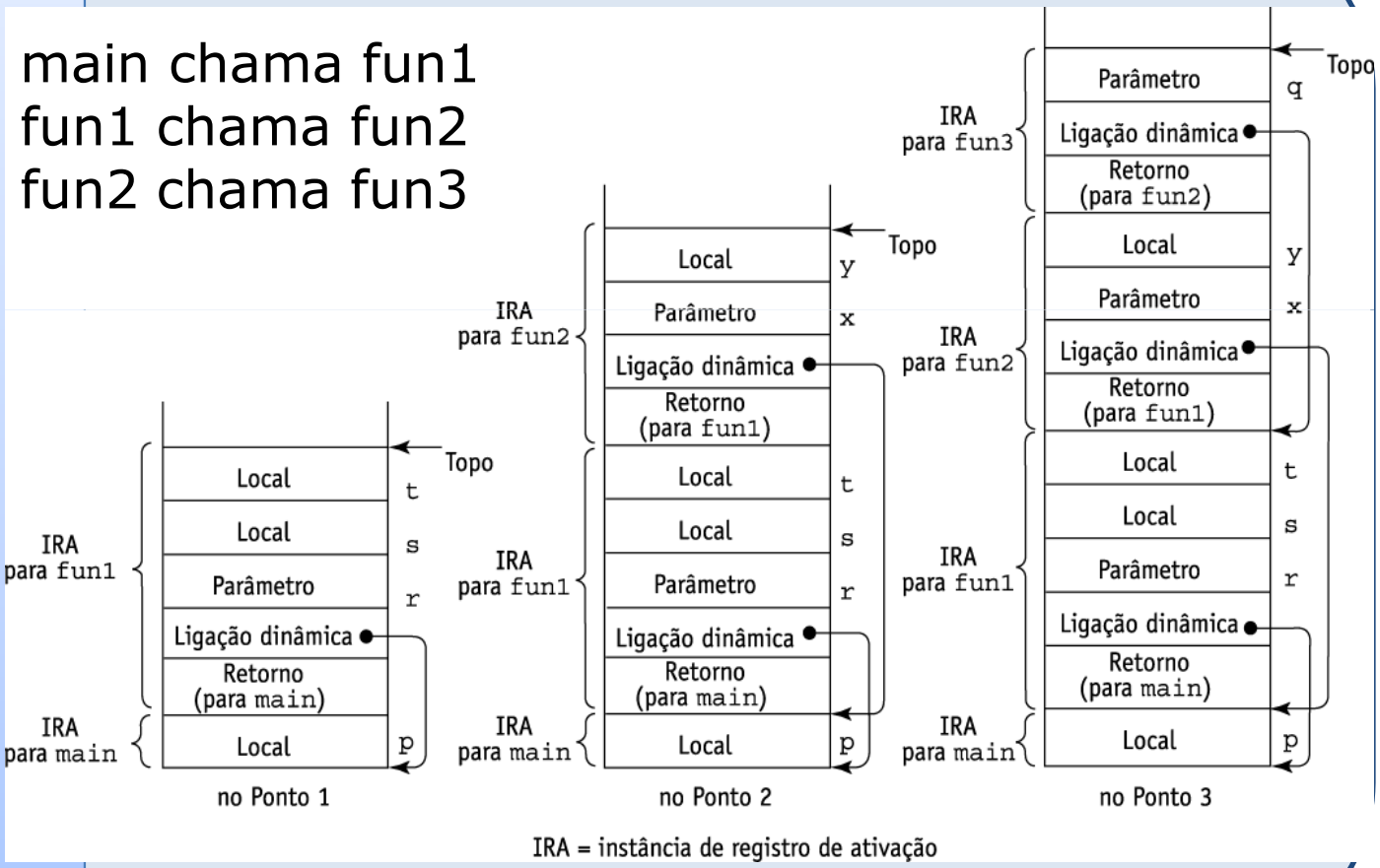
main chama fun1
fun1 chama fun2
fun2 chama fun3



