Armazenamento

Prof. Hugo de Paula

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Departamento de Ciência da Computação

Sumário

Variáveis

- **Variável**: é uma abstração de uma célula de memória.
  - Pode ser inspecionada (semântica de valor) ou atualizada (semântica de referência).
  - é definida pelos seus atributos (nome, localização, valor, tipo de dados, tempo de vida e escopo)
  - variáveis em computação $\neq$ variáveis em matemática
  - *Matematicamente*, não faz sentido uma sentença $x = x + 1$
  - *Computacionalmente* $x = x + 1$ possui a semântica: $ref(x) = val(x) + 1$

Atributos das variáveis

- **Nome**: nem todas as variáveis possuem nome.
- **Endereço (localização)**:
  - pode alterar durante a execução ou dependendo da localização no programa.
  - se dois nomes podem ser usados para acessar a mesma localização na memória, então essas variáveis são chamadas de *aliases*.
  - *Alias* é criado via ponteiro ou referência, e prejudica a legibilidade.
- **Tipo**: determina conjunto de valores, operações, memória e precisão.
- **Valor**: conteúdo associado à variável.
  - *l-value*: é o endereço.
  - *r-value*: é o valor.

Variável de tipo composto

- **Variável de tipo composto**: possui componentes que podem ser inspecionados e atualizados seletivamente ou em conjunto

Exemplo em C++

```
struct {
   int a; int b;
} p, q;
q.a = 10;
q.b = 20;    // atualizacao seletiva
p = q;       // atualizacao total
```

- Em C++, não é possível atualização total de vetores (tratados como ponteiros)

Associação ou *binding*

Associação ou *binding*

binding é uma associação entre uma entidade e um atributo. Tal como a variável e seu tipo ou valor.

- **Tempo de associação** é o instante em que ocorre o *binding*.

Tipos de tempo de associação

- **projeto da linguagem**: associa operadores a seus símbolos.
- **implementação da linguagem**: associa o tipo **float** à representação de ponto flutuante.
- **compilação**: associa uma variável a um tipo em C ou Java.
- **carga**: associa variáveis estáticas a uma célula de memória.
- **execução**: associa variáveis não estáticas a uma célula de memória.

Associação estática e dinâmica

- **associação estática** ocorre antes da execução e permanece imutável durante a execução.
- **associação dinâmica** ocorre durante a execução (*runtime*) e pode se alterar durante a execução.

Type binding

- **Associação estática de tipos**: declaração explícita ou implícita.
  - **Declaração explícita**: uma instrução do programa declara o tipo da variável. Ex.: Java e C.
  - **Declaração implícita**: mecanismo padrão para definir tipos de variáveis através de convenções. Ex.: Basic, Ruby, JavaScript, PHP.
- **Inferência de tipos**: determina o tipo a partir do contexto. Ex.: C#, pela instrução `var`, Haskell, Visual Basic 9.0+.
- **Associação dinâmica de tipos**: tipo especificado pela instrução de atribuição. Ex.: JS, Python, Ruby, PHP.

Tempo de vida

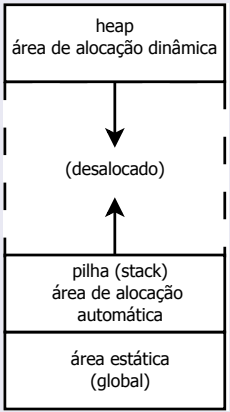
- **Tempo de vida** de uma variável: é o tempo entre a criação e a destruição da variável.
- Classes de armazenamento:
  - **Variável estática**: associada à memória antes da execução e permanece inalterada durante a execução.
  - **Variável dinâmica de pilha**: associação de armazenamento criada no momento em que são declaradas.
  - **Variável dinâmica de *Heap* explícita**: instruções específicas de alocação e desalocação (ex: `new` e `delete`).
  - **Variável dinâmica de *Heap* implícita**: alocação e desalocação causada por instruções de atribuição. Ex.: JavaScript e PHP.

