Disciplina: Linguagens de Programação

Modulo 2 – Variáveis, Nomes, vinculações e escopos

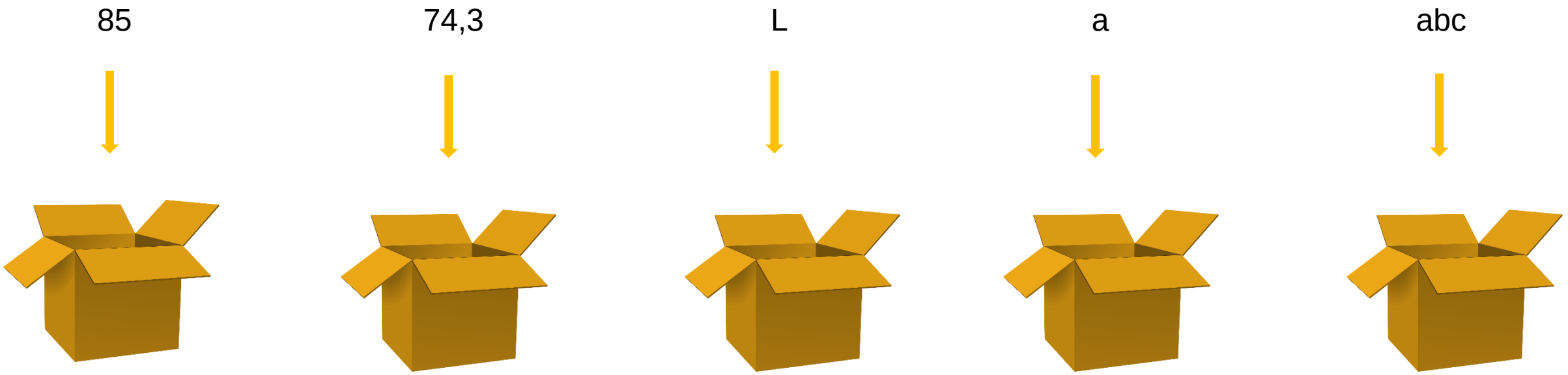
Professor: Adriano Benigno Moreira

benigno@uit.br

Universidade de Itaúna

2.1 Introdução

Uma variável pode ser caracterizada por uma coleção de propriedades, ou atributos, das quais a mais importante é o tipo, um conceito fundamental em linguagens de programação. O projeto dos tipos de dados de uma linguagem exige considerar diversas questões. Entre as mais importantes estão o escopo e o tempo de vida das variáveis.



Universidade de Itaúna

2.3 Variáveis

Uma variável de programa é uma abstração de uma célula de memória de um computador ou de uma coleção de células. Os programadores geralmente pensam em variáveis como nomes para locais de memória, mas elas são muito mais do que apenas um nome.

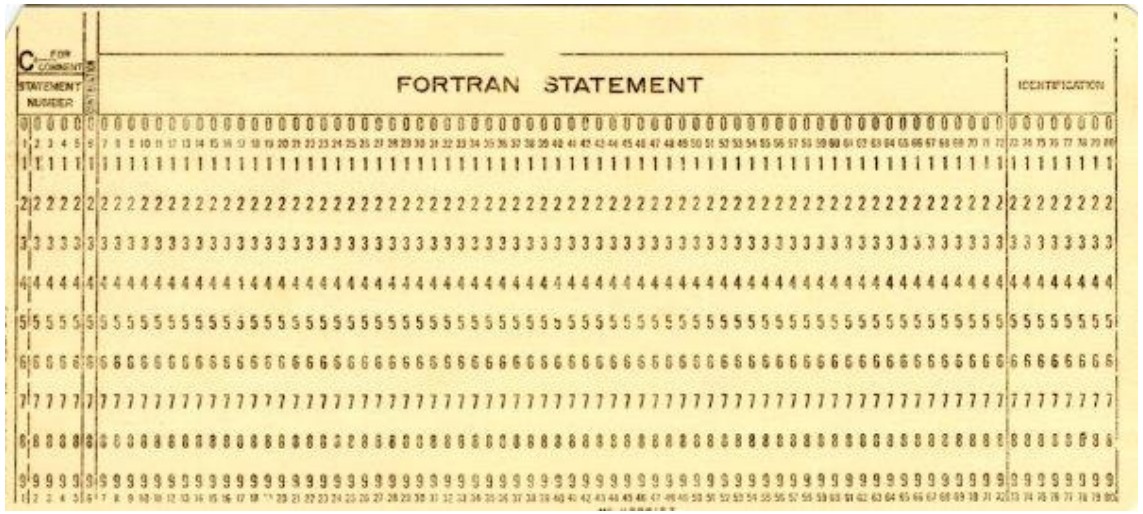
A mudança das linguagens de máquina para as de montagem foi motivada pela necessidade de substituir por nomes os endereços numéricos e absolutos de memória para dados, tornando os programas muito mais legíveis e fáceis de serem escritos e mantidos.

