Ambientes de Execução

Organização da memória

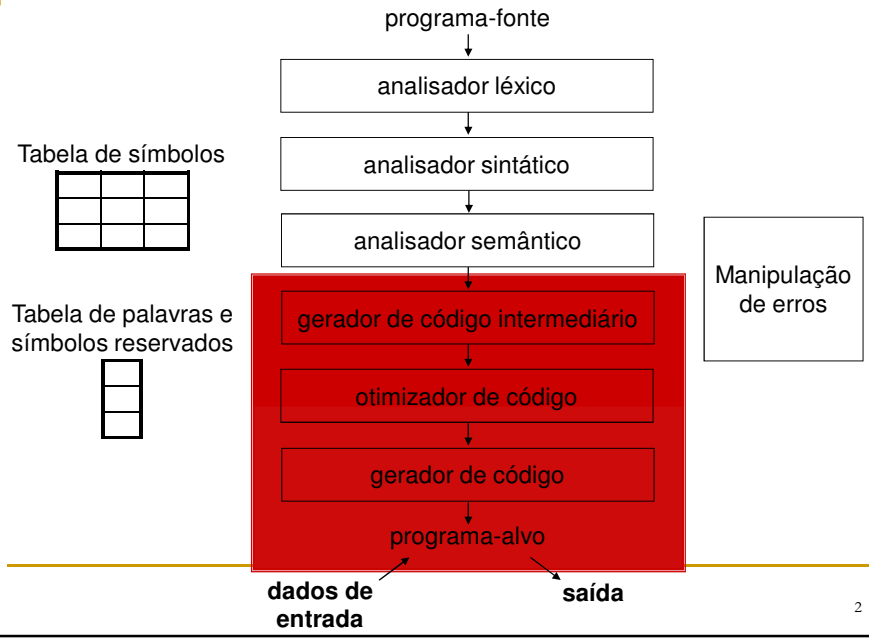
Ambientes totalmente estáticos, baseados em pilhas e totalmente dinâmicos

Passagem de parâmetros

Prof. Thiago A. S. Pardo

1

Ambientes de execução na estrutura do compilador



2

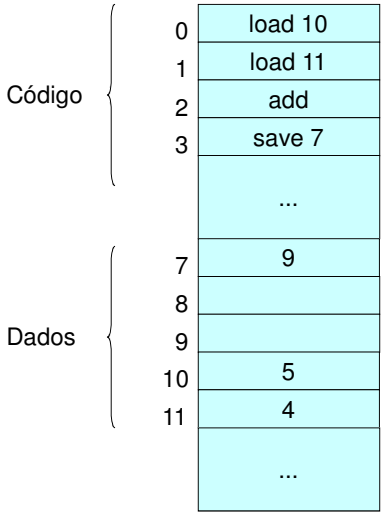
Ambientes de execução

- Análise léxica, sintática e semântica vistas até agora
  - Dependentes apenas das propriedades das linguagens-fonte, independentes da linguagem-alvo e da máquina-alvo e seu sistema operacional
- Geração e otimização de código
  - Maior parte é dependente das propriedades da máquina-alvo, em geral
- Ambientes de execução
  - Estrutura de registros e de memória da máquina-alvo: gerenciamento de memória e manutenção da informação
  - As características gerais são padronizadas para uma ampla variedade de máquinas e arquiteturas

3

Ambientes de execução

- Basicamente, em outras palavras
  - Porção de memória para se carregar o código gerado na compilação
  - Porção de memória para se carregar e trabalhar com os dados necessários que serão manipulados pelo código do programa: variáveis, parâmetros e dados de procedimento, valores temporários e de manutenção do próprio ambiente



4

Ambientes de execução

- Três tipos básicos de ambientes de execução
  - Ambiente totalmente estático
    - Fortran77
  - Ambiente baseado em pilhas
    - Pascal, C, C++
  - Ambiente totalmente dinâmico
    - LISP
- Também é possível haver híbridos

5

Ambientes de execução

- Características da linguagem podem determinar ambientes que sejam mais adequados
  - Questões de escopo e alocação de memória, natureza das ativações de procedimentos, mecanismos de passagem de parâmetros
- O compilador pode manter o ambiente em ordem apenas de forma indireta
  - Gera código para efetuar as operações necessárias de manutenção durante a execução do programa

6

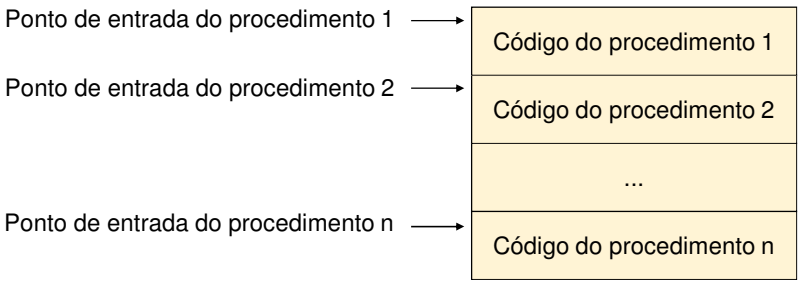
Características gerais

- Memória de um computador típico
  - Registros
  - RAM: mais lenta e de acesso endereçável não seqüencial
    - Pode ser subdividida em área de código e área de dados
      - Maioria das linguagens de programação compiladas não permitem alterar área de código durante a execução
        - Área de código é fixada durante a compilação
        - Apenas parte da área de dados pode ser fixada durante a compilação: variáveis globais, por exemplo

7

Características gerais

- Área de código
  - Conjunto de procedimentos
    - Onde estão os pontos de entrada na memória?
      - Código absoluto vs. realocável



8

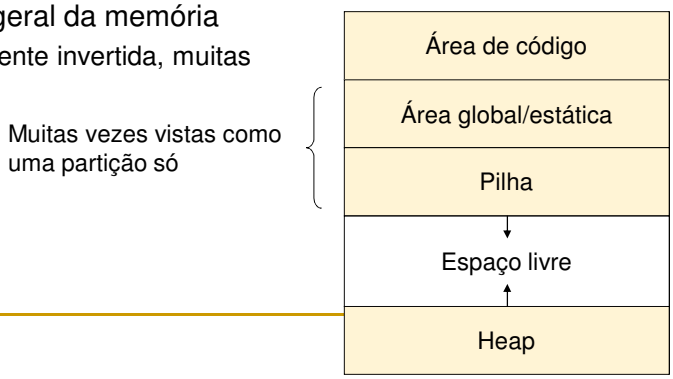
Características gerais

- Exemplos
  - Em Fortran77, todos os dados são globais e podem ter sua posição de memória fixada durante compilação
  - Constantes pequenas como 0 e 1: vale mais a pena inserí-las diretamente no código em vez de colocá-las na área de dados
  - Strings em C são ponteiros na realidade: melhor que fiquem na área de dados

9

Características gerais

- Em geral, área de dados dividida
  - Área para dados globais
  - Pilha, para dados cuja alocação é do tipo LIFO (*last in, first out*)
  - *Heap*, para dados dinâmicos, com a organização que precisar
- Organização geral da memória
  - Conceitualmente invertida, muitas vezes

