

Estruturas de Dados

Algoritmos e Programação de Computadores

Guilherme N. Ramos
gnramos@unb.br

2018/2



Por que todo programa manipula dados [por definição]?

“Acerte as estruturas de dados primeiro, e o resto do programa se escreverá sozinho.”

David Jones

Tipos de dados: numéricos, simbólicos e lógicos.

- *O tipo define o que o programa pode fazer com o dado.*

Representação de Dados

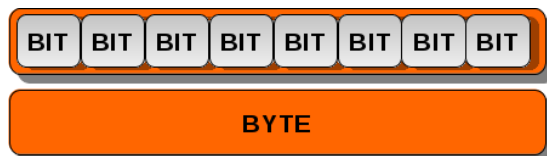
bit (*binary digit*)

Representa um estado binário:

“ligado” é representado pelo símbolo 1.

“desligado” é representado pelo símbolo 0.

A memória é um conjunto ordenado de *bits* que podem conter instruções ou dados.



Representação de Dados

Dados *diferentes* podem ser representados por um mesmo [conjunto de] byte[s].



O mesmo [conjunto de] byte[s] pode ser *interpretado* de formas diferentes.

1010010011110001011100011101010100101011

- A representação do dado é uma só: *binária*!
- A **interpretação** dos bits define a informação.

0xBF400000

Sinal e Magnitude -1061158912_{10}

Complemento de 1 -1086324735_{10}

Complemento de 2 -1086324736_{10}

Ponto Flutuante (32) -0.75_{10}

0x41200000

Inteiro 1092616192_{10}

Ponto Flutuante (32) 10.0_{10}

ASCII A

Sistemas Numéricos

Bits podem representar números pelo sistema numérico posicional¹. Por exemplo, 123_{10} :

$$100 + 20 + 3 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0$$

O valor depende de cada algarismo (base numérica) e de sua posição, e pode ser facilmente obtido com a seguinte fórmula:

$$a_n a_{n-1} \cdots a_2 a_1 a_0 = a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \cdots + a_2 \cdot b^2 + a_1 \cdot b^1 + a_0$$

¹49 em algarismos romanos?

Sistemas Numéricos

Bases:

Hexadecimal $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$

Decimal $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Octal $\{0, 1, 2, 3, 4, 5, 6, 7\}$

Binária $\{0, 1\}$

$$7B_{16} = 123_{10} = 173_8 = 1111011_2$$

$$75_{10} = (\quad)_2 = (\quad)_8 = (\quad)_{16}$$

Números Reais

Reais - IEEE 754

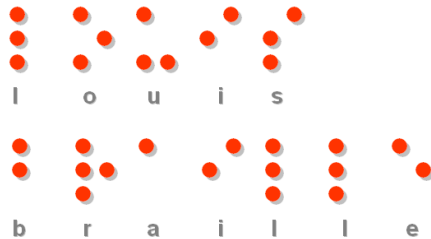
$$(-1)^{\text{signal}} \cdot (1 + \text{mantissa}) \cdot 2^{\text{expoente} - \text{offset}}$$

1 01111110 100000000000000000000000

$$\begin{aligned} & (-1)^1 \cdot 1,1 \cdot 2^{126-127} \\ &= -1,1 \cdot 2^{-1} \\ &= -0,11 \\ &= -(1 \cdot 2^{-1} + 1 \cdot 2^{-2}) \\ &= -(0,5 + 0,25) \\ &= -0,75 \end{aligned}$$

Símbolos

A *codificação de caracteres* é a associação de bits a símbolos.



Por necessidade de diálogos entre os diferentes computadores, foram criados diversos códigos objetivando a padronização.

Ponteiros

Cada variável declarada ocupa um espaço na memória, conforme seu tipo, e nome da variável é apenas uma forma “amigável” de lidar com o endereço deste espaço.

← Ponteiro →

Tipo de dado que armazena um *endereço de memória*, possibilitando leitura e escrita deste endereço.

