Programação Estruturada Subprogramas

Profa. Heloisa – PLP2019

Subprogramas

Sebesta, R.W. *Concepts of Programming Languages*. 9a.edição/ Addison-Weley, 2009. Capítulo 10.

- Duas formas de abstração são possíveis em linguagens de programação: abstração de processo e abstração de dados.
- **Abstração de processos**: aparece na forma de subprogramas, que permitem:
  - o reuso de código
  - economia de tempo e de memória.
  - facilitar a leitura do programa (permite ver a estrutura lógica do programa, escondendo detalhes de codificação)

Características dos subprogramas estudados

- Cada subprograma tem um único ponto de entrada
- A unidade chamadora é suspensa durante a execução da unidade chamada – existe um único subprograma em execução a cada momento
- O controle sempre retorna a unidade chamadora quando acaba a execução da unidade chamada
- Outras formas de subprogramas:
  - corotinas
  - unidades concorrentes
  - tratadores de exceção


```
void fun1 (void);
void fun2 (void);
void fun3 (void);
void main ( ) {
    int a, b, c;
    ....
    fun1(a,b);
}
void fun1 (void) {
    int b, c, d;
    ...
    fun2(a,b);
    …
}
void fun2 (void) {
    int c, d, e;
    ...
}
void fun3 (void) {
    int d, e, f;
    ...
}
```

Chamada de fun1 – para a execução de main, passa o controle para fun1

Após a conclusão de fun1, controle retorna para main

Chamada de fun2 – para a execução de fun1, passa o controle para fun2

Após a conclusão de fun2, controle retorna para fun1

Implementação de subprogramas

- **Ligação de subprograma**: operações de chamada e retorno de subprograma
- A implementação de subprograma deve seguir a semântica da ligação de subprograma da linguagem.
- A chamada e o retorno de subprograma tem diversas operações associadas, executadas tanto pela unidade chamadora como pela unidade chamada.

Implementação de subprogramas

- Durante a chamada, execução e retorno de um subprograma, diversas informações não código precisam ser geradas e armazenadas.
- Essas informações são armazenadas em uma estrutura chamadas registro de ativação, que são alocados na pilha de execução
- Pilha de Execução:
  - parte da memória usada como pilha que guarda informações sobre ativações de subprogramas.
  - parte do sistema de execução

Implementação de subprogramas

- Estudaremos inicialmente a implementação de subprogramas para linguagens:
  - com escopo estático
  - com variáveis locais dinâmicas de pilha
  - sem subprogramas aninhados

Registro de ativação

Ativação de subprograma:

acontece quando o subprograma é chamado.

Registro de Ativação:

formato das informações não código que devem ser armazenadas durante a ativação.

Instância de registro de Ativação:

coleção particular de dados na forma de RA.

Formato de RA X Tamanho de RA

Formato (layout):

- Determina quais as informações fazem parte do RA
- Na maioria das linguagens, conhecido em tempo de compilação

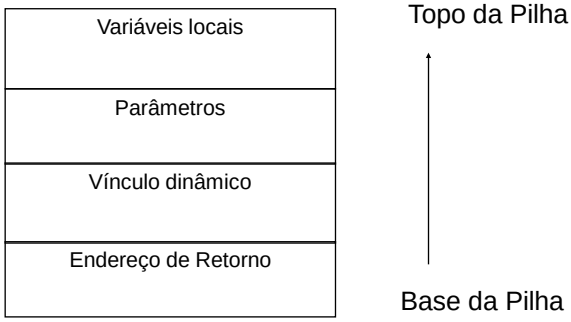
Tamanho:

- Pode mudar quando variáveis locais tem tamanho variável (arrays)
- Conhecido em tempo de compilação quando as variáveis locais tem tamanho fixo.
- Pode depender da chamada em algumas linguagens (Ada), em que o tamanho de arrays locais depende de um parâmetro real

Implementação de subprogramas com variáveis locais dinâmicas de pilha

- Característica principal: implementam recursão.
- O compilador deve gerar código para fazer alocação e liberação implícita de memória para variáveis locais.
- Pode haver múltiplas instâncias de RA para o mesmo subprograma
- O número de instâncias possíveis é limitado apenas pelo tamanho da memória da máquina
- Instâncias de RA devem ser criadas dinamicamente

Formato típico de RA
(sem aninhamento de subprogramas)



Endereço de Retorno, Vínculo Dinâmico e Parâmetros - colocados pelo **chamador**
Variáveis Locais: colocadas pelo **chamado**

- Endereço de Retorno:
  - um ponteiro para a instrução seguinte à chamada do subprograma no segmento de código da unidade chamadora.
- Vínculo Dinâmico:
  - Ponteiro para a base do RA do chamador. Em linguagem de escopo estático é usado para recuperar informação quando ocorre erro em tempo de execução. Em linguagens de escopo dinâmico é usado pra acessar variáveis não locais.

