FACAMP
FACULDADES DE CAMPINAS

Linguagens de Programação

Aula 6

Tipos de dados: primitivos, cadeias, matrizes, registros, uniões e ponteiros

2º semestre de 2019
Prof José Martins Junior

Introdução

- Tipo de dado
  - Domínio de valores de dados e operações predefinidas sobre eles
- Grupos
  - Básicos, que representam unidades de tipos primitivos (numéricos, p.e), e outros (dependendo da linguagem)
  - Estruturados (não escalares), que podem ser
    - Arranjos homogêneos (vetores e matrizes)
    - Registros, ou estruturas compostas
- Um descritor qualifica uma variável
  - Contém atributos como tipo, endereço e tamanho da alocação (offset)
- Tipagem
  - Forte: o tipo é verificado pela LP em tempo de compilação
  - Fraca: permite-se a mudança de tipo da variável durante a execução

Tipos de dados primitivos

- São tipos oferecidos pela plataforma (hardware e SO)
- **Inteiro**
  - Representam valores decimais inteiros, sem casas decimais
  - Há vários tipos e tamanhos
    - Em C: char, short, int, long (ou long int), long long
    - Em Java: byte, short, int, long
  - Bit de sinal
    - Em Java, e se nada for dito em C, os tipos inteiros reservam o MSB para representar o sinal, ou seja, metade do intervalo para cada grupo
  - Em Python, o tamanho longo pode ser ilimitado
    - Deve ser expresso com L no final
    - Exemplo: 243725839182756281923L
    - Além disso, faz conversão direta, quando a variável intermediária com o resultado superar operandos inteiros

Ponto flutuante

- Modelam números reais
  - São representações limitadas (aproximações) para muitos valores reais
  - Operações aritméticas geram perda de precisão
  - IEEE Floating-Point Standard 754 (padrão p/ computadores modernos)
- **Simples precisão (float)**
  - Tamanho padrão geralmente armazenado em 4 bytes de memória
- **Dupla precisão (double)**
  - Normalmente ocupa o dobro de armazenamento (8 bytes)
  - Fornece ao menos o dobro do número de bits de fração

Tipos complexo e decimal

- Algumas LPs suportam o tipo complexo (Fortran, Python, p.e)
  - Valores são representados como pares ordenados de valores de ponto flutuante
  - Em Python, a parte imaginária de um literal complexo é especificada seguindo a por j ou J
    - Exemplo: (7 + 3j)
- Computadores de grande porte geralmente preveem decimal
  - Decimais armazenam um número fixo de dígitos decimais, com o ponto decimal em uma posição fixa no valor
  - São suportados por LPs como COBOL, VB e C# e são úteis para processamentos de dados de negócios

Tipo booleano

- Representam apenas dois elementos, verdadeiro ou falso
  - Em C89 (e também em ANSI C), qualquer valor diferente de zero é verdadeiro, e falso se zero
    - Em linguagens subsequentes, como Java e C#, isso não ocorre
  - Apesar de necessitarem de apenas um único bit, geralmente são armazenados em células de um byte na memória
    - Byte é a menor célula eficientemente endereçável

Tipos e tamanhos em Java

		Valores possíveis			
Tipos	***Primitivo***	***Menor***	***Maior***	***Valor Padrão***	***Tamanho***
Inteiro	byte	-128	127	0	8 bits
	short	-32768	32767	0	16 bits
	int	-2.147.483.648	2.147.483.647	0	32 bits
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits
Caractere	char	0	65535	\0	16 bits
Booleano	boolean	false	true	false	1 bit

Tipos e tamanhos em C

Tipo	Tam (bits)	Formato scanf	Início	Fim
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	16	%d	-32768	32767
unsigned int	16	%u	0	65535
signed int	16	%i	-32768	32767
short int	16	%hi	-32768	32767
unsigned short int	16	%hu	0	65535
signed short int	16	%hi	-32768	32767
long int	32	%li	-2147483648	2147483647
signed long int	32	%li	-2147483648	2147483647
unsigned long int	32	%lu	0	4294967295
long long	64	%lld	-9223372036854775807	9223372036854775807
unsigned long long	64	%llu	0	18446744073709551616

Obs.: o tipo int varia, de acordo com a plataforma. Em compiladores 16 bits, tem o tamanho do short (como acima); em compiladores 32 bits, o tamanho do long

Conversão implícita e explícita

- Os tipos numéricos podem ser convertidos entre si
  - Conversão direta (implícita)
    - Se um tipo de menor tamanho for atribuído a um parâmetro de função ou variável onde se pede um maior
    - Ex. em Java:

```
int n = 10;
double d = n;  //d receberá 10.000000
```