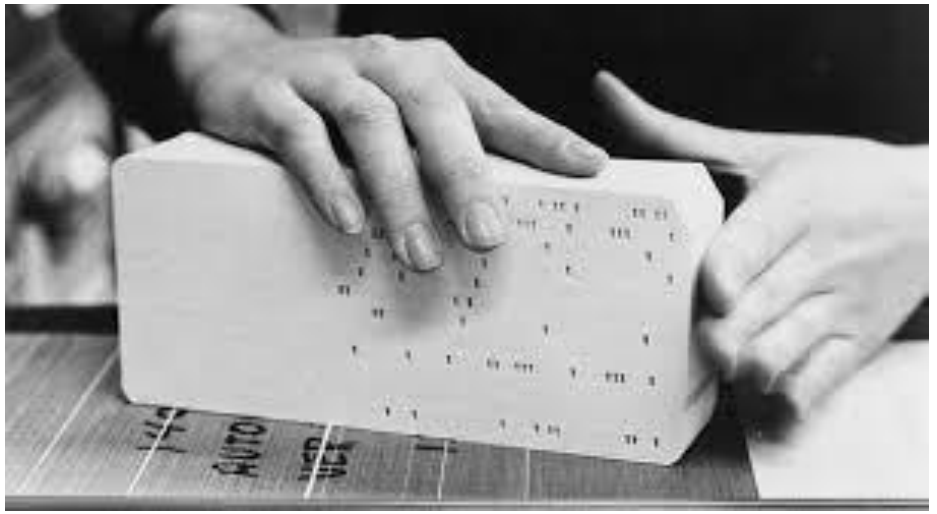
Uma variável pode ser caracterizada por seis atributos: **nome, endereço, valor, tipo, tempo de vida e escopo**. Apesar de isso parecer muito complicado para um conceito aparentemente tão simples, essa caracterização é a maneira mais clara de explicar os vários aspectos das variáveis.

2.2 Nomes

Nomes também são associados a subprogramas, parâmetros formais e outras construções de programa. O termo identificador é muito usado como sinônimo de nome.

- ***2.2.1 Questões de projeto***. As principais questões de projeto para nomes são:
  - Os nomes são sensíveis à diferenciação de maiúsculas/minúsculas?
  - As palavras especiais da linguagem são palavras reservadas ou palavras-chave?





2.2 Nomes

- ***2.2.2 Formato de nomes.*** Um nome é uma cadeia de caracteres usada para identificar alguma entidade em um programa. Na maioria das linguagens de programação, os nomes têm o mesmo formato: uma letra seguida por uma cadeia de letras, dígitos e sublinhados (_). Apesar de o uso de sublinhados ter sido disseminado nos anos 1970 e 1980, a prática é menos popular atualmente. Nas linguagens baseadas em C, eles foram, em grande medida, substituídos pela assim chamada notação camelo (*CamelCase*), na qual, em um nome de várias palavras, todas elas, exceto a primeira, começam com maiúsculas, como em `myStack`. Note que o uso de sublinhados e de caixa mista em nomes é uma questão de estilo de programação, não de projeto de linguagem. Todos os nomes de variáveis em PHP devem começar com um cifrão. Em muitas linguagens, mais notavelmente nas baseadas em C, as letras maiúsculas e minúsculas nos nomes são distintas; ou seja, as linguagens são **sensíveis à diferenciação de maiúsculas/minúsculas**. Por exemplo, os três nomes a seguir são distintos em C++: `rosa, ROSA e Rosa`. Para algumas pessoas, esse é um sério detrimento à legibilidade, porque nomes que se parecem denotam entidades diferentes. Nesse sentido, essa sensibilidade viola o princípio de projeto que diz que as construções de linguagem parecidas devem ter significados parecidos. Mas em linguagens cujos nomes de variáveis são sensíveis à diferenciação de maiúsculas/minúsculas, apesar de `Rosa` e `rosa` parecerem similares, não existe uma conexão entre elas. *(Comentar sobre métodos em Java. Ex. `parseInt`.)*

PH → variáveis iniciam com caractere $

Perl → variáveis iniciam com ‘$’, ‘@’, “5” e indicam o tipo de variável

Ruby → “@” atributos de um objeto e “$” para atributos de classe

2.2 Nomes

- ***2.2.3 Palavras especiais.*** Palavras especiais em linguagens de programação são usadas para tornar os programas mais legíveis ao nomearem as ações a serem realizadas. Elas também são usadas para separar as partes sintáticas das sentenças e programas. Na maioria das linguagens, as palavras especiais são classificadas como palavras reservadas, o que significa que não podem ser redefinidas pelos programadores. Mas, em algumas, como em Fortran, elas são apenas palavras-chave, ou seja, podem ser redefinidas.
- Uma **palavra reservada** é uma **palavra especial** de uma linguagem de programação **que não pode ser usada como um nome**. Há um problema em potencial com as palavras reservadas: se a linguagem inclui um grande número delas, o usuário tem dificuldades para inventar nomes que não sejam reservados. O melhor exemplo disso é COBOL, que tem 300 palavras reservadas. Infelizmente, alguns dos nomes mais escolhidos pelos programadores estão na lista das palavras reservadas – como, por exemplo, LENGTH, BOTTOM, DESTINATION e COUNT.

2.3 Variáveis

- ***2.3.1 Endereço.*** O endereço de uma variável é o endereço de memória de máquina ao qual ela está associada. Essa associação não é tão simples como pode parecer. Em muitas linguagens, é possível que a mesma variável seja associada a diferentes endereços em vários momentos durante a execução do programa. Por exemplo, se um subprograma tem uma variável local alocada a partir da pilha de tempo de execução quando o subprograma é chamado, diferentes chamadas podem resultar em a variável ter diferentes endereços. Essas são, em certo sentido, instanciações diferentes da mesma variável.

