

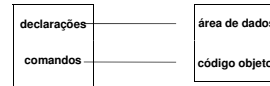
Compiladores

Ambiente de execução

1

Ambientes de Execução

- Relacionar o código fonte com ações a serem executadas em tempo de execução
- Conjunto de rotinas (*run-time support package*) carregado junto com o código objeto gerado
- Composição do código objeto



2

Questões Importantes

- Representações de dados
- Procedimentos e Regras para visibilidade de variáveis
- Alocação de memória para várias classes de armazenamento
- Chamadas de procedimentos:
 - entrada
 - saída
 - retorno

3

Representação de tipos básicos

- Variáveis de tipos simples são representadas por localizações de memória suficientemente grandes para conter cada tipo:
 - char: 1 byte
 - integers: 2 ou 4 bytes
 - floats: 4 a 16 bytes
 - booleans: 1 bit (usualmente 1 byte)

4

Representação de tipos estruturados

- Arranjos:
 - linearizados por linha (row-major)
 - Maioria das linguagens de programação
 - tipo $\text{matriz}[\text{dim}_1, \text{dim}_2, \dots, \text{dim}_n]$
 - O endereço de $\text{matriz}[i_1, i_2, \dots, i_n]$ é:

$$\text{base}(\text{matriz}) + \text{size}(\text{tipo}) \cdot \sum_{k=1, n}^{i_k - \text{dim}_k} \prod_{j=k+1, n} (\text{dim}_j + 1)$$
 - linearizados por coluna (row-major)
 - Fortran

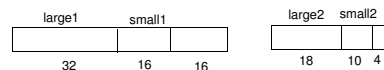
5

Representação de tipos estruturados

```

struct s1 {
    int large1;
    short int small1;
};

struct s2 {
    int large2 : 18;
    short int small2 : 10;
};
    
```



6

Uso de registradores

- Acesso mais rápido que a memória
 - Número limitado pois são caros a nível de hardware
- Metas:
- Alocar as variáveis mais freqüentemente usadas pelo maior tempo possível
 - Acessar variáveis não armazenadas em registradores o mais rápido possível
 - Minimizar o número de registradores usados para gerenciamento do acesso de memória
 - Maximizar a eficiência de chamadas de procedimentos e operações relacionadas

7

Comandos

- Organizados em procedimentos:
 - Declaração com um nome associado que realiza uma dada tarefa
 - Definição:
 - nome
 - variáveis
 - corpo

8

Procedimentos em ação (ativação)

- Fluxo de controle:
 - Execução seqüencial (seqüência de passos)
 - começa no início do corpo
 - Termina no final do corpo
 - Tempo de vida:
 - seqüência de passos executados
 - Chamada de procedimentos:
 - desvio de execução
 - retorna o controle para o ponto imediatamente após o ponto de chamada

9

```
program sort(input, output);
var a : array [0..10] of integer;
procedure readarray;
var i: integer;
begin
    for i:=1 to 9 do read(a[i]);
end;
function partition(y, z: integer): integer;
var i, j, x, v: integer; begin ... end;
procedure quicksort(m, n: integer);
var i: integer;
begin
    if ( n > m ) then begin
        i:= partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    end;
end;
begin
    a[0] := -9999; a[10] := 9999; readarray;
    quicksort(1, 9);
end.
```

Árvores de Ativação

- Cada nodo representa uma ativação de um procedimento
- A raiz representa a ativação do programa principal
- O nodo de 'a' é pai de 'b' se e somente se o fluxo de controle muda de 'a' para 'b'
- O nodo de 'a' está a esquerda de 'b' se e somente se a vida de 'a' ocorre antes de 'b'

