Linguagens de Programação

Aula 7

Celso Olivete Júnior

olivete@fct.unesp.br

Na aula passada

- Nomes, Vinculações, Verificação de Tipos e Tipos de Vinculações

Na aula de hoje

- Tipos de dados – Capítulo 6 - Sebesta – Conceitos de Linguagens de Programação – 9ª Edição

Roteiro

- Introdução
- Tipos de Dados Primitivos
- Tipos Cadeia de Caracteres
- Tipos Definidos pelo Usuário
- Tipos Matriz
- Tipos Registro
- Tipos União
- Tipos Ponteiro
- Verificação e equivalência de tipos

Introdução

- Um **tipo de dado** define uma coleção de dados e um conjunto de operações pré-definidas sobre esses dados
- Um **descritor** é um conjunto de atributos de uma variável (Nome, Endereço, Valor, Tipo, Tempo de vida, Escopo).

Introdução

- É importante que uma LP forneça uma coleção apropriada de **tipos de dados**
- Utilidade:
  - Detecção de erros
  - Modularização
  - Documentação

Introdução

- Sistema de tipos
  - Define como um tipo é associado a uma expressão
  - Inclui regras para equivalência e compatibilidade de tipos

- Entender o sistema de tipos de uma LP é um dos aspectos mais importantes para entender a sua semântica (significado)

- Questão de projeto relativa a todos os tipos de dados:
  - Quais operações são fornecidas para variáveis do tipo e como elas são especificadas?

Tipos de Dados Primitivos

- **Os tipos de dados primitivos** são os tipos de dados não-definidos em termos de outros tipos
- Praticamente todas as LP oferecem um conjunto de tipos de dados primitivos
- Usados com construções de tipo para fornecer os tipos estruturados. Os mais comuns são:
  - Tipos numéricos
  - Tipos booleanos
  - Tipos caracteres

Tipos de Dados Primitivos tipos numéricos: inteiro

- O **tipo de dados primitivo** numérico mais comum é o **inteiro.**
- Quase sempre um reflexo do hardware
  - Assim, seu mapeamento é trivial
- Muitos computadores suportam diferentes tamanhos para inteiros
- Em Java, diferentes tamanhos para inteiros com sinal
  - `byte`, `short`, `int`, `long`

Tipos de Dados Primitivos
tipos numéricos: inteiro em java

Tipo	Tamanho (bits)	Intervalo	
		Início	Fim
byte	8	-128	127
short	16	-32768	32767
int	32	-2.147.483.648	2.147.483.647
long	64	-9223372036854775808	9223372036854775807

Tipos de Dados Primitivos
tipos numéricos: ponto flutuante

- Tipos de dados de **ponto flutuante** modelam os números reais mas as representações são apenas aproximações

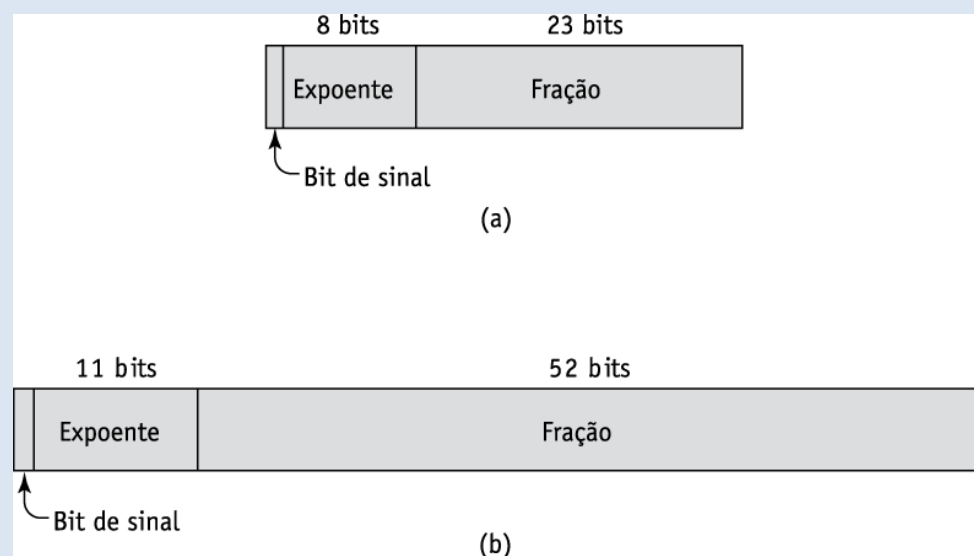
- Valores de **ponto flutuante** são representados como **frações expoentes** (máquinas mais antigas). Máquinas atuais usam o formato padrão IEEE Floating-Point Standard 754.

- A maioria das LP's de fins científicos suportam pelo menos dois tipos de **ponto flutuante**:
  - `float`: é o tamanho padrão. Ocupa 4 bytes de memória
  - `double`: fornecido para situações nas quais partes fracionárias são maiores. Ocupa o dobro do tamanho de float.

Tipos de Dados Primitivos
tipos numéricos: ponto flutuante

- Os valores que podem ser representados pelo tipo ponto flutuante é definido em termos de **precisão** e **faixa**
  - **Precisão:** exatidão da parte fracionária de um valor (medida pela quantidade de bits)
  - **Faixa:** combinação da faixa de **frações** e, o mais importante, de **expoentes.**

