

Linguagens de Programação

Nomes, Escopos e Vinculações (Bindings)

Carlos Bazilio

carlosbazilio@id.uff.br

<http://www.ic.uff.br/~bazilio/cursos/pp>

Nomes

- Forma mais simples de **abstração**
- Um nome é um caracter **mnemônico** que representa alguma coisa:

Mnemónica

Artigo

Discussão

Ler

Editar

Editar código-fonte

[Ver histórico](#)

Pesquisar na Wikipédia



Mnemónica

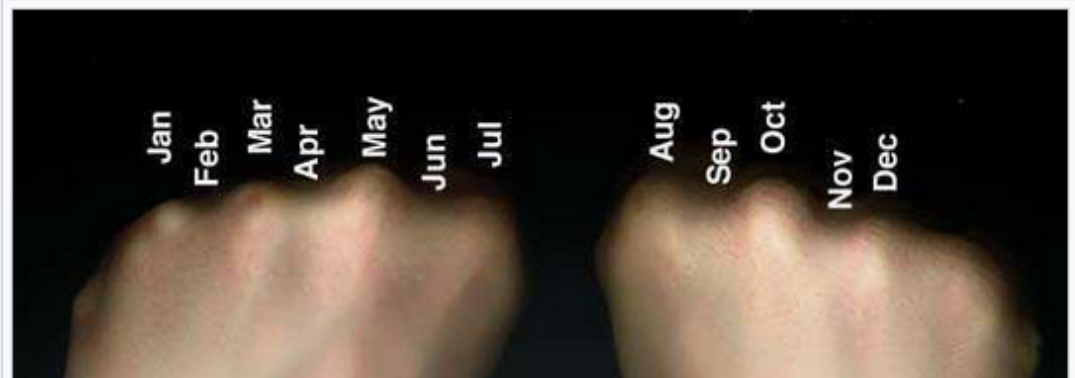
[ocultar]

Origem: Wikipédia, a enciclopédia livre.



Esta página cita **fontes confiáveis**, mas que **não cobrem todo o conteúdo**. Ajude a **inserir referências**. Conteúdo não **verificável** poderá ser **removido**. — *Encontre fontes:* [Google](#) ([notícias](#), [livros](#) e [acadêmico](#)) *(Abril de 2017)*

Uma **mnemónica** ^(pt) ou **mnemônica** ^(pt-BR) é um auxiliar de **memória**. São, tipicamente, verbais, e utilizados para memorizar listas ou fórmulas, e baseiam-se em formas simples de memorizar maiores construções, baseados no princípio de que a **mente humana** tem mais facilidade de memorizar **dados** quando estes são associados a **informação** pessoal, espacial ou de carácter relativamente importante, do



Memorização dos meses do ano que contém 31 dias pelos punhos

Nomes

- Forma mais simples de **abstração**
- Um nome é um caracter **mnemônico** que representa alguma coisa:
 - Um endereço de memória
 - Um valor, uma constante
 - Uma operação, etc
- Se considerarmos o tempo, o período de existência desta associação entre nome e valor é chamado de tempo de vinculação (*binding time*)

Tempo de Vida de um Objeto e Armazenamento

- Eventos que sinalizam o tempo de vida de um objeto:
 - Criação de objetos
 - Criação de associações
 - Referências à variáveis, rotinas, tipos, etc
 - Remoção de associações
 - Destruição de associações
 - Destruição de objetos
- Exemplos fartos para isso são encontrados na manipulação de ponteiros em C/C++

Tempo de Vida de um Objeto e Armazenamento

- O tempo de vida de um objeto geralmente está associado a 3 mecanismos distintos de armazenamento
 - Objetos **estáticos**
 - Objetos armazenados numa **pilha**
 - Objetos armazenados num **heap**

Objetos Estáticos

- Exemplo imediato são **variáveis globais**
- Ou seja, possuem a característica de existirem durante toda a execução de um programa
- **Instruções de um programa** também podem ser consideradas objetos estáticos
- Outros exemplos são:
 - Variáveis estáticas declaradas em rotinas
 - Valores numéricos e strings (constantes)
- **Compiladores** também geram **estruturas auxiliares** (tabelas de símbolos, por exemplo) para auxílio na compilação e execução

Objetos Estáticos

- Em linguagens com **alocação puramente estática**, o compilador previamente **reserva espaço** para ativação de **cada rotina existente**
- Dados comumente armazenados nestes espaços (registros de ativação)
 - Argumentos e valores de retorno
 - Endereço de retorno
 - Variáveis locais, Temporárias, etc
- Com isso, **não é possível haver recursão**, seja esta direta ou indireta

Objetos armazenados numa Pilha

- O empilhamento de **registros de ativação** permite que a recursão possa ocorrer
- Entretanto, naturalmente, devemos abdicar da reserva estática de espaço para rotinas e fazermos **alocação dinâmica** destas
- É de responsabilidade das rotinas chamadora/chamada manter a pilha coerente com estado atual de execução
- A decisão de quem faz o que é questão de implementação da linguagem

