FACAMP
FACULDADES DE CAMPINAS

Linguagens de Programação

Aula 5

Nomes, vinculações e escopos.

2º semestre de 2019
Prof José Martins Junior

Introdução

- Linguagens de programação imperativas são abstrações da arquitetura de Von Neumann
  - Memória: armazena instruções e dados
  - Processador: opera instruções para modificar o conteúdo da memória
- Abstrações para as células de memória da máquina
  - Variáveis e constantes: armazenamento de dados
  - Funções e procedimentos: instruções armazenadas
- Para facilitar, a localização se dá através de **nomes**

Variáveis

- Agrupamentos de bytes endereçados na memória
- Existem durante a execução de um programa (escopo)
  - Podem serem alteradas e lidas diversas vezes durante sua execução
- Devem ser declaradas
  - Declarado um tipo a um nome (exclusivo no programa/escopo)
  - São compatíveis com o tipo de dados declarado
    - Têm o tamanho adequado ao tipo
    - Geralmente não podem receber dados de tipos diferentes
- Devem ser inicializadas (atribuição de valor inicial)
- São usadas para muitas coisas
  - Receber valor informado pelo usuário (ou por outro programa)
  - Armazenar valores intermediários, durante o processamento
  - Retornar dados para o usuário (ou outro programa)

Variável

- Pode ser caracterizada por 6 atributos:
  - Nome: identificador
  - Endereço: localização da memória a ela associado
  - Tipo: domínio para possíveis valores e operações
  - Valor: o que está armazenado na variável num determinado momento
  - Tempo de vida: tempo durante o qual a memória permanece alocada para a variável
  - Escopo: partes do programa onde a variável é acessível

Variáveis

- Variável é uma abstração de uma célula de memória
- Se a memória for um arquivo de aço
  - Variáveis são as gavetas
  - Nomes são etiquetas colocadas nas gavetas
  - Declaração é o processo de reservar e etiquetar gavetas
  - Tipo indica, entre outras coisas, o tamanho e o uso para a gaveta

Nomes

- Nome é uma cadeia de caracteres usada para identificar alguma entidade (variável ou função) em um programa
- Pode impor certas regras
  - Tamanho máximo
  - Restrição a caracteres especiais (pontos, traços, espaços, números)
    - Geralmente, iniciar com **letra** ou _ (e depois, qualquer dígito)
  - Distinção entre caracteres minúsculos e maiúsculos (**case sensitive**)
  - Palavras especiais/reservadas
- Evite nomes pouco significativos, como letras simples
  - O nome da variável ajuda a lembrar o que ela representa
- Convenção em Java para nomes compostos
  - Começar cada nome com maiúscula (o 1º com minúscula)
  - Exs.: nomeAluno, dataAtual, numeroTotal

Palavras especiais

- Palavra-chave
  - Palavra que é especial apenas em certos contextos
  - Exemplos em FORTRAN

    `Integer Real` (variável de nome Real, de tipo Integer)

    `Real Integer` (variável de nome Integer, de tipo Real)
- Palavra reservada
  - Palavra que não pode ser usada como um nome definido pelo usuário
  - Exemplos em Java

    `for, if, while, else, int, char, double,`

    `public, static, final ...`

Blocos

- Blocos delimitam funções e estruturas de controle
  - Entrada no bloco: alocação de memória para as variáveis locais
  - Saída do bloco: liberação da memória
- Blocos em linguagens de programação
  - C, C++, C#, Java, Obj-C, e outras: `{ ... }`
  - Pascal: `begin … end`
  - Python: `\t`
- Linguagens sem blocos
  - 70's FORTRAN
  - 80's BASIC

Vinculação

- Associação (binding) de um (nome) identificador a sua declaração no programa
- Exemplo

```
const int z = 0;
char c;

int func()
{
  const float c = 3.14;
  bool b;
  ...
}

int main()
{
  ...
}
```

c: constante 3.14
b: variável boleana
z: constante 0
func: função

c: variável char
z: constante 0
func: função

Vinculação

- Vinculações de locação
  - Geralmente ocorrem em tempo de loading (quando o programa gerado pelo compilador é alocado à locações específicas na memória)
  - Mas também podem ocorrer em tempo de execução (variáveis em procedimentos e alocação dinâmica)
- Vinculações de nome
  - Ocorrem em tempo de compilação, perante uma declaração
- Vinculações de valor
  - Ocorrem tipicamente em tempo de execução
- Vinculações de tipo
  - Ocorrem geralmente em tempo de compilação, através de declarações de tipo

