Linguagem de Programação C

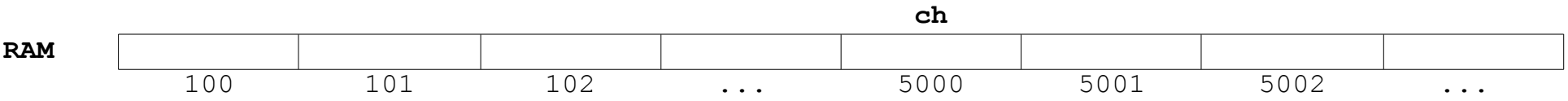
Ponteiros

Cristiano Lehrer

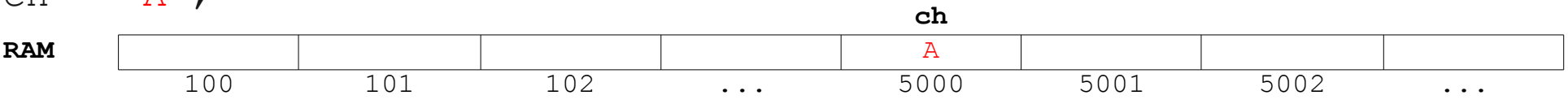
http://www.ybadoo.com.br/

Conceitos Básicos (1/3)

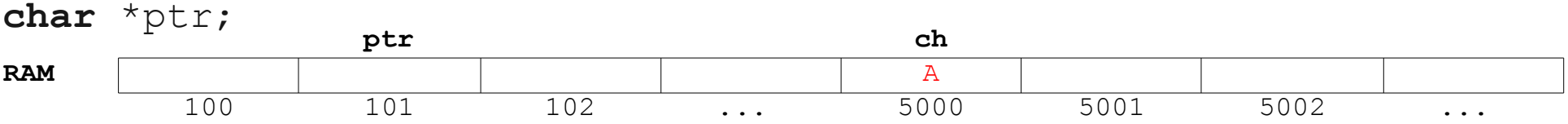
```
char ch;
```



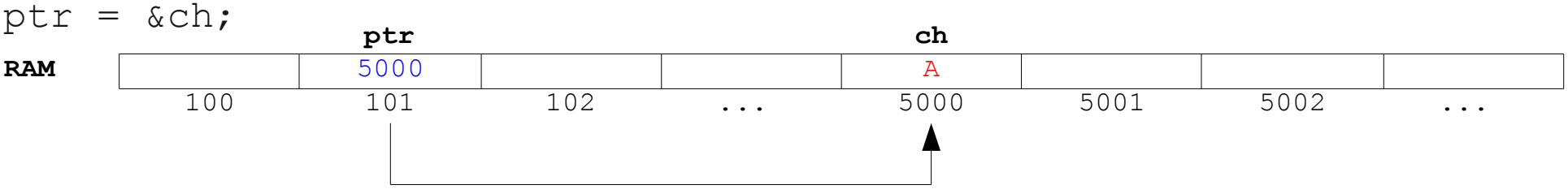

```
ch = 'A';
```



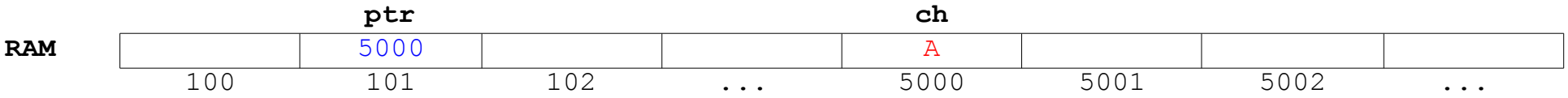

```
char *ptr;
```




```
ptr = &ch;
```



Conceitos Básicos (2/3)



Expressão	Valor	Descrição
`ch`	`A`	valor de ch
`&ch`	`5000`	endereço de ch
`ptr`	`5000`	valor de ptr
`&ptr`	`101`	endereço de ptr
`*ptr`	`A`	valor apontado por ptr

- O operador `&` retorna o endereço de uma variável.
- O operador `*` retorna o valor armazenado no endereço apontado pelo ponteiro.

Conceitos Básicos (3/3)

- Um ponteiro é uma variável que aponta sempre para outra variável de um determinado tipo.
- Para indicar que uma variável é do tipo ponteiro, coloca-se um asterisco antes dela.
- Se a variável `x` contém o endereço da variável `y`, é possível acessar o valor de `y` a partir de `x` colocando um asterisco antes da variável `x` (`*x`):
  - Esse operador denomina-se apontado por.

