Programação de Computadores II

Cap. 2 – Expressões na linguagem C

Livro: Waldemar Celes, Renato Cerqueira, José Lucas Rangel. Introdução a Estruturas de Dados, Editora Campus (2004)

Slides adaptados dos originais dos profs.: Marco Antonio Casanova e Marcelo Gattass (PUC-Rio)

Tópicos de hoje:

- Bits, Bytes e Palavras
- Variáveis e constantes
- Operadores e expressões

Bits, Bytes e Palavras

- Organização da memória
  - Bit:
    - menor unidade
    - armazena 0 ou 1
  - Byte:
    - seqüência de 8 bits
  - Palavra:
    - sequência de bytes
    - número de bytes da palavra varia conforme a arquitetura do computador

		0	1	2	3	4	5	6	7
1	0	0	1	1	1	0	0	1	0
	1	1	1	0	0	1	1	1	0
	2	0	1	1	1	0	0	1	0
	3	0	0	0	0	0	0	0	0
2	0	1	1	1	0	1	0	1	0
	1	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1
	3	0	0	0	0	0	0	0	0

Variáveis e Constantes

Questão 1:

Suponha que:

a = 3

b = a / 2

c = b + 3.1

Qual é o valor de c?

- ❑ c = 4.6
- ❑ c = 4.1
- ❑ c = 4
- ❑ Nenhuma das opções acima
- ❑ Não é possível determinar o valor de c

Representando números inteiros com bits

- Números inteiros: 27, por exemplo,

$$27 = 2\times10^1 + 7\times10^0$$ Base decimal (10)

$$27 = 16+8+2+1$$

$$27 = 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0$$

Base binária (2)

0001 1011

Números inteiros num Byte

Bits	Valor
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Byte	Valor
00000000	0
00000001	1
00000010	2
00000011	3
…	…
11111101	253
11111110	254
11111111	255

(Base hexadecimal)

Byte	Valor	Byte	Valor
0000	0	1000	8
0001	1	1001	9
0010	2	1010	10
0011	3	1011	11
0100	4	1100	12
0101	5	1101	13
0110	6	1110	14
0111	7	1111	15

Byte	Valor	Byte	Valor
0000	**0**	1000	**8**
0001	**1**	1001	**9**
0010	**2**	1010	**A**
0011	**3**	1011	**B**
0100	**4**	1100	**C**
0101	**5**	1101	**D**
0110	**6**	1110	**E**
0111	**7**	1111	**F**

`'\x61'= '0110 0001'`

Little vs Big Endian

Byte3 Byte2 Byte1 Byte0

Processadores Intel
(“Little Endian”)

Base address+0 Byte0
Base address+1 Byte1
Base address+2 Byte2
Base address+3 Byte3

Processadores Motorola
(“Big Endian”)

Base address+0 Byte3
Base address+1 Byte2
Base address+2 Byte1
Base address+3 Byte0

Maior inteiro representável em 32 bits =2^{32}-1= +4.294.967.295

Como representar um número real?

$$123.456 = \underbrace{.123456}_{\text{mantissa}} \times 10^{\overbrace{3}^{\text{expoente}}}$$

IEEE standart 754 Floating Point

Precisão simples (**`float`**)

Precisão dupla (**`double`**)

Notação de constantes numéricas

`32` ← tipo inteiro (depende do *default* da máquina)

`3.1416` ← tipo precisão dupla

`10.14e-3` ← tipo precisão dupla

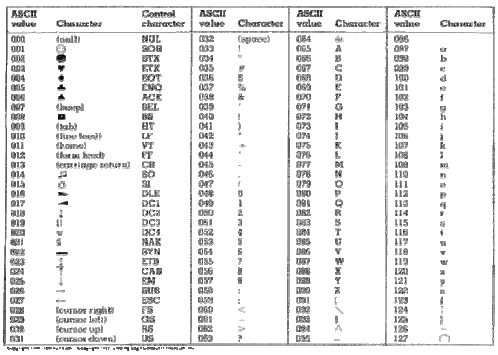
`3.01F` ← tipo precisão simples

Armazenando texto

01101111	01101001	00100000	01100001	01101110	01100001	00000000
111	105	32	97	110	97	0
'o'	**'i'**	**' '**	**'a'**	**'n'**	**'a'**	**'\0'**

"oi ana"

Como representar caracteres?