Objetos armazenados numa Pilha

- Os registros de ativação são também chamados de *frames*
- Da mesma maneira que temos o registrado PC (*Program Counter*) que indica a próxima instrução a ser executada, temos um outro chamado FP (*Frame Pointer*) que aponta para o **endereço base** do registro de ativação da rotina corrente
- Os acessos aos campos de cada frame podem ser feitos somando incrementos (*offsets*) a este ponteiro

Objetos armazenados numa Pilha

- Outro ponteiro comum neste contexto é o SP (*Stack Pointer*) que sempre aponta para o primeiro endereço livre no topo da pilha (ou o endereço do último frame alocado, dependendo da máquina)

Objetos armazenado num Heap

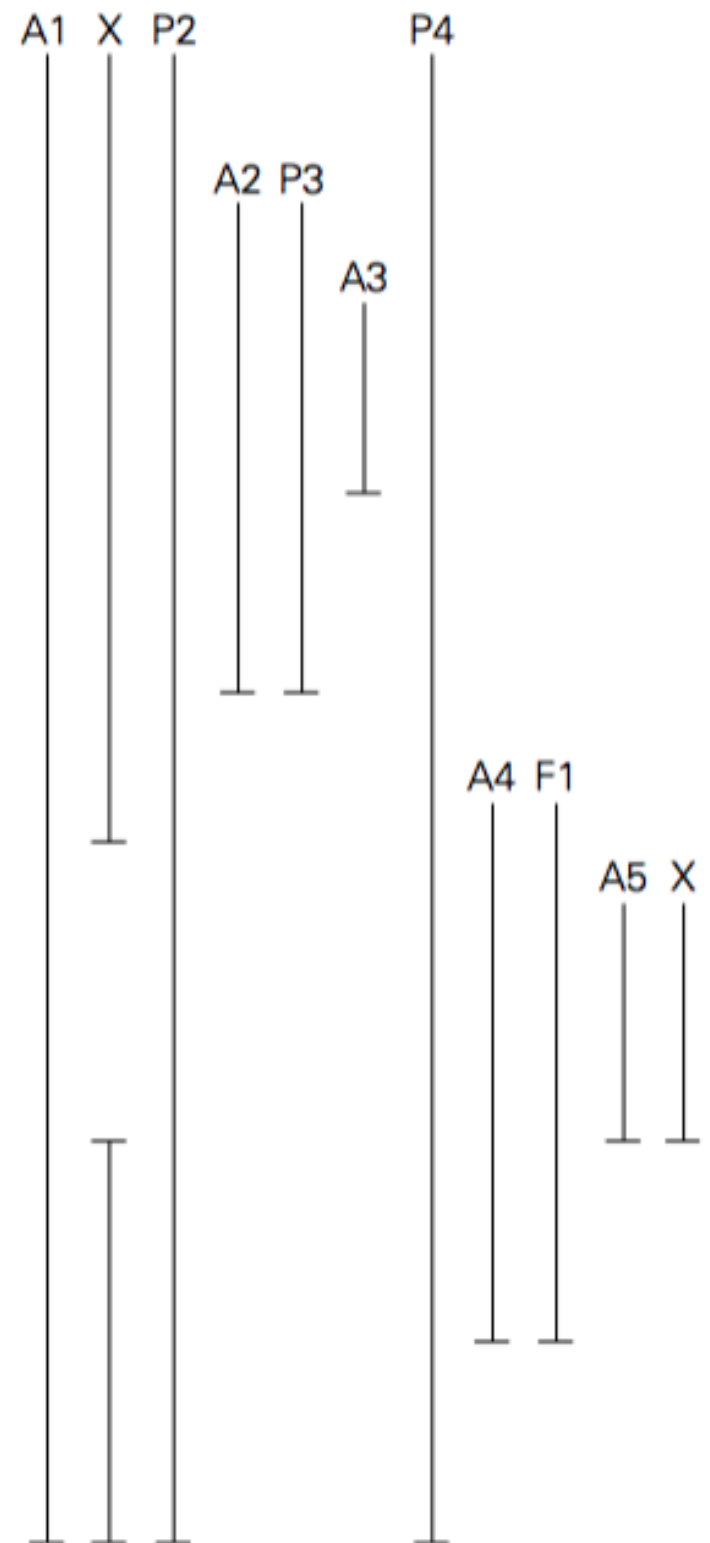
- Heap é uma **área de armazenamento** onde **blocos de memória** podem ser **alocados/desalocados muitas vezes** por um programa de usuário
- **Não há relação** com o nome heap da implementação de fila de prioridades baseada em árvore
- É necessário para **alocação de estruturas de dados dinâmicas** como listas, filas, objetos redimensionáveis dinamicamente, etc