11

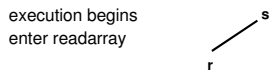
Árvores de ativação

execution begins

s

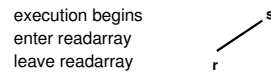
12

Árvores de ativação



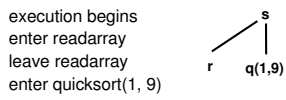
13

Árvores de ativação



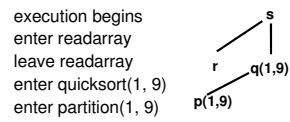
14

Árvores de ativação



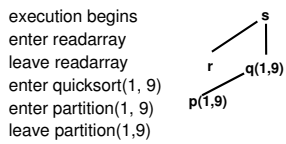
15

Árvores de ativação



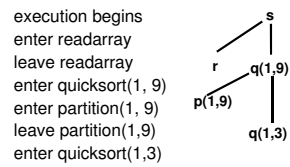
16

Árvores de ativação



17

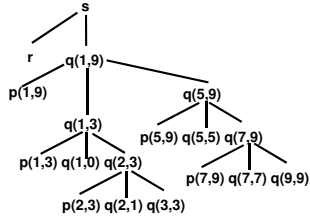
Árvores de ativação



18

Árvores de ativação

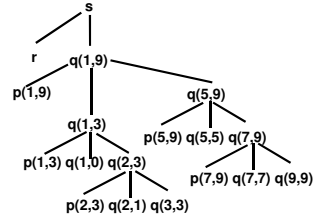
execution begins
 enter readarray
 leave readarray
 enter quicksort(1, 9)
 enter partition(1, 9)
 leave partition(1, 9)
 enter quicksort(1,3)
 ...
 leave quicksort(5, 9)



19

Árvores de ativação

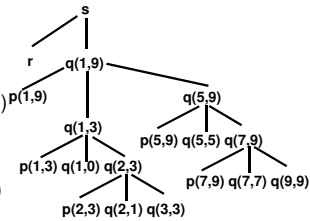
execution begins
 enter readarray
 leave readarray
 enter quicksort(1, 9)
 enter partition(1, 9)
 leave partition(1, 9)
 enter quicksort(1,3)
 ...
 leave quicksort(5, 9)
 leave quicksort(1,9)



20

Árvores de ativação

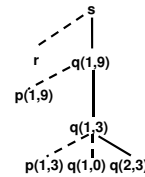
execution begins
 enter readarray
 leave readarray
 enter quicksort(1, 9)
 enter partition(1, 9)
 leave partition(1, 9)
 enter quicksort(1,3)
 ...
 leave quicksort(5, 9)
 leave quicksort(1,9)
 execution terminated



21

Pilhas de controle

- Fluxo de controle corresponde a uma busca em profundidade na árvore de ativação
- Usar uma pilha para controlar as ativações de procedimentos ativos



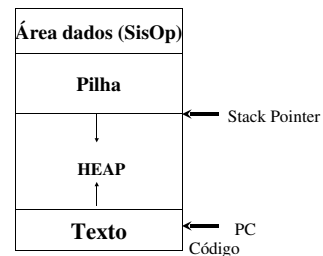
22

Organização de Memória

- Como a memória do programa é armazenada ?
 - Código objeto gerado
 - Espaço para variáveis globais
 - área estática
 - Pilha para ativação de procedimentos
 - Espaço para memória dinâmica (heap)

23

Organização de Memória



24

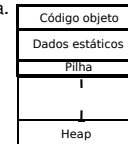
Alocação de memória

- Alocação estática:
 - reserva de memória é feita durante a compilação, de forma estática.
 - Tipo (ou comprimento) do dado é conhecido em tempo de compilação
 - Comprimento não é modificado durante a execução do programa

25

Alocação de memória

- Alocação dinâmica (HEAP):
 - estruturas de dados referenciadas através de ponteiros, as áreas também são reservadas dinamicamente.
 - áreas são alocadas e liberadas, sob o controle do programa
 - alocadas na área de "heap", que cresce no sentido contrário ao da pilha.



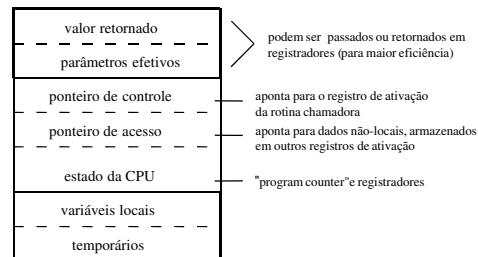
26

Alocação de memória

- Alocação em pilha (STACK):
 - Áreas para dados locais de procedimentos (subrotinas ou funções) devem ser alocadas dinamicamente.
 - A alocação de espaço de memória somente pode ser realizada em tempo de execução, porque a ordem de chamadas é determinada pela execução do programa.
 - Áreas são alocadas numa estrutura em pilha de ativação de procedimentos.
 - Na pilha entram (e saem) **registros de ativação**.