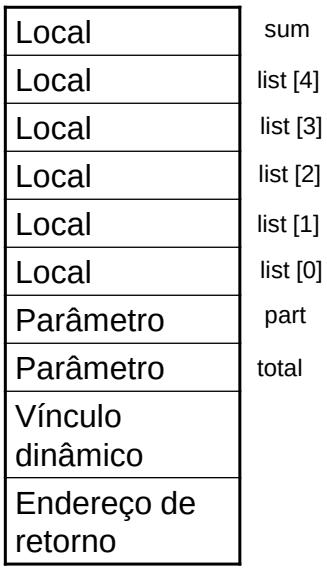
Parâmetros

- valores ou endereços fornecidos pelo chamador
- Passagem por valor: valores são copiados para a pilha. Posição é usada como variável
- Passagem por resultado: Posição usada como variável local. Resultado copiado no parâmetro real.
- Passagem por valor-resultado: combinação dos dois casos anteriores.
- Passagem por referência: endereço do parâmetro real é colocado na pilha.

Exemplo

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

Variáveis locais que são estruturas são as vezes alocadas em outro lugar e apenas seus descritores ficam no RA.



Pilha de execução

A semântica de chamada e retorno dessas linguagens especifica que o último subprograma chamado é o primeiro que termina.

Assim, é adequado criar instâncias de RA para os subprogramas em uma estrutura de pilha.

Pilha de Execução:

- parte da memória usada como pilha que guarda informações sobre ativações de subprogramas.
- parte do sistema de execução

Toda ativação de subprograma, recursiva ou não recursiva, cria uma nova instância do RA na pilha

Isso garante cópias separadas de parâmetros, variáveis locais e endereço de retorno.

Ponteiro de Ambiente de Execução (PE)

- Usado para acessar variáveis locais e parâmetros durante a execução de um subprograma.
- Inicialmente aponta para a base (primeiro endereço) do RA do programa principal
- Durante a execução do programa, aponta para a base do RA do subprograma que está sendo executado
- Quando um subprograma é chamado, PE atual é armazenado no novo registro de ativação como o vínculo dinâmico

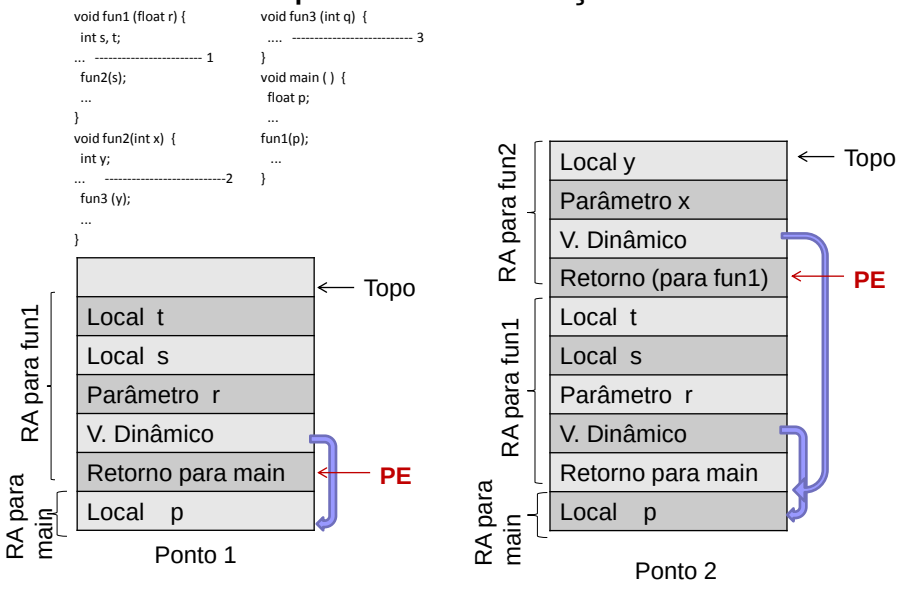
- O novo valor de PE passa a ser a base do novo registro de ativação criado
- Quando ocorre o retorno do subprograma, o ponteiro de topo da pilha é redefinido para o valor atual de PE menos 1
- O novo valor de PE passa a ser o valor do vínculo dinâmico do RA do subprograma que acabou de ser executado.
- Redefinindo o ponteiro de topo da pilha o RA do subprograma que acabou de ser executado é removido da pilha