Um exemplo sem recursão
pilha

```
void fun1(float r) {
   int s,t;
   ...<----------1
   fun2(s);
   ...
}
void fun2(int x) {
   int y;
   ... <----------2
   fun3(y);
   ...
}
void fun3(int q) {
   ... <----------3
}
void main() {
   float p;
   ...
   fun1(p);
   ...
}
```

main chama fun1
fun1 chama fun2
fun2 chama fun3



Cadeia dinâmica e deslocamento local

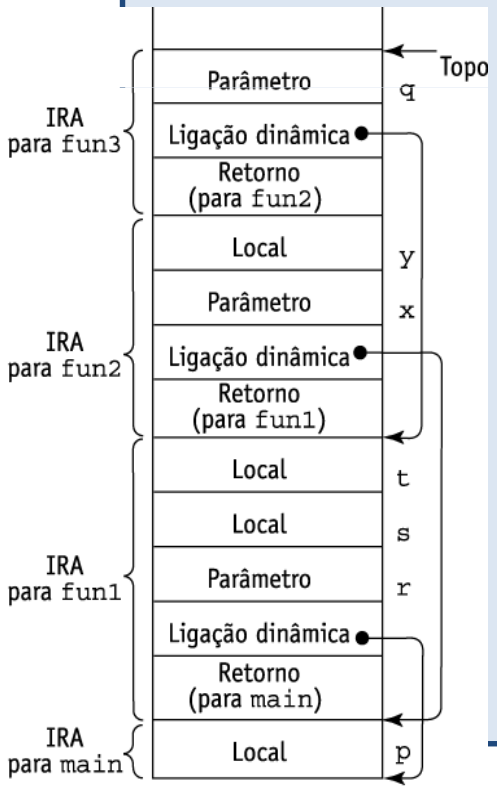
- A coleção de ligações dinâmicas presentes na pilha em um dado momento é chamada de *cadeia dinâmica ou cadeia de chamadas*

- Referências a variáveis locais podem ser representadas no código como deslocamentos a partir do início do RA 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 RA pode ser determinado em tempo de compilação

Cadeia dinâmica e deslocamento local

- Obtendo o deslocamento local ➔ duas posições (end. retorno e lig. dinâmica) + o número de parâmetros a partir da parte inferior



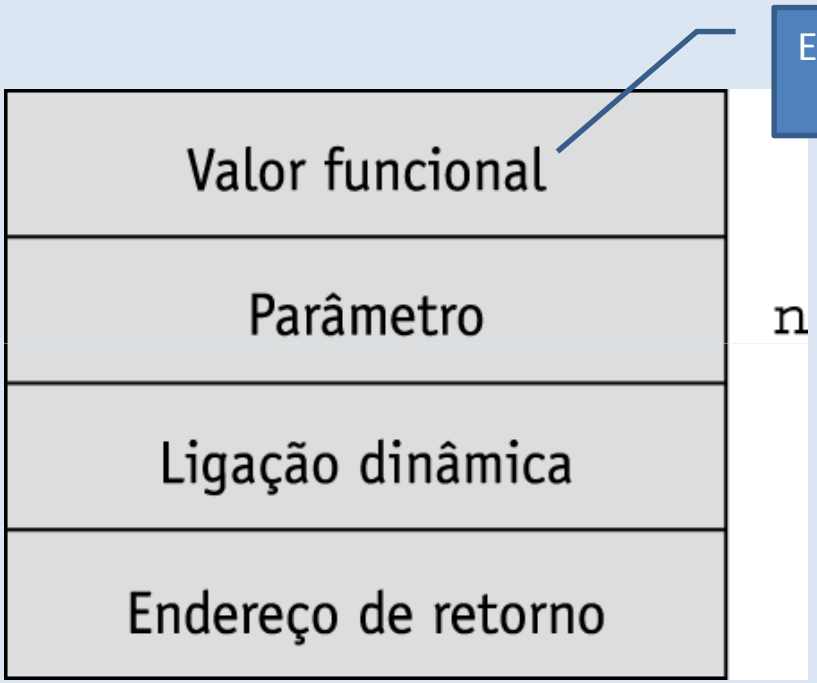
- Deslocamento local
  - s=3 (endereço de retorno → 0 lig. din. → 1 r = 2)
  - t=4
  - y=2