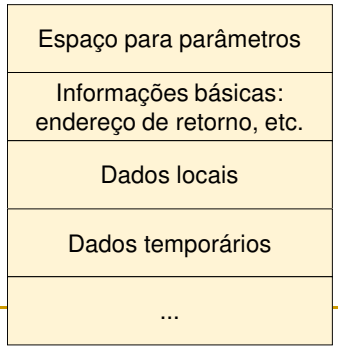


Características gerais

- Unidade de alocação importante: registro de ativação de procedimentos
  - Contém todos os dados relativos a um procedimento: endereço de retorno, parâmetros, dados locais e temporários, etc.
    - Algumas partes têm tamanho fixo para todos os procedimentos, como o endereço de retorno
    - Outras partes não, como parâmetros

Características gerais

- Registro de ativação de procedimentos
  - Dependendo da linguagem, pode estar na área global, na pilha ou na *heap*
    - Se estiver na pilha, é denominado quadro de pilha



Características gerais

- Registros de processadores também são partes do ambiente de execução
  - Contador de programa: para indicar instrução corrente do programa sendo executado
  - Ponteiro da pilha: para indicar dado sendo manipulado
  - Ponteiro de quadro: para registro de ativação de procedimento corrente
  - Ponteiro de argumentos: para parte de parâmetros do registro de ativação do procedimento

13

Características gerais

- Parte importante do projeto de um ambiente de execução: seqüência de operações para ativação de um procedimento
  - Alocação de memória para o registro de ativação, computação e armazenamento dos parâmetros, armazenamento de um valor de retorno, reajuste de registros, liberação de memória
    - Seqüência de ativação
      - Opcionalmente, seqüência de ativação e seqüência de retorno, para operações relativas à ativação e à finalização do procedimento, em vez de se ver tudo dentro da seqüência de ativação

14

Características gerais

- Ponto importante sobre a seqüência de ativação
  - Quem realiza as operações de ativação de um procedimento? O procedimento ativado ou quem o ativou?
    - Deve-se fazer uma boa divisão de tarefas!

15

Ambientes de execução

- Ambientes totalmente estáticos
- Ambientes baseados em pilhas
- Ambientes totalmente dinâmicos

16

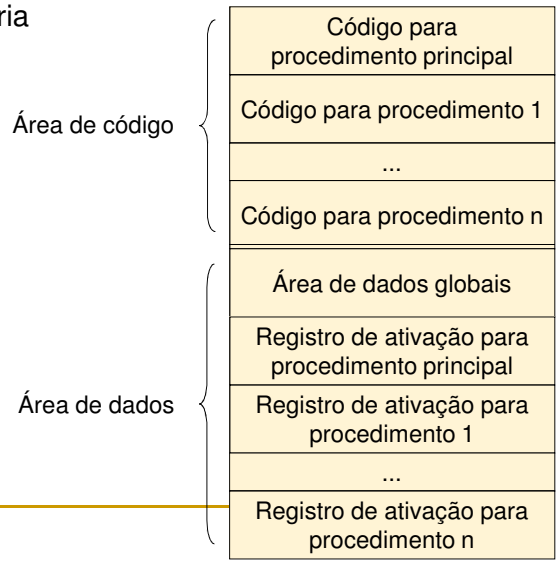
Ambientes totalmente estáticos

- Tipo mais simples
  - Todos os dados são estáticos, permanecendo fixos na memória durante toda a execução do programa
  - Útil quando
    - Não há ponteiros ou alocação dinâmica
    - Não há procedimentos ativados recursivamente

17

Ambientes totalmente estáticos

- Visualização da memória



Ambientes totalmente estáticos

- Relativamente pouca sobrecarga de informação de acompanhamento para preservar cada registro de ativação
- Nenhuma informação adicional do ambiente (além de possivelmente o endereço de retorno)
- Seqüência de ativação particularmente simples
  - Ao se chamar um procedimento, cada argumento é computado e armazenado no registro de ativação do procedimento
  - O endereço de retorno do ativador é gravado
  - Ocorre um salto para o começo do código do procedimento ativado
  - No retorno, um salto simples é efetuado para o endereço de retorno

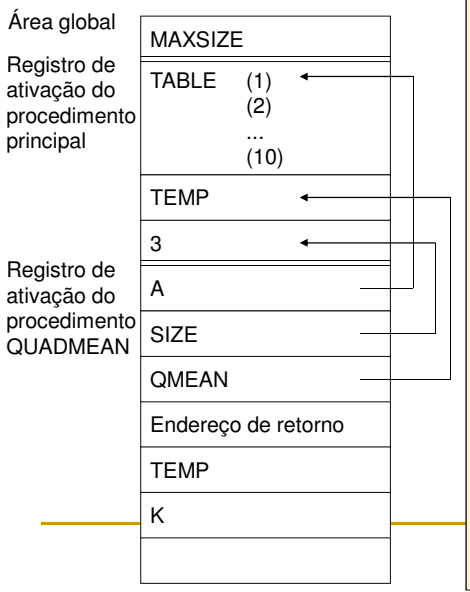
Ambientes totalmente estáticos

- Programa Fortran77

```
PROGRAM TEST
COMMON MAXSIZE
INTEGER MAXSIZE
REAL TABLE(10), TEMP
MAXSIZE=10
READ *, TABLE(1), TABLE(2), TABLE(3)
CALL QUADMEAN(TABLE,3,TEMP)
PRINT *, TEMP
END

SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
COMMON MAXSIZE
INTEGER MAXSIZE, SIZE
REAL A(SIZE), QMEAN, TEMP
INTEGER K
TEMP=0.0
IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO
99
DO 10 K=1, SIZE
        TEMP=TEMP+A(K)*A(K)
10 CONTINUE
99 QMEAN=SQRT(TEMP/SIZE)
RETURN
END
```

Ambientes totalmente estáticos

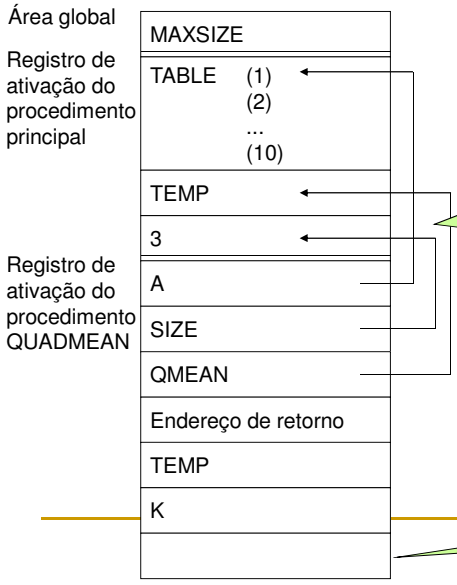
- Programa Fortran77




```
PROGRAM TEST
COMMON MAXSIZE
INTEGER MAXSIZE
REAL TABLE(10), TEMP
MAXSIZE=10
READ *, TABLE(1), TABLE(2), TABLE(3)
CALL QUADMEAN(TABLE,3,TEMP)
PRINT *, TEMP
END

SUBROUTINE QUADMEAN(A,SIZE,QMEAN)
COMMON MAXSIZE
INTEGER MAXSIZE, SIZE
REAL A(SIZE), QMEAN, TEMP
INTEGER K
TEMP=0.0
IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO
99
DO 10 K=1, SIZE
        TEMP=TEMP+A(K)*A(K)
10 CONTINUE
99 QMEAN=SQRT(TEMP/SIZE)
RETURN
END
```