Vinculação estática x dinâmica

- Vinculação Estática
  - Ocorre antes da execução e permanece inalterado durante a execução do programa
  - Declaração de tipo
    - Explícita: é uma sentença declarativa para o tipo da variável
      - Exemplo: `int x;`
    - Implícita: assume o tipo do primeiro valor atribuído
- Vinculação Dinâmica
  - Ocorre durante a execução ou pode ser alterado durante a execução do programa
  - Não há declaração de tipo
  - Vinculação ocorre nas atribuições de valores
  - Exemplo: PHP

```
$varA = 10; //variável int
$varA = "teste"; //passa a ser uma string
```

Variáveis estáticas

- Vinculadas a células estáticas (fixas) de memória antes da execução
- Vantagens
  - Eficiência (endereçamento direto)
  - Não há sobrecarga em tempo de execução para alocação e liberação
- Desvantagens
  - Falta de flexibilidade (sem recursão)
  - Armazenamento não compartilhado entre variáveis

Variáveis dinâmicas de pilha

- Vinculações são criadas quando suas sentenças de declaração são efetuadas, mas o tipo é estaticamente vinculado
  - Variáveis criadas dentro de funções
  - Valores assumidos em execuções recursivas
- Vantagem
  - Permitem recursão e otimizam o uso de espaço em memória
- Desvantagem
  - Sobrecarga de alocação e liberação

Variáveis dinâmicas de heap explícitas

- São células de memória não nomeadas (abstratas) alocadas e liberadas por instruções explícitas, durante a execução
  - Variáveis referenciadas através de ponteiros
  - Exemplo em C

```
int *valor;
valor = (int*) malloc(sizeof(int));
//...
free(valor);
```

- Vantagem
  - Armazenamento dinâmico
- Desvantagem
  - Programador é responsável pelo gerenciamento da memória

Variáveis dinâmicas de heap implícitas

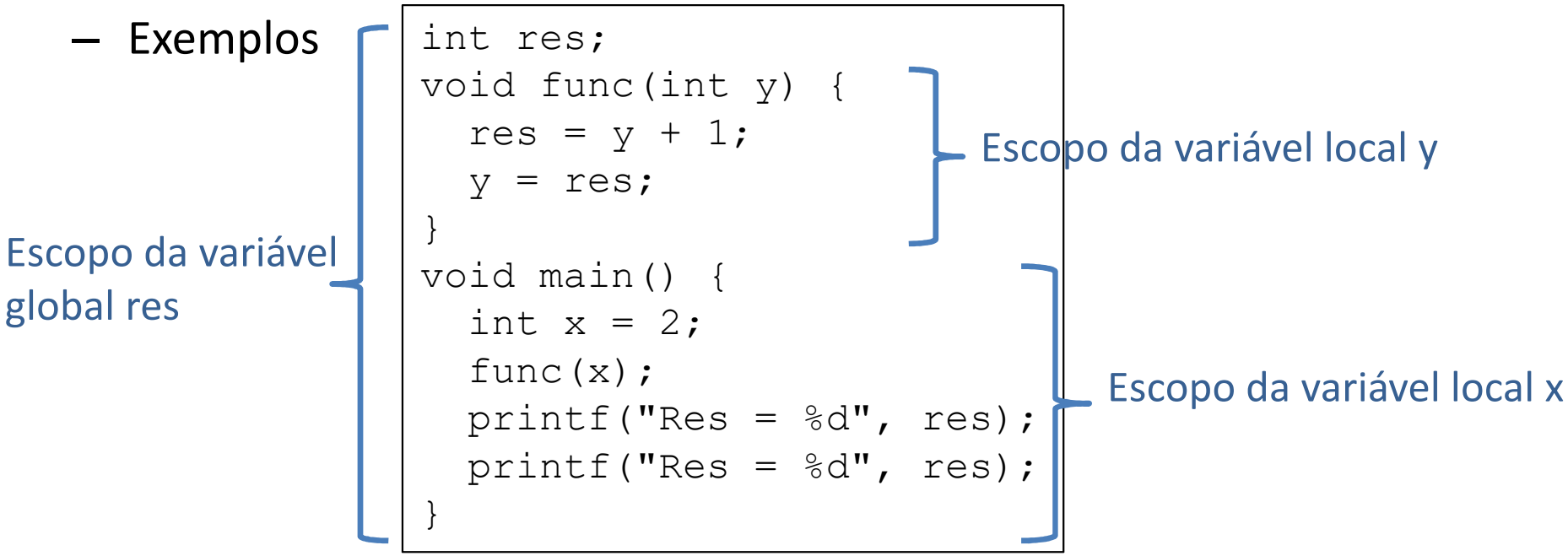
- São vinculadas ao armazenamento no heap apenas quando são atribuídos valores a elas
  - Exemplo em JavaScript

```
highs = [74, 84, 86, 90, 71];
```