Um exemplo com recursão

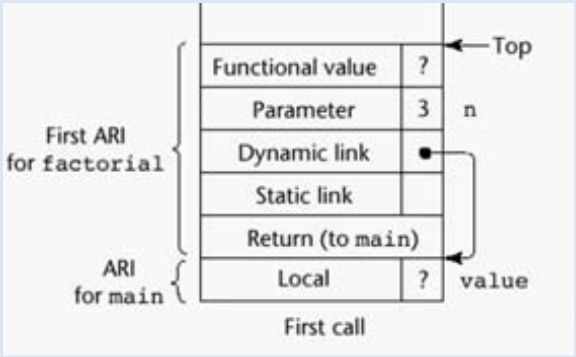
- Exemplo de programa em C que usa recursão para calcular a função fatorial

```
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 de ativação para o `fatorial`



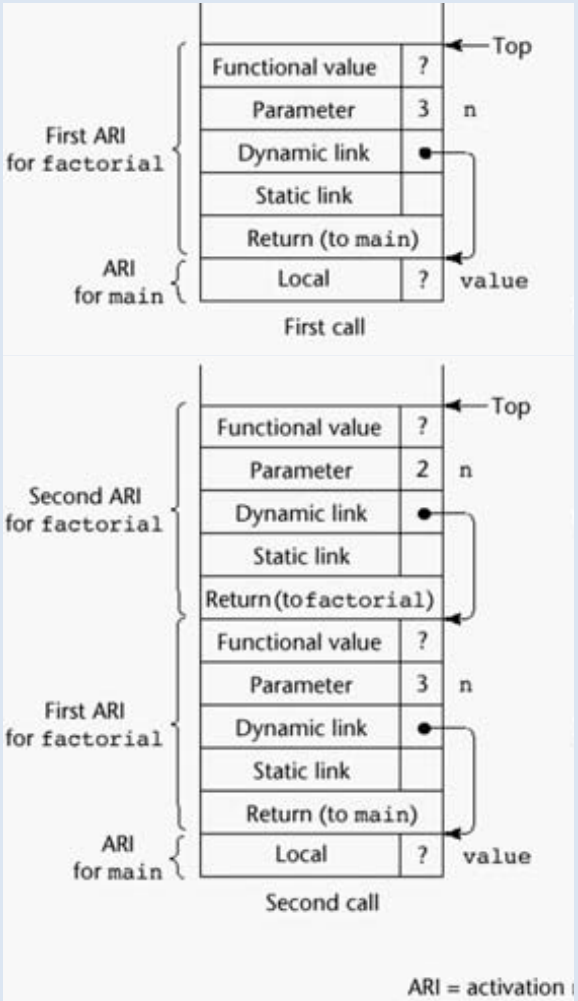
Conteúdo da pilha na posição 1 do `fatorial`