ASCII value	Character	Control character	ASCII value	Character	ASCII value	Character	ASCII value	Character
000	(nul)	NUL	032	(space)	064	@	096	`
001	☺	SOH	033	!	065	A	097	a
002	☻	STX	034	"	066	B	098	b
003	♥	ETX	035	#	067	C	099	c
004	♦	EOT	036	$	068	D	100	d
005	♣	ENQ	037	%	069	E	101	e
006	♠	ACK	038	&	070	F	102	f
007	(beep)	BEL	039	'	071	G	103	g
008	◘	BS	040	(	072	H	104	h
009	(tab)	HT	041	)	073	I	105	i
010	(line feed)	LF	042	*	074	J	106	j
011	(home)	VT	043	+	075	K	107	k
012	(form feed)	FF	044	,	076	L	108	l
013	(carriage return)	CR	045	-	077	M	109	m
014	♫	SO	046	.	078	N	110	n
015	☼	SI	047	/	079	O	111	o
016	►	DLE	048	0	080	P	112	p
017	◄	DC1	049	1	081	Q	113	q
018	↕	DC2	050	2	082	R	114	r
019	‼	DC3	051	3	083	S	115	s
020	¶	DC4	052	4	084	T	116	t
021	§	NAK	053	5	085	U	117	u
022	▬	SYN	054	6	086	V	118	v
023	↨	ETB	055	7	087	W	119	w
024	↑	CAN	056	8	088	X	120	x
025	↓	EM	057	9	089	Y	121	y
026	→	SUB	058	:	090	Z	122	z
027	←	ESC	059	;	091	[	123	{
028	(cursor right)	FS	060	<	092	\	124	|
029	(cursor left)	GS	061	=	093	]	125	}
030	(cursor up)	RS	062	>	094	^	126	~
031	(cursor down)	US	063	?	095	_	127	⌂

Como representar caracteres?

Ctrl	Dec	Hex	Char	Code
^@	0	00		NUL
^A	1	01		SOH
^B	2	02		STX
^C	3	03		ETX
^D	4	04		EOT
^E	5	05		ENQ
^F	6	06		ACK
^G	7	07		BEL
^H	8	08		BS
^I	9	09		HT
^J	10	0A		LF
^K	11	0B		VT
^L	12	0C		FF
^M	13	0D		CR
^N	14	0E		SO
^O	15	0F		SI
^P	16	10		DLE
^Q	17	11		DC1
^R	18	12		DC2
^S	19	13		DC3
^T	20	14		DC4
^U	21	15		NAK
^V	22	16		SYN
^W	23	17		ETB
^X	24	18		CAN
^Y	25	19		EM
^Z	26	1A		SUB
^[	27	1B		ESC
^\	28	1C		FS
^]	29	1D		GS
^^	30	1E	▲	RS
^_	31	1F	▼	US

Dec	Hex	Char
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Dec	Hex	Char
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Dec	Hex	Char
96	60	`
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	⌂

* ASCII code 127 has the code DEL. Under MS-DOS, this code has the same effect as ASCII 8 (BS). The DEL code can be generated by the CTRL + BKSP key.

Constante de caracteres e *strings*

'a' ← armazenado como 97

'1' ← armazenado como 49

1 ← constante inteira

"Maria" ← cadeia de caracteres

"Maria" = 'M' 'a' 'r' 'i' 'a' ' \0'

Variáveis e Constantes

- Tipos básicos na linguagem C:

Tipo	Tamanho	Menor valor	Maior valor
`char`	1 byte	-128	+127
`unsigned char`	1 byte	0	+255
`short int (short)`	2 bytes	-32.768	+32.767
`unsigned short int`	2 bytes	0	+65.535
`int (*)`	4 bytes	-2.147.483.648	+2.147.483.647
`long int (long)`	4 bytes	-2.147.483.648	+2.147.483.647
`unsigned long int`	4 bytes	0	+4.294.967.295
`float`	4 bytes	-10^{38}	$+10^{38}$
`double`	8 bytes	-10^{308}	$+10^{308}$

