PARADIGMAS E LINGUAGENS DE PROGRAMAÇÃO

ENGENHARIA DA COMPUTAÇÃO – UFC/SOBRAL

Prof. Danilo Alves

danilo.alves@alu.ufc.br

INTRODUÇÃO

- As linguagens de programação imperativas são **abstrações** da arquitetura de computadores subjacente de von Neumann
  - **Memória:** armazena instruções e dados
  - **Processador:** fornece operações para modificar o conteúdo da memória

- Célula de memória: **variáveis**

INTRODUÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- O que essas abstrações representam?

```
int quantidade;

int soma(int a, int b) {
    return a + b;
}
```

- Que informações têm as variáveis?
- Variáveis são caracterizadas por atributos
- Para projetar um tipo, deve-se considerar escopo, tempo de vida das variáveis, inicialização e compatibilidade

NOMES

UNIVERSIDADE FEDERAL DO CEARÁ

- Associado a variável, rótulo, subprograma, parâmetros formais e demais construções
  - identificador

- Questões de projeto primárias para nomes:
  - Qual o tamanho máximo de um nome?
  - Caracteres de conexão podem ser usados em nomes?
  - Os nomes são sensíveis a capitalização?
    - Os nomes fazem distinção entre maiúsculas e minúsculas?

NOMES – FORMATO

UNIVERSIDADE FEDERAL DO CEARÁ

- Se for muito pequeno, não pode ser conotativo (ter sentido)

- Exemplos:
  - FORTRAN 95: máximo de 31
  - C/C++: permite combinações de letras maiúsculas, minúsculas e *underscore*
  - Java: permite uso de acentuação

NOMES – CARACTERES ESPECIAIS

UNIVERSIDADE FEDERAL DO CEARÁ

- PHP: todos os nomes de variáveis devem começar com cifrão ($)
  - Ex.: $a = 21; echo ($a);
- Perl: todos os nomes de variáveis começam com caracteres especiais, que especificam o seu tipo
  - $ para escalares: números, strings, referência;
  - @ para conjunto sequencial: vetores;
  - % para mapeamento: hash;
- Ruby: nomes de variáveis que começam com @ são variáveis de instância; as que começam com @@ são variáveis de classe

NOMES – CASE SENSITIVE

UNIVERSIDADE FEDERAL DO CEARÁ

- Desvantagem: legibilidade (nomes que são parecidos, mas são diferentes)

  – Nomes em linguagens baseadas com C são sensíveis à capitalização

  – Nomes em outras não são

  – Em C, o problema é evitado por convenções que orientam a não usar letras maiúsculas em identificadores, dando assim, significado especial em casos especiais

  – Em C++, Java e C#, o problema não pode ser evitado porque muitos dos nomes pré-definidos incluem tanto letras maiúsculas quanto minúsculas

  - Ex.: IndexOutOfBoundsException

- Para Pascal, estes nomes são idênticos:

  – Variavel, variavel, VARIAVEL, vARIAVEL, …

- Para C, C++, Java, C#, Python, … são diferentes.

NOMES – PALAVRAS ESPECIAIS

UNIVERSIDADE FEDERAL DO CEARÁ

- São usadas para tornar os programas mais legíveis ao nomearem as ações a serem realizadas, que podem ser:
  - palavras-chave
  - palavras reservadas

NOMES – PALAVRAS ESPECIAIS

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma **palavra reservada** é uma palavra especial em todos os contextos por exemplo, em Java:

  float varName (**float** é um tipo de dado acompanhado de um nome)

  float = 3.4 (float é uma variável? Erro de sintaxe!)

  - **float** é uma palavra especial sempre!

NOMES – PALAVRAS ESPECIAIS

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma **palavra-chave** é uma palavra especial apenas em alguns contextos por exemplo, em Fortran:

  Real varName (**Real** é um tipo de dado acompanhado de um nome)

  Real = 3.4 (Real é uma variável! Não há erro de sintaxe.)