```
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
    }
```

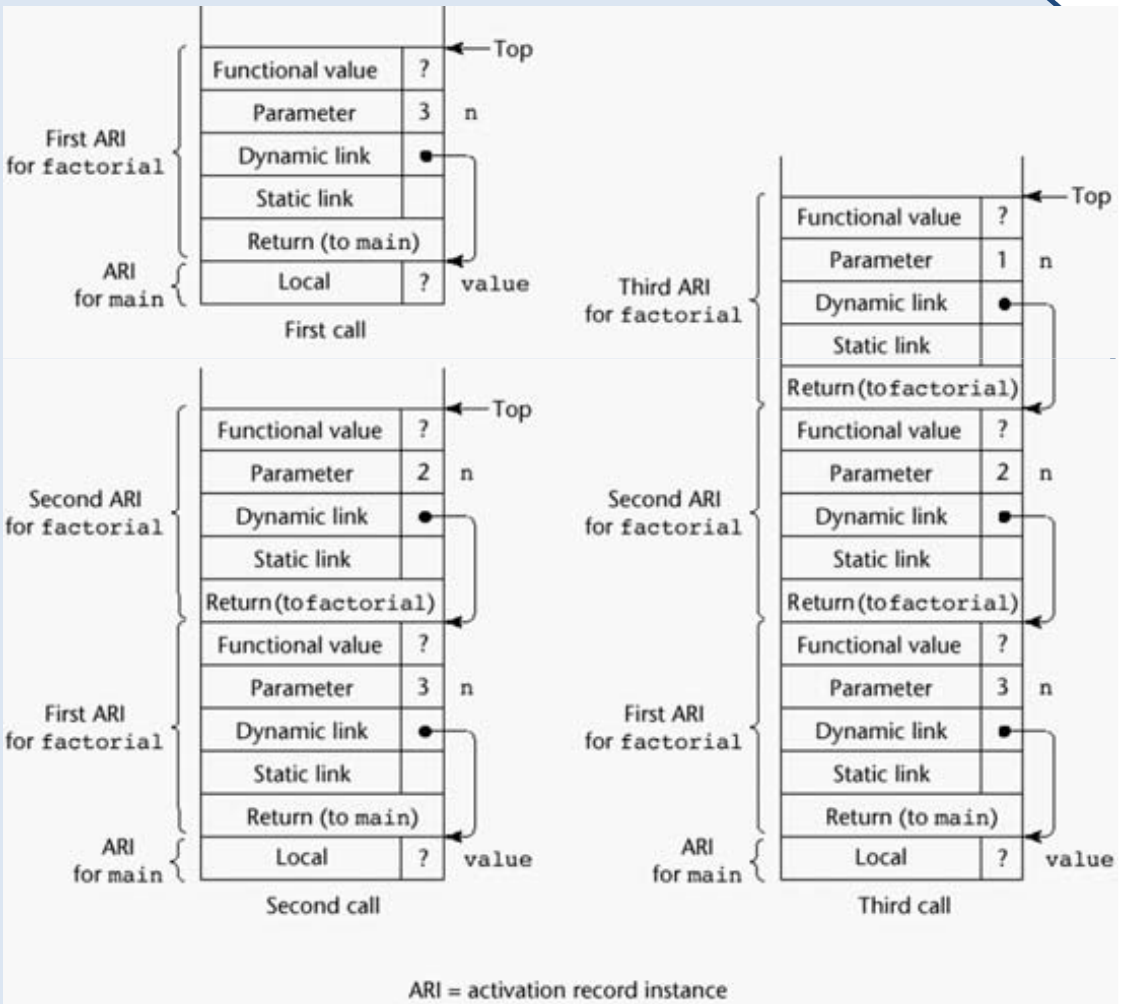
Conteúdo da pilha na posição 1 do `fatorial`

```
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
    }
```



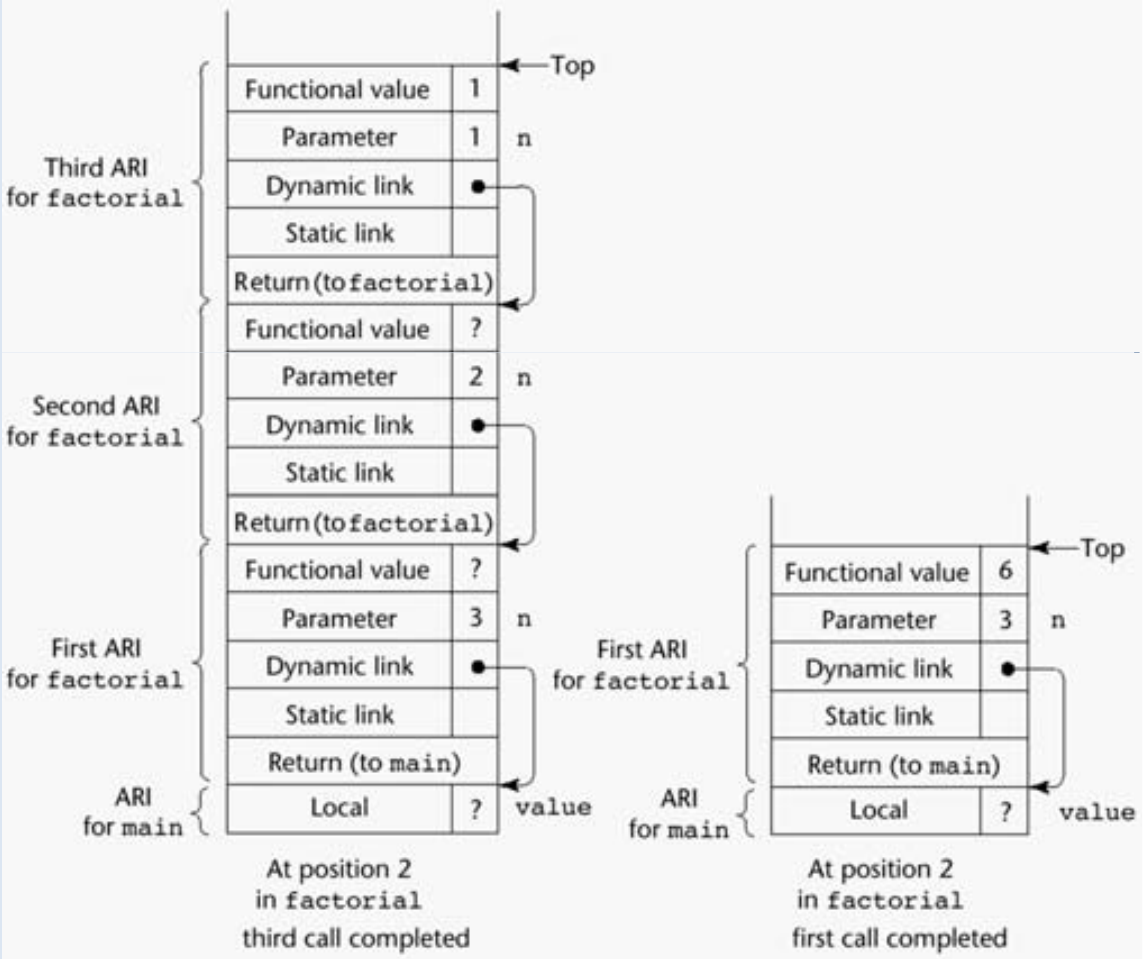
Conteúdo da pilha na posição 1 do `fatorial`