Tipos de Dados Primitivos
tipos numéricos: ponto flutuante padrão IEEE 754



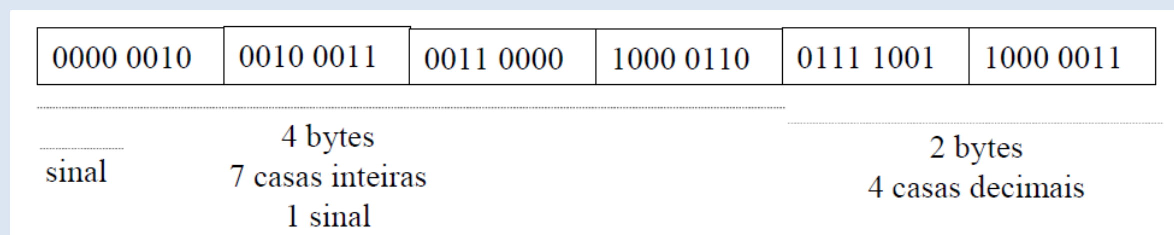
IEEE Padrão de Ponto Flutuante 754

(a) Precisão simples

(b) Precisão dupla

Tipos de Dados Primitivos tipos numéricos: decimal

- Para aplicações comerciais (formato moeda)
  - Essencial para COBOL
  - C# oferece um tipo de dado decimal
- Armazenam um número fixo de dígitos decimais, com um ponto decimal em uma posição fixa no valor



- Vantagem
  - Precisão
- Desvantagens
  - Faixa de valores restrita
  - Desperdício de memória

Tipos de Dados Primitivos
booleanos

- Mais simples de todos
- Faixa de valores
  - Dois elementos, um para "*true*" e um para "*false*"
- Um inteiro poderia ser utilizado para representá-lo, porem dificulta a legibilidade.
- C não apresenta o tipo booleano
  - 0 falso; 1 verdadeiro

Tipos de Dados Primitivos
caracteres

- Armazenados como codificações numéricas
- Tradicionalmente, a codificação mais utilizada era o ASCII de 8 bits
  - Faixa de valores entre 0 e 127 para codificar 128 caracteres diferentes
  - Se tornou inadequado (>computadores; >necessidade de comunicação)
- Uma alternativa, codificação de 16 bits: Unicode (UCS-2)
  - Inclui caracteres da maioria das linguagens naturais
  - Usado em Java
  - C# e JavaScript também suportam Unicode

Tipos de Dados Primitivos
caracteres

- Outra codificação
  - UTF-32 (ou UCS-4) publicado em 2000
    - ✓ Codificação de caracteres em 4 bytes
    - ✓ Suportada por Fortran, começando em 2003

Tipos Cadeias de Caracteres (strings)

- Valores consistem em sequências de caracteres
- Questões de projeto:
  - É um tipo primitivo ou apenas um tipo especial de vetores de caracteres?
  - As cadeias devem ter tamanho estático ou dinâmico?

Tipos Cadeias de Caracteres (strings)

- String de Tamanho **Estático**:
  - Tamanho especificado na declaração.
  - “Strings cheias”, caso uma string mais curta for atribuída, os caracteres vazios são definidos como brancos (espaços).
- String de Tamanho **Dinâmico Limitado**:
  - Possuem tamanhos variáveis até um máximo declarado e fixo estabelecido pela definição da variável.
  - Tais variáveis podem armazenar qualquer número de caracteres entre zero e o máximo.

Tipos Cadeias de Caracteres (strings)

- String de Tamanho **Dinâmico**:
  - Possuem tamanhos variáveis sem limites.
  - Tal opção exige a alocação e desalocação dinâmica de armazenamento, mas proporciona máxima flexibilidade.

Tipos Cadeias de Caracteres:

strings e suas operações

- Operações típicas:
  - Atribuição e cópia
  - Comparação (=, >, etc.)
  - Concatenação
  - Referências a subcadeias (fatias)

Tipos Cadeias de Caracteres: cadeias nas linguagens

- C e C++
  - Não são definidas como tipo primitivo
  - Usam vetores `char` e uma biblioteca de funções que oferecem operações (string.h)
  - Finalizadas por um caractere especial, nulo, representado com um zero → operações são realizadas até encontrar este caractere.
  - Ex: `char str[] = "teste"; //`vetor de elementos do tipo char → `teste` **0**
  - Exemplos de bibliotecas:
    - ✓ strcpy → copia o conteúdo
    - ✓ strcmp → compara o conteúdo
    - ✓ strlen, strcat

Tipos Cadeias de Caracteres: avaliação

- São importantes para a capacidade de escrita de uma linguagem
- A adição de cadeias de caracteres em uma LP não é custoso
  - Por que não tê-los em uma linguagem?
  - Por exemplo, se C não tivesse a função `strcpy`?
- Tamanho dinâmico é mais flexível
  - Mas o custo compensa?

Tipos Cadeias de Caracteres: implementação

- Tamanho estático
  - descritor em tempo de compilação
- Tamanho dinâmico limitado
  - Podem exigir um descritor em tempo de execução para armazenar tanto o tamanho máximo como o tamanho atual
- Tamanho dinâmico
  - Exigem um descritor em tempo de execução
  - Exigem um gerenciamento de armazenagem mais complexo
    - ✓ Alocação e desalocação é o maior problema

Tipos Cadeias de Caracteres: implementação

- Descritor em tempo de compilação para cadeias estáticas

Cadeia estática
Tamanho
Endereço

Tipos Cadeias de Caracteres: implementação

- Descritor em tempo de execução para cadeias dinâmicas de tamanho limitado

Cadeia dinâmica de tamanho ilimitado
Tamanho máximo
Tamanho atual
Endereço

Tipos Definidos pelo Usuário
tipos ordinais

- Um tipo ordinal é aquele cuja faixa de valores possíveis pode ser associada ao conjunto dos números inteiros positivos. Podem ser:
  - Tipos Enumeração
  - Tipos Subfaixa
- Exemplos de tipos primitivos ordinais em Java
  - integer
  - char
  - boolean

Tipos Definidos pelo Usuário
tipos enumeração

- Todos os valores possíveis, os quais se tornam constantes simbólicas e são enumerados na definição
- Exemplo C#

```
enum dias {seg, ter, qua, qui, sex, sab, dom};
```

- As constantes de enumeração são preenchidas implicitamente por atribuições de valores inteiros (0, 1, 2,...6 no caso acima)