  - No primeiro caso, **Real** é uma palavra especial, mas não no segundo caso

NOMES – PALAVRAS ESPECIAIS

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma **palavra reservada** é uma palavra especial que não pode ser usada como um nome – independente do contexto

– Para Java, **float** é uma palavra reservada, independente do contexto

– Para Fortran, **Real** é uma palavra-chave, pois ela é especial em alguns contextos

- Problema em potencial com as **palavras reservadas**: se houver muitas, o usuário tem dificuldade para inventar nomes (por exemplo, COBOL tem 300 palavras reservadas!)
- Problema em potencial com as **palavras-chave:** a habilidade de redefini-las pode ser confusa, como em:

Integer Real – Real é uma variável do tipo **Integer**

Real Integer – Integer é uma variável do tipo **Real**

VARIÁVEIS

- Abstração de uma célula de memória
- É comum associar a uma variável, um nome e um valor, e em alguns casos, um tipo – porém, há diversas informações acerca esse elemento
- Uma variável pode ser caracterizada como um conjunto de seis atributos:

– Nome

– Endereço

– Valor

– Tipo

– Tempo de vida

– Escopo

ATRIBUTOS DE VARIÁVEIS

UNIVERSIDADE FEDERAL DO CEARÁ

- Nome – identificadores; nem todas as variáveis têm (uso de referências)

- Endereço – é o endereço de memória de máquina ao qual ela está associada

– Uma variável pode ter diferentes endereços em diferentes momentos durante a execução

– Uma variável pode ter diferentes endereços em diferentes lugares em um programa

– Quando dois nomes de variáveis podem ser usados para acessar a mesma posição de memória, elas são chamadas de apelidos (*aliases*)

– Apelidos são criados por ponteiros, variáveis de referência, **tipos de união C e C++**

– Apelidos são um problema para a legibilidade (um leitor do programa deve sempre se lembrar de todos eles)

UNION

- A definição de uma **union** é semelhante à de um registro

```
union u_nomeUniao{
   tipoDadoMembro_1 nomeMembro_1;
   tipoDadoMembro_2 nomeMembro_2;
   ...
   tipoDadoMembro_n nomeMembro_n;
}; //end union
```

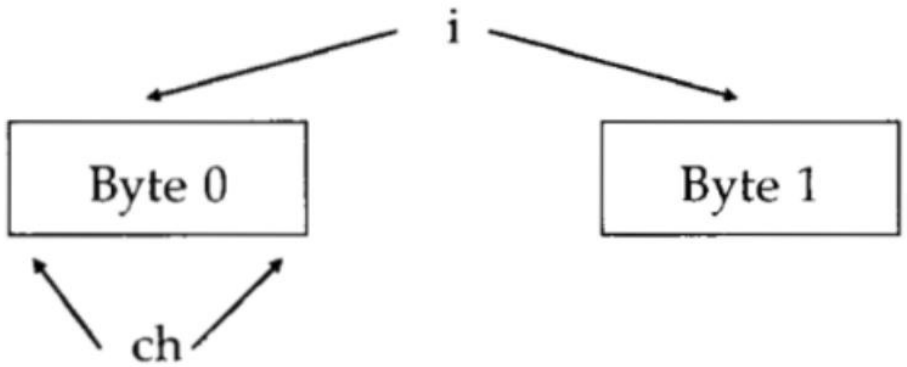
```
union u_type {
   int i;
   char ch;
};//end union

union u_type cnvt;
```

UNIÃO

- Uma **union** é uma posição de memória que é compartilhada por duas ou mais variáveis diferentes – duas variáveis ocupando o mesmo espaço de memória – geralmente de tipos diferentes, em momentos diferentes

UNION

- Na **union cnvt**, tanto inteiro **i** quanto o caractere **ch** compartilham a mesma posição de memória
  - **i** ocupa 2 (ou 4) bytes
  - **ch** ocupa 1 byte



- Quando uma **union** é definida, o compilador aloca um espaço na memória grande o bastante para conter o maior tipo dentro da **union**
- Para o exemplo, **cnvt** possui 2 bytes (ou 4)
- Para acessar um elemento da **union**, usa-se a mesma sintaxe das **structs**
  - Operador ponto (.) e operador seta (->)

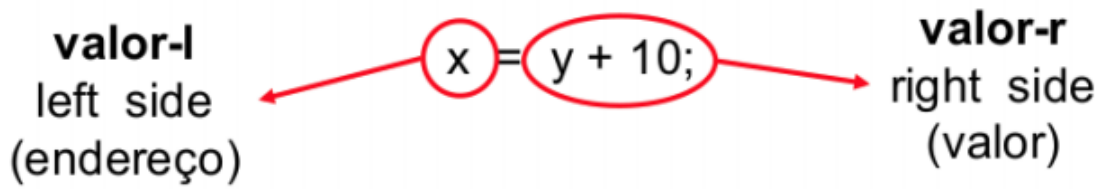
ATRIBUTOS DE VARIÁVEIS

- Tipo – determina a **faixa de valores** que a variável pode armazenar e o **conjunto de operações** definidas para valores do tipo