Ambientes totalmente estáticos

- Programa Fortran77




```
PROGRAM TEST
COMMON
INTEGER
REAL TAB
MAXSIZE=
READ *, TA
CALL QUA
PRINT *, T
END
                                    AN)
COMMON
INTEGER
REAL A(SIZE), QMEAN, TEMP
INTEGER K
TEMP=0.0
IF ((SIZE.GT.MAXSIZE).OR.(SIZE.LT.1)) GOTO
99
DO 10 K=1, SIZE
        TEMP=TEMP+A(K)*A(K)
10 CONTINUE
99 QMEA
RETU
END
```

Valores dos parâmetros são referências de memória. Conseqüências:
- referência adicional para se acessar um valor de parâmetro
- matrizes não precisam ser copiadas, tendo sua base referenciada
- constantes precisam ser armazenadas

Espaço extra para cálculos de temporários

Ambientes totalmente estáticos

- Questão
  - Por que esse tipo de ambiente não suporta procedimentos recursivos?

23

Ambientes de execução

- Ambientes totalmente estáticos
- Ambientes baseados em pilhas
- Ambientes totalmente dinâmicos

24

Ambientes baseados em pilhas

- Limitações de ambientes estáticos
  - Procedimentos recursivos necessitam de mais de um registro de ativação criado no momento de sua chamada
  - A cada ativação, variáveis locais dos procedimentos recebem novas posições de memória
  - Em um dado momento, vários registros de ativação de um mesmo procedimento podem existir na memória
- Solução: ambiente baseado em pilha
  - Na ativação de um procedimento, empilha-se um novo registro de ativação e os novos dados necessários
  - Quando um procedimento termina, os dados correspondentes são desempilhados
    - Maior controle da informação é necessário!

25

Ambientes baseados em pilhas

- Estudaremos estes ambientes com níveis crescentes de complexidade
  - Sem procedimentos locais
  - Com procedimentos locais
  - Com procedimentos como parâmetros

26

Ambientes baseados em pilhas

- Ambientes com níveis crescentes de complexidade
  - Sem procedimentos locais
  - Com procedimentos locais
  - Com procedimentos como parâmetros

27

Ambientes baseados em pilhas sem procedimentos locais

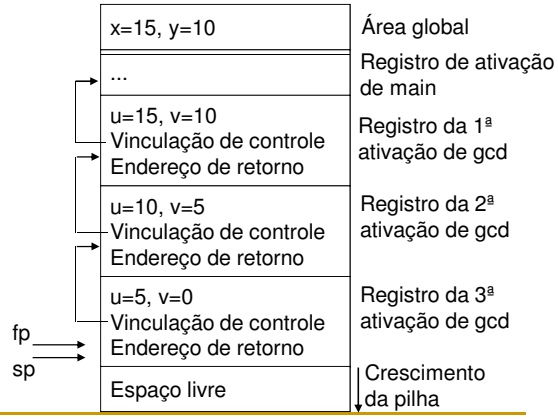
- Em linguagens com procedimentos globais somente (por exemplo, C), são necessários
  - Ponteiro para o registro de ativação corrente
    - Ponteiro de quadro (fp = *frame pointer*)
  - Posição do registro do ativador, de forma que este possa ser recuperado quando o procedimento chamado terminar
    - Ponteiro no registro corrente para o registro anterior
      - Essa informação é chamada "vinculação de controle" ou "vinculação dinâmica" (pois acontece em tempo de execução)
  - Muitas vezes, guarda-se também o topo da pilha (sp = *stack pointer*)

28

Ambientes baseados em pilhas sem procedimentos locais

- Exemplo: algoritmo de Euclides para calcular máximo divisor comum

```
#include <stdio.h>

int x, y;

int gcd(int u, int v) {
        if (v==0) return u;
        else return gcd(v, u%v);
}

main() {
        x=15; y=10;
        printf("%d\n", gcd(x,y));
        return 0;
}
```



29

Ambientes baseados em pilhas sem procedimentos locais

- Atenção
  - A cada procedimento ativado, um novo quadro de pilha é empilhado e fp é incrementado
  - Quando um procedimento termina, o quadro mais ao topo é desempilhado e a vinculação de controle retorna o controle para o procedimento ativador
  - Quando os procedimentos acabarem, sobrarão na pilha somente a área global e o registro de ativação de main

30

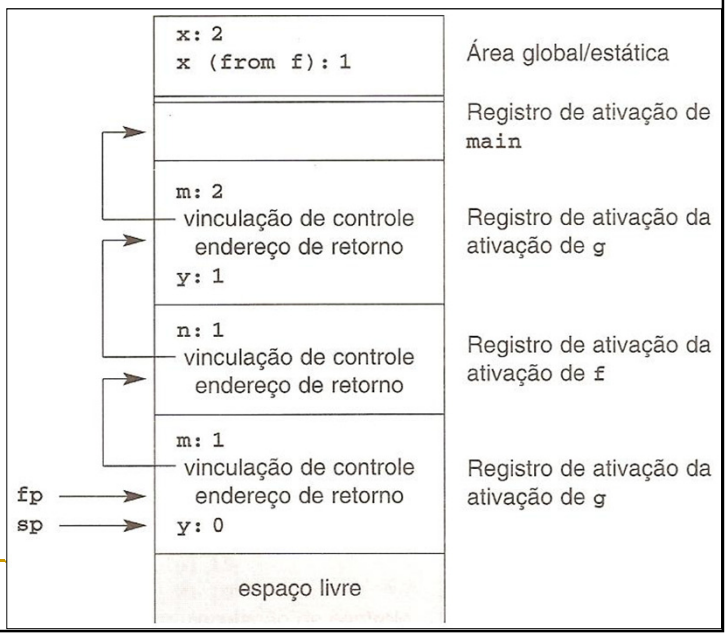
Exercício

- Monte a pilha com os registros de ativação apropriados para a execução do programa ao lado
  - Atenção: como a variável x do procedimento f é static, precisa estar na área global em vez de estar no registro de ativação de f

```
int x=2;

void g(int); /*protótipo*/

void f(int n) {
        static int x=1;
        g(n);
        x--;
}

void g(int m) {
        int y=m-1;
        if (y>0) {
                f(y);
                x--;
                g(y);
        }
}

main() {
        g(x);
        return(0);
}
```