  É possível haver diversas variáveis com o mesmo endereço. Quando mais de um nome de variável pode ser usado para acessar a mesma posição de memória, as variáveis são chamadas de **apelidos** (*aliases*).

  O uso de apelidos é um problema para a legibilidade, porque permite que uma variável tenha seu valor modificado por uma atribuição a uma variável diferente.

  Duas variáveis de ponteiro são apelidos quando apontam para a mesma posição de memória. O mesmo ocorre com as variáveis de referência. Esse tipo de uso de apelidos é um efeito colateral da natureza dos ponteiros e das referências. Quando um ponteiro em C++ é configurado para apontar para uma variável nomeada, o ponteiro (quando desreferenciado) e o nome da variável são apelidos.

2.3 Variáveis

- ***2.3.2 Tipo.*** O tipo de uma variável determina a faixa de valores que ela pode armazenar e o conjunto de operações definidas para valores do tipo. Por exemplo, o tipo **`int`** em Java especifica uma faixa de valores de –2147483648 a 2147483647 e operações aritméticas para adição, subtração, multiplicação, divisão e módulo.
- Java – biginteger
- Haskell linguagens ditas com valores infinitos de inteiros...

- ***2.3.3 Valor.*** O valor de uma variável é o conteúdo da(s) célula(s) de memória associada(s) a ela. É conveniente pensar na memória de um computador em termos de células abstratas, em vez de em termos de células físicas. As células físicas, ou unidades endereçáveis individualmente, da maioria das memórias de computador contemporâneas tem o tamanho de um byte, que, por sua vez, tem oito bits. Esse tamanho é pequeno demais para a maioria das variáveis de programa. Uma célula abstrata de memória tem o tamanho exigido pela variável à qual está associada. Por exemplo, apesar de os valores de ponto flutuante poderem ocupar quatro bytes físicos em uma implementação de determinada linguagem, um valor de ponto flutuante é visto como se ocupasse uma única célula abstrata de memória.

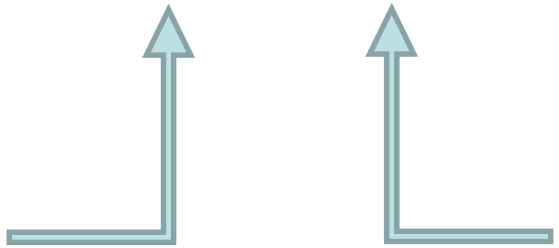
2.3 Variáveis

- ***Endereço vs. Valor***

- ***Endereço: Localização da variável na memória***
- ***Valor: Conteúdo armazenado na variável***

Valor esquerdo ou L-Value **Valor direito ou R-Value**

```
int x = y + 1
```



Preciso de seu endereço para saber onde armazenar o resultado da expressão

Preciso de seu valor para poder calcular o resultado da expressão

2.4 O CONCEITO DE VINCULAÇÃO

- **Vinculação.** Vinculação é a associação entre um atributo e uma entidade, como entre uma variável e seu tipo ou valor, ou entre uma operação e um símbolo. O momento no qual uma vinculação ocorre é chamado de tempo de vinculação. A vinculação e os tempos de vinculação são conceitos proeminentes na semântica das linguagens de programação. As vinculações podem ocorrer em tempo de projeto da linguagem, em tempo de implementação da linguagem, em tempo de compilação, em tempo de carga, em tempo de ligação ou em tempo de execução. Por exemplo, o símbolo de asterisco (*) é geralmente ligado à operação de multiplicação em tempo de projeto de uma linguagem. Um tipo de dados, como **int** em C, é vinculado a uma faixa de valores possíveis em tempo de implementação da linguagem. Em tempo de compilação, uma variável em um programa Java é vinculada a um tipo de dados em particular. Uma variável pode ser vinculada a uma célula de armazenamento quando o programa é carregado na memória. A mesma vinculação não acontece até o tempo de execução, em alguns casos, como as variáveis declaradas em métodos Java. Uma chamada a um subprograma de uma biblioteca é vinculada ao código do subprograma em tempo de ligação. Considere a seguinte sentença de atribuição em C++:

```
count = count + 5;
```

Algumas das vinculações e seus tempos de vinculação para as partes dessa sentença são:

- O tipo de count é vinculado em tempo de compilação.
- O conjunto dos valores possíveis de count é vinculado em tempo de projeto do compilador.
- O significado do símbolo de operador + é vinculado em tempo de compilação, quando os tipos dos operandos tiverem sido determinados.
- A representação interna do literal 5 é vinculada ao tempo de projeto do compilador.
- O valor de count é vinculado em tempo de execução a essa sentença.

2.4 O CONCEITO DE VINCULAÇÃO

- ***2.4.1 Vinculação de atributos a variáveis.*** Uma vinculação é **estática** se ocorre pela primeira vez antes do tempo de execução e permanece inalterada ao longo da execução do programa. Se a vinculação ocorre pela primeira vez durante o tempo de execução ou pode ser mudada ao longo do curso da execução do programa, é chamada de **dinâmica**. A vinculação física de uma variável a uma célula de armazenamento em um ambiente de memória virtual é complexa, porque a página ou o segmento do espaço de endereçamento no qual a célula reside pode ser movido para dentro ou para fora da memória muitas vezes, durante a execução do programa. De certa forma, tais variáveis são vinculadas e desvinculadas repetidamente. Contudo, essas vinculações são mantidas pelo hardware do computador, e as alterações são invisíveis para o programa e para o usuário.