  – Em Java: int

  -2.147.483.648 a 2.147.483.647

  Operadores: +, –, *, /, %, etc



- Valor – é o conteúdo da(s) célula(s) de memória associada(s) a ela

  – O lado esquerdo (l-value) de uma variável é seu endereço

  – O lado direito (r-value) de variável é seu valor

- Célula abstrata de memória – célula física ou coleção de células associadas a uma variável

VINCULAÇÃO

- Uma vinculação é uma **associação**, como entre uma operação e um símbolo

  \+ operador **soma**

  – operador de **subtração**

  * operador de **multiplicação**

  / operador de **divisão**

- Tempo de vinculação é o momento no qual uma vinculação ocorre

TEMPOS DE VINCULAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- Tempo de projeto de linguagem
  - Ex.: vincula símbolos de operadores a operações
- Tempo de implementação de linguagem
  - Ex.: vincula tipo de ponto-flutuante a uma representação
- Tempo de compilação
  - Ex.: vincula uma variável a um tipo, exemplo: em C ou Java
- Tempo de carga
  - Ex.: vincula uma variável **static** de C ou C++ a uma célula de memória

- As variáveis em linguagem C possuem um tempo de vida que é condizente com o ambiente em que ela foi criada
  - Ex.: Uma variável declarada dentro de uma função. Quando a função começa a ser executada é **reservado** um **espaço de memória** para a variável em questão. Quando a função termina o espaço de memória que foi reservado para a variável é **liberado**
- Já algumas variáveis possuem um espaço reservado de modo permanente. Durante todo o tempo em que a placa estiver ligada aquele espaço será utilizado exclusivamente para a variável
  - Para forçar uma variável ter seu espaço sempre no mesmo local durante a execução do sistema, usa-se o modificador **static**

TEMPOS DE VINCULAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- Tempo de projeto de linguagem
  - Ex.: vincula símbolos de operadores a operações
- Tempo de implementação de linguagem
  - Ex.: vincula tipo de ponto-flutuante a uma representação
- Tempo de compilação
  - Ex.: vincula uma variável a um tipo, com exemplo em C ou Java
- Tempo de carga
  - Ex.: vincula uma variável **static** de C ou C++ a uma célula de memória
- **Tempo de ligação**
  - Ex.: vincula uma variável local não estática a uma célula de memória
- **Tempo de execução**
  - Ex.: vincula um valor a uma variável

TEMPOS DE VINCULAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

```
int cont; (...) cont = cont + 5;
```

- Conjunto dos tipos possíveis para cont
  - Vinculado no tempo de projeto da linguagem
- Tipo de cont
  - Vinculado no tempo de compilação

- Valor de cont
  - Vinculado no tempo de execução com essa instrução
- O significado para o símbolo int
  - Vinculado no tempo de compilação
- Representação interna do literal 5
  - Vinculada no tempo de projeto