Tipos Definidos pelo Usuário
tipos enumeração

- Todos os valores possíveis, os quais se tornam constantes simbólicas e são enumerados na definição
- Exemplo C: forma de se declarar constantes em C

```
define Brasil 0
define ITALIA 1
define PORTUGAL 2
define ALEMANHA 3
```

Tipos Definidos pelo Usuário

tipos enumeração

- Todos os valores possíveis, os quais se tornam constantes simbólicas e são enumerados na definição
- Exemplo C: forma de se declarar constantes em C

```
define Brasil 0
define ITALIA 1
define PORTUGAL 2
define ALEMANHA 3
```

```
Utilizando enum
enum Paises
{
        BRASIL,
        ITALIA,
        PORTUGAL,
        ALEMANHA
};
```

Tipos Definidos pelo Usuário

tipos enumeração

- Todos os valores possíveis, os quais se tornam constantes simbólicas e são enumerados na definição
- Exemplo C: forma de se declarar constantes em C

```
define Brasil 0
define ITALIA 1
define PORTUGAL 2
define ALEMANHA 3
```

```
Utilizando enum
enum Paises
{
        BRASIL,
        ITALIA,
        PORTUGAL,
        ALEMANHA
};
```

enum Paises pais;
//C e C++

Tipos Definidos pelo Usuário
tipos enumeração

- **Questões de projeto**
  - Deve-se permitir que uma constante de enumeração apareça em mais de uma definição de tipo e, se assim for, como o tipo de uma ocorrência de tal constante é verificado no programa?
  - Os valores de enumeração são convertidos para inteiros?
  - Existem outros tipos que são convertidos para um tipo enumeração?

Todas essas questões estão relacionadas com a verificação de tipos

Tipos Definidos pelo Usuário
tipos enumeração

- Em linguagens que não tem o tipo enumeração
  - Simula-se com o uso de valores inteiros. Ex: Representar um conjunto de cores
    - int vermelho = 0, azul = 1;
    - Problema:
      - Não existe um tipo para cores, ou seja, não tem verificação de tipos quando são usadas.
        - Seria permitido adicionar as duas juntas
        - Poderia ser atribuído qualquer valor inteiro a elas, destruindo o relacionamento de cores

Tipos Definidos pelo Usuário
tipos enumeração

- Linguagens C e Pascal incluem enumeração
  - Ex:

```
enum colors {vermelho, azul, verde, amarelo, preto}
colors MinhaCor = vermelho
```

  - Como o tipo **colors** armazena internamente 0,1,2.. para as cores
  - Caso ocorre-se **`MinhaCor ++`**
  - `MinhaCor` passaria ser `azul`

Tipos Definidos pelo Usuário
tipos enumeração - avaliação

- Vantagens
  - Legibilidade
    - Valores nomeados são facilmente reconhecidos, enquanto os codificados não.
  - Confiabilidade:
    - Nenhuma operação aritmética é permitida
    - Nenhuma variável de enumeração pode ter um valor atribuído a ela fora da faixa definida.

Tipos Definidos pelo Usuário

tipos subfaixa

- Um tipo subfaixa (subrange) é uma subsequência contígua de um ordinal.
- Por exemplo, 10..14 é uma subfaixa do tipo inteiro → incluídos em Pascal e Ada

➔ enumeração

```
enum dias {seg, ter, qua, qui, sex, sab, dom};
```

➔ subfaixa

```
subtype DIASSEMANA is DIAS range seg .. sex;
subtype NOME is LETRAS range A .. Z;
```

Tipos Definidos pelo Usuário

tipos subfaixa

- Outro exemplo em Pascal:

```
program primeiro;
type faixames = 1 ..12;
type faixadias = 1..31;
type maiuscula = 'A' .. 'Z';
var letra: maiuscula ;
begin
    letra:= 3;
    writeln(letra); //exibe 'C', índice começa em 1
end.
```

Tipos Definidos pelo Usuário

tipos subfaixa: avaliação

- Melhora legibilidade
  - Informam ao leitor que as variáveis podem armazenar apenas uma faixa de valores
- Melhora a confiabilidade
  - A atribuição de um valor a uma variável fora de sua faixa pode ser detectado pelo compilador como sendo um erro

Implementação de tipos ordinais definidos pelo usuário

- Tipos enumeração são implementados como inteiros

- Tipos subfaixas são implementados como seus tipos ancestrais, exceto que as verificações de faixas devem ser implicitamente incluídas pelo compilador em cada atribuição de uma variável

Tipos Matrizes

- Uma matriz (arrays) é um agregado de dados homogêneo em que cada elemento é uma variável e pode ser acessado por sua posição relativa, a partir da origem (primeiro elemento), através de um índice.
- Exemplo:
  - Muitas vezes em programas, é necessário que uma variável contenha muitos valores
  - Indexação - mapeamento de índices para elementos de um array.
  - Mapeamento (nome_array, valor-índice)→ elemento

```
float dados [3] [2];
```

5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0

Tipos Matrizes
questões de projeto

- Quais tipos são legais para os índices?
- As expressões de índices em referências a elementos são verificados quanto à faixa?
- Quando as faixas de índices são vinculadas?
- Quando a alocação/liberação da matriz ocorre?
- Matrizes podem ser inicializadas quando têm seu armazenamento alocado?
- Quais tipos de fatias são permitidos, se for o caso?

