Variáveis

Nomes, Vinculações, Checagem de Tipos e Escopo

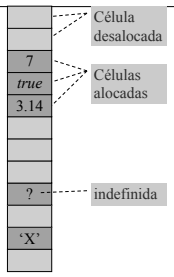
Paradigmas de Programação – BCC/UFRPE
Lucas Albertins – lucas.albertins@deinfo.ufrpe.br

Tópicos

- Introdução
- Nomes
- Variáveis
- O conceito de Vinculação (Binding)
- Escopo
- Tempo de Vida
- Referências
- Constantes

Modelo Abstrato de Memória

- Vamos assumir um modelo abstrato de memória
  - Um **memória** é uma coleção de **células de memória**. Cada célula possui endereço único
  - Cada célula de memória pode estar *alocada* ou *desalocada*
  - Cada célula alocada contém um valor simples or está *indefinida*



3-3

Introdução

- Em LP's estritamente funcionais/lógicas, não há o conceito de alocação de memória ou variável
- Em LP's imperativas, uma **variável** é um repositório para um valor, que pode ser inspecionado e atualizado quando desejado
- Pode ser usada para modelar um objeto do mundo real cujo estado muda ao longo do tempo

Nomes

- Questões de projeto de Nomes
  - Case sensitive?
  - As palavras especiais são palavras-chave ou palavras reservadas

Nomes

- Tamanho
  - Se muito pequeno, dificulta a compreensão
  - Exemplos:
    - FORTRAN 95: máximo de 31
    - C99: sem limite mas só os primeiros 63 são significantes;
    - C#, Ada, e Java: sem limites
    - C++: sem limites, mas compiladores geralmente impõem algum limite

Nomes

- Caracteres Especiais
  - PHP: todos os nomes de variáveis devem começar com o caractere dólar.
  - Perl: todos os nomes de variáveis começam com caracteres especiais que especificam o tipo da variável
  - Ruby: nomes de variáveis que começam com @ são variáveis de instância; e os que começam com @@ são variáveis de classe

Variáveis

- Uma variável é uma abstração de uma célula de memória
- Variáveis podem ser caracterizadas como uma tupla de seis atributos:
  - Nome
  - Endereço
  - Valor
  - Tipo
  - Tempo de Vida
  - Escopo

Atributos de Variáveis

- Nome – nem todas as variáveis tem um
- Endereço – o endereço de memória com o qual ela é associada
  - Uma variável pode ter diferente endereços de memória em diferentes momentos durante a execução
  - Uma variável pode ter endereços diferentes em diferentes lugares em um programa
  - Se dois nomes de variáveis podem ser usados para acessar a mesma célula de memória, ambos são chamados aliases
  - Aliases são criados via ponteiros, referências, etc
  - Aliases dificultam a legibilidade

Atributos de Variáveis

- *Tipo* – determina o intervalo de valores da variável e o conjunto de operações que são definidas para tais valores; no caso de ponto flutuante, o tipo também determina a precisão do tipo
- *Valor* – Os conteúdos do local com o qual a variável está associada
- *Célula de memória abstrata* – pode ser uma célula ou um conjunto de células físicas associadas à variável

O Conceito de Vinculação (Binding)

Uma Vinculação (Binding) é uma associação entre uma entidade e um atributo, como entre uma variável e seu tipo ou valor, or entre uma operação e um símbolo

- *Tempo de Vinculação (Binding time)* é o tempo no qual a vinculação ocorre.