Objetos armazenados num Heap

- Há várias estratégias de manipulação de espaço num heap
- Estas estratégias levam em consideração 2 fatores: **velocidade** e **espaço**
- Dois problemas a serem tratados:
 - Fragmentação Interna
 - Fragmentação Externa
- Algoritmos de alocação no heap, visando amenizar o problema de fragmentação externa:
 - First fit, Next fit, Best fit, Worst fit

Exemplo Escopo em Pascal

```
procedure P1(A1)
  var X      -- local to P1
  ...
  procedure P2(A2)
    ...
    procedure P3(A3)
      ...
      begin
        ...      -- body of P3
      end
    ...
  begin
    ...      -- body of P2
  end
  ...
  procedure P4(A4)
    ...
    function F1(A5)
      var X  -- local to F1
      ...
      begin
        ...      -- body of F1
      end
    ...
  begin
    ...      -- body of P4
  end
  ...
begin
  ...
end
```

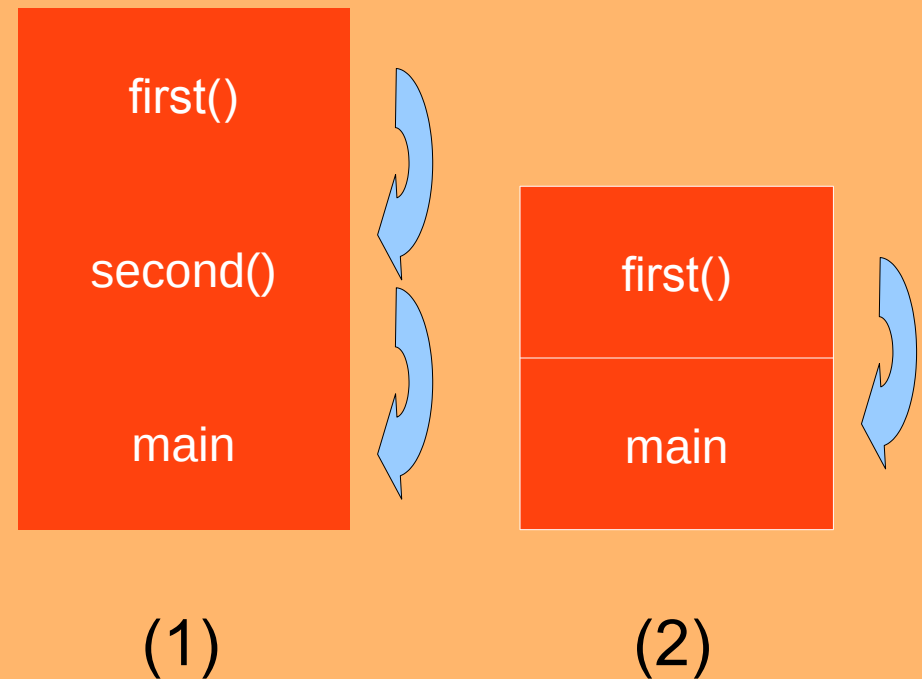


Escopo

- 2 tipos:
 - Estático (Léxico): refere-se à estrutura do programa; ou seja, a visibilidade dos objetos pode ser definida em tempo de compilação
 - Dinâmico: a visibilidade depende da ordem de execução das instruções num programa; o efeito desse escopo é a sobrescrita do escopo global

Escopo

```
int a;  
void first() {  
    a = 1;  
}  
void second() {  
    int a;  
    first();  
}  
main() {  
    int v;  
    a = 2;  
    scanf("%d",&v);  
    if v > 0  
        second(); // (1)  
    else  
        first(); // (2)  
    printf(a);  
}
```



Escopo Perl

- Perl permite tanto tratamento estático quanto dinâmico no acesso à variáveis
- A diferenciação é feita pelos modificadores *my* (estático) e *local* (dinâmico)

```
$x = 0;  
sub f { return $x; }  
sub g { my $x=1; return f(); }  
print g()."\n";
```

```
$x = 0;  
sub f { return $x; }  
sub f2 { my $x=2; return f(); }  
sub g { local $x=1; return f2(); }  
print g()."\n";
```

Escopo

```
type tpessoa = record
    idade : integer
limite : integer
pessoas : collection
```

```
function mais_antigo (p : tpessoa) : boolean
    return p.idade >= limite
```

```
procedure imprime_pessoa (p : tpessoa)
    // Chamar rotinas apropriadas de I/O
    // Fazer uso da variável não local tamanho_linha
```

```
procedure imprime_registros_selecionados
    (bd : collection, predicado, rotina_impressao : procedure)
    tamanho_linha : integer
    if device_type(stdout) = terminal
        tamanho_linha = 80
    else
        tamanho_linha = 132
    foreach record r in bd
        if predicado(r)
            rotina_impressao(r)
```

```
limite := 35
```

```
imprime_registros_selecionados(pessoas, mais_antigo, imprime_pessoa)
```