Tempo de vida

- Alocação e desalocação de *heap* e manipulação explícita de ponteiros são consideradas operações inseguras (*unsafe*)
- Algumas linguagens como o Java exigem que a desalocação do *heap* seja feita de forma automática: coleta de lixo

Ambiente de execução

Estrutura típica de um ambiente de execução



Declarações, blocos e escopo

- **Bloco** sequência de declarações seguida de uma sequência de instruções
  - Ling. estruturadas em blocos permitem aninhamento de blocos e redeclaração de nomes no interior desses blocos.
  - **Declarações locais** são associadas com algum bloco.
  - **Declarações não locais** são associadas com blocos relacionados.
  - **Variáveis globais** são tipo particular de variável não local.

Exemplo em C++

```
class Shulambs { // bloco de Shulambs
   int i; //declaracao nao local de i para o metodo "f"

   void f() {  // bloco aninhado no bloco de Shulambs
      int i;   // declaracao local de i em "f"
   }
}
```

Alocação, tempo de vida e o ambiente

- A memória para as variáveis locais declaradas dentro de uma função não serão alocadas até que a função seja chamada.
- **Ativação**: chamada a uma função.
- **Registro de ativação**: a região de memória alocada para aquela chamada de função.
- Em uma linguagem baseada em blocos e com escopo léxico, o mesmo nome de variável pode estar associado a diferentes localizações (objetos), mas apenas um destes objetos pode ser acessado de cada vez.

Escopo

- **Escopo**: trecho do programa em que uma declaração tem efeito (visibilidade)
  - **Variável local**: escopo é o bloco em que a variável foi alocada.
  - **Variável global**: escopo é todo o programa.
- **Escopo léxico**: em linguagens baseadas em blocos, o escopo é limitado ao bloco na qual a declaração aparece (e outros blocos internos a ele)
- **Regra da "Declaração antes do uso"**: em linguagens como o C, o escopo de uma declaração se inicia no ponto da declaração até o final do bloco em que a declaração está localizada

Escopo de bloco

Exemplo em Pascal

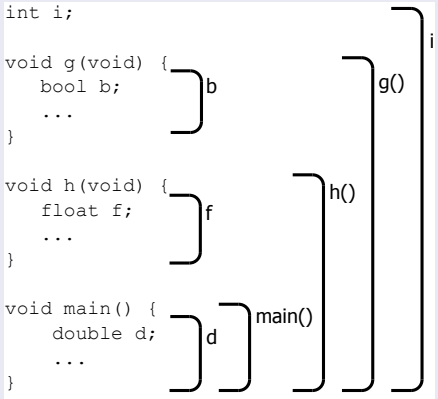
```
program ESCOPO;

var
    global : integer;

procedure bloco()
var
    local : integer;
begin
    ....
end;

begin
    ....
end.
```

global

local

Regra da “Declaração antes do uso”

Exemplo em C



Tipos de escopo

- Dado um identificador, a que ele está associado? Pode ser:
  - *Binding* estático (escopo estático ou escopo léxico)
  - *Binding* dinâmico (escopo dinâmico)

Exemplo em C

```
1  const int s = 2;
2  int scale(int x) {
3     return x * s; // a qual declaracao s esta associado ?
4  }
5  int triplica(int x) {
6    const int s = 3;
7    return scale(x);
8  }
9  int main ()  {
10   cout << scale(10) << " e " << triplica(10);
11 }
```

Escopo estático

- Identificador é associado à declaração realizada no bloco mais próximo do ponto de uso do identificador
  - Blocos são pesquisados “de dentro para fora”
  - *Binding* é determinado em tempo de compilação
- Assim, exemplo anterior irá imprimir: 20 e 20
- Usado pela grande maioria das linguagens: Algol, Pascal, C, C++, Ada, Java etc
- Vantagem:
  - Programas mais legíveis e de mais fácil entendimento