```
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
    }
```



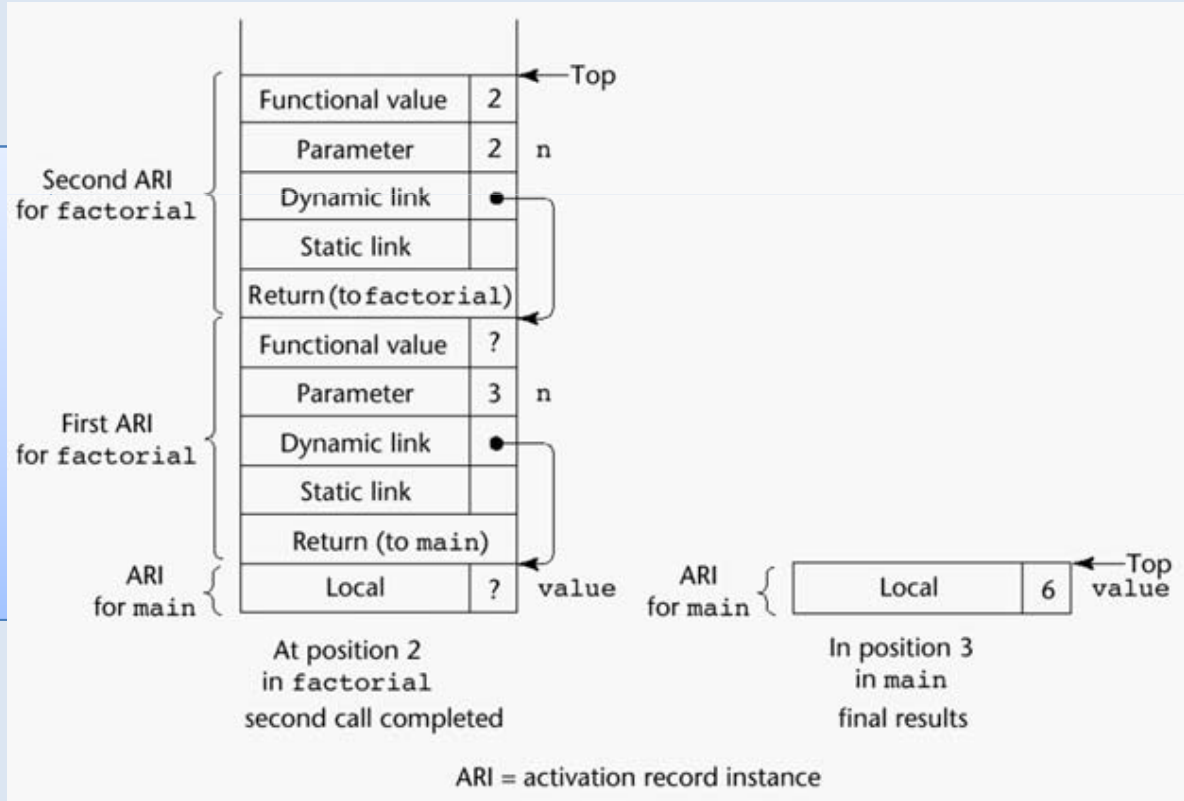
Conteúdo da pilha na posição 2 do `fatorial`