Tempos de Vinculação

- Language design time -- liga símbolos de operadores a operações
- Language implementation time– liga o tipo ponto flutuante para uma representação
- Compile time -- liga uma variável para um tipo em C ou Java
- Load time – liga uma variável estática em C ou C++ para uma célula de memória
- Runtime – liga uma variável local de um método para uma célula de memória

Binding Estático ou Dinâmico

- Um *binding* é *estático* se ele ocorre antes do tempo de execução (runtime) e não se altera ao longo da execução do programa.
- Um *binding* é dinâmico se ele ocorre durante a execução ou pode ser alterado durante a execução do programa

Vinculação de Tipo

- Como o tipo é especificado?
- Quando o *binding* acontece?
- Se estático, o tipo pode ser especificado por uma declaração explícita ou implícita

Declaração Explícita/Implícita

- Uma declaração explícita é uma linha do programa usada para declarar os tipos de variáveis
- Uma declaração implícita é um mecanismo default para especificar tipos de variáveis usando convenções ao inves de declarações
  - Ex. Se uma variável começa com I,J,K, L,M, N são integer
- Fortran, BASIC, Perl, Ruby, JavaScript, e PHP fornecem declarações implícita
  - Vantagem: conviniência
  - Desvantagem: pouca confiabilidade

Vinculação Dinâmica de Tipo

- Vinculação Dinâmica de Tipo (JavaScript, Python, Ruby, PHP, and C#)
- A atribuição determina o tipo. Exemplo: JavaScript

```
list = [2, 4.33, 6, 8];

list = 17.3;
```

  - Vantagem: flexibilidade
  - Desvantagens:
    - Alto custo (checagem e interpretação dinâmicas)
    - Deteção de erros de tipo pelo compilador se torna difícil

Vinculação de Armazenamento e Tempo de Vida

- Alocação: célula de memória a qual uma variável é vinculada de um conjunto de células disponíveis
- Liberação: é colocar uma célula de memória que foi desvinculada de uma variável no conj. de células disponíveis
- Tempo de vida: é tempo que uma variável permanece vinculada a uma posição específica

Vinculação de Armazenamento e Tempo de Vida

- Variáveis estáticas: vinculadas antes do início da execução
  - Vantagens: eficiência, variáveis globais
  - Desvantagens: redução da flexibilidade, não permite recursividade.
- Variáveis dinâmicas da pilha: são aquelas vinculadas quando declaradas, com tipos vinculados estaticamente, em tempo de execução
  - Vantagens: Permite recursividade.
  - Desvantagens: Custo das operações de reserva e libertação implícita de memória. Ex.: C, C++ (variáveis normais)

Vinculação de Armazenamento e Tempo de Vida

- Variáveis dinâmicas da heap explícitas: reservadas e libertadas de memória em tempo de execução por declarações explícitas do programador.
  - Heap é uma coleção de células de armazenamento altamente desorganizada, por causa da imprevisibilidade do seu uso.
- Ex. C++

```
int *intnode;

intnode = new int;

delete intnode;
```

Vinculação de Armazenamento e Tempo de Vida

- Variáveis dinâmicas da heap implícitas: vinculadas apenas quando atribuídos valores
- Ex.: JavaScript

```
Vetor = [74, 84, 30];
```

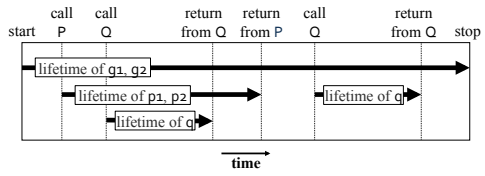
Exemplo: variáveis globais e locais - Ada *(1)*

```
procedure main is
   g1: Integer; g2: Float;
begin … P; … Q; … end;
procedure P is
   p1: Float; p2: Integer;
begin … Q; … end;
procedure Q is
   q: Integer;
begin … end;
```

3-21

Exemplo: variáveis globais e locais - Ada*(2)*

- Tempo de vida de variáveis globais e locais



- Tempo de vida de variáveis globais e locais é aninhado

3-22

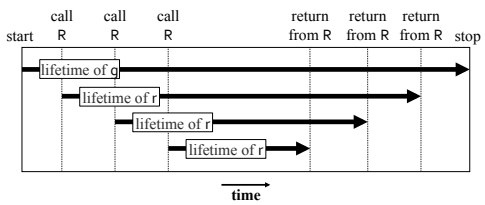
Exemplo: variáveis locais de procedimentos recursivos - Ada *(1)*

```
procedure main is
   g: Integer;
begin
   … R; …
end;
procedure R is
   r: Integer;
begin
   … R; …
end;
```