VINCULAÇÕES ESTÁTICAS E DINÂMICAS

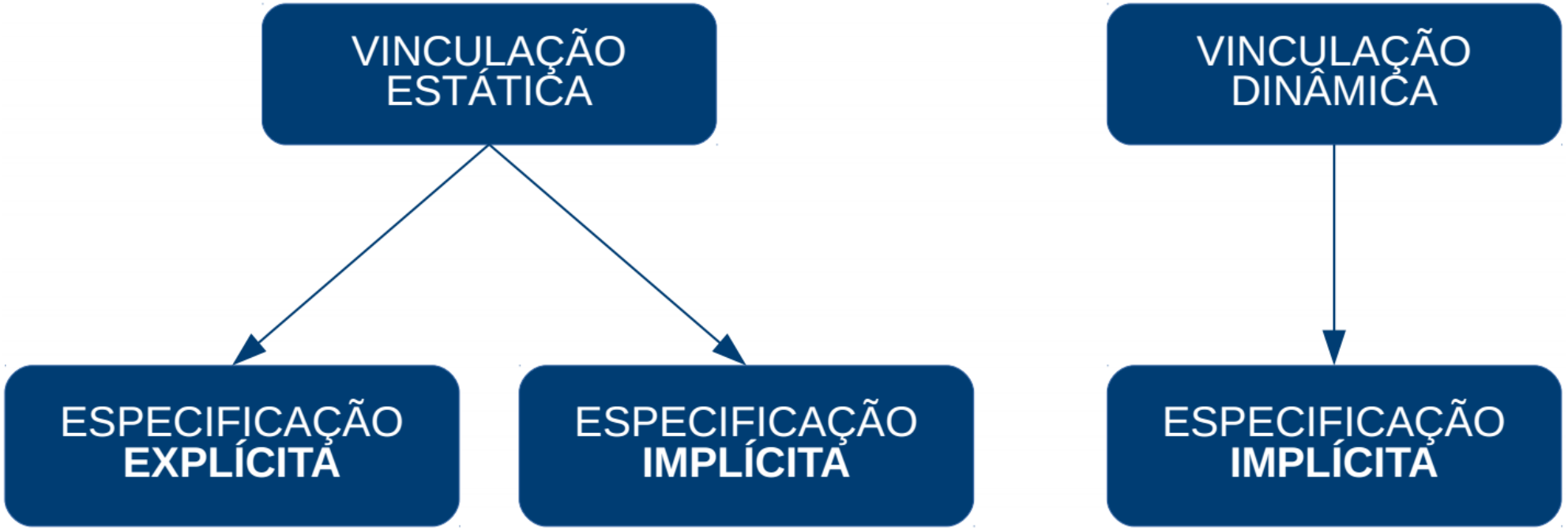
UNIVERSIDADE FEDERAL DO CEARÁ

- Antes de referenciar uma variável num programa, ela deve ser vinculada a um **tipo de dados**

  – Essa vinculação – especificação dos tipos – pode ocorrer de forma **estática** ou **dinâmica**

- Uma vinculação é **estática** se ocorre pela primeira vez antes do tempo de execução e permanece intocada ao longo da execução do programa
- Uma vinculação é **dinâmica** se ocorre pela primeira vez durante o tempo de execução ou pode ser mudada ao longo do curso da execução do programa

VINCULAÇÕES ESTÁTICAS E DINÂMICAS DE TIPOS



VINCULAÇÃO DE TIPOS

UNIVERSIDADE FEDERAL DO CEARÁ

- Como o tipo é especificado?

  – Especificação **explícita** versus **implícita**

- Quando a vinculação ocorre?

  – Vinculação **estática** versus **dinâmica**

- Se for **estático**, o tipo pode ser especificado por alguma forma de declaração **explícita** ou **implícita**

  – Uma **declaração explícita** é uma sentença em um programa que lista nomes de variáveis e especifica que elas são de um 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 (primeira aparição de um nome de variável em um programa)

- Se for **dinâmico**, o tipo é especificado implicitamente

DECLARAÇÃO EXPLÍCITA/IMPLÍCITA

UNIVERSIDADE FEDERAL DO CEARÁ

- FORTRAN, BASIC e Perl têm declarações implícitas (Fortran tem explícita e implícita)

  – Em FORTRAN, um identificador que não é declarado explicitamente, é declarado implicitamente – se começar com **i, j, k, l, m, n: Integer**, senão: **Real**

  – Perl: todos os nomes de variáveis começam com caracteres especiais, que especificam o seu tipo

  - $ para escalares: números, strings, referência;
  - @ para conjunto sequencial: vetores;
  - % para mapeamento: hash;

  - **Vantagem:** facilidade de escrita
  - **Desvantagem:** confiabilidade – previnem que no processo de compilação erros (de programação, de digitação) sejam detectados

DECLARAÇÃO EXPLÍCITA/IMPLÍCITA

UNIVERSIDADE FEDERAL DO CEARÁ

- C, C++, Java, Pascal possuem declaração explícita
- PHP, Ruby, Python têm declarações implícitas

VINCULAÇÃO DE TIPOS DINÂMICA

UNIVERSIDADE FEDERAL DO CEARÁ

- Vinculação de tipos dinâmica (JavaScript, PHP, Pyhon)

- Especificada por meio de uma sentença de atribuição

– por exemplo, Python

- list = [2, 4.33, 6, 8]
- list = 17.3

– **Vantagem:** flexibilidade (unidades de programa genéricas)

– Desvantagens:

- Custo elevado
- Detecção de erros de tipo pelo compilador é difícil

INFERÊNCIAS DE TIPOS

UNIVERSIDADE FEDERAL DO CEARÁ

- Em vez de por sentenças de atribuição, tipos são determinados (pelo compilador) a partir do contexto da referência – o programador não precisa especificar os tipos das variáveis
- Ex. em Python

– **def** circunf(r):

    return 3.14159 * r * r

– **def** quadrado(x):

    return x * x

– **def** quadrado(x):

    return int(x) * **int**(x)

– **def** vezes10(x):

    return 10 * x

– **def** quadrado(x):

    return int(x * x)