Escopo dinâmico

- Identificador é associado à declaração mais próxima na pilha de chamada de subprogramas
  - *Binding* é determinado em tempo de execução
- Assim, exemplo anterior irá imprimir: 20 e 30
- Usado por linguagens como: LISP (primeiras versões), APL, SNOBOL, Smalltalk, etc
  - Em geral, linguagens com verificação dinâmica de tipos
- Desvantagens:
  - Dificulta leitura e entendimento dos programas (*binding* varia com a sequência de chamadas de funções)
  - Execução mais lenta

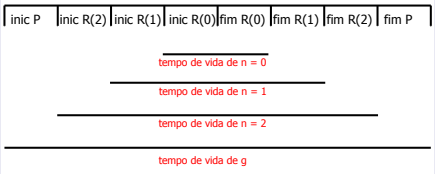
Tempo de Vida e Escopo

Exemplo em Pascal

```
program ESCOPO
var
   g: integer;
procedure R(n : integer);
begin
  if n > 0 then R(n-1)
end;

begin
    R(2)
end.
```

escopo de g

escopo de n



Variáveis estáticas

- Variável com escopo local e tempo de vida global

Exemplo em C++

```
int a = 1;

void f() {
   int b = 1;          // inicializado a cada chamada de f
   static int c = a;   // inicializado somente uma vez

   cout << "a = " << a++ << "b = " << b++;
   cout << "c = " << c++ << endl;
   c = c + 2
}

int main() {
   while (a < 4)  f();
}
```

Alias

- **Alias** ocorre quando o mesmo objeto é associado a dois nomes diferentes ao mesmo tempo
- Pode surgir durante a chamada de uma função ou procedimento através do uso de variáveis ponteiro ou referência

Exemplo em C

```
int *a, *b;
a = (int *) malloc(sizeof(int));
*a = 10;

b = a;                  // a e b sao alias
*b = 20;

printf("\%d\n", *a); // ira imprimir 20
```

Alias

- Alias pode causar efeitos colaterais indesejáveis
- **Efeito colateral** é qualquer mudança no valor de uma variável que persiste após a execução de uma instrução (altera o estado global do sistema)
- Atribuições a partir de alias são potencialmente danosas por serem difíceis de controlar
  - Não podem ser determinadas diretamente a partir do código fonte
  - não se sabe claramente para onde o ponteiro está apontando em um determinado momento.

Referência *Dangling*

- Ponteiro que aponta para uma área de memória que foi liberada
  - Pode surgir quando o endereço de uma variável local é atribuído a uma variável com tempo de vida mais longo

Exemplo em C++

```
int *r;

int f() {
    int v;
    r= &v;
}
int main() {
    f();
    *r = 1; // r e uma referencia dangling
}
```

Lixo

- **Lixo** é a memória que foi alocada no ambiente mas se torna inacessível ao programa
  - Pode surgir quando um programador se esquece de desalocar uma variável dinâmica antes de alterar o estado do ponteiro que referencia esta região de memória

Exemplo em C de vazamento de memória

```
int *i;

i = (int *) malloc(sizeof(int));
i = 0;
```

- Programa produz lixo, também chamado de vazamento de memória (*memory leak*) e pode esgotar a memória do sistema

Coletores de lixo

- **Coletor de lixo** é um processo que automaticamente elimina o lixo, liberando a memória que não é mais utilizada
- Coletores de lixo eliminam o vazamento de memória
- Coletores de lixo eliminam referências *dangling*
  - eliminam a necessidade de se desalocar memória explicitamente
- Por exemplo: Java
  - não possui ponteiros explícitos (apenas semântica de referência)
  - não possui operadores de desalocação de memória (*free* ou *delete*)
  - possui coletor de lixo que faz a gestão da desalocação de memória automaticamente