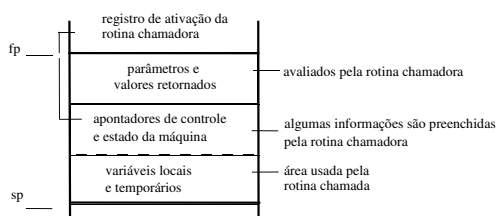
27

Registro de Ativação



28

Ponteiros de Acesso



2 ponteiros:

- frame_pointer (fp): aponta para o registro de ativação correto
- stack_pointer (sp): topo da pilha de registros de ativação

Código de Chamada

- Rotina chamadora:
 - Avalia os parâmetros efetivos e os coloca na pilha
 - Registradores em uso pelo chamador são salvos em memória
 - Armazena o endereço de retorno e o valor antigo do frame_pointer no registro de ativação da rotina chamada e atualiza o valor do frame_pointer;
- Rotina chamada:
 - Salva valores de registradores e outras informações do estado da máquina;
 - Inicializa variáveis locais e começa sua execução.

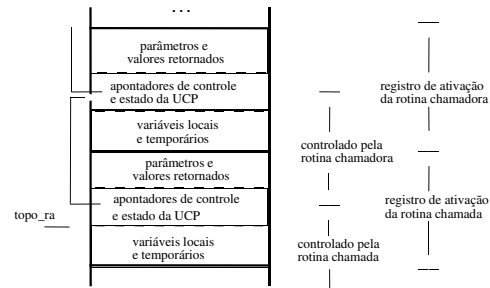
30

Código de Retorno

- Rotina chamada:
 - Armazena o valor de retorno logo após o registro de ativação da rotina chamadora;
 - Restaura o apontador topo_ra e os registradores da máquina e desvia para o endereço de retorno dentro da rotina chamadora;
- Rotina chamadora:
 - Copia o valor retornado no seu próprio registro de ativação

31

Controle



32

Escopo de declarações

- Escopo: área de atuação de uma variável
- Níveis de escopo:
 - Diferentes lugares onde variáveis são definidas
 - Escopos aninhados: uma linha de código pode pertencer a mais de um escopo
 - Escopo corrente: o mais interno
 - Escopos abertos e fechados

33

Funções associadas a nomes (bindings)

- Ambiente:
 - Mapear um nome para uma localização de memória
- Estado:
 - Mapear uma posição de memória ao valor armazenado



34

Funções associadas a nomes

Noção estática	Noção dinâmica
definição de um procedimento	ativações de procedimento
declaração de um nome	associação de um nome
escopo de uma declaração	vida de uma associação

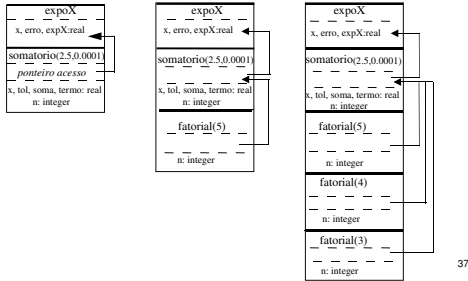
35

Checagem de escopo

- Como fazer ?
 - Pilha de escopos (pilha de tabela de símbolos):
 - Uma entrada para cada escopo
 - Escopo corrente está no topo da pilha
 - Escopo global está no final da pilha
 - Empilhar/desempilhar escopos
 - Busca de nomes: acessar escopos do topo para o início da pilha