– **def** quadrado(x):

    return int(x) * x

VINCULAÇÕES DE ARMAZENAMENTO E TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Alocação – obter uma célula de um conjunto de células disponíveis
  - Célula de memória vinculada a uma variável é tomada da memória disponível
- Liberação – colocar a célula de volta no conjunto
  - Devolve a célula de memória desvinculada de uma variável à memória disponível
- Tempo de vida – tempo durante o qual a variável está vinculada a uma posição da memória, que começa na alocação e termina na liberação

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Para investigar as vinculações de armazenamento, é conveniente separar variáveis escalares em quatro categorias, de acordo com seus tempos de vida
  - Estáticas
  - Dinâmicas da pilha
  - Dinâmicas do monte explícitas
  - Dinâmicas do monte implícitas

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Estáticas – vinculadas a células de memória antes do início da execução de um programa e permanecem vinculadas a essas mesmas células de memória até que a execução do programa termine

  – Vantagens: uso global, eficiência (endereçamento direto), subprogramas sensíveis ao histórico

  – Desvantagens: redução da flexibilidade (sem recursão)

  – Ex. em C, C++ e em Java: aplicação de static

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Dinâmicas da pilha – Vinculações de armazenamento são criadas quando suas sentenças de declaração são elaboradas, mas cujos tipos são estaticamente vinculados. (A declaração é elaborada quando a execução alcança o código com o qual a declaração está anexada)
- Se escalar, todos os atributos exceto o endereço são vinculados estaticamente
  - Variáveis locais em subprogramas C e métodos Java
- Vantagem: permite recursão; conserva o armazenamento
- Desvantagens:
  - Sobrecarga da alocação e liberação
  - Subprogramas não são sensíveis ao histórico
  - Referências ineficientes (endereçamento indireto)

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Dinâmicas do monte explícitas – Células de memória abstratas (sem nome) alocadas e liberadas por instruções explícitas em tempo de execução

  – Monte – Conjunto de células de armazenamento altamente desorganizado devido à imprevisibilidade do uso

- Referenciadas apenas por ponteiros ou variáveis de referência, por exemplo, objetos dinâmicos em C++ (via new e delete), todos os objetos em Java

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

```
int *intnode; // Cria um ponteiro
intnode = new int; // Cria uma variável dinâmica do monte
...
delete intnode; // Libera a variável dinâmica do monte
// para qual intnode aponta
int *intnode; // Cria um ponteiro
intnode = (int*) malloc(sizeof(int)); // Cria monte uma variável
... // dinâmica do monte
free(intnode); // Libera a variável dinâmica do monte
// para qual intnode aponta
```

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Variáveis dinâmicas do monte explícitas são usadas para construir estruturas dinâmicas, como listas ligadas e árvores, que precisam crescer e/ou diminuir durante a execução
- Vantagem: prevê o gerenciamento de armazenamento dinâmico
- Desvantagem: ineficientes e não confiáveis

CATEGORIAS DE VARIÁVEIS POR TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Dinâmicas do monte implícitas – Alocadas e liberadas por sentenças de atribuição
  - Todas as variáveis em APL;
  - Todas cadeias e matrizes em Perl, JavaScript e PHP

  Ex, em JavaScript.: var notas = [7.4, 8.4, 8.6, 9.0, 7.1]

  Independentemente se a variável notas foi previamente usada no programa ou para que foi usada, ela agora é um array de cinco valores numéricos
- Vantagem:
  - Flexibilidade (código genérico)
- Desvantagens:
  - Ineficiente, pois todos os atributos são dinâmicos
  - Perda de detecção de erro

ESCOPO

UNIVERSIDADE FEDERAL DO CEARÁ

- O escopo de uma variável é a faixa de sentenças nas quais a variável é visível
- As variáveis não locais de uma unidade ou de um bloco de programa são aquelas que estão visíveis, mas não são declaradas nessa unidade ou bloco
- As regras de escopo de uma linguagem determinam como uma ocorrência em particular de um nome é associada com uma variável

Escopo está relacionado ao **espaço**

Uma variável é visível em uma sentença se ela pode ser referenciada nessa sentença

ESCOPO

UNIVERSIDADE FEDERAL DO CEARÁ

- Baseado no texto do programa
- Para ligar um nome de referência para uma variável, você (ou o compilador) deve encontrar a declaração
- **Processo de busca:** busca por declaração, primeiro localmente, depois em escopos cada vez maiores, até encontrar o nome dado
- Elementos ancestrais de escopo estático (para um escopo específico) são chamados **ancestrais estáticos**; o mais próximo é chamado de **pai estático**
- Algumas linguagens permitem subprogramas aninhados, que criam escopos estáticos aninhados (por exemplo, Ada, Python, JavaScript, Fortran 2003 e PHP)
- Java e C# não permitem