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);
   ...
}
```

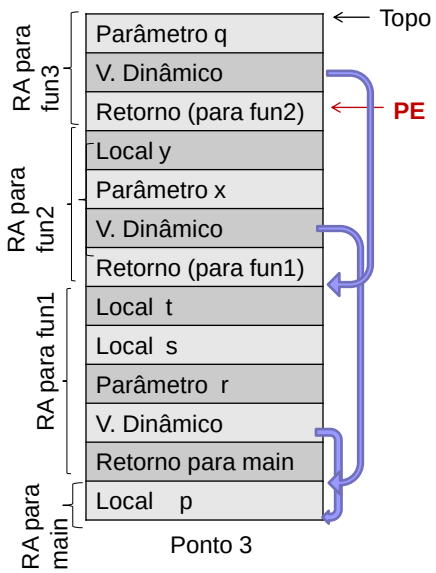
Sequência de chamadas:
main chama fun1
fun1 chama fun2
fun2 chama fun3

Conteúdo da pilha de execução



```
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);
  ...
}
```



Funções

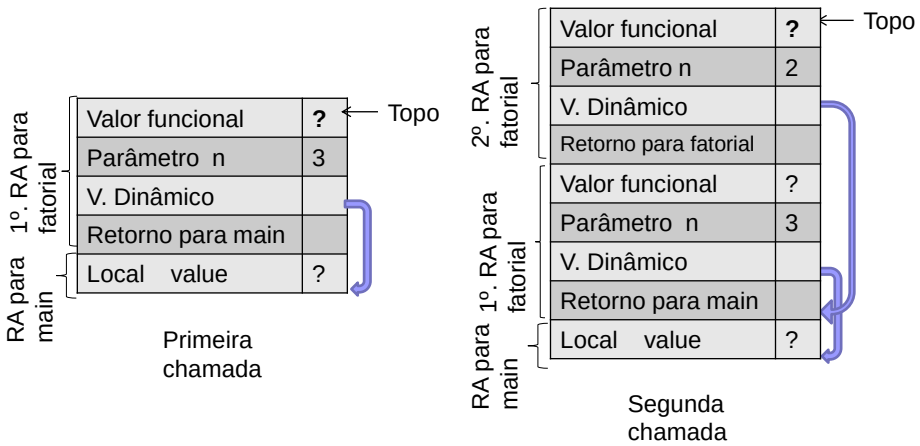
- O formato típico de registros de ativação para funções inclui uma posição adicional, para o valor que a função retorna.

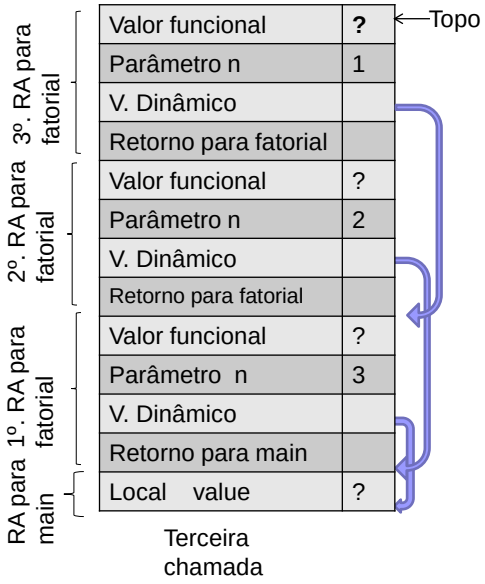
Valor da função
Variáveis locais
Parâmetros
Vínculo dinâmico
Endereço de retorno

Recursão