Atenção

Há uma diferença conceitual entre **endereço** e **conteúdo**. O endereço indica a localização na memória (onde está armazenado), o conteúdo indica o valor dos bits (o que está armazenado).

Ponteiros na Linguagem C

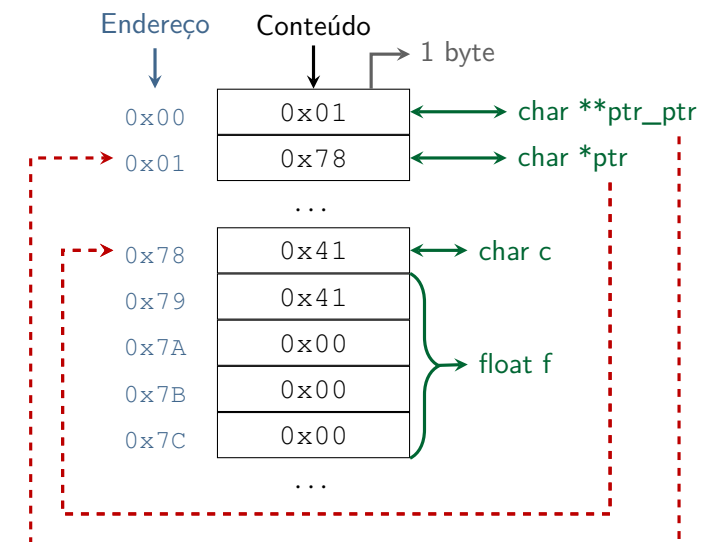
Em linguagem C, um ponteiro é declarado da seguinte forma:

```
tipo* identificador;
```

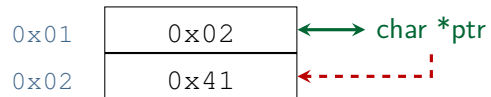
Por exemplo:

```
1 int* ptr_int; // ponteiro para inteiro
2 float* ptr_float; // ponteiro para real
3 char* ptr_char; // ponteiro para caractere
4
5 int** ptr_ptr_int; // ponteiro para (ponteiro para inteiro)
```

Ponteiros na Linguagem C



Ponteiros na Linguagem C



00-ponteiro.c

```
1 char c = 'A';
2 char* ptr = &c; /* Armazena o endereço de c */
3
4 /* O conteúdo de c é: */
5 printf("  c = %c\n", c);
6 /* O conteúdo de ptr é: */
7 printf(" ptr = %p\n", ptr);
8 /* O conteúdo do endereço apontado por ptr é: */
9 printf(" *ptr = %c\n", *ptr);
10 /* O endereço de ptr é: */
11 printf("&ptr = %p\n", &ptr);
```

Ponteiros na Linguagem C

Ponteiros permitem “sair” do escopo

```
1 int dobra(int x) {
2     return 2*x;
3 }
```

```
1 void dobra(int* x) {
2     (*x) = 2*(*x);
3 }
```

```
1 void troca(int a, int b) {
2     int aux = a;
3     a = b;
4     b = aux;
5 } /* ? */
```

```
1 void troca(int* a, int* b) {
2     int aux = *a;
3     *a = *b;
4     *b = aux;
5 }
```

Ponteiros na Linguagem Python

Python não utiliza ponteiros como tipos de dados, mas lida com referências de forma “diferente”...

Os objetos em Python são:

imutáveis: int, float, str, ...

mutáveis: lista, ...

Ponteiros na Linguagem Python

Toda variável em Python é uma referência para algum objeto da memória.

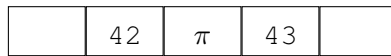
Nesta abordagem para lidar com a memória, o valor de nome ‘a’ não varia a não ser que você o mude explicitamente².

²Explícito é melhor que implícito.

Ponteiros na Linguagem Python

O operador `=` não é de *atribuição*, mas de *referência*.

