

Estruturas de Dados

Algoritmos e Programação de Computadores

Guilherme N. Ramos
gnramos@unb.br

2015/2



Representação de Dados

Por que todo programa manipula dados [por definição]?

Tipos de dados: numéricos, simbólicos e lógicos.

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

Como representar os dados [na memória] do computador?

Representação de Dados

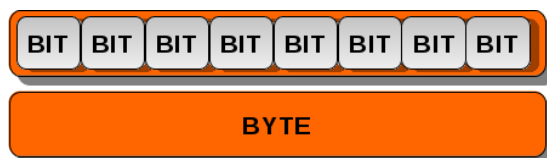
bit (*binary digit*)

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.



1010010011110001011100011101010100101011

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

0xBF400000

Sinal e Magnitude -4145152_{10}

Complemento de 1 -8486911_{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:

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

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

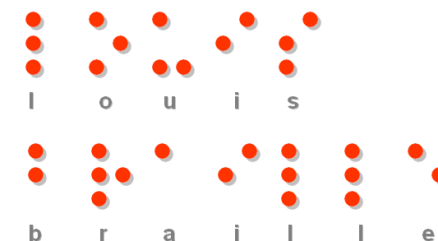
$$(-1)^{sinal} \cdot (1 + mantissa) \cdot 2^{expoente - offset}$$

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

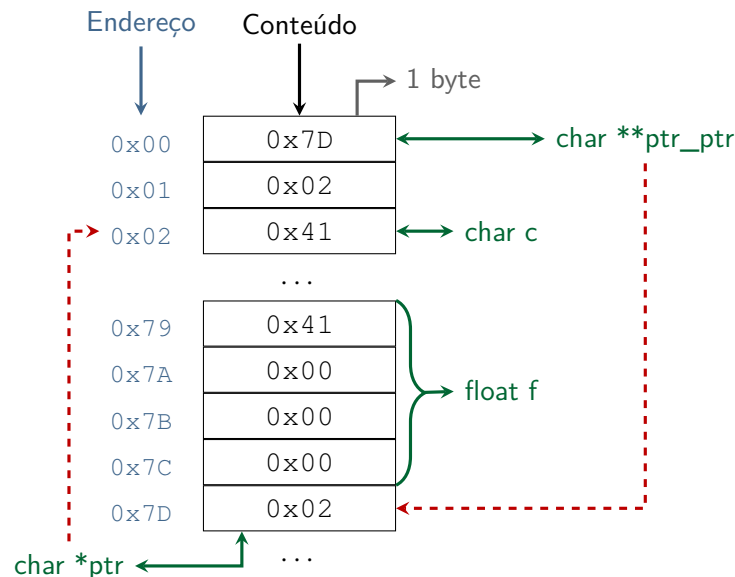
Em linguagem C, um ponteiro é declarado da seguinte forma:

```
tipo* identificador;
```

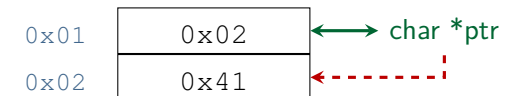
Por exemplo:

```
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



Ponteiros



0-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

```
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 }
```

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

Lembrando como funcionam laços de repetição e [aritmética de] ponteiros, como seria um algoritmo que mostre n caracteres? 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 /* ... */  
  
999 /* ... */  
1000 printf("c999=%c\n", c999);
```

Vetores

Vetor (array)

É um conjunto *finito* e *ordenado*² 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
?	?	?	?	?	?	?	?	?	?

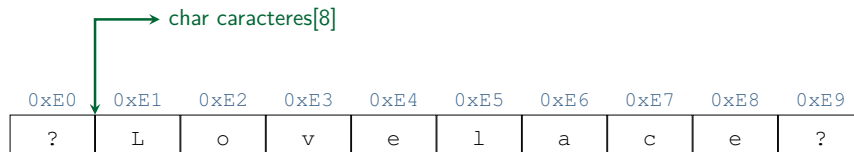
RAM + indexação \Rightarrow **velocidade**

²Em relação a posição de memória.