3-23

Exemplo: variáveis locais de procedimentos recursivos - Ada *(2)*

- Tempo de vida de variáveis globais e locais (assumindo 3 execuções de R):



3-24

Exemplo: variáveis *heap* – Ada *(1)*

```
procedure main is
   type IntNode;
   type IntList is access IntNode;
   type IntNode is record
          elem: Integer;
          succ: IntList;
        end record;
   odds, primes: IntList := null;
   function cons (h: Integer; t: IntList)
        return IntList is
   begin
     return new IntNode'(h, t);
   end;
```

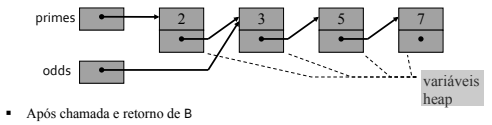

Exemplo: variáveis *heap* – Ada *(2)*

+ *continuação*

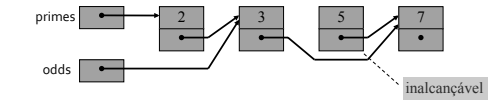
```
      procedure A is
      begin
        odds := cons(3, cons(5, cons(7, null)));
        primes := cons(2, odds);
      end;
      procedure B is
      begin
        odds.succ := odds.succ.succ;
      end;
   begin
      … A; … B; …
   end;
```


Exemplo: variáveis *heap* – Ada *(3)*

+ Após chamada e retorno de A

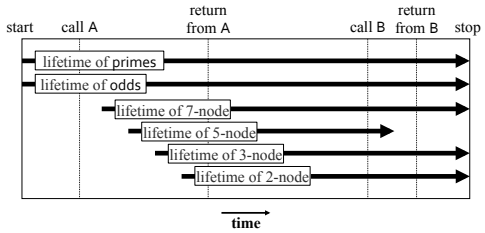


- Após chamada e retorno de B



Exemplo: variáveis *heap* – Ada*(4)*

+ Tempo de vida de variáveis globais e heap



- Tempo de vida de variáveis *heap* não segue padrão

Escopo

+ O escopo de uma variável é o intervalo de comandos que ela é visível.
+ As variáveis locais de uma unidade de programa são aquelas declaradas dentro da referida unidade
+ Variáveis não-locais de uma unidade de programa são aquelas que são visíveis dentro da unidade mas não são declaradas nela
+ *Variáveis Globais* são uma categoria especial de variáveis não-locais
+ As regras de escopo de uma linguagem determinam como referências a nomes são associadas com variáveis

Escopo Estático

+ Baseado no texto do programa
+ Para conectar uma referencia a uma variável, você (ou o compilador) deve achar a declaração da variável
+ *Processo de Busca*: busca declarações, primeiro locais, depois ao escopos mais próximos até que uma declaração seja encontrada para o nome correspondente
+ Algumas linguagens permitem definições de subprogramas aninhados, os quais criam escopos estáticos aninhados (e.g., Ada, JavaScript, Common LISP, Scheme, Fortran 2003+, F#, e Python)

Blocos

- Um método para criar escopos estáticos dentro de unidades de programa
- Exemplo em C:

```
void sub() {
  int count;
  while (...) {
      int count;
    count++;
    ...
  }
  …
}
```

- Nota: permitido em C e C++, mas não em Java e C#

Escopo Estático

```
int z=1,k=2;

func1 (){

z=z*2;

}

func2 (){

int z=10;

k=k+2;

}
```

```
void main (){

printf("z=%d \t k=%d",z,k);

func1();

func2();

printf("\nz=%d \t k=
%d",z,k);

}
```

Qual o valor das variáveis z e k?