0xB1 0xB5 0xB9 0xBD 0xC0



```
1 a = 42
2 a += 1
3 b = a
4 a = 3.14
```

Ponteiros na Linguagem Python

00-identificador.py

```
1 x = 'A'
2 id_x = id(x)
3
4 # O identificador não é necessariamente o endereço de
5 # memória do objeto!
6
7 # O conteúdo de x é:
8 print('    x = {}'.format(x))
9 # O conteúdo de id_x é:
10 print(' id_x = {}'.format(id_x))
11
12 # É possível obter o objeto a partir de seu identificador,
13 # mas isso NÃO é um procedimento adequado (vai contra muitas
14 # decisões de projeto da linguagem).
```

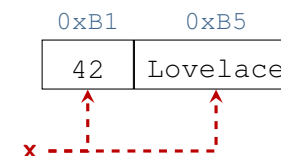
Ponteiros na Linguagem Python

01-identificador.py

```
1 x, y = 'A', 'B'
2 id_x, id_y = id(x), id(y)
3
4 print('x={} ({}), y={} ({}).format(x, id_x, y, id_y))
5
6 del x # Forçando o fim do ciclo de vida do objeto x
7
8 # Criando um novo objeto 'y' (mesmo valor que x tinha!)
9 y = 'A'
10 id_y = id(y)
11 print('y={} ({}).format(y, id_y))
12
13 if id_y == id_x:
14     print('y tem o mesmo id que x tinha...')
```

Ponteiros na Linguagem Python

Um tipo é *dinâmico e forte* (em tempo de execução, sabe-se o tipo de dado referenciado e como ele é representado na memória).



```
1 x = 42
2 x = 'Lovelace'
```

Ponteiros na Linguagem Python

É possível retornar de múltiplos valores.

```
05-bhaskara.py
1 def bhaskara(a, b, c):
2     delta = (b ** 2) - (4 * a * c)
3     raizes_reais = (delta >= 0)
4
5     r1 = (-b + (delta ** 0.5)) / 2 if raizes_reais else 0.0
6     r2 = (-b - (delta ** 0.5)) / 2 if raizes_reais else 0.0
7
8     return (raizes_reais, r1, r2)
```

Vetores

É fácil manipular um dado para resolver um problema:

```
1 z = min(x, y);
```

Mas e 2 3 n problemas?

```
1 z = min(x1, min(x2, min(x3, /* ... */ min(xk, xn) /* ... */)));
```

Vetores

Mostrar 1000 caracteres não seria agradável... Mas suponha eles estão magicamente armazenados sequencialmente, começando em um endereço de memória que você conhece...

```
1 printf("c0=%c\n", c0); 1 for(i = 0; i < n; ++i)
2 printf("c1=%c\n", c1); 2     printf("c%d=%c\n", i, *(c+i));
3 printf("c2=%c\n", c2);
4 printf("c3=%c\n", c3);
5 /* ... */

997 /* ... */
998 printf("c997=%c\n", c997);
999 printf("c998=%c\n", c998);
1000 printf("c999=%c\n", c999);
```

Vetores

Vetor (array)

É um conjunto *finito* e *ordenado*³ de elementos *homogêneos*.

Quais elementos?

O vetor é um modo particular de organizar dados para facilitar o acesso e manipulação dos dados.

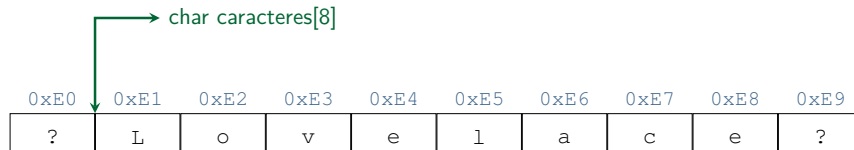
0	1	2	3	4	5	6	7	8	9
?	?	?	?	?	?	?	?	?	?

³Em relação a posição.