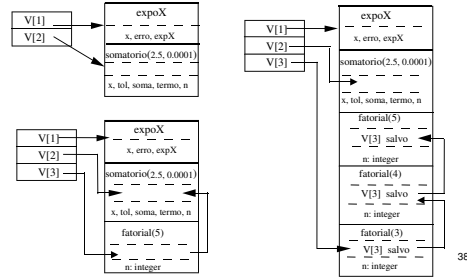
36

Pilha de ativação com ponteiros de acesso



37

Pilhas de ativação com ponteiros de níveis



38

Checgem de escopo

- Como fazer ?
 - Pilha de escopos (pilha de tabela de símbolos):
 - Uma entrada para cada escopo
 - Escopo corrente está no topo da pilha
 - Escopo global está no final da pilha
 - Empilhar/desempilhar escopos
 - Busca de nomes: acessar escopos do topo para o início da pilha
 - Única tabela de escopos com identificador de escopo
 - Usar um contador de escopos como identificador
 - Busca deve varrer toda tabela

39

Passagem de Parâmetros

- Passagem de parâmetros por valor
 - Método mais simples de passagem de parâmetros
 - Parâmetros são avaliados, e seus valores são passados para o procedimento chamado.
 - Implementação:
 - um parâmetro formal é tratado exatamente como um nome local, de maneira que a memória para os parâmetros formais é reservada no registro de ativação do procedimento chamado;
 - o procedimento chamador avalia os parâmetros reais e armazena seus valores na memória reservada para os parâmetros formais.

40

Passagem de Parâmetros

- Passagem de parâmetros por endereço
 - Procedimento chamador passa o endereço de cada parâmetro real.
 - Se um parâmetro real é:
 - Identificador:
 - seu endereço é passado;
 - Expressão:
 - expressão é avaliada num temporário, e o endereço desse temporário é passado;
 - Referências a parâmetros formais, no procedimento chamado, são feitas de forma indireta.

41

Passagem de Parâmetros

- Passagem de parâmetros por cópia-restauração
 - Meio-termo entre passagem por valor e passagem por referência (ADA inout e FORTRAN)
 - Também é conhecido como passagem por valor-resultado (call-by-value-result):
 - Antes da transferência para o procedimento chamado:
 - os parâmetros reais são avaliados.
 - argumentos que correspondem a valores são passados por valor
 - argumentos que correspondem a endereços são determinados.
 - No retorno do procedimento:
 - os valores dos argumentos computados pelo procedimento são copiados nos endereços anteriormente determinados.

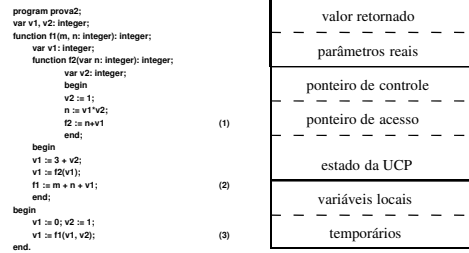
42

Passagem de Parâmetros

- Passagem de parâmetros por nome
 - Tradicional da linguagem Algol 60 (pouco usado)
 - Procedimento chamado é tratado como se fosse uma macro:
 - sua chamada é substituída pelo corpo do procedimento
 - parâmetros formais são literalmente substituídos pelos parâmetros de chamada
 - nomes locais do procedimento chamado são mantidos distintos dos nomes do procedimento chamador (cada nome sistematicamente renomeado antes de ser realizada a expansão da macro)

43

Passagem de Parâmetros



A passagem por parâmetros é feita por valor (default) e por referência (precedendo o parâmetro com var).
 Desenhe o estado da pilha de execução para os 3 pontos indicados, mostrando os valores dentro da pilha logo após executar o comando da linha marcada.

14

Síntese

Os assuntos apresentados trazem respostas às perguntas seguintes:

1. Pode a memória ser alocada dinamicamente ?
2. Pode a memória ser liberada explicitamente ?
 - Heap vs. Pilha
3. Procedimentos podem ser recursivos ?
 - Pilha de ativação
4. O que acontece com valores de nomes quando o controle retorna de uma ativação ?
5. Pode um procedimento referenciar nomes não locais ?
 - Ponteiros de acesso
6. Como os parâmetros são passados quando procedimentos são chamados ?
 - Passagem de parâmetros
7. Podem procedimentos ser chamados como parâmetros ?
8. Podem procedimentos ser retornados como resultados ?

45