Tipos Matrizes
matrizes e índices

- Índices (ou subscritos) fazem mapeamento para elementos
- Nome_matriz(lista_valores_índices) → elemento
- Sintaxe do índice
  - FORTRAN, PL/I e Ada **usam parênteses** (Problemas quando se definem funções parametrizadas → uma matriz se confunde com a chamada da função → reduz legibilidade)
    - ✓ `mat(i)(j)`
  - Maioria das linguagens usam colchetes
    - ✓ `mat[i][j]`

Ling. C
int notas [100];

Tipos Matrizes
tipos de índices

- FORTRAN, C: apenas inteiros
- Pascal: qualquer tipo ordinais
  - inteiro, Boolean, Char, enumeração
- Ada: inteiro ou enumeração (incluindo Boolean e Char)
- Java: apenas inteiros
- Verificação de faixas de índices
  - C, C++, Perl e Fortran não especificam faixa para checagem
  - Java, ML e C# especificam a checagem da faixa

Tipos Matriz
exemplos de tipos de índices

```
C:
int mat[5][4];
float a[10];
```

```
Pascal :
var a: array[1 .. 10] of real;
     b: array[1 .. 10, 1..10] of real;
```

```
Fortran 90:
integer vetor1(1:10), vetor2(1:10)
                         ...
vetor1 = vetor1 + vetor2
```

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- **Matriz Estática**
  - As faixas de índice estão estaticamente vinculadas e a alocação de armazenamento é estático (feita antes da execução).

    Ex: int alunos[100];
  - Vantagem: eficiência (nem alocação nem desalocação é necessária)
  - Desvantagem: armazenamento permanece durante toda execução e é necessário saber o tamanho antes da execução do programa

- No exemplo acima, a matriz de 100 elementos do tipo ***int*** irá requerer 100*2 ou 200 bytes de memória.
  - Se fosse do tipo float, 100*4 ou 400 bytes de memória

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- Matriz Estática

```
#include <stdio.h>

void main(void)
      {
        int notas[100];
        float salar[100];
        char string[100];

        printf("Memoria para conter int notas[100] %d bytes\n",
           sizeof(notas));
        printf("Memoria para conter float salar[100] %d bytes\n",
           sizeof(salar));
        printf("Memoria para conter char string[100] %d bytes\n",
           sizeof(string));
     }
```

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- **Matriz Fixa Dinâmica na Pilha**
  - Faixas de índice estão estaticamente vinculadas, mas a alocação é feita no momento da declaração durante a execução
  - Vantagem: eficiência de espaço
  - Desvantagem: tempo necessário para alocação e desalocação

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- **Matriz Dinâmica na Pilha**
  - Faixas de índices estão dinamicamente vinculadas e a alocação de armazenamento é dinâmica (feita durante a execução)
  - Uma vez que as faixas de índices são vinculadas e o armazenamento é alocado, ambas permanecem fixas durante todo o tempo de vida
  - Vantagem: flexibilidade (o tamanho de uma matriz não precisa ser conhecido antes da sua utilização)

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- **Matriz Dinâmica no Monte**
  - A vinculação das faixas dos índices e a alocação são dinâmicas e podem mudar várias vezes
  - Vantagem: flexibilidade (matrizes podem crescer ou encolher durante a execução do programa)
  - Desvantagem: alocação e liberação levam mais tempo e podem ocorrer durante várias vezes na execução

```
int* v;
…
v = (int*) malloc(n * sizeof(int));
```

Tipos Matrizes

Vinculações de índices e Categorias de Matrizes

- Matrizes C e C++ que incluem o modificador **static** são estáticas
- Matrizes C e C++ sem **static** são fixas dinâmicas na pilha
- C e C++ também oferecem matrizes dinâmicas (malloc e free)
- PHP e JavaScript suportam matrizes dinâmicas
- C# inclui inclui uma segunda classe de matrizes, *ArrayList*, que fornece matrizes dinâmicas da pilha

Tipos Matrizes
inicialização de matrizes

- Algumas linguagens permitem a inicialização no momento em que o armazenamento é alocado
- Exemplos: C, C++, Java e C#

```
int list [] = {4, 5, 7, 83}
```

- Cadeias de caracteres em C e C++

```
char name [] = "freddie";
```

- Matrizes de strings em C e C++

```
char *names [] = {"Bob","Jake","Joe"];
```

- Java: inicialização de objetos String

```
String[] names = {"Bob", "Jake", "Joe"};
```

Tipos Matrizes
fatias

- Uma fatia (slice) de uma matriz é alguma subestrutura desta. Nada mais do que um mecanismo de referência

- Fatias são úteis em linguagens que possuem operadores sobre matrizes

Tipos Matrizes
fatias

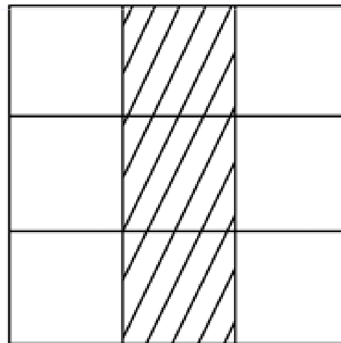
- Ex: Operações com ***arrays*** → Uma operação de ***array*** é aquela em que ele opera como uma unidade. Ex: Fortran 90
- Exemplo:
  - INTERGER VETOR (1:10)
  - MAT (1:3, 1:3)
  - CUBO (1:3, 1:3, 1:4)

Fatias

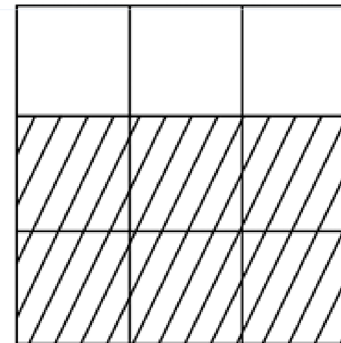
- VETOR (3:6)
- MAT (1:3, 2) 2ª coluna de mat
- MAT (3, 1:3) 3ª linha de mat

Tipos Matrizes
fatias

- Mais exemplos....Fortran



Mat (:, 2)



Mat (2:3, :)

Tipos Matrizes
avaliação

- Matrizes são incluídas na maioria das linguagens de programação
  - Vantagem: permitem a inclusão de todos os tipos ordinais como possíveis tipos de índices, fatias e matrizes dinâmicas

Implementação de Matrizes

- O código para permitir o acesso aos elementos de uma matriz deve ser gerado em tempo de compilação.