Declaração de Ponteiros (1/2)

```
tipo * ptr
```

- `ptr` → é o nome da variável do tipo ponteiro.
- `tipo` → é o tipo da variável para a qual apontará.
- `*` → indica que é uma variável do tipo ponteiro.
  - O asterisco utilizado na declaração de ponteiros é o mesmo que é usado para a operação de multiplicação, no entanto não provoca qualquer confusão, pois o seu significado depende do contexto em que é usado.
- Exemplos:

```
char a, b, *p, c, *q;

int idade, *p_idade;
```

Declaração de Ponteiros (2/2)

- O asterisco na declaração de um ponteiro pode ser colocado em qualquer um dos três lugares:
  - `int * ptr; /* separado do tipo e da variável */`
  - `int* ptr;  /* junto ao tipo */`
  - `int *ptr;  /* junto à variável */`
- Cuidado:
  - `int* x, y, z;`
  - Apenas a variável `x` é um ponteiro para inteiro, as demais são apenas variáveis inteiras, e não ponteiros para inteiro!

Carga Inicial Automática de Ponteiros (1/2)

- A carga inicial de ponteiros se faz através do operador endereço de (&).
- Um bom hábito para evitar problemas de programação é sempre a carga inicial dos ponteiros:

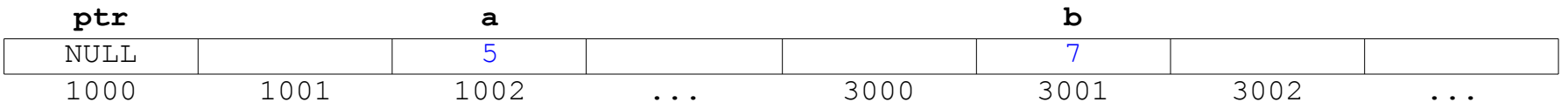
```
int x = 5;

float pi = 3.14;

int *ptr_x = &x;

float *ptr_pi = &pi;
```

Carga Inicial Automática de Ponteiros (2/2)

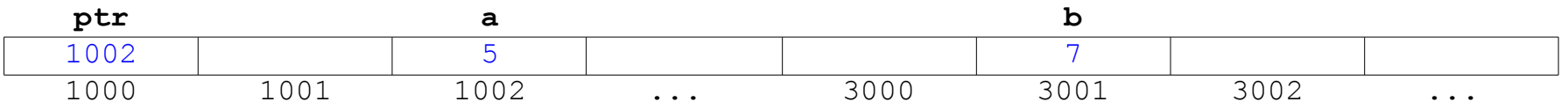
- Caso não se queira que o ponteiro aponte para variável alguma, inicializamos o ponteiro com `NULL`:
  - `int *ptr = NULL;`
- A constante simbólica `NULL`, quando colocada em um ponteiro, indica que ele não aponta para nenhuma variável:
  - Embora o seu valor tenha sofrido uma ou outra alteração ao longo da evolução da linguagem, o valor de `NULL` é obtido por `#define ((void *) 0)`, isto é, representa o endereço de memória ZERO.

Ponteiros em Ação (1/3)

```
int a = 5, b = 7;

int *ptr = NULL;
```

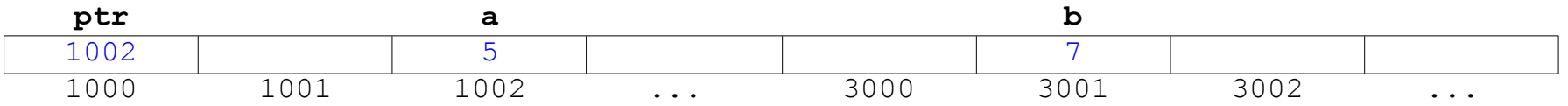



```
ptr = &a;
```

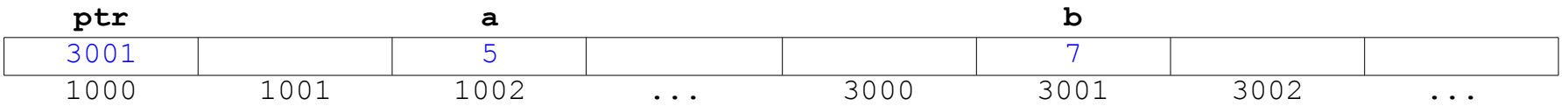



```
printf("%d", a);        /* 5 */

printf("%d", *ptr);     /* 5 */
```