2.4 O CONCEITO DE VINCULAÇÃO

- ***2.4.2* Vinculações de tipos.** Antes de uma variável poder ser referenciada em um programa, ela deve ser vinculada a um tipo de dados. Os dois aspectos importantes dessa vinculação são: como o tipo é especificado e quando a vinculação ocorre. Os tipos podem ser especificados estaticamente por alguma forma de declaração explícita ou implícita.
  - **Vinculação de tipos estática** Uma **declaração explícita** é uma sentença em um programa que lista nomes de variáveis e especifica que elas são de certo tipo. Uma declaração implícita é uma forma de associar variáveis a tipos por meio de convenções padronizadas, em vez de por sentenças de declaração. Nesse caso, a primeira aparição de um nome de variável em um programa constitui sua declaração implícita. Tanto as declarações explícitas quanto implícitas criam vinculações estáticas a tipos. A maioria das linguagens de programação amplamente usadas, que empregam exclusivamente vinculação estática a tipos e foram projetadas desde meados dos anos 1960, exige declarações explícitas de todas as variáveis (Visual Basic e ML são duas exceções). Apesar de serem uma pequena conveniência para os programadores, as declarações implícitas podem ser prejudiciais à confiabilidade, pois impedem o processo de compilação de detectar alguns erros de programação e de digitação.
  - **Vinculação de tipos dinâmica**, o tipo de uma variável não é especificado por uma sentença de declaração, nem pode ser determinado pelo nome da variável. Em vez disso, a variável é vinculada a um tipo quando é atribuído um valor a ela em uma sentença de atribuição. Quando a sentença de atribuição é executada, a variável que recebe um valor atribuído é vinculada ao tipo do valor da expressão no lado direito da atribuição. Tal atribuição também pode vincular a variável a um endereço e a uma célula de memória, pois diferentes valores de tipo podem exigir diferentes quantidades de armazenamento. Qualquer variável pode receber qualquer valor de tipo. Além disso, o tipo de uma variável pode mudar qualquer número de vezes durante a execução do programa. É importante perceber que o tipo de uma variável, quando é vinculado dinamicamente, pode ser temporário.

    ```
    list = [10.2, 3.5]
    ```

  - Independentemente do tipo anterior da variável chamada list, essa atribuição faz com que ela se torne o nome de um vetor unidimensional de tamanho 2. Se a sentença

    ```
    list = 47;
    ```

2.4 O CONCEITO DE VINCULAÇÃO

- Há uma convergência de opiniões que a principal desvantagem da vinculação de tipos dinâmica seja o custo. O custo de implementar a vinculação de atributos dinâmica é considerável, principalmente em tempo de execução. A verificação de tipos deve ser feita em tempo de execução. Além disso, cada variável deve ter um descritor em tempo de execução associado a ela, de forma a manter o tipo atual. O armazenamento usado para o valor de uma variável deve ser de tamanho variável, porque valores de tipos diferentes precisam de quantidades distintas de armazenamento.

  As linguagens que têm vinculação de tipos dinâmica são normalmente implementadas por meio de interpretadores puros, em vez de compiladores. Por outro lado, as linguagens com vinculações de tipo estáticas raramente são implementadas pela interpretação pura, pois os programas nessas linguagens podem ser facilmente traduzidos para versões em código de máquina muito eficientes

Vídeo

https://www.youtube.com/watch?v=GnQArD_kjPo

2.4 O CONCEITO DE VINCULAÇÃO

- ***2.4.3* Vinculações de armazenamento e tempo de vida.** O caráter fundamental de uma linguagem de programação imperativa é, em grande parte, determinado pelo projeto das vinculações de armazenamento para suas variáveis. Dessa forma, é importante ter um claro entendimento dessas vinculações. A célula de memória à qual uma variável é vinculada deve ser obtida, de alguma forma, de um conjunto de células de memória disponíveis. Esse processo é chamado de alocação. Liberação é o processo de colocar uma célula de memória que foi desvinculada de uma variável de volta no conjunto de células de memória disponíveis.

  O tempo de vida de uma variável é aquele durante o qual ela está vinculada a uma posição específica da memória. Então, o tempo de vida de uma variável começa quando ela é vinculada a uma célula específica e termina quando ela é desvinculada dessa célula. Para investigar as vinculações de armazenamento das variáveis, é conveniente separar as variáveis escalares (não estruturadas) em quatro categorias, de acordo com seus tempos de vida. As categorias são: estáticas, dinâmicas da pilha, dinâmicas do monte explícitas e dinâmicas do monte implícitas.