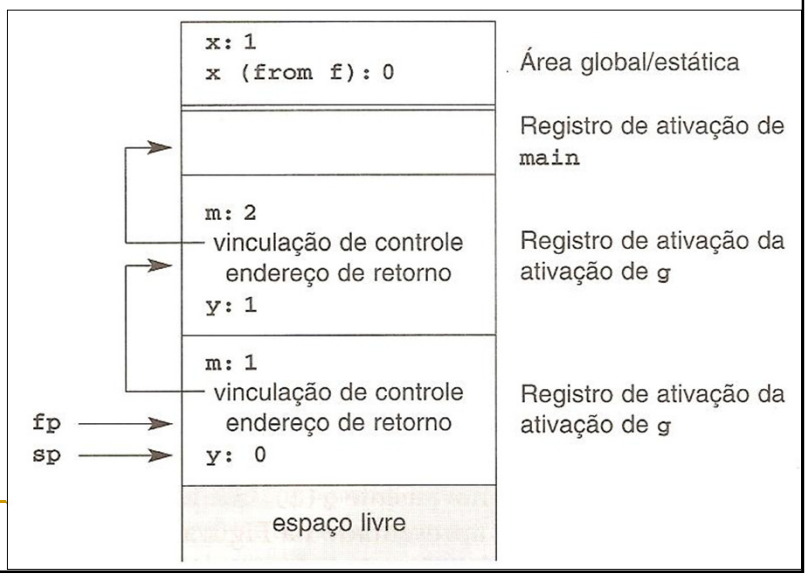
Exercício

- Ambiente imediatamente após a 2ª ativação de g



Exercício

- Ambiente imediatamente após a 3ª ativação de g



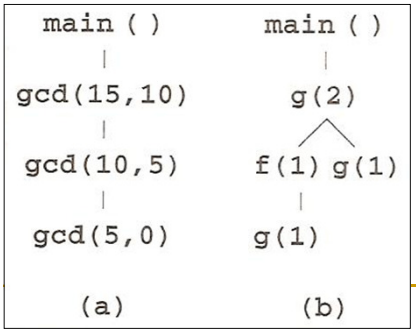
Exercício

- Atenção
  - O registro da 3ª ativação de g ocupa a área de memória previamente ocupada pelo registro de f (que foi encerrado e retirado da pilha)
  - A variável estática x de f não é confundida com a x global porque a tabela de símbolos determina quem é quem no programa e determina seus acessos de forma correta no código gerado

34

Ambientes baseados em pilhas sem procedimentos locais

- Árvore de ativação
  - Ferramenta útil para a análise de estruturas de ativação (que podem ser muito complexas)
    - Exemplos: algoritmo de Euclides (a) e programa anterior (b)



Atenção:

- cada registro de ativação é um nó na árvore

- os descendentes de cada nó representam as ativações efetuadas a partir dele

- a pilha de registros em um tempo t corresponde ao caminho do nó correspondente na árvore até a raiz

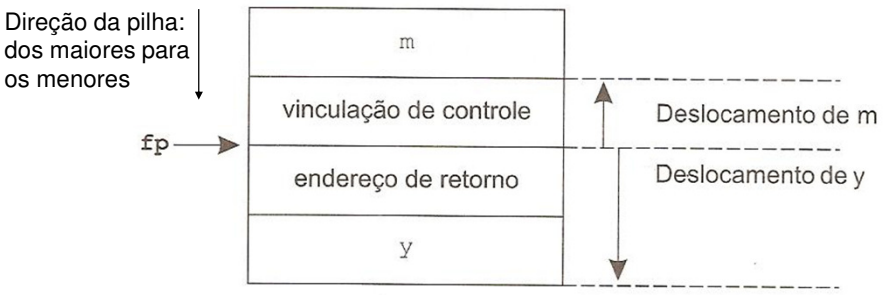
Ambientes baseados em pilhas sem procedimentos locais

- No ambiente baseado em pilhas, parâmetros e variáveis locais dos procedimentos não podem ser acessados como no ambiente totalmente estático
  - Eles não estão na área global, mas na área dos registros de ativação dos procedimentos...
    - mas se os registros têm o mesmo tamanho, é possível calcular a posição relativa dos registros

36

Ambientes baseados em pilhas sem procedimentos locais

- Esquema geral



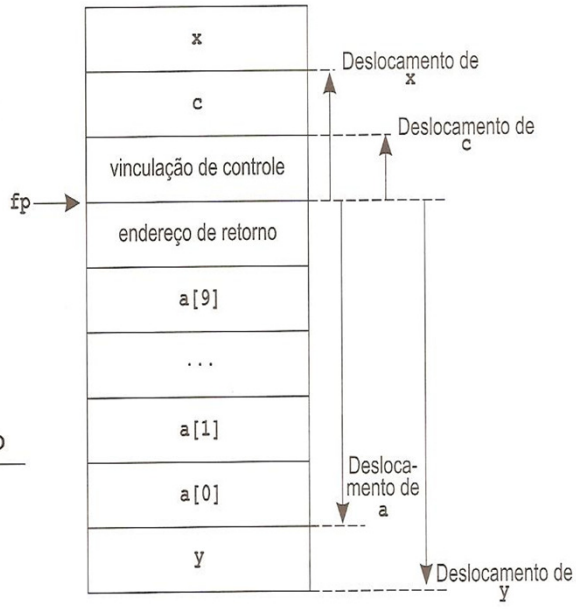
37

Exemplo

```
void f(int x, char c)
{ int a[10];
  double y;
  ...
}
```

int=2 bytes, char=1 byte
endereços=4 bytes
double=8 bytes

Nome	Deslocamento
x	+5
c	+4
a	–24
y	–32



38

Ambientes baseados em pilhas sem procedimentos locais

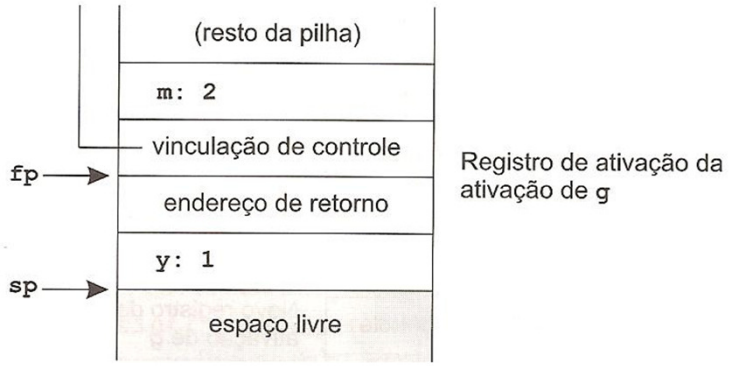
- Passos básicos da seqüência de ativação
  1. Compute os argumentos e os armazene nas posições corretas no novo registro de ativação do procedimento
  2. Armazene na pilha o valor de fp (corresponde ao endereço do registro anterior)
  3. Faça fp apontar para o novo registro (se houver sp, faça fp=sp)
  4. Armazene o endereço de retorno no novo registro
  5. Salte para o código do procedimento ativado