Vetores

Vetor em C: *endereço do primeiro elemento e quantidade de elementos.*

```
1 int    inteiros[1000];
2 float   reais[50];
3 char   caracteres[8];
```



```
1 for(i = 0; i < n; ++i)    1 for(i = 0; i < n; ++i)
2 printf("c%d=%c\n", i, *(c+i)); 2 printf("c%d=%c\n", i, c[i]);
```

RAM + indexação ⇒ **velocidade**

Vetores

Em Python, o termo que descreve um conjunto de elementos é coleção, e a mais simples é a lista. O funcionamento é similar ao de um vetor em C, mas com uma série de facilidades.

```
1 inteiros = [0, 1, 2, 3]
2 reais = [0.0] * 10
3 caracteres = ['L', 'o', 'v', 'e', 'l', 'a', 'c', 'e']
```

```
1 for i in range(len(inteiros)):    1 for c in caracteres:
2     print(inteiros[i])            2     print(c);
```

Vetores

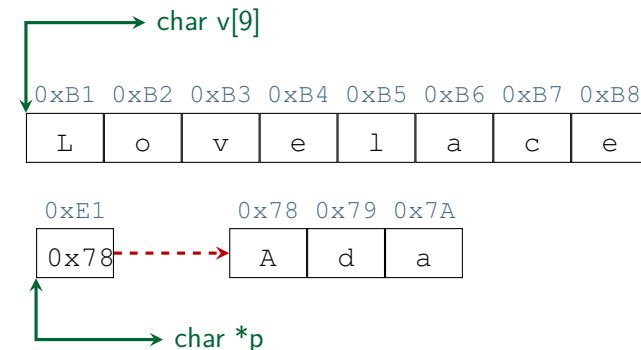
Considerações para vetores:

- Muito cuidado com os índices utilizados, use somente $i \in [0, n)$.
- Pode não ser preciso ocupar todas as posições do vetor, usar um vetor maior que o necessário muitas vezes facilita a vida...
- Em linguagem C: *alocação estática de memória.*
- Em linguagem Python: *alocação "dinâmica" de memória.*

Vetores

Linguagem C

Vetores *não* são ponteiros.



Strings

Uma *palavra/frase* é um conjunto finito e ordenado de letras.

apc_vetor.h

```
1 void mostra_n_chars(char* str, int n) {
2     int i;
3     printf("string = ");
4     for(i = 0; i < n; ++i)
5         putchar(str[i]);
6     printf("]\n");
7 }
```

Vetor de caracteres, com tamanho fixo?

Cada string tem

- 1 um inteiro associado a seu tamanho (04-string.c); ou
- 2 um caracter específico que indica o fim do string (06-string.c).

Strings

Em linguagem C, usa-se o caracter `'\0'` para determinar o fim do vetor, portanto pode-se ignorar o tamanho do vetor e simplesmente percorrê-lo até encontrar o caractere de término.:

07-string.c

```
1 /* Assume-se que o string termina em '\0'. */
2 mostra_ate_char(frase, '\0');
```