2.4 O CONCEITO DE VINCULAÇÃO

- **2.4.3.1 Variáveis estáticas.** Variáveis estáticas são vinculadas a células de memória antes do início da execução de um programa e permanecem vinculadas a essas mesmas células até que a execução do programa termine. Variáveis vinculadas estaticamente têm diversas aplicações importantes em programação. Variáveis acessíveis globalmente são usadas ao longo da execução de um programa, o que implica tê-las vinculadas ao mesmo armazenamento durante essa execução. Algumas vezes é conveniente ter subprogramas sensíveis ao histórico. Tais subprogramas devem ter variáveis locais estáticas. Uma vantagem das variáveis estáticas é a eficiência. Todo o endereçamento de variáveis estáticas pode ser direto; outros tipos de variáveis frequentemente exigem endereçamento indireto, que é mais lento. Além disso, não há sobrecarga em tempo de execução para a alocação e a liberação de variáveis estáticas, apesar de esse tempo ser normalmente insignificante. Uma desvantagem da vinculação estática para armazenamento é a redução da flexibilidade; em particular, uma linguagem que tem apenas variáveis estáticas não permite o uso de subprogramas recursivos. Se os vetores são estáticos, eles não podem compartilhar o mesmo armazenamento para seus vetores.

2.4 O CONCEITO DE VINCULAÇÃO

- **2.4.3.2 Variáveis dinâmicas.** Variáveis dinâmicas da pilha são aquelas cujas vinculações de armazenamento são criadas quando suas sentenças de declaração são elaboradas, mas cujos tipos são vinculados estaticamente. A elaboração de tal declaração se refere à alocação do armazenamento e ao processo de vinculação indicado pela declaração, que ocorre quando a execução alcança o código no qual a declaração está anexada. Logo, a elaboração ocorre apenas em tempo de execução. Por exemplo, as declarações de variáveis que aparecem no início de um método Java são elaboradas quando o método é chamado, e as variáveis definidas por essas declarações são liberadas quando o método completa sua execução. Como seu nome indica, as variáveis dinâmicas da pilha são alocadas a partir da pilha de tempo de execução.
- As vantagens das variáveis dinâmicas da pilha são as seguintes: para serem úteis, ao menos na maioria dos casos, os subprogramas recursivos exigem armazenamento dinâmico local, de forma que cada cópia ativa do subprograma recursivo tenha sua própria versão das variáveis locais. Essas necessidades são convenientemente satisfeitas pelas variáveis dinâmicas da pilha.
- As desvantagens das variáveis dinâmicas da pilha em relação às variáveis estáticas são a sobrecarga em tempo de execução da alocação e da liberação, acessos possivelmente mais lentos em função do endereçamento indireto necessário, e o fato de os subprogramas não poderem ser sensíveis ao histórico de execução.

2.4 O CONCEITO DE VINCULAÇÃO

- **2.4.3.3 Variáveis dinâmicas do monte** explícitas são células de memória não nomeadas (abstratas), alocadas e liberadas por instruções explícitas em tempo de execução escritas pelo programador. Essas variáveis, alocadas a partir do monte e liberadas para o monte, podem apenas ser referenciadas por ponteiros ou variáveis de referência. O monte (*heap*) é uma coleção de células de armazenamento altamente desorganizadas, devido à imprevisibilidade de seu uso. O ponteiro (ou a variável de referência) usado para acessar uma variável dinâmica do monte explícita é criado como qualquer outra variável escalar. Uma variável dinâmica do monte explícita é criada por meio de um operador ((por exemplo, em C). Como uma variável dinâmica do monte explícita é vinculada a um tipo em tempo de compilação, essa vinculação é estática. Entretanto, tais variáveis são vinculadas ao armazenamento no momento em que são criadas, durante o tempo de execução.
- Elas são usadas para construir estruturas dinâmicas, como listas ligadas e árvores, que precisam crescer e/ou diminuir durante a execução. Tais estruturas podem ser construídas de maneira conveniente por meio de ponteiros ou referências e variáveis dinâmicas do monte explícitas. As desvantagens das variáveis dinâmicas do monte explícitas são a dificuldade de usar ponteiros e variáveis de referência corretamente, o custo das referências às variáveis e a complexidade da implementação do gerenciamento de armazenamento exigido. Esse é essencialmente o problema do gerenciamento do monte, que é dispendioso e complicado.

2.4 O CONCEITO DE VINCULAÇÃO

- **2.4.3.4 Variáveis dinâmicas do monte implícitas** são vinculadas ao armazenamento no monte apenas quando são atribuídos valores a elas. De fato, todos os seus atributos são vinculados cada vez que elas recebem valores atribuídos. Por exemplo, considere a seguinte sentença de atribuição em *JavaScript*:

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

- Independentemente de a variável chamada *highs* ter sido usada anteriormente no programa ou de para que foi usada, ela agora é um vetor de cinco valores numéricos.
- A vantagem de tais variáveis é que elas têm o mais alto grau de flexibilidade, permitindo a escrita de códigos altamente genéricos. Uma desvantagem das variáveis dinâmicas do monte implícitas é a sobrecarga em tempo de execução para manter todos os atributos dinâmicos, o que pode incluir tipos e faixas de índices de vetores, entre outros.

https://www.youtube.com/watch?v=fTsaj5-g_DM
tempo: 12:40

Universidade de Itaúna

2.5 ESCOPO

- O escopo de uma variável é a faixa de sentenças nas quais ela é visível. Uma variável é visível em uma sentença se ela pode ser referenciada ou atribuída nessa sentença. Uma variável é local a uma unidade ou a um bloco de programa se for declarada lá. As variáveis não locais de uma unidade ou de um bloco de programa são aquelas visíveis dentro da unidade ou do bloco de programa, mas não declaradas nessa unidade ou nesse bloco.