- Passos básicos da seqüência de retorno
  1. Copie fp no sp
  2. Carregue a vinculação de controle no fp
  3. Efetue um salto para o endereço de retorno
  4. Altere o sp para retirar da pilha os argumentos

39

Exemplo

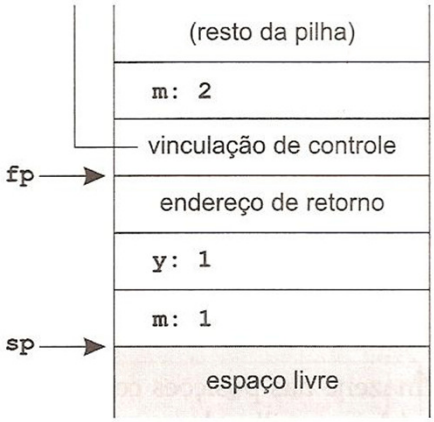
- Considere a pilha no seguinte estado



40

Exemplo

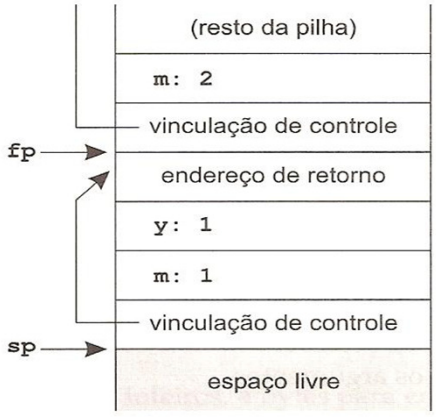
- Quando g é ativado, o parâmetro m é empilhado



41

Exemplo

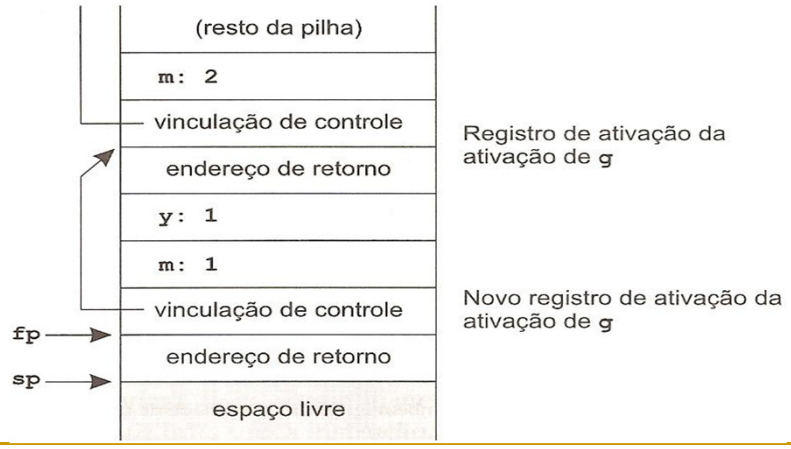
- O valor de fp (que corresponde à vinculação de controle) é empilhado



42

Exemplo

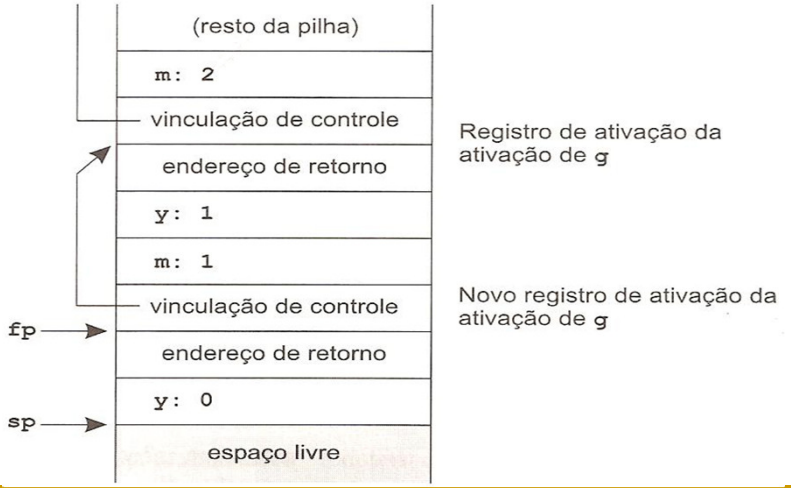
- fp aponta para o novo registro (fp=sp), o endereço de retorno é empilhado (e salta-se para o código de g)



43

Exemplo

- Ao executar o código de g, y é empilhado com seu novo valor



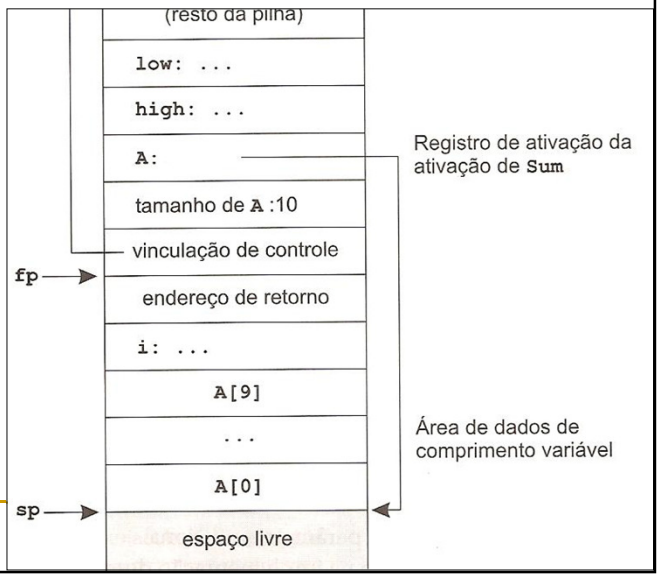
44

Ambientes baseados em pilhas sem procedimentos locais

- Às vezes, dados de tamanho variáveis precisam ser acomodados na pilha
  - Número de argumentos variáveis
    - Exemplo: printf em C (número de argumentos depende da cadeia de formato)
      - Solução: uma posição a mais na pilha indicando o número de argumentos
  - Arranjos de tamanho variável
    - Exemplo: arranjos em Ada (que podem ser definidos em tempo de execução)
      - Solução: reservar uma parte da pilha para o espaço extra e fazer a variável do tipo arranjo apontar para essa área