- Vantagem
  - Flexibilidade
- Desvantagem
  - Sobrecarga em tempo de execução

Escopo e Ciclo de Vida

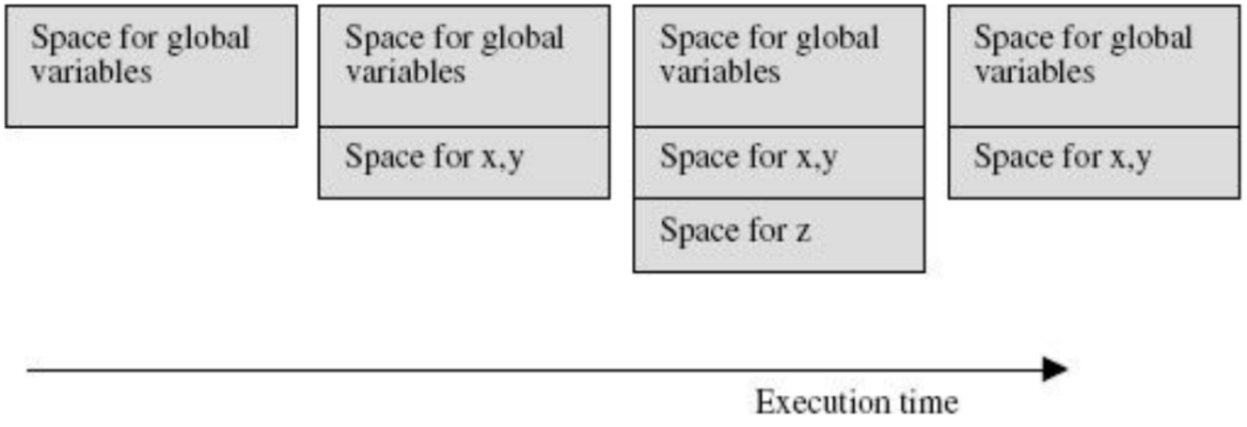
- Escopo é a região do código onde a declaração é visível
- Ciclo de Vida é o tempo, durante a execução de um programa, que o espaço alocado para uma declaração existe
- Escopos em relação a um bloco
  - Local: interno ao bloco
  - Global: externo ao bloco
  - Exemplos