(*) depende da máquina, sendo 4 bytes para arquiteturas de 32 bits

Variáveis e Constantes

- Valor Constante:
  - armazenado na memória
  - possui um tipo, indicado pela sintaxe da constante

```
123          /* constante inteira do tipo “int” */
12.45        /* constante real do tipo “double” */
1245e-2     /* constante real do tipo “double” */
12.45F       /* constante real do tipo “float” */
```


Variáveis e Constantes

- Variável:
  - espaço de memória para armazenar um dado
  - não é uma variável no sentido matemático
  - possui um tipo e um nome
    - nome: identifica o espaço de memória
    - tipo: determina a natureza do dado

Variáveis e Constantes

- Declaração de variável:
  - variáveis devem ser explicitamente declaradas
  - variáveis podem ser declaradas em conjunto

```
int a;          /* declara uma variável do tipo int */
int b;          /* declara uma variável do tipo int */
float c;        /* declara uma variável do tipo float */

int d, e;       /* declara duas variáveis do tipo int */
```

Variáveis e Constantes

- Declaração de variável:
  - variáveis só armazenam valores do mesmo tipo com que foram declaradas

```
int a;        /* declara uma variável do tipo int */
a = 4.3;       /* a armazenará o valor 4 */
```


Variáveis e Constantes

- Variável com valor indefinido:
  - uma variável pode receber um valor quando é definida (inicializada), ou através de um operador de atribuição

```
int a = 5, b = 10;     /* declara e inicializa duas variáveis do tipo int */

float c = 5.3; /* declara e inicializa uma variável  do tipo float */
```


Variáveis e Constantes

- Variável com valor indefinido:
  - uma variável deve ter um valor definido quando é utilizada

```
int a, b, c; /* declara e inicializa duas variáveis do tipo int */
a = 2;
c = a + b;              /* ERRO: b contém "lixo" */
```


Operadores e Expressões

- Operadores:
  - aritméticos: **+ , - , * , / , %**
  - atribuição: **= , += , -= , *= , /= , %=**
  - incremento e decremento: **++ , --**
  - relacionais e lógicos: **< , <= , == , >= , > , !=**
  - outros

Operadores e Expressões

- Operadores aritméticos (**+ , - , * , / , %**):
  - operações são feitas na precisão dos operandos
    - o operando com tipo de menor expressividade é convertido para o tipo do operando com tipo de maior expressividade
    - divisão entre inteiros trunca a parte fracionária

```
int a;
double b, c;
a = 3.5;              /* a recebe o valor 3 */
b = a / 2.0;   /* b recebe o valor 1.5 */
c = 1/3 + b; /* 1/3 retorna 0 pois a operação será sobre inteiros */
              /* c recebe o valor de b */
```


Operadores e Expressões

- Operadores aritméticos (cont.):
  - o operador módulo, "%", aplica-se a inteiros
  - precedência dos operadores: * , / , - , +

```
x % 2          /* o resultado será 0, se x for par;
                  caso contrário, será 1 */

a + b * c / d  é equivalente a       (a + ((b * c) / d))
```

Operadores e Expressões

- Operadores de atribuição :
  (= , += , -= , *= , /= , %=)
  - C trata uma atribuição como uma expressão
    - a ordem é da direita para a esquerda
  - C oferece uma notação compacta para atribuições em que a mesma variável aparece dos dois lados

    var *op*= expr é equivalente a var = var *op* (expr)

```
i += 2;              é equivalente a      i = i + 2;
x *= y + 1;          é equivalente a      x =  x * (y +1);
```


Operadores e Expressões

- Operadores de incremento e decremento **(++ , --):**
  - incrementa ou decrementa de uma unidade o valor de uma variável
    - os operadores não se aplicam a expressões
    - o incremento pode ser antes ou depois da variável ser utilizada

      n++ incrementa n de uma unidade, depois de ser usado

      ++n incrementa n de uma unidade, antes de ser usado

```
n = 5;
x = n++;              /* x recebe 5; n é incrementada para 6 */
n=5;
x = ++n;              /* n é incrementada para 6; x recebe 6 */
a = 3;
b = a++ * 2;   / b termina com o valor 6 e a com o valor 4 */
```


