1º Semestre

Técnicas de Programação A

Luiz Fernando Carvalho

luizfcarvalhoo@gmail.com

Armazenamento de dados

- É preciso estabelecer um **tamanho padronizado** que as variáveis devem possuir na **memória** do computador;
  - Se não tivesse um tamanho padrão, como saberíamos onde começa e onde termina cada variável?

Armazenamento de dados

- Há cinco tipos **básicos** de dados em C:
  - Caracteres (`char`);
  - Inteiro (`int`);
  - Ponto flutuante (`float`);
  - Precisão dupla (`double`);
  - Sem valor (`void`).
- O tamanho e a faixa desses tipos de dados variam de acordo com o tipo de processor e com a implementação do compilador;
- Números inteiros **COSTUMAM** ser armazenados em espaços de 4 bytes, enquanto caracteres são armazenados em 1 byte.

Armazenamento de dados

- A menor unidade possível de informação é o ***bit***;
- Porém, a menor unidade de informação que pode ser de fato acessada é o **byte;**

1 byte = 8 bits

- No entanto, um **byte** sozinho não serve para muita coisa.

Armazenamento de dados

- Geralmente...

Tipo	Bytes	Faixa de valores
char	1	-128 a 127 ou 0 a 255
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
long int	4	Mesmo que int
unsigned long int	4	Mesmo que o unsigned int
float	4	1,2E-38 a 3,4E+38
double	8	2,3E-308 a 1,7E+308
long double	10	3,4E-4932 a 1,1E+4932

Tabela ASCII

Armazenamento de dados

- O tipo ***int*** geralmente tem o tamanho de 32 ***bits***, mas possui alguns subtipos com tamanho diferentes
  - *short int*, *long int* e *long long int*;

Quantidade de valores de acordo com bits b é dado por:

$2^b - 1$

Tipo	Bits	Bytes	Faixa	Versão unsigned
short	16	2	–32.768 a 32.767	65.535
int, long *	32	4	–2.147.483.648 a 2.147.483.647	4.294.967.295
long long	64	8	–9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	18.446.744.073.709.551.615

* O tipo **int** costuma ser do tipo **long**

Todos esses subtipos podem (e costumam) ser abreviados, tirando deles a palavra **int**. Por exemplo, **short int** geralmente é escrito apenas como **short**.

Armazenamento de dados

Tipo	Bits	Bytes	Maior valor
short	16	2	65.535
int, long	32	4	4.294.967.295
long long	64	8	18.446.744.073.709.551.615

Em C é possível armazenar valores tanto positivos e negativos (com sinal) quanto só positivos sem sinal. Usa-se a cláusula ***signed*** e ***unsigned int*** para designar com sinal e sem sinal, respectivamente.

Tipo	Maior valor
short (16 bits)	-32.768 a 32.767
int, long (32 bits)	-2.147.483.648 a 2.147.483.647
long long (64 bits)	-9.223.372.036.854.755.808 a 9.223.372.036.854.775.807

Usa-se o “%u” para imprimir valores no formato unsigned

Armazenamento de dados

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

A tabela ASCII que apresenta até o valor 255 (128 – 255) dos caracteres é chamada de estendida.

Armazenamento de dados

- Em C, os números reais (ponto flutuante) são definidos em dois tipo: ***float*** e ***double***.
- Os dois funcionam mais ou menos do mesmo jeito; a diferença entre eles é a precisão de cada um.
  - ***float*** *(4 bytes)* – precisão simples;
  - ***double*** *(8 bytes)* – precisão dupla;

- Podem ser escritos em forma de notação científica

3.14159e-7	1.234E+26	4.56e5
$(3{,}14159 \times 10^{-7})$	$(1{,}234 \times 10^{26})$	$(4{,}56 \times 10^{5})$

A letra **e** pode ser minúscula ou maiúscula

```
printf("%.nf", variavel);
```


n é a quantidade de casas decimais

Armazenamento de dados

Tipo	Intervalo	Precisão
float (32 bits)	10^{-45} ~ 10^{38}	6 decimais
double (64 bits)	10^{-324} ~ 10^{308}	15 decimais
long double (80 bits)	10^{-4950} ~ 10^{4932}	19 decimais

É necessário utilizar o código **%lf** em vez de apenas **%f** quando queremos ler com `scanf` uma variável ***double*** (ou **%Lf** para ***long double***). Nesse caso também é necessário utilizar esse mesmo código para o printf (em contraste com o ***double*** que continua usando o código **%f**).