- ***2.5.1*** **Escopo estático**, O escopo estático é assim chamado porque o escopo de uma variável pode ser determinado estaticamente – ou seja, antes da execução. Isso permite a um programador (e ao compilador) determinar o tipo de cada variável no programa simplesmente examinando seu código-fonte. Existem duas categorias de linguagens de escopo estático: aquelas nas quais os subprogramas podem ser aninhados, as quais criam escopos estáticos aninhados, e aquelas nas quais os subprogramas não podem ser aninhados. Na última categoria, os escopos estáticos também são criados para subprogramas, mas os aninhados são criados apenas por definições de classes aninhadas ou de blocos aninhados. Ada, JavaScript, Common Lisp, Scheme, Fortran 2003+, F# e Python permitem subprogramas aninhados, mas as linguagens baseadas em C não.

2.5 ESCOPO

- Quando o leitor de um programa encontra uma referência a uma variável, os atributos dessa variável podem ser determinados por meio da descoberta da sentença na qual ela está declarada (explícita ou implicitamente). Em linguagens de escopo estático com subprogramas, esse processo pode ser visto da forma a seguir. Suponha que uma referência é feita a uma variável `x` em um subprograma `sub1`. A declaração correta é encontrada primeiro pela busca das declarações do subprograma `sub1`. Se nenhuma declaração for encontrada para a variável lá, a busca continua nas declarações do subprograma que declarou o subprograma `sub1`, chamado de **pai estático**. Se uma declaração de `x` não for encontrada lá, a busca continua para a próxima unidade maior que envolve o subprograma em que está sendo feita a busca (a unidade que declarou o pai de `sub1`), e assim por diante, até que seja encontrada uma declaração para `x` ou a maior unidade de declaração tenha sido pesquisada sem sucesso. Nesse caso, é informado um erro de variável não declarada.

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

De acordo com o escopo estático, a referência à variável `x` em `sub2` é para o `x` declarado no procedimento big. Isso é verdade porque a busca por `x` começa no procedimento no qual a referência ocorre, `sub2`, mas nenhuma declaração para `x` é encontrada lá. A busca continua no **pai estático** de `sub2`, big, onde a declaração de `x` é encontrada. O `x` declarado em `sub1` é ignorado porque não está nos ancestrais estáticos de `sub2`.

2.5 ESCOPO

- ***2.5.2*** **Blocos**, Muitas linguagens permitem que novos escopos estáticos sejam definidos no meio do código. Esse poderoso conceito, introduzido em ALGOL 60, permite a uma seção de código ter suas próprias variáveis locais, cujo escopo é minimizado. Tais variáveis são dinâmicas da pilha, de forma que seu armazenamento é alocado quando a seção é alcançada e liberado quando a seção é abandonada. Essa seção de código é denominada bloco. Os blocos dão origem à frase linguagem estruturada por blocos. As linguagens baseadas em C permitem que quaisquer sentenças compostas (uma sequência de sentenças envoltas em chaves correspondentes – {}) tenham declarações e, dessa forma, definam um novo escopo. Tais sentenças compostas são denominadas blocos. Por exemplo, se `list` fosse um vetor de inteiros, alguém poderia escrever o seguinte:

```
if (list[i] < list[j]) {
        int temp;
        temp = list[i];
        list[i] = list[j];
        list[j] = temp;
}
```

2.5 ESCOPO

- A referência a `count` no laço de repetição **while** é para o `count` local do laço. Nesse caso, o `count` de sub é ocultado do código que está dentro do laço **while**. Em geral, a declaração de uma variável efetivamente oculta qualquer declaração de uma variável com o mesmo nome em um escopo envolvente maior.

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

Note que esse código é válido em C e C++, mas inválido em Java e C#. Os projetistas de Java e C# acreditavam que o reúso de nomes em blocos aninhados era muito propenso a erros para ser permitido.

https://www.youtube.com/watch?v=EXcj8jhZj8I

2.5 ESCOPO

- ***2.5.3* Ordem de Declaração**, como em algumas linguagens, todas as declarações de dados em uma função, exceto aquelas em blocos aninhados, devem aparecer no início da função. Entretanto, algumas linguagens – como C99, C++, Java, JavaScript e C# – permitem que as declarações de variáveis apareçam em qualquer lugar onde uma sentença poderia aparecer em uma unidade de programa. Declarações podem criar escopos não associados com sentenças compostas ou subprogramas. Por exemplo, em C99, C++ e Java, o escopo de todas as variáveis locais abarca desde suas declarações até o final dos blocos nos quais essas declarações aparecem. Em C#, o escopo de quaisquer variáveis declaradas em um bloco é o bloco inteiro, independentemente da posição da declaração, desde que ele não seja aninhado. O mesmo é válido para os métodos. Lembre-se de que C# não permite que a declaração de uma variável em um bloco aninhado tenha o mesmo nome de uma variável em um escopo que faça o aninhamento. Por exemplo, considere o código C# a seguir:

```
{int x;
    ...
   {int x; // Inválido
    ...
    }
    ...
}
```