```
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
    }
```



Conteúdo da pilha na posição 2 e 3 do `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
    }
```



Subprogramas aninhados

- Algumas das LP's 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 IRA existentes e, logo, estão em algum lugar na pilha

- O processo de referência para uma variável não local:
  1. Encontrar a IRA 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 IRA correta
  - Regras de semântica estática garantem que todas as variáveis não locais que podem ser referenciadas foram alocadas em alguma IRA que está na pilha quando a referência é feita

Encadeamentos estáticos

- Um *encadeamento estático* é uma cadeia de ligações estáticas que conectam certas IRA na pilha
- A *ligação estática* aponta para o final da IRA de uma ativação do ancestral estático
- A cadeia estática de uma IRA 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

Encadeamentos estáticos

- *Exemplo de Profundidade estática* é um inteiro associado com um escopo estático que indica o quão profundamente ele está aninhado no escopo mais externo

```
procedure A is   <----- profundidade estática de A = 0
    procedure B is <----- profundidade estática de B = 1
        procedure C is <----- profundidade estática de C = 2
            ....
        end; <--de C
    ....
    end; <--de B
  .....
end; <--de A
```

- Se C referencia uma variável declarada em A, o deslocamento de encadeamento dessa referência seria 2 (C=2;A=0;ref=2-0=2)

Encadeamentos estáticos (cont...)

- O *deslocamento de encadeamento ou profundidade de aninhamento* 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 pelo par:

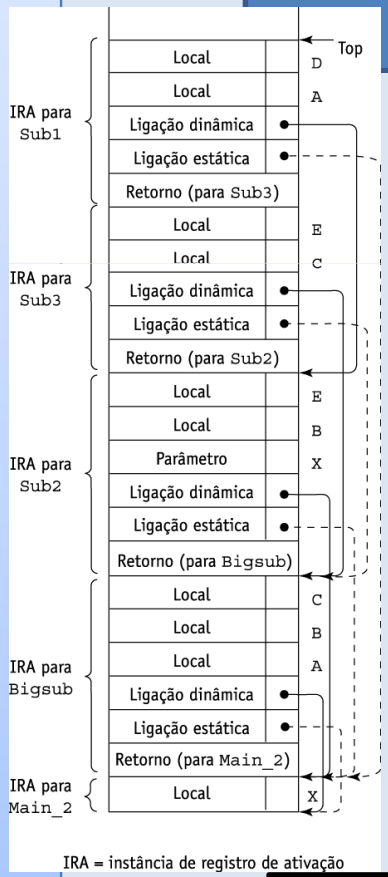
  (deslocamento de encadeamento, deslocamento local)

Exemplo de subprograma 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 sequência de chamadas a procedimentos é:

Main_2 chama Bigsub
Bigsub chama Sub2
Sub2 chama Sub3
Sub3 chama Sub1



Referência real das variáveis:

Na posição 1 em SUB1:
A - (0, 3)
B - (1, 4)
C - (1, 5)

Na posição 2 em SUB3:
E - (0, 4)
B - (1, 4)
A - (2, 3)

Na posição 3 em SUB2:
A - (1, 3)
D - erro
E - (0, 5)

(desl.encadeamento, desl. local)
(prof.aninhamento, desl. local)

Manutenção de cadeias estáticas

- No momento da chamada,
  - A IRA deve ser encontrada
    - ✓ Dois métodos:
      1. Busca a cadeia dinâmica
      2. Trata chamadas a subprogramas e definições como referências a variáveis e definições

Avaliação de cadeias estáticas

- Problemas:
  - Uma referência não local é lenta se a profundidade de aninhamento é grande
  - Código com tempo limitado é difícil:
    a) Custos de referências não locais são difíceis de determinar
    b) 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 IRA