ESCOPO ESTÁTICO

UNIVERSIDADE FEDERAL DO CEARÁ

- Variáveis podem ser ocultas de outros segmentos de códigos, em uma unidade com uma variável com o mesmo nome
- É possível definir uma forma de acesso à variável do escopo ancestral:

  – Em Ada, variáveis ocultas de escopos ancestrais podem ser acessadas com referências seletivas, que incluem o nome do escopo ancestral

  - Exemplo, unit.name

  – Python permite funções dentro de outras funções

  - Python3 inclui a palavra **nonlocal** que permite referenciar uma variável não local

ESCOPO ESTÁTICO

UNIVERSIDADE FEDERAL DO CEARÁ

```
def f1():
    x = 2
    def f2():
        x = 3
        print(x)

    f2()
    print(x)

f1();
```

ESCOPO ESTÁTICO

UNIVERSIDADE FEDERAL DO CEARÁ

```
def f1():
    x = 2
    def f2():
        nonlocal x
        x = 3
        print(x)

    f2()
    print(x)

f1();
```

BLOCOS

- Um método de criar escopos estáticos em unidades de programa – do ALGOL 60
- Exemplo em C:

```
void sub(  ) {
    int count;

    while ( ... ) {
        int count;
        count++;
        ...

    }
    ...
}
```

- Note que o código é legal em C e C++, mas ilegal em Java e C#

  – Em Java → error: variable count is already defined in method sub()

BLOCOS

UNIVERSIDADE FEDERAL DO CEARÁ

- Variáveis locais aos blocos são dinâmicas na pilha – espaço de memória é alocado quando o bloco é alcançado e liberado quando a seção é abandonada
- Origem do termo (linguagem ou programação) estruturada [em blocos]
- Linguagens baseadas em C permitem que quaisquer sentenças compostas tenham declarações e, dessa forma, definam um novo escopo

```
int* func(int tam, int init) {
    int* v;
    ...
    {
        int i;
        v = (int*) malloc(sizeof(int) * tam);
        for (i = 0; i < tam; ++i) v[i] = init;
    }
    ...
}
```

ORDEM DE DECLARAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- C99, C++, Java e C# permitem que as declarações de variáveis apareçam em qualquer lugar que uma sentença poderia aparecer em uma unidade de programa

  – Em C99, C++ e Java, o escopo de todas as variáveis locais é de suas declarações até o final dos blocos

  – Em C#, o escopo de quaisquer variáveis declaradas em um bloco é o bloco inteiro

    - C# ainda requer que todas as variáveis sejam declaradas antes de serem usadas

- Em C++, Java e C#, variáveis podem ser declaradas em sentenças for

  – O escopo dessas variáveis é restrito à construção for

ORDEM DE DECLARAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

- Em JavaScript, variáveis podem ser declaradas atribuindo um valor diretamente a uma variável, fazendo com que esta tenha escopo global
- Para declarar variáveis com escopo restrito as palavras **var**, **let** e **const** são usadas
- const permite que uma "variável imutável" seja declarada em um escopo
- O uso de var e de let está relacionado ao escopo
  - var permite que uma variável seja "vista" antes de sua declaração, em um escopo
  - let somente tornará "visível" a variável após a sua declaração

ORDEM DE DECLARAÇÃO

UNIVERSIDADE FEDERAL DO CEARÁ

```
function exemplo() {
    // x poderia ser acessado aqui – hoisting
    for(var x = 0; x < 5; x++) {
        // x existe aqui
    };
    // x está visível aqui novamente
};

function exemplo() {
    // x não existe aqui
    for(let x = 0; x < 5; x++) {
        // x existe aqui
    };
    // x não está visível aqui novamente
};
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

- C, C++, PHP 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

  – Definição de variáveis fora de funções em um arquivo criam variáveis globais

- C e C++ têm declarações e definições

  – Declarações especificam tipo, nome, mas não a alocação de armazenamento.

  Definições causam alocação de armazenamento

  – Uma variável global em C é implicitamente visível em todas as funções subsequentes no arquivo, exceto aquelas que incluem uma declaração local de mesmo nome

  – Em C++, a variável ocultada pode ser acessada pelo operador de escopo ::

  – Uma variável global definida após uma função pode ser tornada visível na função declarando-a como externa

  - extern int sum;

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