  - Conversão indireta ou casting (explícita)
    - Quando se deseja converter um tipo maior para um menor (atribuição, parâmetro, etc) deve-se explicitar (casting) a conversão
    - Implica em perda de precisão
    - Ex. em Java:

```
double d = 10.5;
int n = (int) d;  //n receberá 10
```

Caractere

- Representam um único símbolo, ou caractere
  - Em muitas linguagens, são armazenados como números
  - Padrão ASCII codifica valores de 0 a 255 (1 byte) para representar dígitos alfanuméricos, pontuações, entre outros caracteres
  - Exemplo em C:

```
char c = 'a';
printf("%c\n", c); //imprimirá a
printf("%d\n", c); //imprimirá 97
```

  - Em 1991, o consórcio Unicode criou o padrão UCS-2, como extensão do ASCII, com tamanho de 16 bits,
    - Permitiu incluir símbolos da maioria das linguagens naturais do mundo
    - Java foi a primeira linguagem amplamente usada a adotar o Unicode
    - Desde então, ele foi adotado em JavaScript, Python, Perl e C#

Tabela ASCII (e estendida)

Regular ASCII Chart (character codes 0 - 127)

000		(nul)	016	►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001	☺	(soh)	017	◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002	☻	(stx)	018	↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥	(etx)	019	‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦	(eot)	020	¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005	♣	(enq)	021	§	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♠	(ack)	022	▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	•	(bel)	023	↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	◘	(bs)	024	↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	○	(tab)	025	↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010		(lf)	026	→	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	♂	(vt)	027	←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	♀	(np)	028	∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	¦
013		(cr)	029	↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♫	(so)	030	▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	☼	(si)	031	▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂ (del)

Extended ASCII Chart (character codes 128 - 255)

128	Ç	143	Å	158	×	172	¼	186	‖	200	╚	214	í	228	õ	242	=
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	Î	229	Õ	243	¾
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	Ï	230	µ	244	¶
131	â	146	Æ	161	í	175	»	189	¢	203	╦	217	┘	231	Þ	245	§
132	ä	147	ô	162	ó	176	░	190	¥	204	╠	218	┌	232	Þ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Ú	247	¸
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Û	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	¤	221	¦	235	Ù	249	¨
136	ê	151	ù	166	ª	180	┤	194	┬	208	ð	222	Ì	236	ý	250	·
137	ë	152	ÿ	167	º	181	Á	195	├	209	Ð	223	▀	237	Ý	251	¹
138	è	153	Ö	168	¿	182	Â	196	─	210	Ê	224	Ó	238	¯	252	³
139	ï	154	Ü	169	®	183	À	197	┼	211	Ë	225	ß	239	´	253	²
140	î	155	ø	170	¬	184	©	198	ã	212	È	226	Ô	240	-	254	■
141	ì	156	£	171	½	185	╣	199	Ã	213	ı	227	Ò	241	±	255	
142	Ä	157	Ø														

Cadeias de caracteres

- Sequências de caracteres (strings) que formam palavra e frases
  - C e C++ usam vetores para armazenar cadeias de caracteres
  - Exemplo: `char str[] = "apples";`
    - Cria um vetor com 7 posições e na última insere um terminador `\0`
    - O terminador é de suma importância para as funções que manipulam strings em C, como strlen, strcat, strtok, e até mesmo o formato %s no printf
  - Java e C# possuem classes para representar strings e outras mais para complementar suas manipulações (StringBuilder, StringTokenizer, p.e)
    - As strings em Java e C# não têm tamanho prefixado (declaração), como em C
  - A maioria das linguagens permitem operações com strings usando expressões regulares

Tipo enumeração

- Permite definição de conjunto restrito de possíveis opções
  - São como constantes agrupadas e indexadas
  - Exemplo em Java:

```
enum Day {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
Day d = Day.Mon;
System.out.println(d.Mon); //imprimirá Mon
System.out.println(d.ordinal()); //imprimirá 0
```