- Em tempo de execução, esse código deve ser gerado para produzir endereços de elementos

Implementação de Matrizes

- Exemplo: não existe uma maneira de computar previamente o endereço a ser acessado por

List[k] → uma matriz de uma dimensão é implementada como uma lista de células adjacentes

Obtendo o Endereço:

Endereço(List[k]) = endereço(list[0]) + k * tamanho_do_elemento

elemento float	8.0	9.0	5.0	43.0	6.0	7.0	2.0	67.0
Índice	0	1	2	3	4	5	6	7
endereço	100	104	108	112	116	120	124	128

Implementação de Matrizes

- Descritor em tempo de compilação para uma matriz de uma dimensão

Matriz
Tipo do elemento
Tipo do índice
Limite inferior do índice
Limite superior do índice
Endereço

Acessando matrizes multidimensionais

- Duas maneiras comumente usadas
  - Ordem principal de linhas → usado na maioria das linguagens
- Ordem principal de coluna
  - Usando em Fortran

Acessando matrizes multidimensionais

- Exemplo de matriz

3 4 7

6 2 5

1 3 8

- Ordem principal de linhas
  - 3,4,7,6,2,5,1,3,8
- Ordem principal de colunas
  - 3,6,1,4,2,3,7,5,8

Implementação de Matrizes

- Matriz multidimensional

Número de elementos por linha

endereço(a[i,j]) = endereço a[1,1] + ((((i-1)*n) + (j-1)) *tamanho_do_elemento)

	1	2	. . .	j−1	j	. . .	n
1							
2							
⋮							
i−1							
i					⊗		
⋮							
m							

5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0

Qual o endereço de 40 considerando a alocação ao lado?

	60.0
	55.0
	50.0
	45.0
	40.0
	35.0
	30.0
	25.0
	20.0
	15.0
	10.0
104	5.0

Implementação de Matrizes

- Descritor em tempo de compilação para uma matriz multidimensional

Matriz multidimensional
Tipo do elemento
Tipo do índice
Número de dimensões
Faixa de índices 1
...
Faixa de índices n
Endereço

Implementação de Matrizes
exemplo

```
float mat[4][3] =
{{5.0,10.0,15.0},
{20.0,25.0,30.0},
{35.0,40.0,45.0},
{50.0,55.0,60.0}};
```

- a matriz de 4*3 elementos do tipo ***float*** irá requerer 12*4 = 48 bytes

5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0

Valor	Endereço
60.0	148
55.0	
50.0	
45.0	
40.0	
35.0	
30.0	
25.0	
20.0	
15.0	
10.0	
5.0	104

Tipos Registro
definição

- Um *registro* é um agregado, possivelmente heterogêneo, de elementos de dados
- Cada elemento individual é identificado por seu nome e acessados por meio de deslocamentos a partir do início da estrutura. Exemplo de definição em C

```
struct ESTUDANTE{
string nome;
float Media_ano;
} reg_estudante;
```

Tipos Registro

- Diferenças entre matrizes heterogêneas e registros
  - Em matrizes os elementos são referenciados por objetos de dados que residem em posições espalhadas, geralmente no **monte**
  - Registros residem em posições de memória adjacentes

Tipos Registro

- Em C, C++ e C# são suportados pelo tipo *struct*
- Em Pascal pelo tipo *record*

Tipos Registro
referência a campos

- A maioria das linguagens usam um ponto na notação

`NomeRegistro.NomeCampo`

- **Ex:** considerando a sintaxe do Pascal → Para armazenar a data 02/12/2004 na variável nasc, podemos fazer:

nasc.dia := 2; nasc.mes := dez; nasc.ano := 2004

```
type Data =
record
dia : 1..31;
mes : (jan, fev, mar, abr, mai, jun, jul, ago, set, out,
nov, dez );
ano : integer
end;
var nasc : Data;
```

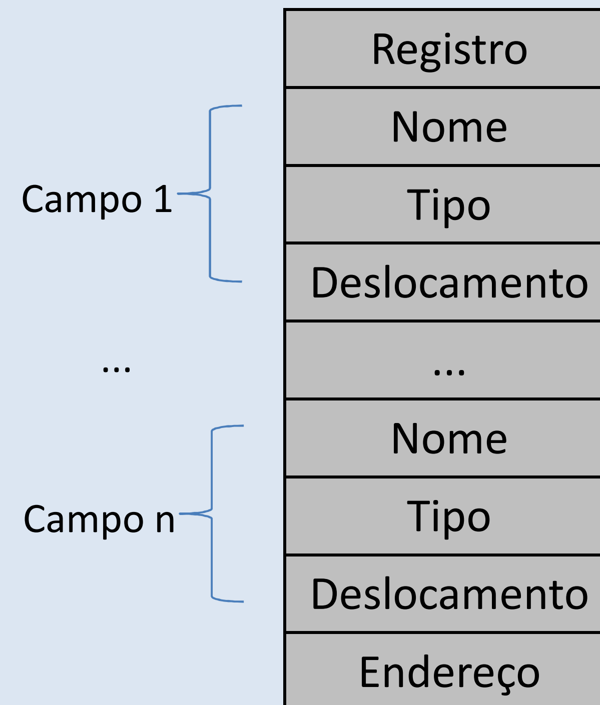
Tipos Registro

avaliação

- Registros e matrizes são fortemente relacionados com formas estruturais.
  - Matrizes são utilizadas quando todos os valores dos dados são do mesmo tipo e são processados da mesma forma
  - Registros são utilizados quando os dados são heterogêneos e os campos diferentes não são processados da mesma maneira. Os campos também não precisam ser processados em uma ordem particular

Tipos Registro implementação

- Os campos são armazenados em posições de memória adjacentes
  - O tamanho dos campos também (geralmente) são diferentes → método de acesso diferente do utilizado em matrizes
  - Utiliza-se um endereço de deslocamento relativo ao início do registro, que é associado a cada um dos seus campos



Tipos União

- Uma **união é um tipo que pode** armazenar diferentes valores de tipo durante a execução do programa
- A declaração de uma **union** é similar à declaração de um **registro**.
  - A diferença é que com uma **struct (registro)** é alocado de uma vez espaço suficiente para todos os objetos, enquanto que com uma **union** só é alocado espaço para o maior dos objetos que a compõem

Tipos União
exemplo

```
union Valor{
int ivalor;
double dvalor;
char cvalor;
}val;
```

- significa que **val** poderá armazenar ou um int, ou um double, ou um char.