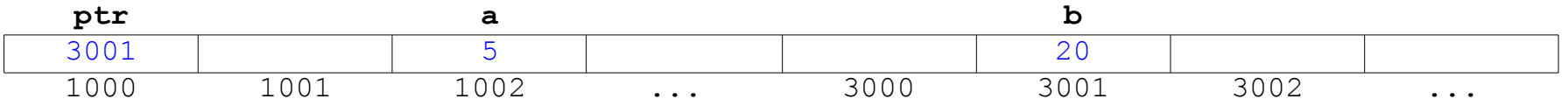
Ponteiros em Ação (2/3)




```
printf("%d %d %d", a, b, *ptr); /* 5 7 5 */

ptr = &b;
```

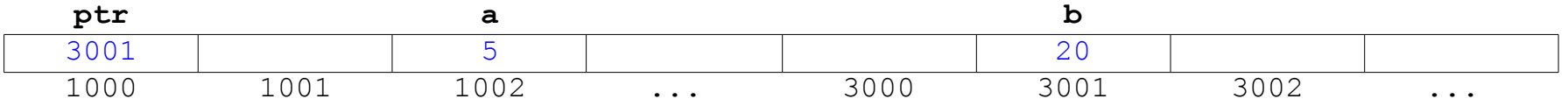



```
printf("%d %d %d", a, b, *ptr); /* 5 7 7 */

*ptr = 20;
```




```
printf("%d %d %d", a, b, *ptr); /* 5 20 20 */
```

Ponteiros em Ação (3/3)



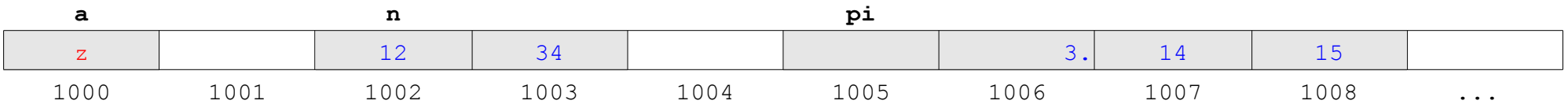
Expressão	Valor
a	5
&a	1002
b	20
&b	3001
ptr	3001
&ptr	1000
*ptr	20

`*a` ou `*b` não faz sentido, pois tais variáveis não foram declaradas como ponteiros.

Ponteiros e Tipos de Dados (1/3)

- Por que razão os ponteiros têm que possuir um determinado tipo e não são simplesmente ponteiros genéricos?
  - Devido a arquitetura dos dados manipulados.
  - Exemplo:

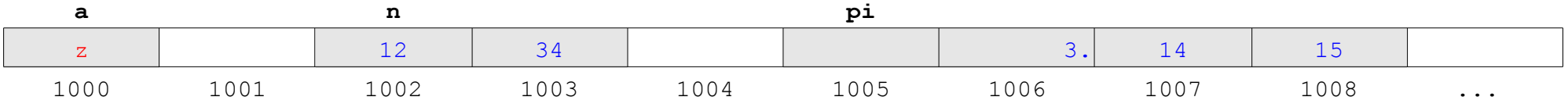
```
char a = 'z';

int n = 1234;

float pi = 3.1415;
```



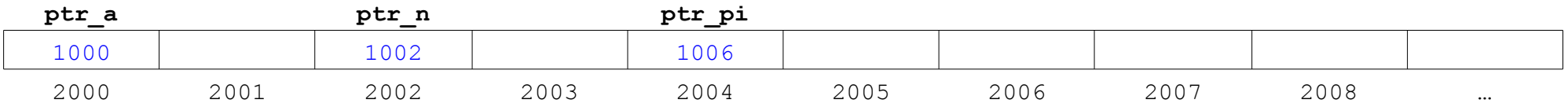
O endereço de uma variável é sempre o menor dos endereços que ela ocupa em memória.

Ponteiros e Tipos de Dados (2/3)

- Um ponteiro para o tipo `xpto` endereça sempre o número de *bytes* que esse tipo ocupa em memória, isto é, endereça sempre `sizeof(xpto)` *bytes*:

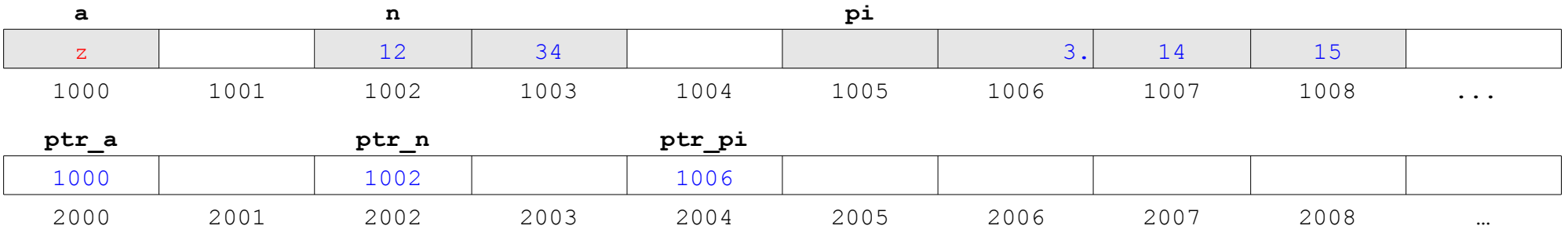



```
char *ptr_a = &a;

int *ptr_n = &n;

float *ptr_pi = &pi;
```



Ponteiros e Tipos de Dados (3/3)

- `*ptr_a`
  - Considera `sizeof(char)` *bytes* a partir do endereço contido em `ptr_a`.
- `*ptr_n`
  - Considera `sizeof(int)` *bytes* a partir do endereço contido em `ptr_n`.
- `*ptr_pi`
  - Considera `sizeof(float)` *bytes* a partir do endereço contido em `ptr_pi`.



Ponteiros e Vetores (1/2)

- O nome de um vetor corresponde ao endereço do seu primeiro elemento, isto é, se `v` for um vetor, então:

```
v == &v[0]
```

- Embora o nome de um vetor seja um ponteiro para o primeiro elemento do vetor, esse ponteiro não pode ser alterado durante a execução do programa a que pertence:
  - Se tal fosse possível, estaríamos nos arriscando a perder o vetor previamente declarado.
- Os elementos de um vetor ocupam posições consecutivas de memória, sendo o nome do vetor igual ao endereço do primeiro elemento, isto é, o menor endereço do vetor.

Ponteiros e Vetores (2/2)

```
int v[3] = {10, 20, 30};

int *ptr;

/* apontar para o primeiro elemento do vetor */

ptr = v;                          /* ou ptr = &v[0] */

printf("%d %d\n", v[0], *ptr); /* 10 10 */

ptr = &v[2];

printf("%d %d\n", v[2], *ptr); /* 30 30 */
```

Aritmética de Ponteiros (1/3)

- Incremento e decremento:

```
int x = 5, *px = &x;

float y = 5.0, *py = &y;


printf("%d %ld\n", x, (long)px);              /* 5 1211048978 */

printf("%d %ld\n", x + 1, (long)(px + 1));/* 6 1211048980 */


printf("%f %ld\n", y, (long)py);              /* 5.0 1211048970 */

printf("%f %ld\n", y + 1, (long)(py + 1));/* 6.0 1211048974 */
```

- Um ponteiro para o tipo `xpto` avança sempre `sizeof(xpto)` *bytes* por unidade de incremento.

Aritmética de Ponteiros (2/3)

- Diferença e comparação:
  - A diferença e a comparação entre ponteiros só podem ser realizadas entre ponteiros do mesmo tipo.

```
int strlen(char *s)
{
  char *ptr = s; /* guardar o endereço inicial */
  while(*s != '\0') /* enquanto não chegar ao fim */
  {
    s++;
  }

  /* retornar a diferença entre os endereços */
  return (int)(s - ptr);
}
```

Aritmética de Ponteiros (3/3)

Operação	Exemplo	Observações
Atribuição	`ptr = &x`	Podemos atribuir um valor (endereço) a um ponteiro. Se quisermos que aponte para nada podemos atribuir-lhe o valor da constante NULL.
Incremento	`ptr = ptr + 2`	Incremento de 2 * sizeof(tipo) de ptr.
Decremento	`ptr = ptr - 10`	Decremento de 10 * sizeof(tipo) de ptr.
Apontado por	`*ptr`	O operador asterisco permite obter o valor existente na posição cujo endereço está armazenado em ptr.
Endereço de	`&ptr`	Tal como qualquer outra variável, um ponteiro ocupa espaço em memória. Dessa forma podemos saber qual o endereço que um ponteiro ocupa em memória.
Diferença	`ptr1 - ptr2`	Permite-nos saber qual o número de elementos entre ptr1 e ptr2.
Comparação	`ptr1 > ptr2`	Permite-nos verificar, por exemplo, qual a ordem de dois elementos num vetor através do valor dos seus endereços.