Vetores e matrizes

- São agregados homogêneos de dados
  - Cada elemento é referenciado por seu índice (posição)
    - Em várias linguagens, como C e Java, o índice começa em 0
    - Em outras, como Pascal e VB, pode-se definir o índice
    - Muitas linguagens (como Java) alertam sobre acessos fora do intervalo do índice, gerando um erro
    - Cuidado! Em C, pode-se acessar índices inválidos para o limite do arranjo
  - Na maioria das LPs são alocados estaticamente na declaração
    - Assume um lugar fixo (tamanho e endereço) na memória
    - Para fins de desempenho, a maioria das linguagens passam a referência do endereço, quando uma matriz é fornecida como parâmetro de função
    - Em algumas, permite-se cópia, realocação e redimensionamento
  - Em muitas LPs, todos os elementos dever ser do mesmo tipo
    - Em outras, como JavaScript, permite-se a combinação de elementos de tipos diferentes

Acesso a posições da matriz

- As matrizes possuem um índice por dimensão que permite buscar cada dado alocado na memória
  - A combinação dos índices indica a posição relativa do item na matriz
  - Em Java, cada índice vai de **0** até **n-1** (onde **n** é o **tamanho** da dimensão)
  - Para a matriz mXn (m=3 linhas, por n=4 colunas)

```
int[][] matriz2 = {{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

- Possui **3** linhas, com índice de **0** a **2** (**m-1**)
- Possui **4** colunas, com índice de **0** a **3** (**n-1**)

matriz2

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
matriz2[1][2] = 17;
```

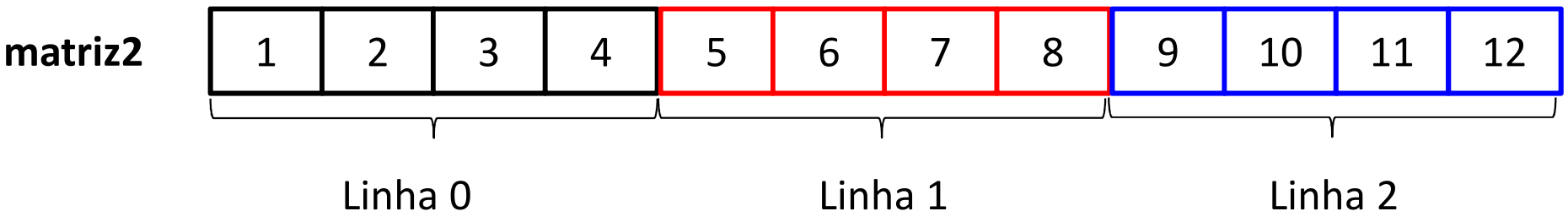
- Atribui o valor 17 ao elemento na segunda linha e terceira coluna da matriz (onde havia o valor 7)

```
System.out.println(matriz2[2][1]);
```

- Acessa e imprime o elemento da terceira linha e segunda coluna da matriz (o valor 10)

Posição na memória

- Apesar de manipularmos matrizes com diferentes índices, para linha e coluna, a alocação na memória é um bloco sequencial (como fosse um vetor com a mesma quantidade de itens)
  - O índice nesse “vetor” corresponde a **(i * colunas + j)**
  - Onde
    - **i** é o índice da linha desejada
    - **colunas** corresponde à quantidade de colunas da matriz
    - **j** é o índice da coluna desejada
- Exemplo



Registros

- Registros são conjuntos de dados não indexados
  - Podem conter campos nomeados, de diferentes tipos
  - Usados para representar dados compostos (entidades de dados)

Exemplo em COBOL:

```
01 EMPLOYEE-RECORD.
   02 EMPLOYEE-NAME.
      05 FIRST    PICTURE IS X(20).
      05 MIDDLE   PICTURE IS X(10).
      05 LAST     PICTURE IS X(20).
   02 HOURLY-RATE PICTURE IS 99V99.
```

Referência a campo em COBOL:

```
MIDDLE OF EMPLOYEE-NAME OF EMPLOYEE-
RECORD
```

Exemplo em ADA:

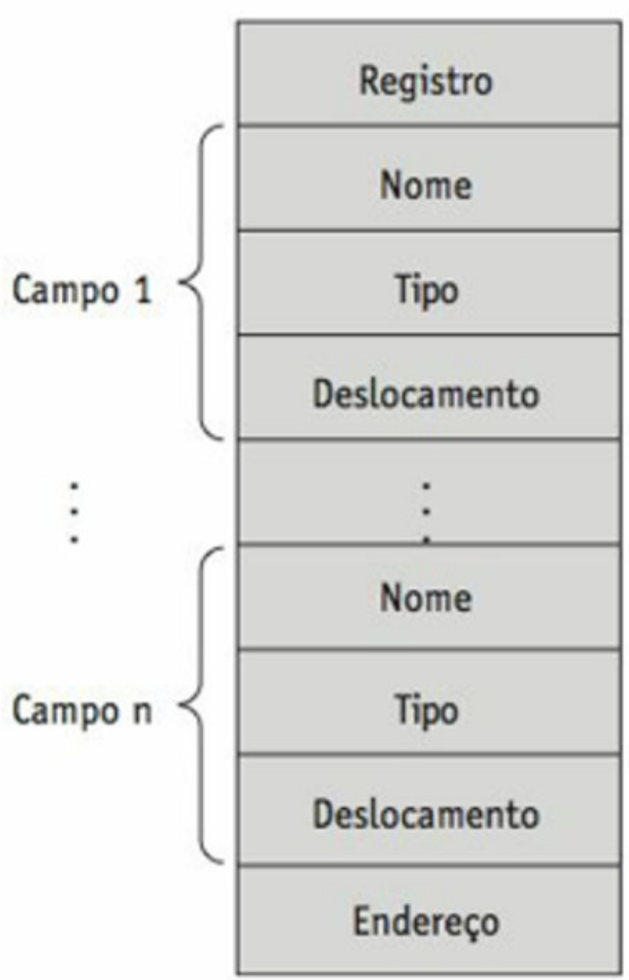
```
type Employee_Name_Type is record
    First : String (1..20);
    Middle : String (1..10);
    Last : String (1..20);
end record;
type Employee_Record_Type is record
    Employee_Name: Employee_Name_Type;
    Hourly_Rate: Float;
end record;
Employee_Record: Employee_Record_Type;
```

Referência a campo em ADA:

```
Employee_Record.Employee_Name.Middle
```

Uniões

- Permite variáveis armazenarem valores de diferentes tipos em vários momentos durante a execução de um programa
  - No descritor de uma união, todos os campos são descriminados quanto ao nome, tipo e offset (deslocamento)
  - Em sua instância, reserva-se quantidade de memória equivalente o suficiente para armazenar o campo de maior tipo envolvido
  - Geralmente, a referência aos tipos não é checada em tempo de compilação



Exemplo de união em C

```
union flexType {
    int intEl;
    double doubleEl;
};
union flexType el1;
int x;
el1.doubleEl = 27.5;
x = el1.doubleEl;
printf("%d\n", x);
printf("%u\n", sizeof(el1));
```

- Quando executado, imprimirá

```
27
8
```

Estruturas em C

- Estruturas em C são equivalentes a registros em outras LPs
  - Sua implementação deriva de uniões
  - É reservada quantidade de memória em bytes equivalente ao múltiplo do maior tipo envolvido, sendo maior ou igual a soma dos campos
  - Exemplo:

```
struct exemplo {
    int cod;
    double val
};
struct exemplo x;
x.cod = 1;
x.val = 5.5;
printf("Cod: %d\tVal: %f\n", x.cod, x.val);
printf("Tamanho: %u\n", sizeof(x));
```

  - Imprimirá:

```
Cod: 1  Val: 5.500000
Tamanho: 16
```

Ponteiros

- Variáveis especiais que contém o endereço de outras variáveis
  - Muitas linguagens, como Java e Python, escondem ponteiros
  - Exemplo em C:

```
struct exemplo *p;      //declaração de ponteiro
p = &x;                 //p recebe o endereço de x
printf("Cod: %d\tVal: %f\n", p->cod, p->val);
```

  - Imprimirá:

```
Cod: 1  Val: 5.500000
```

Alocação dinâmica

- Alocação dinâmica de memória
  - Alocação de espaço na memória, em tempo de execução
  - Usada para guardar dados simples ou compostos (registros)
  - Função em C: `void *malloc(size_t size);`
    - Retorna um ponteiro para void que pode ser convertido para o tipo de ponteiro apropriado para o registro armazenado
  - Exemplo em C

```
struct exemplo *p2;
p2 = (struct exemplo *) malloc(sizeof(struct exemplo));
```

Aritmética de ponteiros

- As operações aritméticas sobre ponteiros são diferentes
  - Geralmente, o valor unitário, aplicado a um incremento, é a quantidade de bytes referentes ao tipo do ponteiro
  - Exemplo em C:

```
int v[5] = {10, 20, 30, 40, 50}, i;
int *pv = v;                    //vetores são tratados como ponteiros
while(*pv != 50) {
  printf("%d\t", *pv);          //imprime o conteúdo apontado por pv
  pv++;                         //soma 4 ao endereço contido em pv
}
printf("%d\n", *pv);
```

  - Imprimirá:

```
10      20      30      40      50
```

Bibliografia

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