Tipos União
avaliação

- Uniões são construções potencialmente inseguras
  - Não permite verificação de tipos
- Java e C# não suportam uniões
  - Reflexo da crescente preocupação com a segurança em linguagens de programação

```
union Valor{
int ivalor;
double dvalor;
char cvalor;
}val;
```

Tipos Ponteiro

- Um tipo *ponteiro* é aquele em que as variáveis têm uma faixa de valores que consistem em endereços de memória e um valor especial, *nil* (que diz que o ponteiro não pode ser usado atualmente para referenciar uma célula de memória)
- Oferece o poder de endereçamento indireto
- Oferece uma alternativa para gerenciar o endereçamento dinâmico, chamado de **monte** (heap)
- Um ponteiro pode ser usado para acessar uma posição na área onde o armazenamento é dinamicamente alocado, o qual é chamado de monte (*heap*)

letra (0x240ff5f)

‘a’

p (0x240ff58)

0x240ff5f

.

.

.

A variável letra é do tipo char

A variável p é um apontador

Neste caso, dizemos que p “aponta” para letra, sendo possível ler e alterar o conteúdo de letra via p.

Tipos Ponteiro
questões de projeto

- Quais é o escopo e o tempo de vida de uma variável do tipo ponteiro?
- Qual é o tempo de vida de uma variável dinâmica no monte?
- Os ponteiros são usados para gerenciamento de armazenamento dinâmico, endereçamento indireto ou ambos?
- A linguagem deveria suportar tipos ponteiro, tipos de referência ou ambos?

Tipos Ponteiro

- As variáveis alocadas dinamicamente no **monte** são chamadas de **variáveis dinâmicas do monte (*heap*)**
  - Não tem identificadores associadas a elas
  - Acessadas a partir de referências

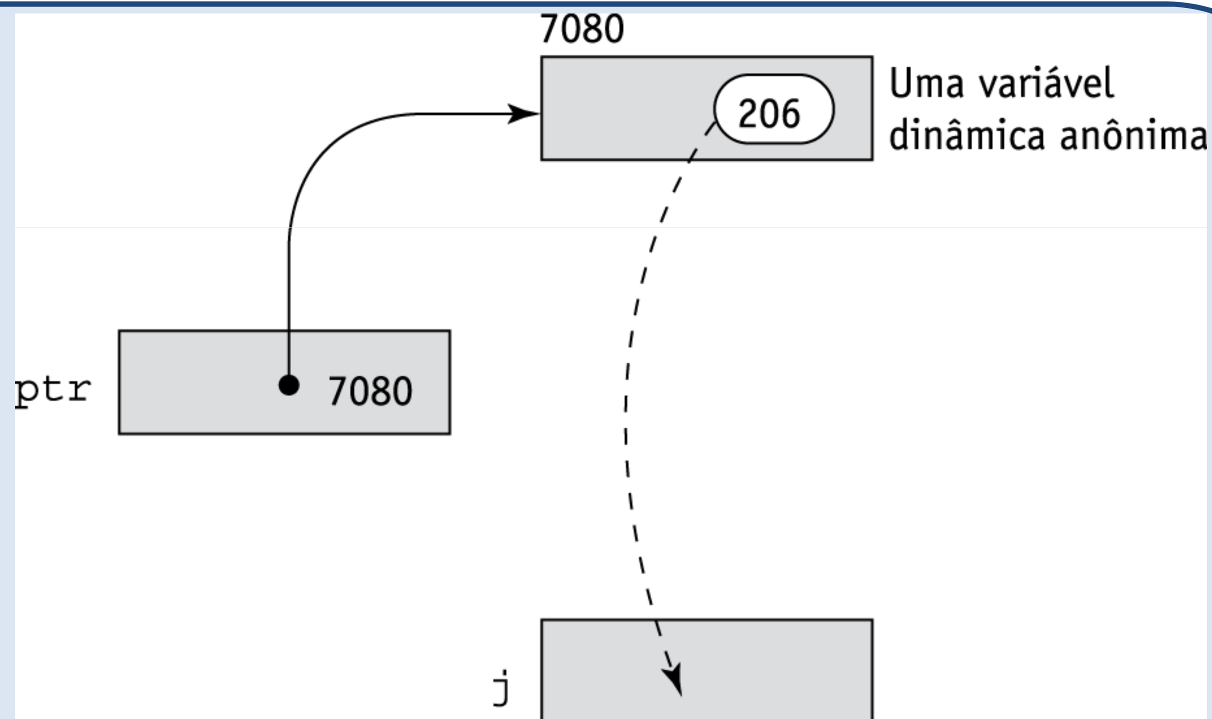
Tipos Ponteiro
operações com ponteiros

- Duas operações fundamentais
  - Atribuição (alocação→ definiar/inicializar variável) e desreferenciamento (desalocação)
- Atribuição é usada para fixar o valor de uma variável de ponteiro em um endereço útil
- Desreferenciamento referencia o valor da célula de memória (não apenas o endereço)
  - Desreferenciamento pode ser implícito ou explícito
  - C++ usa uma operação explícita asterisco (*)

```
j = *ptr
```