Gerenciando o escopo de variáveis

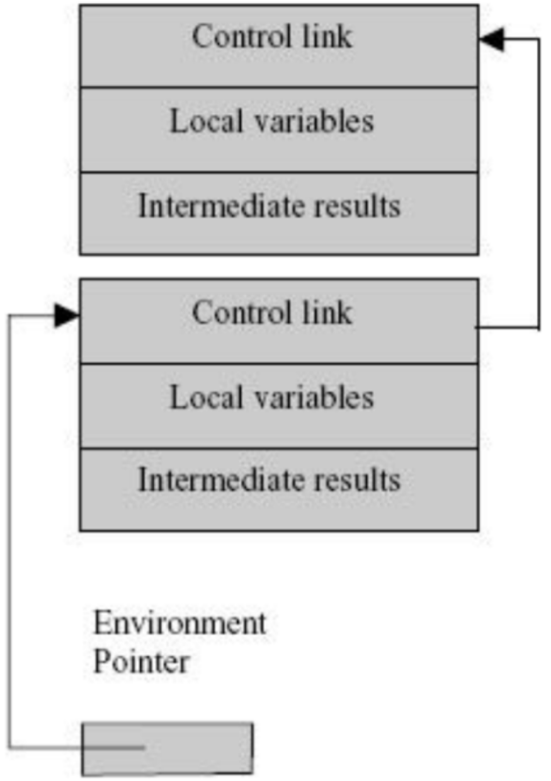
- Activation Records e variáveis locais
  - Quando entra no bloco: aloca espaço
  - Quando sai do bloco: desaloca espaço
  - Espaço = Activation Record (Stack frame)
  - Conhecido em tempo de execução
  - Pode variar bastante (funções recursivas)

```
{ int x=0;
  int y=x+1;
      { int z=(x+y)*(x-y);
        };
};
```



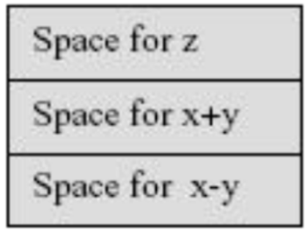
Controle da pilha e valores intermediários

- Tamanho de cada Activation Record é diferente
  - Link de controle
    - Usado para operação das funções pop e push da pilha
  - Push record on stack
    - Set new control link to point to old env ptr
    - Set env ptr to new record
  - Pop record off stack
    - Follow control link and reset



- Activation record pode armazenar valores intermediários

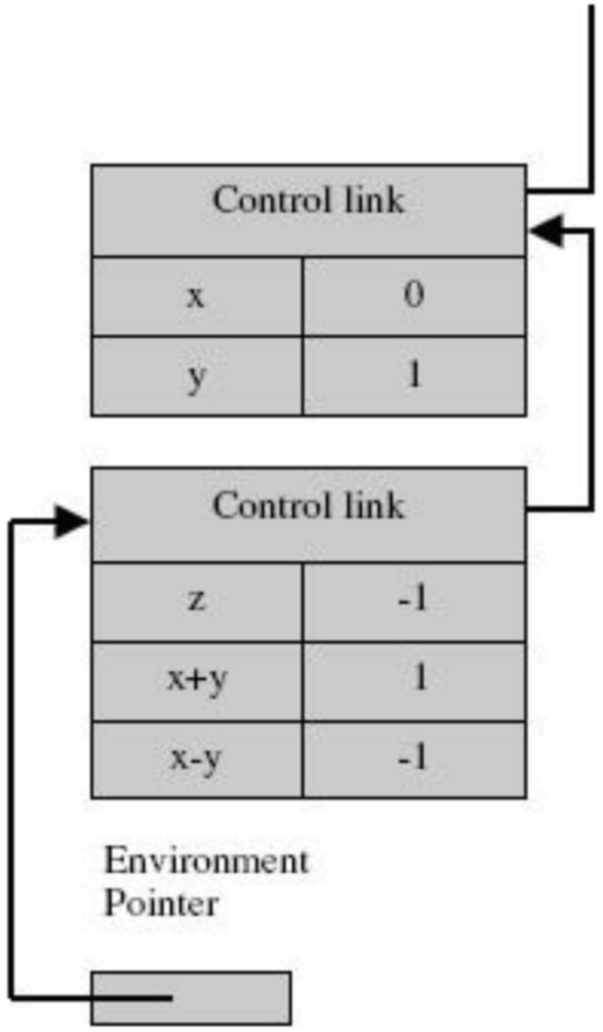
```
{ int z = (x+y)*(x-y);
}
```



Acesso a contextos externos

- Dentro do contexto mais interior
  - Necessário usar o ponteiro para o Activation Record do escopo anterior para acessar x e y

```
{ int x=0;
  int y=x+1;
      { int z=(x+y)*(x-y);
        };
        };
```



Bibliografia

- SEBESTA, R. W. Conceitos de Linguagens de Programação. Porto Alegre: Bookman, 2011. 792p.