2.5 ESCOPO

- Observe que C# ainda exige todas as variáveis declaradas antes de serem usadas. Logo, independentemente de o escopo de uma variável se estender da declaração até o topo do bloco ou subprograma no qual ela aparece, a variável ainda não pode ser usada acima de sua declaração. Em JavaScript, variáveis locais podem ser declaradas em qualquer lugar em uma função, mas o escopo de uma variável assim é sempre a função inteira. Se for usada antes de sua declaração na função, essa variável terá o valor `undefined`. A referência não é inválida. As sentenças for de C++, Java e C# permitem a definição de variáveis em suas expressões de inicialização. Nas primeiras versões de C++, o escopo de tal variável era de sua definição até o final do menor bloco que envolvia o for. Na versão padrão, entretanto, o escopo é restrito à construção for, como é o caso de Java e C#. Considere o esqueleto de método

```
void fun() {
      . . .
    for (int count = 0; count < 10; count++){
            . . .
        }
        . . .
}
```

2.5 ESCOPO

- ***2.5.4*** **Escopo Global**, Algumas linguagens, incluindo C, C++, PHP, JavaScript e Python, permitem uma estrutura de programa que é uma sequência de definição de funções, nas quais as definições de variáveis podem aparecer fora das funções. Definições fora de funções em um arquivo criam variáveis globais, potencialmente visíveis para essas funções. C e C++ têm tanto declarações quanto definições de dados globais. Declarações especificam tipos e outros atributos, mas não causam a alocação de armazenamento. As definições especificam atributos e causam a alocação de armazenamento. Para um nome global específico, um programa em C pode ter qualquer número de declarações compatíveis, mas apenas uma definição. Uma declaração de uma variável fora das definições de funções especifica que ela é definida em um arquivo diferente. Uma variável global em C é implicitamente visível em todas as funções subsequentes no arquivo, exceto naquelas que incluem uma declaração de uma variável local com o mesmo nome. Uma variável global definida após uma função pode se tornar visível na função sendo declarada como externa, como no seguinte:

```
extern int sum
```

https://www.youtube.com/watch?v=HQAUOnJG0zE

2.5 ESCOPO

- Em C99, as definições de variáveis globais normalmente têm valores iniciais. As declarações de variáveis globais nunca têm valores iniciais. Se a declaração estiver fora das definições de função, ela não precisa incluir o qualificador **`extern`**. Essa ideia de declarações e definições é usada também para funções em C e C++, em que os protótipos declaram nomes e interfaces de funções, mas não fornecem seu código. As definições de funções, em contrapartida, são completas. Em C++, uma variável global oculta por uma local com o mesmo nome pode ser acessada por meio do operador de escopo (::). Por exemplo, se x é uma variável global que está oculta em uma função por uma variável local chamada x, a variável global pode ser referenciada como ::x. Sentenças em PHP podem ser interpoladas com definições de funções. As variáveis em PHP são implicitamente declaradas quando aparecem em sentenças. Qualquer variável implicitamente declarada fora de qualquer função é global; variáveis implicitamente declaradas em funções são variáveis locais. O escopo das variáveis globais se estende desde suas declarações até o fim do programa, mas ignora quaisquer definições de funções subsequentes. Logo, as variáveis globais não são implicitamente visíveis em nenhuma função. Elas podem se tornar visíveis em funções em seu escopo de duas formas: (1) se a função contém uma variável local com o mesmo nome de uma global, a global pode ser acessada por meio do vetor $GLOBALS, usando-se o nome da global como índice do vetor, e (2) se na função não houver nenhuma variável local com o mesmo nome da global, a global poderá se tornar visível por meio de sua inclusão em uma sentença de declaração global. Considere o seguinte exemplo:

2.5 ESCOPO

2.5.4 Escopo Global,

```
$dia= "Segunda";
$mes= "Janeiro";
function calendario() {
 $dia= "terca";
 global $mes;
 print "dia local e $dia";
 $gdia = $GLOBALS['dia'];
 print "dia global $gdia <br \>";
 print "mes global e $mes";
}
calendario()
```

A interpretação desse código produz:

```
Dia local e Terca
Dia global e Segunda
Mes global e Janeiro
```

2.5 ESCOPO

- ***2.5.5*** **Avaliação do escopo estático**, O escopo estático fornece um método de acesso a não locais que funciona bem em muitas situações. Entretanto, isso não ocorre sem problemas. Primeiro, na maioria dos casos ele permite mais acesso que o necessário às variáveis e aos subprogramas. É simplesmente uma ferramenta rudimentar demais para especificar concisamente tais restrições. Segundo, e talvez mais importante, é um problema relacionado à evolução de programas. Sistemas de software são altamente dinâmicos – programas usados regularmente mudam com frequência. Essas mudanças normalmente resultam em reestruturação, destruindo a estrutura inicial que restringia o acesso às variáveis e aos subprogramas em uma linguagem de escopo estático. Para evitar a complexidade de manter essas restrições de acesso, os desenvolvedores normalmente descartam a estrutura quando ela começa a atrapalhar. Logo, tentar contornar as restrições do escopo estático pode levar a projetos de programas que mantêm pouca semelhança com seu original, mesmo em áreas do programa nas quais não foram feitas mudanças. Os projetistas são encorajados a usar muito mais variáveis globais que o necessário.

- `Exemplo em PHP:  https://www.youtube.com/watch?v=GP7wY316l0Y`
- `Exemplo em C#: https://www.youtube.com/watch?v=gMalh0cYbXo`

2.5 ESCOPO

- ***2.5.6*** **Avaliação do escopo dinâmicos**, O escopo de variáveis em APL, SNOBOL4 e nas primeiras versões de Lisp era dinâmico. Perl e Common Lisp também permitem que as variáveis sejam declaradas com escopo dinâmico, apesar de o mecanismo de escopo padrão dessas linguagens ser estático. O escopo dinâmico é baseado na sequência de chamadas de subprogramas, não em seu relacionamento espacial uns com os outros. Logo, o escopo pode ser determinado apenas em tempo de execução.