Blocos

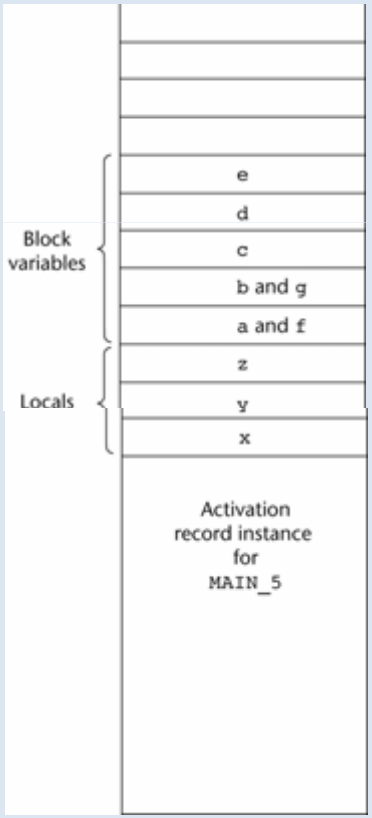
- Blocos permitem criar novos escopos locais para variáveis
- Um exemplo em C

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

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

Exemplo de blocos

```
main_5()
{ int x, y, z;
    while(...)
    { int a, b, c;
      ...
      while(...)
      { int d, e;
        ...
      }
    }
    while(...)
     { int f,g;
       ...
     }
}
```



Variáveis de blocos são armazenadas depois das variáveis locais

As variáveis f e g ocuparão as posições de a e b

Implementando blocos

- Dois métodos:
  1. Blocos são tratados como subprogramas sem parâmetros e 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 RA

Implementando escopo dinâmico

- *Acesso profundo*: as referências a variáveis não locais podem ser resolvidas com buscas por meio das IRA 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 C() {
  int x, z;
  x = u + v;
  …
}
void B() {
  int w, x;
  …
}
void A() {
  int v, w;
  …
}
void main_6() {
  int v, u;
  …
}
```

```
main chama A
A chama A
A chama B
B chama C
C chama C
C chama A
...
```

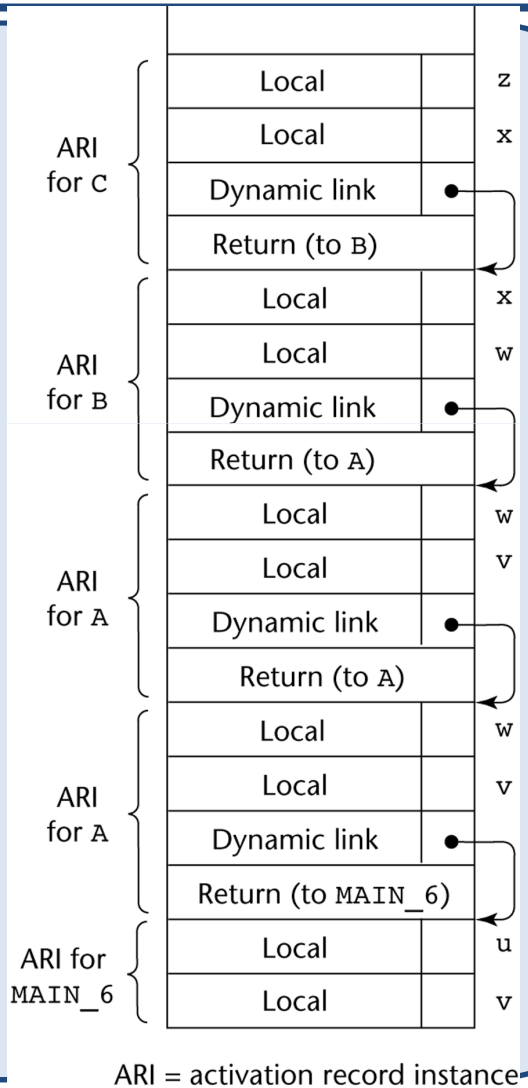
	A			B
	A	C		A
MAIN_6	MAIN_6	B	C	A
u	v	x	z	w

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

Conteúdo da Pilha para programa de escopo dinâmico

	A			B
	A	C		A
MAIN_6	MAIN_6	B	C	A
u	v	x	z	w

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



unesp

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

Resumo (cont...)

- IRA 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

Exercícios

- Questões de revisão
  - 1,2,3,6,7,18,19
- Conjunto de problemas
  - 1,2,3,4