  - fixará j ao valor ao valor alocado em ptr

Tipos Ponteiro
Ilustração de atribuição de ponteiro



- A operação de atribuição **j = *ptr**

Tipos Ponteiro
problemas com ponteiros

- **Ponteiros soltos** ou **referência solta**
  - Ponteiro que contém o endereço de uma variável dinâmica do monte já **liberada**
  - Perigo:
    - A posição sendo apontada pode ter sido realocada para alguma outra variável dinâmica no monte nova
      - Se forem de tipos diferentes, como fica a verificação de tipos?!?
    - Se o ponteiro solto é usado para modificar a variável dinâmica no monte, o valor da nova variável será destruído
    - ...
  - Vazamento de memória

Tipos Ponteiro
problemas com ponteiros

- **Ponteiros soltos** ou **referência solta**

```
int * arrayPtr1;
int * arrayPtr2 = new int[100];
arrayPtr1 = arrayPtr2;
delete [] arrayPtr2;
```

- Agora arrayPtr1 é solto, pois o armazenamento no monte para o qual ele apontava foi liberado

Tipos Ponteiro
em C e C++

- Extremamente flexíveis, mas devem ser usados com muito cuidado
- Podem apontar para qualquer variável, independentemente de onde ela estiver alocada
- Usado para o gerenciamento de armazenamento dinâmico e endereçamento

Tipos Ponteiro
em C e C++

- A aritmética de ponteiros é também possível de algumas formas restritas

- C e C++ incluem ponteiros do tipo **void ***, que podem apontar para valores de quaisquer tipos. São, para todos os efeitos, ponteiros genéricos

Tipos Ponteiro
aritmética em C e C++

```
int lista[100];
int *p;
p = lista; //atribui o endereço de lista[0] a p
           //endereço base de uma matriz
```

Exemplos:

`*(p+5)` é equivalente a `lista[5]` e `p[5]`

`*(p+i)` é equivalente a `lista[i]` e `p[i]`

Tipos Ponteiro
tipos de referência

- Uma variável de *tipo de referência* é similar a um ponteiro, com uma diferença importante e fundamental:
  - um ponteiro se refere a um endereço em memória, enquanto uma referência se refere a um objeto ou a um valor em memória

Tipos Ponteiro
tipos de referência

- Em Java, variáveis de referência são estendidas da forma de C++ para uma forma que as permitem substituírem os ponteiros inteiramente
- Variáveis do *tipo de referência* são especificadas precedidas de &. Ex:

```
int resultado = 0;
int &ref_resultado = resultado;
...
ref_resultado = 100;
```

Tipos Ponteiro

avaliação

- Ponteiros soltos e lixo são problemas, tanto quanto o gerenciamento do monte (*heap)*

- Ponteiros são como a instrução *goto* - que aumenta a faixa de células que podem ser acessadas por uma variável

- Ponteiros e referências são necessários para estruturas de dados dinâmicas – não podemos projetar uma linguagem sem eles

Tipos Ponteiro
representação de ponteiros

- Em computadores de grande porte, ponteiros são valores únicos armazenados em células de memória

- Nos primeiros microcomputadores baseados em microprocessadores Intel, os endereços possuíam duas partes: um segmento e um deslocamento

Tipos Ponteiro

Solução para o problema dos ponteiros soltos ➔ coleta de lixo

- Gerenciamento do *Heap*
  - Processo em tempo de execução complexo
  - Duas abordagens para a coleta de lixo
    - Contadores de referências (abordagem ansiosa): a recuperação da memória é incremental e é feita quando células inacessíveis são criadas
    - Marcar-e-varrer (abordagem preguiçosa): a recuperação ocorre apenas quando a lista de espaços disponíveis se torna vazia

Tipos Ponteiro
Contador de referência

- **Contadores de referência**: usa um contador (inteiro) para cada elemento para servir de contador de referências

  - Esse contador indica a quantidade de ponteiros ou referências a esse elemento

    - ✓ Sempre que um ponteiro a um elemento **A** é retirado e passa a ser redirecionado para apontar para um outro elemento **B**, então o contador de referências de A deve ser **diminuído** de **1**, ao passo que o contador de referências de **B** deve ser **acrescido** de **1**.

Tipos Ponteiro
Contador de referência

- **Contadores de referência**:
  - Quando o contador de referências de um elemento atinge o **valor zero,** o elemento **é lixo** por definição, pois nenhum outro elemento estaria apontando a ele.
  - O elemento é então recolhido ao espaço livre. Caso esse elemento aponte para outros elementos, esses outros elementos apontados devem ter seus contadores de referências atualizados, podendo resultar em lixo também caso esses contadores atingem zero com a atualização.
  - Com isso o lixo pode ser identificado e recolhido assim que ele surgir.

Tipos Ponteiro
Contador de referência

- **Contadores de referência**: a recuperação de memória é incremental e é feita quando células inacessíveis são criadas

  - Desvantagens: o espaço necessário para os contadores é significativo, algum tempo de execução é necessário e complicações quando uma coleção de células é conectada circularmente

  - Vantagens: é intrinsecamente incremental, suas ações são intercaladas com aquelas da aplicação, então ela nunca causa demoras significativas na execução da aplicação

Tipos Ponteiro
marcar – e – varrer

- O sistema de tempo de execução aloca células de armazenamento conforme solicitado e desconecta ponteiros de células conforme a necessidade; marcar-varrer começa

  - Cada célula do monte possui um bit ou campo indicador extra que é usado pelo algoritmo de coleta

  - Todas as células no monte têm seus indicadores configurados para indicar que eles são lixo

Tipos Ponteiro
marcar – e – varrer

- Cada ponteiro no programa é rastreado no monte, e todas as células alcançáveis são marcadas como não sendo lixo