Ambientes baseados em pilhas sem procedimentos locais

- Possível exemplo de arranjo em Ada

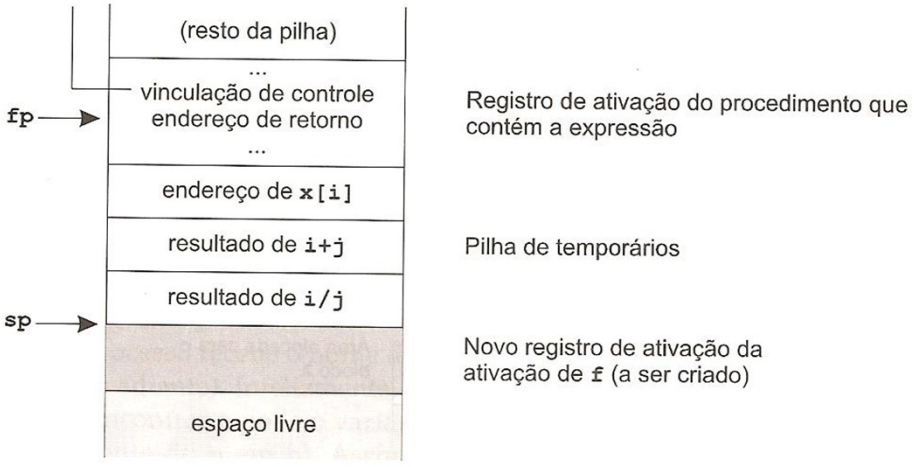


Ambientes baseados em pilhas sem procedimentos locais

- Atenção com temporários locais
  - Por exemplo: x[i] := (i+j) * (i/k + f(j))
    - Os valores de x[i], (i+j) e (i/k) devem ficar "pendurados" até f(j) retornar seu valor
      - Esses dados poderiam ser gravados em registradores ou na pilha como temporários

Ambientes baseados em pilhas sem procedimentos locais

- Atenção com temporários locais



Ambientes baseados em pilhas sem procedimentos locais

- Atenção com declarações aninhadas

```
void p( int x, double y)
{  char a;
   int  i;
   ...
A:{  double x;
     int j;
     ...
   }
   ...
B:{  char * a;
     int k;
     ...
   }
   ...
}
```

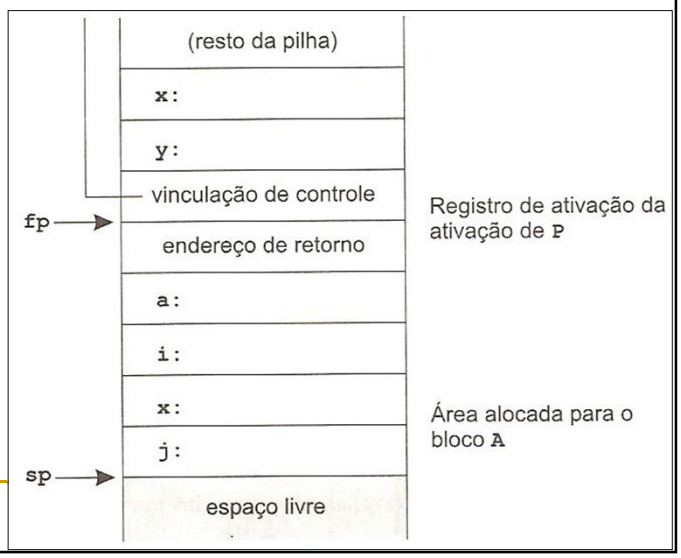
Solução: tratar os blocos internos como dados temporários que desaparecem no final do bloco

Alternativamente, os blocos poderiam ser tratados como procedimentos, mas não vale a pena, pois os blocos têm comportamento mais simples do que os procedimentos

49

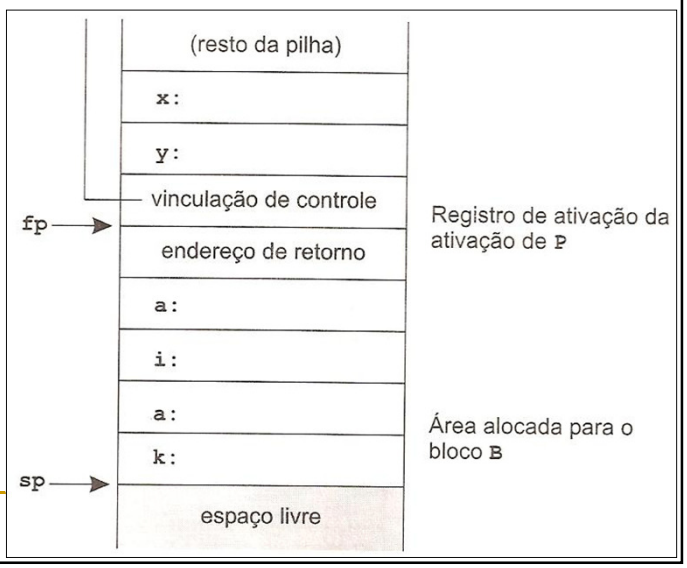
Ambientes baseados em pilhas sem procedimentos locais

- Com o processamento do bloco A



Ambientes baseados em pilhas sem procedimentos locais

- Com o processamento do bloco B



Ambientes baseados em pilhas

- Ambientes com níveis crescentes de complexidade
  - Sem procedimentos locais
  - Com procedimentos locais
  - Com procedimentos como parâmetros

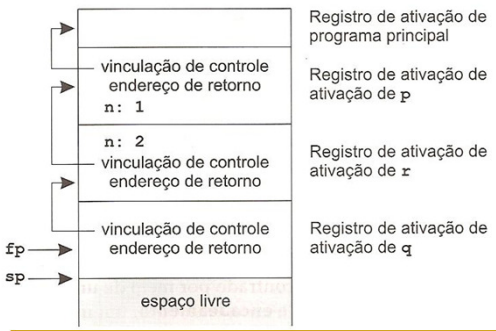
Ambientes baseados em pilhas com procedimentos locais

- Se procedimentos locais permitidos, modelo anterior de ambiente de execução não basta
  - Não foram previstas referências não locais e não globais
    - No programa em PASCAL ao lado
      - main ativa p
      - p ativa r
      - r ativa q
        - o n que q referencia não é global, nem local
    - No ambiente de execução visto até agora, n não seria encontrado

```
program nonLocalRef;

procedure p;
var n: integer;

  procedure q;
  begin
    (* uma referência a n é agora
       não local e não global *)
  end; (* q *)

  procedure r(n: integer);
  begin
    q;
  end; (* r *)

begin (* p *)
  n := 1;
  r (2);
end; (* p *)

begin (* main *)
  p;
end.
```

Ambientes baseados em pilhas com procedimentos locais

- Ambiente de execução após ativação de q
  - n não está onde a vinculação de controle aponta




```
program nonLocalRef;

procedure p;
var n: integer;

  procedure q;
  begin
    (* uma referência a n é agora
       não local e não global *)
  end; (* q *)

  procedure r(n: integer);
  begin
    q;
  end; (* r *)

begin (* p *)
  n := 1;
  r (2);
end; (* p *)

begin (* main *)
  p;
end.
```

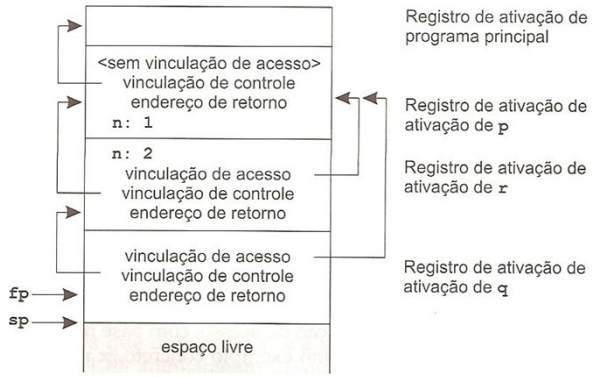
Ambientes baseados em pilhas com procedimentos locais