Strings

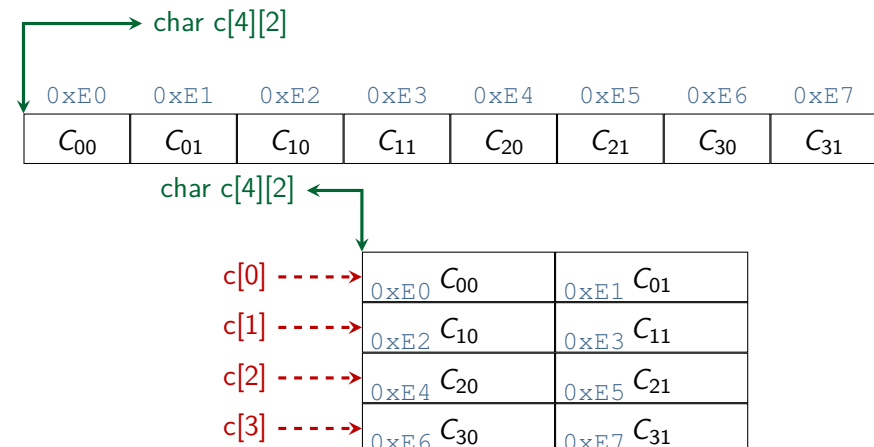
Linguagem Python

Strings (imutáveis) *não são listas* (mutáveis).

```
1 nomes = ['Turing', 'Hopper', 'Von Neumann']
2 alan = 'Turing'
3
4 nomes[0] # 'Turing'
5 alan[0] # 'T'
6
7 nomes[0] = 'Alan' # ['Alan', 'Hopper', 'Von Neumann']
8 alan[0] = 't' # ERRO!
9
10 nomes[1:3] # ['Hopper', 'Von Neumann']
11 alan[0:2] # 'Tu'
```

Vetor 2D

Um vetor é um bloco de memória (suficiente para N elementos).



Vetor 2D

"Grandes poderes trazem grandes responsabilidades."

Ben Parker

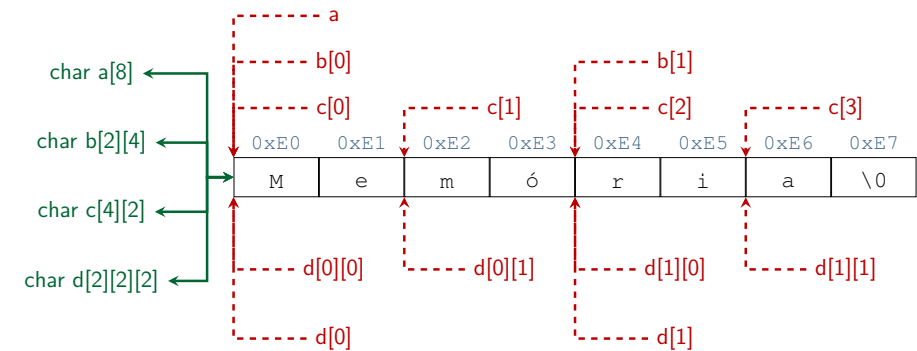
O acesso a blocos de memória com ponteiros é algo extremamente útil se feito com a devida cautela (e sem maldade ou malícia).

12-main.c

```
1 int main(int argc, char* argv[]) {
2     /* A ideia é cumprimentar o usuário... */
3     printf("Boa tarde, %s.\n", argv[1]);
4
5     return 0;
```

Vetores N-dimensionais

Os mesmos princípios que se aplicam a 2, se aplicam a N vetores.



Registros

Registro

Estrutura que armazena diferentes tipos de dados em uma única variável.

```
1 Algoritmo LeFuncionários
2 Definições
3     funcionario : registro (nome, endereço : string;
4                             sexo : caractere;
5                             código : inteiro;
6                             salário : real)
7 Variáveis
8     funcionários : vetor[1000] de funcionario
9 Início
10     /* ... */
11     Para i de 0 a 999 Faça
12         Leia(funcionários[i])
13     FimPara
14     /* ... */
15 Fim
```

Registros

Na linguagem C, o registro é definido pela palavra-chave `struct`, e o acesso a seus componentes pelo identificador e o caractere `'.'`.

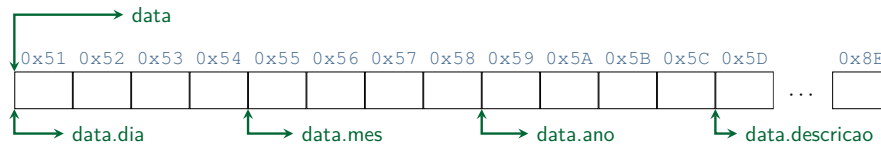
00-data.c

```
1 struct {
2     int dia, mes, ano; /* A "data" em si. */
3     char descricao[50]; /* Uma descrição da data. */
4 } data;
5
6 printf("Digite a descrição: ");
7 scanf("%[^\n]", data.descricao);
8 printf("Digite o ano: ");
9 scanf("%d", &(data.ano));
10 printf("Digite o mês: ");
11 scanf("%d", &(data.mes));
12 printf("Digite o dia: ");
13 scanf("%d", &(data.dia));
14
```