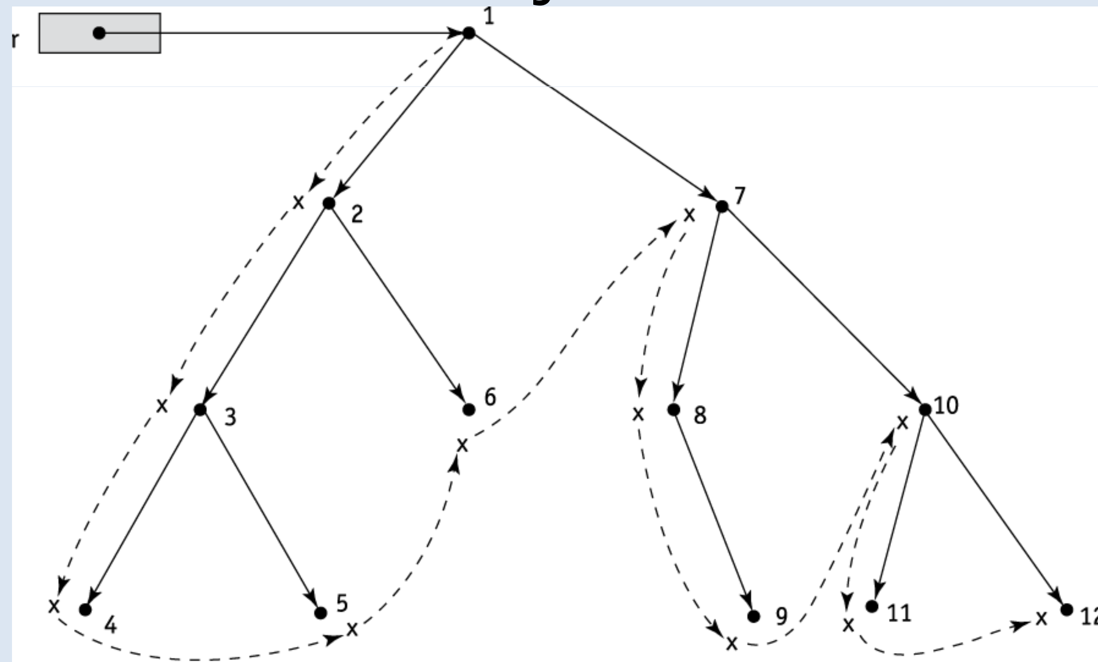
- Todas as células retornam para a lista de espaço disponível

- Desvantagens: Quando feita, causava atrasos na execução da aplicação

unesp

Tipos Ponteiro
marcar – e – varrer

- Algoritmo de marcação



Linhas tracejadas mostram a ordem de marcação dos nós

Verificação de tipos

- *Verificação de tipos* é a atividade de garantir que os operandos de um operador são de tipos compatíveis
- Um *tipo compatível* é um que legal para o operador ou é permitido a ele, dentro das regras da linguagem, ser implicitamente convertido pelo código gerado pelo compilador (ou pelo interpretador) para um tipo legal
- Essa conversão automática é chamada de *coerção*
- Um erro de tipo é a aplicação de um operador a um operando de um tipo não apropriado

Verificação de tipos

- Se todas as vinculações são estáticas, a verificação de tipos pode ser feita praticamente sempre de maneira estática
- Se as vinculações de tipo são dinâmicas, a verificação de tipos deve ser dinâmica
- Uma linguagem de programação é fortemente tipada se os erros de tipo são sempre detectados
- Vantagem de tipagem forte: permite a detecção da utilização indevida de variáveis que resultam em erros de tipo

Verificação de tipos

- **Tipagem forte**
  - Exemplos de linguagens:
    - FORTRAN 95 não é fortemente tipada
    - C e C++ também não: ambas incluem tipos união, que não são verificados em relação a tipos
    - Ada é quase fortemente tipada
    - (Java e C# são similares a Ada)

Verificação de tipos

- As regras de *coerção* de uma linguagem têm efeito importante no valor da verificação de tipos - eles podem enfraquecer consideravelmente

- Java e C# têm cerca de metade das coerções de tipo em atribuições que C++. Então, sua detecção de erros é melhor do que a de C++, mas não é nem perto de ser tão efetiva quanto a de Ada

Equivalência de tipos
por nome

- *Equivalência de tipos por nome* significa que duas variáveis são equivalentes se elas são definidas na mesma declaração ou em declarações que usam o mesmo nome de tipo
- Fácil de implementar, mas é mais restritiva:
  - Subfaixas de tipos inteiros não são equivalentes a tipos inteiros
  - Parâmetros formais devem ser do mesmo tipo que os seus correspondentes parâmetros reais

Equivalência de tipos

por estrutura

- *Equivalência de tipos por estrutura* significa que duas variáveis têm tipos equivalentes se seus tipos têm estruturas idênticas

- Mais flexível, mas mais difícil de implementar

Equivalência de tipos

- Considere o problema de dois tipos estruturados:
  - Dois tipos de registros são equivalentes se eles são estruturalmente o mesmo, mas usarem nomes de campos diferentes?
  - Dois tipos de matrizes são equivalentes se eles são o mesmo, exceto se os índices são diferentes?
    - (por exemplo, [1..10] e [0..9])
  - Dois tipos de enumeração são equivalentes, se seus componentes são escritos de maneira diferente?
  - Com o tipo de equivalência estrutural, não é possível diferenciar os tipos de a mesma estrutura?

Resumo

- Os tipos de dados de uma linguagem são uma grande parte do que determina o estilo e a utilidade de uma linguagem

- Os tipos de dados primitivos da maioria das linguagens imperativas incluem os tipos numéricos, de caracteres e booleanos

- Os tipos de enumeração e de subfaixa definidos pelo usuário são convenientes e melhoram a legibilidade e a confiabilidade dos programas

- Matrizes fazem parte da maioria das linguagens de programação

- Ponteiros são usados para lidar com a flexibilidade e para controlar o gerenciamento de armazenamento dinâmico

Exercícios

- Questões de revisão
  - 3, 5, 6, 8, 12, 18, 26, 36, 37 e 39
- Conjunto de problemas
  - 2, 5, 10, 12 e 21
- Exercícios de programação
  - 7