- Soluções
  - Permitir que se navegue pelas vinculações de controle de vários registros de ativação até que se encontre a referência procurada
    - Processo chamado "encadeamento"
      - Controle e acesso complicados e típicos de ambientes dinâmicos
  - Adicionar mais um tipo de vinculação
    - Vinculação de acesso, que representa o ambiente de definição do procedimento em questão
      - Também chamada "vinculação estática", apesar de não ocorrer em tempo de compilação

55

Ambientes baseados em pilhas com procedimentos locais

- Exemplo
  - Vinculação de q e r apontam para p
  - p não define sua vinculação de acesso (pois não tem)



56

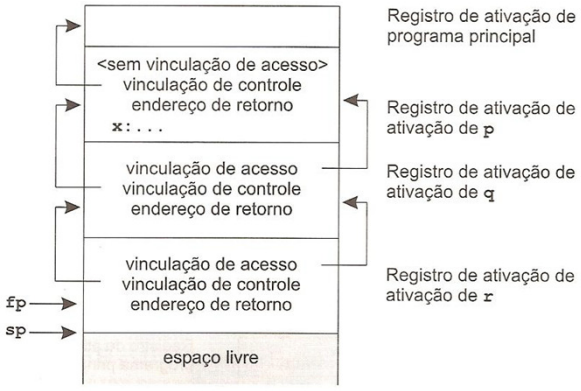
Ambientes baseados em pilhas com procedimentos locais

- Um caso um pouco mais complicado
  - r declarado dentro de q, que é declarado dentro de p
  - r acessa x, que não está em sua vinculação de acesso
    - Novamente, é necessário que se navegue pelas vinculações de acesso (“encadeamento de acesso”)

```
program chain;

procedure p;
var x: integer;

  procedure q;
    procedure r;
    begin
      x := 2;
      ...
      if ... then p;
    end; (* r *)
  begin
    r;
  end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Ambientes baseados em pilhas com procedimentos locais

- Ambiente de execução após ativação de r
  - Quantos acessos são necessários até x?




```
program chain;

procedure p;
var x: integer;

  procedure q;
    procedure r;
    begin
      x := 2;
      ...
      if ... then p;
    end; (* r *)
  begin
    r;
  end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Ambientes baseados em pilhas com procedimentos locais

- Possível solução
  - Guardar o nível de aninhamento de cada identificador, começando-se por 0 e incrementando sempre que o nível muda (decrementa-se quando se volta)
    - p=0, x=1 (o nível é incrementado quando se inicia p), q=1, r=2, dentro de r=3
  - Número de acessos necessários é igual ao nível de aninhamento do registro atual menos o nível de aninhamento do identificador procurado
    - no caso anterior, estando em r e acessando x, número de acessos = 3-1 = 2 acessos

```
program chain;

procedure p;
var x: integer;

  procedure q;
    procedure r;
    begin
      x := 2;
      ...
      if ... then p;
    end; (* r *)
  begin
    r;
  end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Ambientes baseados em pilhas com procedimentos locais

- O cenário não é tão ruim assim
  - Na prática, os níveis de aninhamento raramente são maiores do que 2 ou 3
  - A maioria das referências não locais são para variáveis globais, que podem ser acessadas diretamente na área global
    - Ou seja, o encadeamento de acesso não é tão ineficiente quanto se espera

60

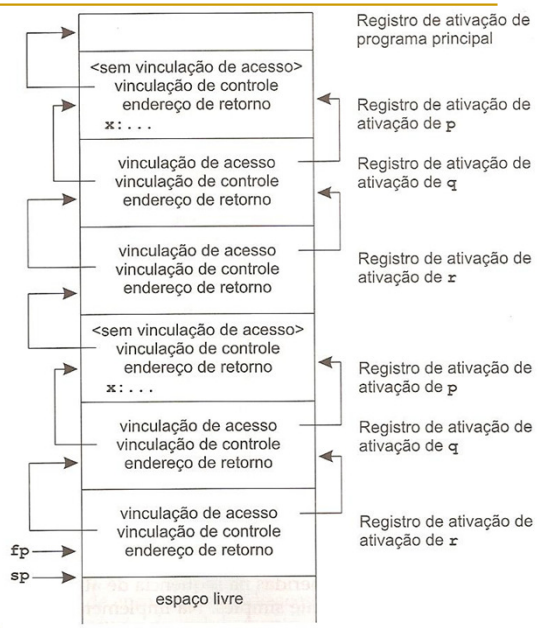
Exercício

- Monte a pilha para a situação após a segunda ativação de r
  - Há conflitos nas vinculações?

```
program chain;

procedure p;
var x: integer;

  procedure q;
    procedure r;
    begin
      x := 2;
      ...
      if ... then p;
    end; (* r *)
  begin
    r;
  end; (* q *)

begin
  q;
end; (* p *)

begin (* main *)
  p;
end.
```

Exercício

- Monte a pilha para a situação após a segunda ativação de r
  - Há conflitos nas vinculações?
    - Não



62

Ambientes baseados em pilhas com procedimentos locais

- Exercício para casa
  - Monte a seqüência de ativação para este novo modelo de ambiente com vinculações de acesso

63

Ambientes baseados em pilhas

- Ambientes com níveis crescentes de complexidade
  - Sem procedimentos locais
  - Com procedimentos locais
  - Com procedimentos como parâmetros

64

Ambientes baseados em pilhas com procedimentos como parâmetros

- Em algumas linguagens, além de procedimentos locais, são permitidos procedimentos como parâmetros
  - É um problema?

```
program closureEx(output);

procedure p(procedure a);
begin
   a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
         writeln(x);
      end;

begin
   x := 2;
   p(r);
end; (* q *)

begin (* main *)
   q;
end.
```

Ambientes baseados em pilhas com procedimentos como parâmetros

- Em algumas linguagens, além de procedimentos locais, são permitidos procedimentos como parâmetros
  - No modelo de ambiente anterior, o ambiente de definição do procedimento (sua vinculação de acesso) seria perdido
  - No exemplo ao lado, o procedimento r é passado como parâmetro para p; r casa com a, que, quando ativado, não sabe quem é seu ambiente de definição