```
int fatorial (int n)  {
←-----------------------1
  if (n <= 1)
      return 1;
  else return ( n * fatorial (n-1));
     ←-----------------------------2
}
void main ( )   {
   int value;
   value = fatorial(3)
}
```

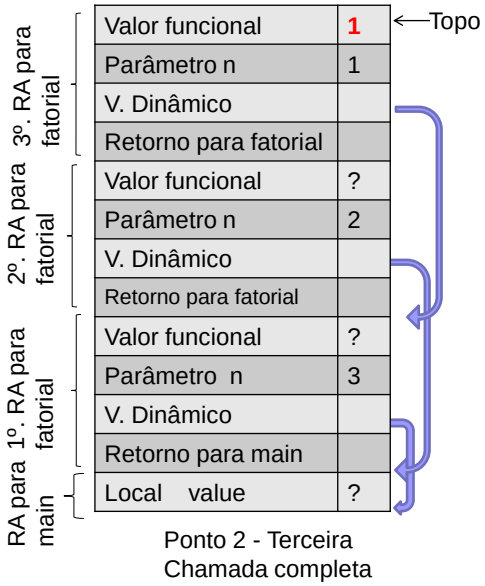

Conteúdo da pilha de execução





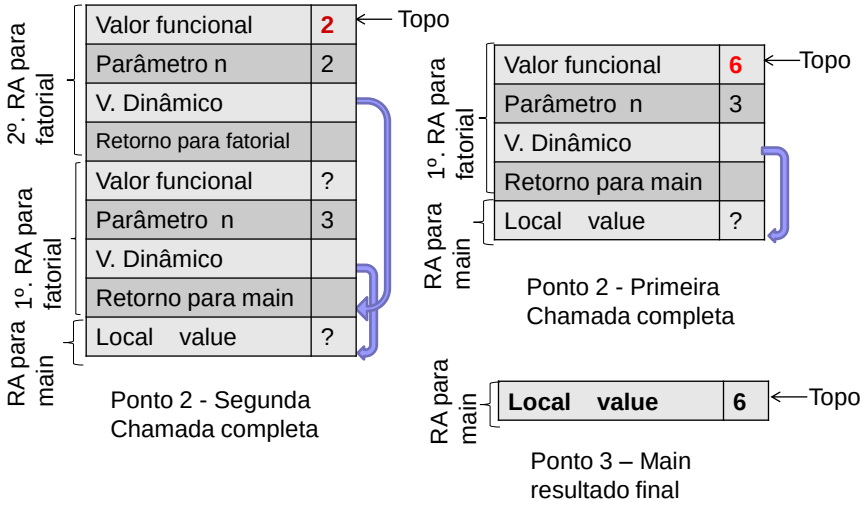
Terceira chamada

Conteúdo da pilha no ponto 2



Ponto 2 - Terceira Chamada completa

Conteúdo da pilha de execução



Subprogramas aninhados

- Algumas linguagens permitem que subprogramas sejam definidos dentro de outros subprogramas (Pascal, Ada, Python, JavaScript, Ruby)
- Pela regra semântica das linguagens de escopo estático, temos que:
  - em um dado subprograma, apenas variáveis declaradas em escopos de **antecessores estáticos** são visíveis e podem ser acessadas.
  - um subprograma é “chamável” apenas quando seus antecessores estáticos estão ativos.

Subprogramas aninhados

- Variáveis não locais (definidas em unidades mais externas) precisam ser acessadas e podem estar em subprogramas diferentes.
- Variáveis não locais estão em registros de ativação que já foram criados e estão em algum lugar da pilha.
- Para referências não locais em linguagens de escopo estático é necessário encontrar todos os RA na pilha que correspondem aos ancestrais estáticos do subprograma

Encadeamento estático

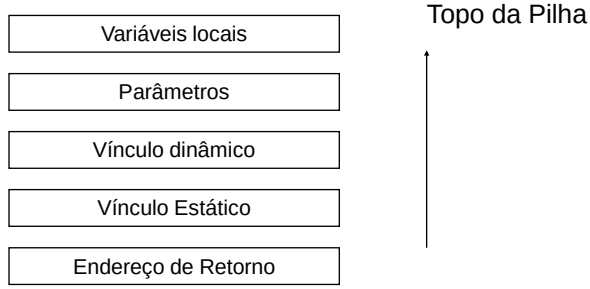
- É a maneira mais comum de implementar escopo estático em linguagens que permitem aninhamento de subprogramas.
- Utiliza o **vínculo estático**: ponteiro que aponta para a base do registro de ativação do pai estático do subprograma e é utilizado para acessar variáveis não locais.
- **Encadeamento estático** é uma sequência de vínculos estáticos que conectam registros de ativação de subprogramas aninhados.
- O encadeamento estático liga cada registro de ativação aos seus ancestrais estáticos.

Encontrando a instância de RA correta para referências não locais

- Quando é feita uma referência a uma variável não local, os ponteiros estáticos são percorridos até encontrar um RA que contenha essa variável.
- Alternativamente, como o aninhamento de escopos é conhecido em tempo de compilação, o compilador pode determinar o tamanho da cadeia a ser seguida para acessar a variável.

Vínculo Estático

Para as linguagens com aninhamento de subprogramas, o registro de ativação tem uma informação a mais:
Vínculo Estático



Endereço de Retorno, Vínculo Estático, Vínculo Dinâmico e Parâmetros - colocados pelo **chamador**
Variáveis Locais: colocadas pelo **chamado**

```
procedure  Example is
  A, B, : Integer;
  ……
  procedure Sub1  is
       X, Y, : Integer;
       begin  { sub1 }
         sub3( );          <---------- 2
       end;  { sub1 }
  procedure Sub2   is;
       X : Integer;
       ……
       procedure Sub3   is;
          X : Integer;
          begin  { Sub3}
           .....            <----------1
           end;  { Sub3 }
        begin  { Sub2}
          sub1( );          <----------3
        end;  { Sub2 }
  begin {  Example}
     sub2( );               <----------4
  end; { Example }
```