- `Exemplo em PHP:  https://www.youtube.com/watch?v=GP7wY316l0Y`
- `https://www.youtube.com/watch?v=Zursy21ZzIs`

2.5 ESCOPO

- ***2.5.7*** **Avaliação do escopo dinâmico**, O efeito do escopo dinâmico na programação é profundo. Quando o escopo dinâmico é usado, os atributos corretos das variáveis não locais visíveis a uma sentença de um programa não podem ser determinados estaticamente. Além disso, uma referência ao nome de tal variável nem sempre é para a mesma variável. Uma sentença em um subprograma que contém uma referência para uma variável não local pode se referir a diferentes variáveis não locais durante diferentes execuções do subprograma. Diversos tipos de problemas de programação aparecem devido ao escopo dinâmico.
- O escopo dinâmico também torna os programas muito mais difíceis de serem lidos, porque a sequência de chamadas de subprogramas deve ser conhecida para se determinar o significado das referências a variáveis não locais. Essa tarefa é praticamente impossível para um leitor humano. Por fim, o acesso a variáveis não locais em linguagens de escopo dinâmico demora muito mais do que o acesso a não locais, quando o escopo estático é usado.

2.6 ESCOPO E TEMPO DE VIDA

- ***2.6*** **Escopo e tempo de vida**, Algumas vezes, o escopo e o tempo de vida de uma variável parecem ser relacionados. Por exemplo, considere uma variável declarada em um método Java que não contém nenhuma chamada a método. O escopo dessa variável se estende desde sua declaração até o final do método. O tempo de vida dessa variável é o período de tempo que começa com a entrada no método e termina quando a execução do método chega ao final. Apesar de o escopo e o tempo de vida da variável serem diferentes, devido ao escopo estático ser um conceito textual, ou espacial, enquanto o tempo de vida é um conceito temporal, eles ao menos parecem ser relacionados nesse caso. Esse relacionamento aparente entre escopo e tempo de vida não se mantém em outras situações. Em C e C++, por exemplo, uma variável declarada em uma função por meio do especificador **`static`** é estaticamente vinculada ao escopo dessa função e ao armazenamento. Logo, seu escopo é estático e local à função, mas seu tempo de vida se estende pela execução completa do programa do qual faz parte.

O escopo e o tempo de vida também não são relacionados quando estão envolvidas chamadas a subprogramas. O escopo da variável sum é completamente contido na função compute. Ele não se estende até o corpo da função `printheader`, apesar de `printheader` executar no meio da execução de compute. Entretanto, o tempo de vida de sum se estende por todo o período em que `printheader` é executada. Qualquer que seja a posição de armazenamento à qual sum está vinculada antes da chamada `a printheader`, esse vínculo continuará durante e após a execução de `printheader`.

```
void printheader() {
  . . .
}    /* fim de printheader */
void compute() {
  int sum;
  . . .
  printheader();
} /* fim de compute */
```

2.7 CONSTANTES NOMEADAS

- ***2.7* Constantes Nomeadas,** é uma variável vinculada a um valor apenas uma vez. Constantes nomeadas são úteis para auxiliar a legibilidade e a confiabilidade dos programas. A legibilidade pode ser melhorada, por exemplo, ao ser usado o nome pi em vez da constante `3.14159265`. Outro uso importante de constantes nomeadas é na parametrização de um programa. Por exemplo, considere um que processa valores de dados um número fixo de vezes, digamos, 100. Tal programa normalmente usa a constante 100 em diversos locais para declarar as faixas de índices de vetores e para controlar os limites dos laços de repetição.

```
void example() {
   int[] intList = new int[100];
   String[] strList = new    String[100];
   . . .
   for (index = 0; index < 100; index++) {
   . . .
   }
    . . .
   for (index = 0; index < 100; index++) {
      . . . }
      . . .
     average = sum / 100
      . . .
}
```

2.7 CONSTANTES NOMEADAS

- Quando esse programa for modificado para lidar com um número diferente de valores de dados, todas as ocorrências de 100 devem ser encontradas e modificadas. Em um programa extenso, isso pode ser tedioso e propenso a erros. Um método mais fácil e confiável é usar uma constante nomeada como um parâmetro de programa, como segue:

```
void example() {
   final int len = 100;
   int[] intList = new int[len];
   String[] strList = new String[len];
   . . .
   for (index = 0; index < len; index++) {
    . . . }
   . . .
   for (index = 0; index < len; index++) {
    . . . }
   . . .
   average = sum / len;
  . . .
}
```

Agora, quando o tamanho precisar ser alterado, apenas uma linha deverá ser modificada (a variável len), independentemente do número de vezes em que ela é usada no programa. Esse é outro exemplo das vantagens da abstração. O nome len é uma abstração para o número de elementos em alguns vetores e para o número de iterações em alguns laços de repetição. Isso ilustra como constantes nomeadas podem auxiliar na facilidade de modificação.

Referência

- SEBESTA, Robert W.. Conceitos de linguagens de programação. 11. ed. Porto Alegre, RS: Bookman, 2018. E-book (757 p.) ISBN 9788582604694.