Registros

00-data.c

```
1  /* Definição da estrutura do registro (identificado como
   "data"): */
2  struct {
3      int dia, mes, ano; /* A "data" em si. */
4      char descricao[50]; /* Uma descrição da data. */
5  } data;
```



Registros

03-mp3.c

```
1  /** @file: 03-mp3.c
2   * @author: Guilherme N. Ramos (gnramos@unb.br)
3   * @disciplina: Algoritmos e Programação de Computadores
4   *
5   * Exemplo de uso de registro (ID3v1) para armazenar as
6   * informações de um arquivo no formato MP3. Veja mais em:
7   * http://en.wikipedia.org/wiki/ID3#ID3v1 */
8
9  typedef struct{
10     char header[3];
11     char titulo[30];
12     char artista[30];
13     char album[30];
14     char ano[4];
15     char comentario[30];
16     unsigned char genero;
17 } mp3_ID3v1;
```

Registros

03-mp3.py

```
1 class mp3_ID3v1():
2     self.header = ''
3     self.titulo = ''
4     self.artista = ''
5     self.album = ''
6     self.ano = ''
7     self.comentario = ''
8     self.genero = ''
```

Binários

O computador trabalhar apenas com bit e bytes, portanto todos os arquivos são conjuntos binários.

A manipulação é extremamente simples, tem-se o endereço do arquivo, basta ler/escrever a quantidade de bytes desejada.

Linguagem C

```
1 size_t fread(void *destino, size_t tam, size_t qte, FILE *origem);
2 size_t fwrite(void *origem, size_t tam, size_t qte, FILE *destino);
```

Linguagem Python

```
1 instancia.read(bytes)
2 instancia.write(bytes)
```

Binários

É muito fácil manipular arquivos binários, mas os procedimentos de leitura não podem ser dissociados dos de escrita (e vice-versa).

Texto

Humanos não se comunicam por bytes...

Linguagem C

```
1 int fprintf(FILE *fp, const char *formato, ... );
2 int fscanf(FILE *fp, const char *formato, ... );
3 int fputc(int caractere, FILE *fp );
4 int fgetc(FILE *fp);
5 int fputs(const char *string, FILE *fp );
6 char *fgets(char *string, int num_caracteres, FILE *fp );
```

Linguagem Python

```
1 f.write(string)
2 f.read(string)
```

Cor

Um padrão comum de representação de cor é o sistema RGB, em que cada cor é composta pelos três componentes (*Red - Green - Blue*).

Cada componente tem um valor definido por 1 byte indicando a intensidade: 0xRRGGBB

(ausência da cor) 00 ⇔ FF (intensidade máxima)

São, portanto, $2^8 \cdot 2^8 \cdot 2^8 = 2^{24} = 16,777,216$ cores possíveis.

0xFF0000 vermelho

0x00FF00 verde

0x0000FF azul

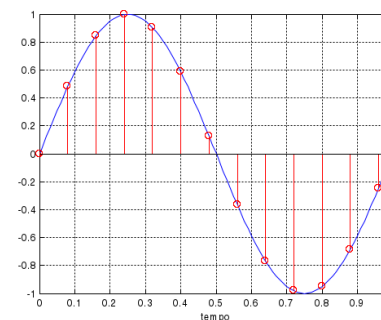
0x000000 preto

0xFFFFFFFF branco

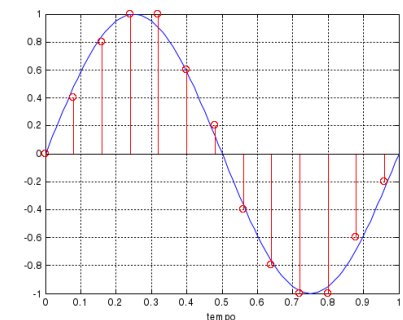
0xFFFF00 amarelo

Áudio

- 44.1kHz
- 16 bits
- Estéreo (2 canais)



Amostragem



Quantização