```
#include <iostream>

int x = 20;

void imprimeX(void) {
    int x = 30;
    std::cout << ::x << “, ” << x << std::endl;
}

int main(void) {
    imprimeX();

    return 0;
}
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

```c
#include <stdio.h>

void imprimeX(void) {
    extern int x;
    printf("%d\n", x);
}

int x = 20;

int main(void) {
    imprimeX();

    return 0;
}
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

- PHP

  – Programas são embutidos em documentos HTML

  – O escopo de uma variável (implicitamente) declarada em uma função é local à função

  – O escopo das variáveis globais se estende de suas declarações até o fim do programa, mas pulam sobre quaisquer definições de funções subsequentes

  - Variáveis globais podem ser acessadas em uma função por meio do vetor $GLOBALS

```
<?php
$a  = 10;

function teste() { echo $GLOBALS['a']; }

teste();

?>
```

```
e: Undefined variable:
shell code on line 2
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

```
$day = "Monday";
$month = "January";

function calendar() {
    $day = "Tuesday";
    global $month;
    print "local day is $day <br />";
    $gday = $GLOBALS['day'];
    print "global day is $gday <br \>";
    print "global month is $month <br />";
}
calendar();
```

```
local day is Tuesday
global day is Monday
global month is January
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

- Python
  - Uma variável global pode ser referenciada em uma função, mas uma variável global pode ter valores atribuídos a ela apenas se ela tiver sido declarada como global na função

```
x = 10
def f():
    x = 20
    print(x)

f()
print(x)
```

```
x = 10
def f():
    global x
    x = 20
    print(x)

f()
print(x)
```

ESCOPO GLOBAL

UNIVERSIDADE FEDERAL DO CEARÁ

- Java e C# não possuem variáveis de escopo global
- Pode ser contornado por meio da definição de classe com atributo static

```
class Global {
    public static int x;
    public static readonly int MAX = 10000;
}

class MainProgram {
    public static void Main(string[] args) {
        Global.x = 9;
        Global.x++;

        System.Console.WriteLine(Global.x);
        System.Console.WriteLine(Global.MAX);
    }
}
```

```
class Global {
    public static int x;
    public static final int MAX = 10000;
}

class MainProgram {
    public static void Main(String[] args) {
        Global.x = 9;
        Global.x++;

        System.out.println(Global.x);
        System.out.println(Global.MAX);
    }
}
```

AVALIAÇÃO DO ESCOPO ESTÁTICO

UNIVERSIDADE FEDERAL DO CEARÁ

- Funciona bem em muitas situações

- Problemas:

  – Em muitos casos, fornece mais acesso do que é necessário

  – Como um programa evolui, a estrutura inicial é destruída e as variáveis locais muitas vezes se tornam globais

ESCOPO DINÂMICO

UNIVERSIDADE FEDERAL DO CEARÁ

- **Baseado na sequência de chamadas de subprogramas, não em seu relacionamento espacial uns com os outros**

- As referências a variáveis são conectadas com as declarações por meio de buscas pela cadeia de chamadas a subprograma que forçaram a execução até esse ponto

EXEMPLO DE ESCOPO


```
Big
    - declaração de X
      Sub1
        - declaração de X -
        ...
        chama Sub2
        ...

      Sub2
        ...
        - referência a X -
        ...
    ...
    chama Sub1
    ...
```

- **Big chama Sub1**
- **Sub1 chama Sub2**
- **Sub2 usa X**

EXEMPLO DE ESCOPO

- Escopo estático

  – Referência a X é ao X declarado em Big

- Escopo dinâmico

  – Referência a X é ao X declarado em Sub1 – depende dos subprogramas ativos

- Avaliação do escopo dinâmico:

  – Vantagem: conveniência – não precisa passar parâmetros às chamadas a subprogramas

  – Desvantagens:

    1. Enquanto um subprograma é executado, suas variáveis são visíveis aos subprograma que ele chama
    2. Impossibilidade de verificar os tipos das referências a não locais estaticamente
    3. Programas são mais difíceis de ler

ESCOPO E TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

- Escopo e tempo de vida às vezes parecem ser relacionados, mas são conceitos diferentes
- Considere uma variável static em uma função C ou C++