```
program closureEx(output);

procedure p(procedure a);
begin
   a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
         writeln(x);
      end;

begin
   x := 2;
   p(r);
end; (* q *)

begin (* main *)
   q;
end.
```

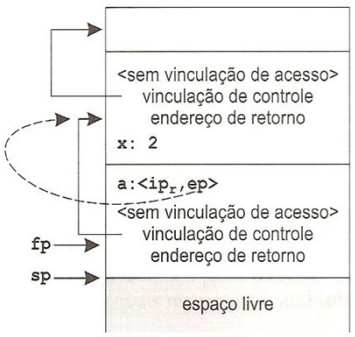
Ambientes baseados em pilhas com procedimentos como parâmetros

- Possível solução
  - Usar dois novos ponteiros
    - Ponteiro de instrução ip para código do procedimento
    - Ponteiro de vinculação de acesso ep (ou ponteiro de ambiente)
      - O par de ponteiros é comumente denominado “fechamento” do procedimento, pois fornece tudo que é necessário para o entendimento do mesmo

67

Ambientes baseados em pilhas com procedimentos como parâmetros

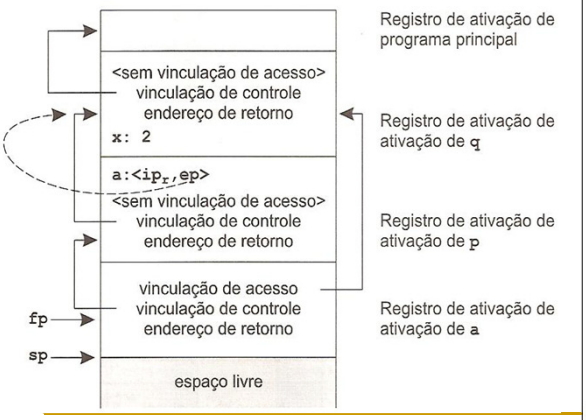
- Exemplo: ambiente após ativação de **p**




```
program closureEx(output);

procedure p(procedure a);
begin
   a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
         writeln(x);
      end;

begin
   x := 2;
   p(r);
end; (* q *)

begin (* main *)
   q;
end.
```

Ambientes baseados em pilhas com procedimentos como parâmetros

- Exemplo: ambiente após ativação de **a**




```
program closureEx(output);

procedure p(procedure a);
begin
   a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
        writeln(x);
      end;

begin
   x := 2;
   p(r);
end; (* q *)

begin (* main *)
   q;
end.
```

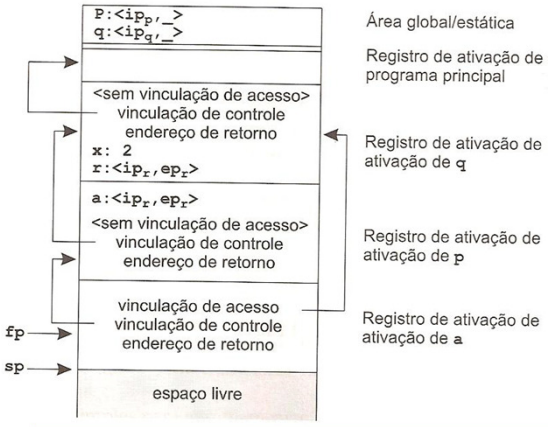
Ambientes baseados em pilhas com procedimentos como parâmetros

- Pontos importantes
  - O ponteiro ip pode não ser conhecido no momento da ativação, precisando ser resolvido indiretamente pelo compilador
  - Por questões de robustez e uniformidade, para algumas linguagens, todos os procedimentos (comuns ou como parâmetros) podem ser representados na pilha via par <ip,ep>

70

Ambientes baseados em pilhas com procedimentos como parâmetros

- Exemplo: ambiente após ativação de **a**




```
program closureEx(output);

procedure p(procedure a);
begin
   a;
end;

procedure q;
var x:integer;

      procedure r;
      begin
         writeln(x);
      end;

begin
   x := 2;
   p(r);
end; (* q *)

begin (* main *)
   q;
end.
```

Ambientes baseados em pilhas com procedimentos como parâmetros

- Como algumas linguagens de programação lidam com a questão
  - C não tem procedimentos locais, embora tenha variáveis e parâmetros que possam ser procedimentos
  - Em Modula-2, somente procedimentos globais podem ser variáveis ou procedimentos
  - Em Ada, não há procedimentos como variáveis ou parâmetros

72

Ambientes de execução

- Ambientes totalmente estáticos
- Ambientes baseados em pilhas
- Ambientes totalmente dinâmicos

Ambientes totalmente dinâmicos

- Limitações de ambientes baseados em pilha
  - Não permite que uma variável local de um procedimento seja retornada e manipulada pelo ativador do procedimento
    - O registro de ativação do procedimento ativado é desempilhado, junto com as variáveis locais
    - O ativador fica com uma posição de memória inválida
      - Exemplo (incorreto em C, mas com paralelos possíveis em LISP e ML, por exemplo)

```
int* prog(void) {
    int x;
    ...
    return &x;
}
```

- Solução: ambiente totalmente dinâmico

Ambientes totalmente dinâmicos

- Características de ambientes totalmente dinâmicos
  - Devem permitir manter na memória registros de ativação de procedimentos (ou funções) terminados enquanto for necessário
  - Registros podem ser liberados dinamicamente em tempos arbitrários de execução do programa
    - Coleta de lixo para retirar o que não é mais necessário
  - Funcionamento mais complexo do que ambientes baseados em pilhas, mas com coisas em comum, como a estrutura básica dos registros de ativação
    - São necessárias operações mais genéricas e robustas de inserção e remoção de registros

Ambientes totalmente dinâmicos

- Gerenciamento do *heap*
  - Essencial para ambientes totalmente dinâmicos e importante para ambientes baseados em pilhas também
  - Boa manutenção do espaço de memória livre e em uso
    - Deve-se usar uma boa estrutura de dados, por exemplo, listas circulares de espaços disponíveis
  - Deve-se evitar fragmentação, ou seja, espaços livres contíguos muito pequenos para serem utilizados
    - Pode-se fazer compactação de memória: deixa-se todo o espaço alocado adjacente, sem espaços vazios no meio
  - Pode-se fazer coalescimento (junção) de espaços livres adjacentes
  - Coletas de lixo periódicas para eliminar registros não utilizados mais

Ambientes de execução

- Um ponto importante, independente do tipo do ambiente, é a forma de passagem de parâmetros para procedimentos
  - Quando o argumento se liga ao parâmetro, tem-se o que se chama de "amarração" ou "ligação"
    - Se por valor, pode-se copiar o valor do argumento para o registro de ativação do procedimento
    - Se por referência, pode-se usar ponteiro e fazer referência indireta (como ocorre em C)
      - Alternativamente, pode-se copiar o valor para o registro e devolver o valor atualizado depois (na verdade, esta estratégia se chama passagem por valor-resultado, típico de Ada, e tem implicações importantes para o funcionamento da linguagem)

77

Ambientes de execução

- Para ter na cabeça

Programa	Execução
Definição de procedimento	Ativação de um procedimento
Declaração de um identificador	Ligações de um identificador
Escopo de uma declaração	Tempo de vida de uma ligação

78

Ambientes de execução

- Algumas perguntas a responder para se saber como deve ser o ambiente de execução
  1. Os procedimentos podem ser recursivos?
  2. O que acontece com os valores de identificadores locais quando o controle retorna da ativação de um procedimento?
  3. Os procedimentos podem fazer referências a identificadores não locais?
  4. Como os parâmetros são passados quando um procedimento é chamado?
  5. Procedimentos podem ser passados como parâmetros?
  6. Localizações de memória podem ser alocadas dinamicamente quando um procedimento roda?