Ordem de Declaração

- Em C

```
void func2 (){

printf("%d",f);

int z=10;

int f=0;

k=k+2;

}
```

O Construtor LET

- A maioria das linguagens funcionais incluem alguma forma do construtor `let`
- Um construtor `let` tem duas partes:
  - A primeira liga nomes a valores
  - A segunda usa os nomes definidos na primeira
- In ML:

```
let
  val name1 = expression1
  …
  val namen = expressionn
in
 expression
end;
```

Escopo Global

- C, C++, PHP, e Python suportam uma estrutura do programa que consiste de uma sequencia de definições de funções em um arquivo
  - Estas linguagens permitem que declarações de variáveis apareçam fora das definições das funções
- C e C++ têm ambos declarações e definições
  - Uma declaração fora de uma função especifica que ela é definida em um outro arquivo
- Python
  - Uma variável global pode ser referenciada em funções, mas pode ser atribuída dentro de uma função só se ela foi declarada como global dentro da função.

Escopo Global

- Em Python

```
>>> dia="segunda"

>>> def tester():

print "Global dia é: ", dia

dia="terça";

>>> tester()
```

- UnboundLocalError
- Solução: global dia (dentro da função)

Avaliação de Escopo Estático

- Problemas:
  - Na maioria dos casos, muito acesso é possível
  - As variáveis globais são visíveis a todos os procedimentos/funções/métodos;
  - Dificulta futuras modificações ao programa;
  - Usa-se variáveis de escopo global mais que o necessário

Escopo Dinâmico

- Baseado na sequência de chamadas de subprogramas, não no seu relacionamento espacial
- Referências a variáveis são conectadas a declarações pela busca através da cadeia de chamadas a subprogramas que forçaram a execução naquele ponto.

Escopo Dinâmico

- Exemplo Java Script:

```
function big() {
        function sub1() {
                var x = 7;
                sub2();
        }
        function sub2() {
                var y = x;
        }
        var x = 3;
        sub1();
}
```

- Escopo Estático
  - Referência a `x` em `sub2` é para `x` de `big`
- Escopo Dinâmico
  - Referência a `x` em `sub2` é para `x` de `sub1`

Escopo e Tempo de Vida

- Escopo e Tempo de Vida são geralmente relacionados, mas são conceitos diferentes
- Ex1: Qual é o escopo e o tempo de vida de uma varíavel declarada dentro de um método de uma classe Java?
- Ex2: Qual o escopo e tempo de vida da variável x no código C abaixo?

```
void sub1() {
        ...
}
void sub2() {
        int x;
        ...
        sub1();
}
```

Ambiente de Referenciamento

- É a coleção de todas as variáveis visíveis na sentença
- No escopo estático: variáveis local mais todas as variáveis de seus escopos ascendentes visíveis.
- No escopo dinâmico: variáveis local mais todas as variáveis visíveis em todos subprogramas ativos
- Um subprograma está ativo se sua execução começou mas ainda não terminou

Ambiente de Referenciamento

```
void sub1(){
        int a, b;
        ... (1)
}
void sub2(){
        int b, c;
        ...(2)
        sub1();
}
void main(){
        int c, d;
        ... (3)
        sub2();
}
```

- Ponto
- 1 – a, b de sub1, c de sub2, d de main()
- 2 – b e c de sub2, d de main
- 3 – c e d de main

Constantes

- Uma constante é uma variável vinculada a um valor apenas uma vez.
- O valor das constantes não pode ser alterado.
- Ajudam na legibilidade e confiabilidade
- Em java: **final int** len=100;
- Em C: **const int** len = 100;

Constantes

- O que acontece com o seguinte código em C?

```
const int z=1,k=2;
void func1 (){
      z=z*2;
}
void main (){
      func1();
      printf("\nz=%d \t k=%d",z, k);
}
```

Leitura Adicional

- Capítulo 5 – Nomes, vinculações e escopos. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.
- Próxima aula
  - Capítulo 6 – Tipos de Dados. SEBESTA, R. W. Conceitos de Linguagens de Programação. 9ª ED. BOOKMAN, 2011.