```
void f() {
    static int i = 0;

    i += 1;

    printf("%d\n", i);
}
```

```
int main(void) {
    f();
    f();
    f();


    return 0;
}
```

ESCOPO E TEMPO DE VIDA

UNIVERSIDADE FEDERAL DO CEARÁ

```
void printheader() {
    ...
} /* Fim de printheader */


void compute() {
    int sum;              /* Qual o escopo e tempo de vida
    ...                      de sum? */
    printheader();
} /* Fim de compute */
```

Escopo está relacionado ao **espaço**

Tempo de vida está relacionado ao **tempo**

AMBIENTES DE REFERENCIAMENTO

UNIVERSIDADE FEDERAL DO CEARÁ

- O ambiente de referenciamento de uma sentença é a coleção de todas as variáveis visíveis na sentença
- Em uma linguagem de escopo estático, é composto pelas variáveis declaradas em seu escopo local mais a coleção de todas as variáveis de seus escopos ancestrais visíveis

- Um subprograma está ativo se sua execução começou, mas ainda não terminou
- Em uma linguagem de escopo dinâmico, é composto pelas variáveis declaradas localmente, mais as variáveis de todos os outros subprogramas que estão atualmente ativos

AMBIENTES DE REFERENCIAMENTO

UNIVERSIDADE FEDERAL DO CEARÁ

```
int c = 3;


int main(void) {
    int a = 1;

    {
        int b = 2;
        ... <------?
    }
    <---------------?

    return 0;
}
```

```
int c = 3;
int a = 0;

int main(void) {
    int a = 1;

    {
        int b = 2;
        ... <------?
    }
    <---------------?

    return 0;
}
```

AMBIENTES DE REFERENCIAMENTO

UNIVERSIDADE FEDERAL DO CEARÁ

```
c = 3

def f():

    a = 1

    while True:
        b = 2
        <----------?
    <--------------?

f()
<--------------------?
```

```
c = 3
a = 0

def f():

    a = 1

    while True:
        b = 2
        <----------?
    <--------------?

f()
<--------------------?
```

AMBIENTES DE REFERENCIAMENTO

UNIVERSIDADE FEDERAL DO CEARÁ

```
c = 3

def f():

    a = 1

    while True:
        b = 2
        <----------?
    <--------------?

f()
<------------------?
```

```
c = 3
a = 0

def f():
    global a
    a = 1

    while True:
        b = 2
        <----------?
    <--------------?

f()
<------------------?
```

AMBIENTES DE REFERENCIAMENTO


```
procedure Example is
    A, B : Integer;
    ...
    procedure Sub1 is
        X, Y : Integer;
        begin -- de Sub1
        ... <---------------- 1
        end;  -- de Sub1
    procedure Sub2 is
        X : Integer;
        ...
        procedure Sub3 is
            X : Integer;
            begin -- de Sub3
            ... <------------ 2
            end;  -- de Sub3
        begin -- de Sub2
        ... <---------------- 3
        end;  -- de Sub2
    begin -- de Example
    ... <-------------------- 4
    end.  -- de Example
```

Os ambientes de referenciamento dos pontos de programa indicados são:

1) X e Y de Sub1, A e B de Example
2) X de Sub3, (X de Sub2 está oculto), A e B de Example
3) X de Sub2, A e B de Example
4) A e B de Example

AMBIENTES DE REFERENCIAMENTO

UNIVERSIDADE FEDERAL DO CEARÁ

- O ambiente de referenciamento de uma sentença em uma linguagem de escopo dinâmico é composto pelas variáveis declaradas localmente, mais as variáveis de todos os subprogramas **ativos**
- Mais uma vez, algumas variáveis em subprogramas ativos podem ser ocultadas do ambiente de referenciamento
- Ativações de subprogramas recentes podem ter declarações para variáveis que ocultam variáveis com o mesmo nome em ativações prévias de subprogramas

CONSTANTES NOMEADAS

- Uma constante nomeada é uma variável que está vinculada a um valor apenas uma vez
- Vantagens: legibilidade e confiabilidade
- Usadas para parametrizar programas: LEN, MIN, MAX, ...

```
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;
    ...
}
```

CONSTANTES NOMEADAS

UNIVERSIDADE FEDERAL DO CEARÁ

- Uma constante nomeada é uma variável que está vinculada a um valor apenas uma vez
- Vantagens: legibilidade e confiabilidade
- Usadas para parametrizar programas: LEN, MIN, MAX, ...
- A vinculação de valores a constantes nomeadas podem ser estáticas (chamadas de constantes de manifesto) ou dinâmicas
  - const int result = 2 * width + 1;
  - Linguagens:
    - FORTRAN 95: apenas expressões constantes
    - Ada, C++ e Java: expressões de qualquer tipo
    - C# tem dois tipos, readonly e const
      - Os valores de const são vinculados em tempo de compilação
      - Os valores de readonly são dinamicamente vinculados