Operadores e Expressões

- Operadores relacionais
  (< , <= , == , >= , > , !=):
  - o resultado será 0 ou 1 (não há valores booleanos em C)

```
int a, b;
int c = 23;
int d = c + 4;

c < 20         retorna 0
d > c          retorna 1
```


Operadores e Expressões

- Operadores lógicos (**&& , || , !**)
  - a avaliação é da esquerda para a direita
  - a avaliação pára quando o resultado pode ser conhecido

```
int a, b;
int c = 23;
int d = c + 4;

a = (c < 20) || (d > c);      /* retorna 1 */
                          /* as duas sub-expressões são avaliadas */
b = (c < 20) && (d > c);      /* retorna 0 */
              /* apenas a primeira sub-expressão é avaliada */
```


Operadores e Expressões

- *sizeof:*
  - retorna o número de bytes ocupados por um tipo

```
int a = sizeof(float)         /* armazena 4 em a */
```


Operadores e Expressões

- conversão de tipo:
  - conversão de tipo é automática na avaliação de uma expressão
  - conversão de tipo pode ser requisita explicitamente

```
float f;              /*  valor 3 é convertido automaticamente para "float" */
float f = 3;          /* ou seja, passa a valer 3.0F, antes de ser atribuído a f  */

int g, h;             /* 3.5 é convertido (e arredondado) para "int"  */
g = (int) 3.5;        /* antes de ser atribuído à variável g      */
h = (int) 3.5 % 2     /* e antes de aplicar o operador módulo "%" */
```

Entrada e Saída

- Função “printf”:
  - possibilita a saída de valores segundo um determinado formato

```
printf (formato, lista de constantes/variáveis/expressões...);
```

```
printf ("%d %g", 33, 5.3);

tem como resultado a impressão da linha:
33 5.3
```

```
printf ("Inteiro = %d   Real = %g", 33, 5.3);

com saída:
Inteiro = 33   Real = 5.3
```

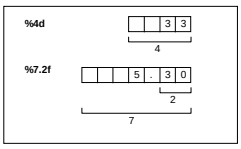

Entrada e Saída

- Especificação de formato:

%c	*especifica um char*
%d	*especifica um int*
%u	*especifica um unsigned int*
%f	*especifica um double (ou float)*
%e	*especifica um double (ou float) no formato científico*
%g	*especifica um double (ou float) no formato mais apropriado (%f ou %e)*
%s	*especifica uma cadeia de caracteres*

Entrada e Saída

- Especificação de tamanho de campo:



Entrada e Saída

- Impressão de texto:

```
printf("Curso de Estruturas de Dados\n");

exibe na tela a mensagem:
Curso de Estruturas de Dados
```


Entrada e Saída

- Função “scanf”:
  - captura valores fornecidos via teclado

```
    scanf (formato, lista de endereços das variáveis...);
```

```
int n;
scanf ("%d", &n);

valor inteiro digitado pelo usuário é armazenado na variável n
```


Entrada e Saída

- Especificação de formato:

%c	*especifica um char*
%d	*especifica um int*
%u	*especifica um unsigned int*
%f,%e,%g	*especificam um float*
%lf, %le, %lg	*especificam um double*
%s	*especifica uma cadeia de caracteres*

Entrada e Saída

- Função “scanf” (cont.):
  - caracteres diferentes dos especificadores no formato servem para cercar a entrada
  - espaço em branco dentro do formato faz com que sejam "pulados" eventuais brancos da entrada
  - %d, %f, %e e %g automaticamente pulam os brancos que precederem os valores numéricos a serem capturados

```
scanf ("%d:%d", &h, &m);

valores (inteiros) fornecidos devem ser separados pelo
caractere dois pontos (:)
```


Para casa:

Ler: Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004) Capítulo 2 – Expressões

Ler: Felipe Bergo. Representação de Números. (disponível no site)

Programar:

Escrever alguns programas que fixem as ideias apresentadas

Fazer a lista de exercícios 1