Armazenamento de dados

- Exemplo

```
int main(){
   double num1, num2;

   num1 = 2.1e+21;
   num2 = 1.2e+21;

    printf("%lf\n", num1 + num2); //valor cheio

    printf("%.2lf", num1 + num2); //valor com apenas 2 casas decimais

    return 0;
}
```

Armazenamento de dados

- Como saber a quantidade de memória de um tipo de dado em C?
  - Função `sizeof`

```
int main()
{
    char a;
    int b;
    float c;
    double d;

    printf("%d %d %d %d", sizeof(a), sizeof(b), sizeof(c), sizeof(d));

    return 0;
}
```

A função `sizeof` tem como resultado um valor inteiro. Esse valor indica a quantidade de bytes que a variável está utilizando para ser armazenada na memória.

Conversão explícita de tipos

- As operações entre valores do tipo ponto flutuante ocorrem normalmente como as de números inteiros;
- Em operações mistas, o inteiro é convertido (momentaneamente) para ponto flutuante.

```
float x;
x = 1/7;
printf("%f", x);
```

```
0.000000
```

Divisão entre inteiros, gera um resultado inteiro e esse valor é guardado numa variável **float** (convertido para float)

```
float x;
x = 1.0/7;
x = 1/7.0;
x = 1.0/7.0;
printf("%f", x);
```

```
0.142857
```

Quem decide a operação são os operandos

Como efetuar a operação com números que não são conhecidos *a priori?*

Conversão explícita de tipos

- A conversão explícita de tipos de variáveis é chamada de *casting*.

(novo_tipo) **variável_ou_valor**

```
int a, b;
float x;

/* leitura dos números */

x = (float)a / b;
printf("%f\n", x);
```

O valor da variável **a** é convertido para **float**, logo, a operação é mista (**float** e **int**). Portanto, o resultado é um **float.**

Obs.: O valor é convertido apenas para a operação. Seu valor continua o mesmo "fora" da operação.

Definindo Constantes (pt 1)

- A linguagem C permite que programador dê nomes à constantes usadas no programa.
- Isso ajuda na organização do código
  - Evita a ocorrência de vários valores anônimos difíceis de entender;
  - Mudar sua definição permite que seu valor seja alterado em todos os lugares onde a constante é usada;

```
#define NOME valor
```

Exemplos:

```
#define PI 3.14159

#define PRECO 9.90

#define LETRA ‘a’
```

Por questão de legibilidade, geralmente as definições de constantes são feitas após a inserção das bibliotecas

Definindo Constantes (pt 1)

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#define LUZ 299792458
#define PI 3.14159265

int main(){
    double energia;
    float massa = 0.02, raio=2.0, area;

    energia = massa * pow(LUZ, 2); //e = m *c^2
    printf("Segundo Einstein, a energia do corpo e': %lf", energia);

    area = PI * pow(raio, 2);
    printf("\n\n A area de um circulo com raio %f e': %f", raio, area);

    return 0;
}
```

Definindo Constantes (pt 2)

- Uma outra maneira de definir constantes é usar o comando **const**.

```
const <tipo> <nome_variável> = <valor>;
```

```c
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

const float PI = 3.14159265; //outro jeito de definir a constante pi

int main(){
    float raio = 2.0, area;

    area = PI * pow(raio, 2);
    printf("\n\n A area de um circulo com raio %f e': %f", raio, area);

    return 0;
}
```