Vetores

Vetor: *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]);
```

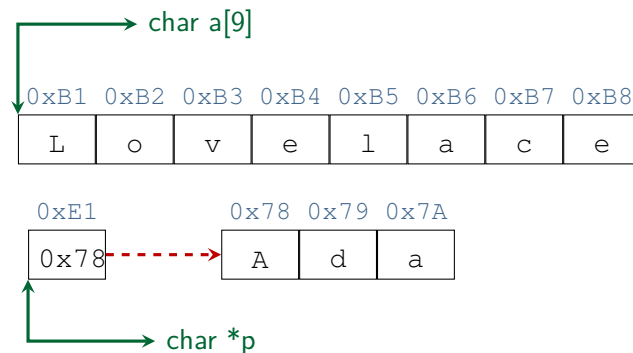
Funções com Vetores

Considerações para vetores em linguagem C:

- 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...
- Alocação estática de memória.

Funções com Vetores

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 (4-string.c); ou
- 2 um caracter específico que indica o fim do string (6-string.c).

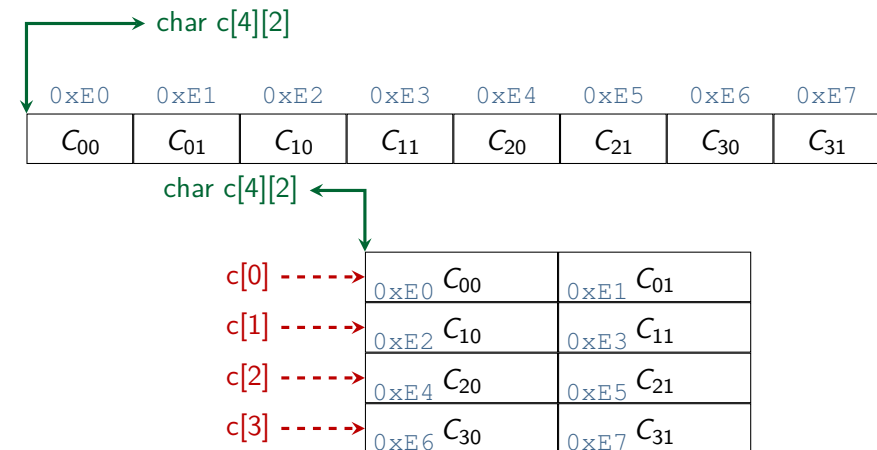
Strings

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.:

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

Vetor 2D

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



Vetor 2D

"Grandes poderes trazem grandes responsabilidades."

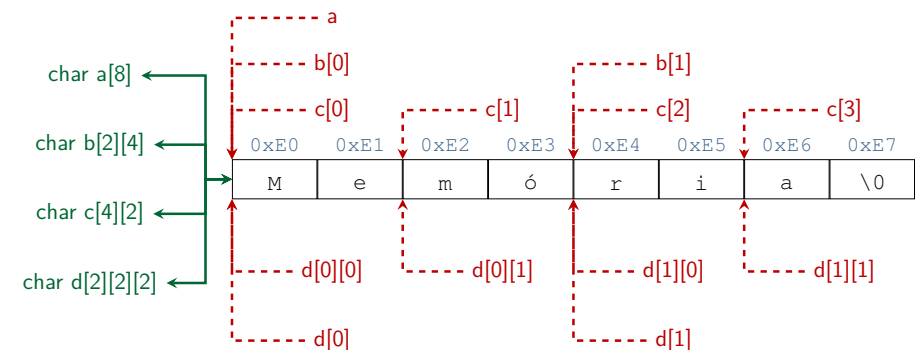
Ben Parker

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 `'.'`.

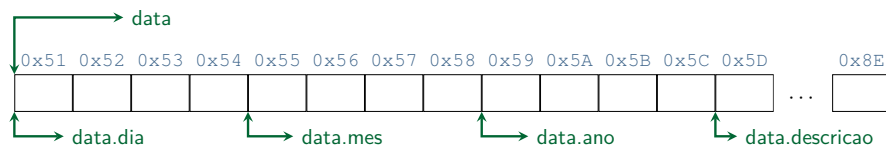
0-data.c

```
1 struct {
2     int dia, mes, ano;
3     char descricao[50];
4 } data;
5
6 leia_string("Digite a descrição: ", data.descricao);
7 data.ano = leia_int("Digite o ano: ");
8 data.mes = leia_int("Digite o mes: ");
9 data.dia = leia_int("Digite o dia: ");
10
11 printf("%s:\n%02d/%02d/%04d\n", data.descricao,
12        data.dia,
13        data.mes,
14        data.ano);
15
```

Registros

0-data.c

```
1 struct {
2     int dia, mes, ano;
3     char descricao[50];
4 } data;
```



Registros

3-mp3.c

```
1 /** @file: 3-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;
```

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.

Pseudo-código

```
1 Função Void Leia(arquivo origem, tipo destino)
2 Função Void Escreva(arquivo destino, tipo origem)
```

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);
```

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...

```
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 );
```

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

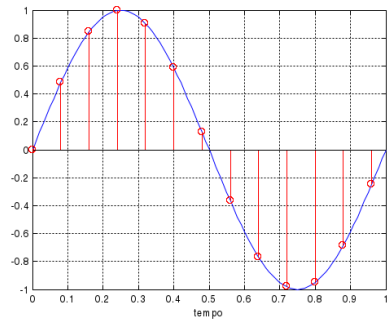
0x000000 preto

0xFFFFFFFF branco

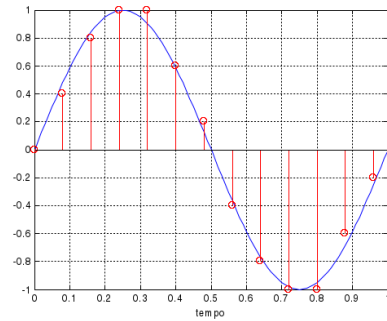
0xFFFF00 amarelo

Áudio

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



Amostragem



Quantização