Vetores unidimensionais, strings e matrizes

Mutirão C 2018

Igor Rafael Hashi

21-02-2018

Revisão da leitura 1

Comandos para compilação

Utilização do `gcc` na linha de comando:

1. `gcc -Wall -std=c99 -pedantic -Og -c arq1.c` (repetir para todos arquivos)
2. `gcc arq1.o .... -o executavel` (listar todos os arquivos `.o` gerados)

Se modificamos um dos arquivos só precisamos executar sua compilação e depois repetir o segundo passo.

Make

Sintaxe simples:

```
target_name: dependencia1 dependencia2
(tab) comando1
(tab) comando2
(tab) ...
```

Chamar `make (target_name)` para compilar um *target* específico ou `make` para compilar todos.

Cuidado para seu editor não substituir o (tab) por 4 espaços!

Aviso

Dúvidas?

Aviso

Dúvidas?

Todos os exercícios usarão *make* e *gcc* a partir de agora.

Vetores unidimensionais

Sintaxe

- Declaração:

```
long vetor[100]; /* Tamanho FIXO */
long vetor2[] = {1, 2, 3, 4, 5}; /* Inicialização. */
```

- Acesso a elemento:

```
scanf("%ld\n", &vetor[0]); /* vetor[pos] */
```

- Uso como parâmetro de função:

```
void funcao1(long vetor[], int tamanho);
void funcao2(long vetor[100]); /* tamanho FIXO */
```

Boas práticas

Evite números mágicos no seu código! Use macros!

```
#define MAX_SIZE 100

long vec[MAX_SIZE];

...

for(i = 0; i < MAX_SIZE; i++) {
    ...
}
```

- Por que isto é uma boa ideia?

Organização na memória

Os dados de um vetor são organizados sequencialmente, *sem intervalos*

```
long A[4];
```

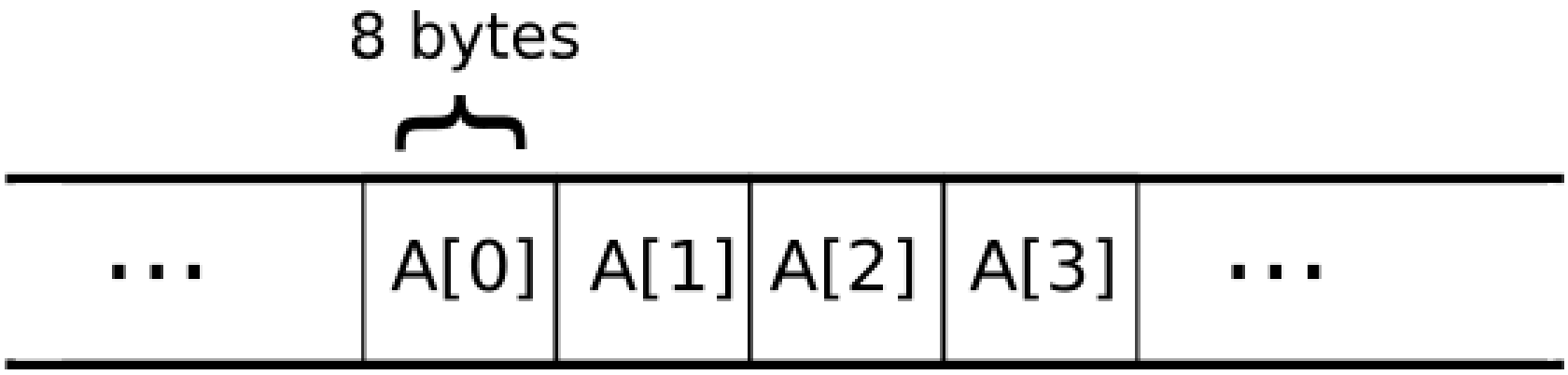

Figure 1:

Quiz

- Declare um vetor com tamanho suficiente para guardar a altura dos alunos de uma disciplina de engenharia no Insper.

- Declare um buffer para armazenar 2 segundos de áudio a 44100Hz com resolução de 16bits por amostra.

- Declare um vetor para contar a quantidade de visualização por dia de um vídeo no Youtube.

Quiz

- Declare um vetor com tamanho suficiente para guardar a altura dos alunos de uma disciplina de engenharia no Insper.

- Declare um buffer para armazenar 2 segundos de áudio a 44100Hz com resolução de 16bits por amostra.

- Declare um vetor para contar a quantidade de visualização por dia de um vídeo no Youtube.

PSY breaks youtube counter

Escopo de variáveis “normais”

Qual é o resultado deste código:

```
void soma_um(long n) {
    n += 1;
}

int main(int argc, char *argv[]) {
    long n = 10;
    soma_um(n);
    printf("%ld\n", n);
}
```

Escopo de variáveis “normais”

Qual é o resultado deste código:

```
void soma_um(long n) {
    n += 1;
}

int main(int argc, char *argv[]) {
    long n = 10;
    soma_um(n);
    printf("%ld\n", n);
}
```

n 10

Escopo de arrays

Qual é o resultado deste código:

```
void soma_um(long arr[], long pos) {
    arr[pos]+=1;
}

int main(int argc, char *argv[]) {
    long arr[] = {1, 2, 3, 4, 5, 10};
    soma_um(arr, 5);
    printf("arr[5] %ld\n", arr[5]);
}
```

Escopo de arrays

Qual é o resultado deste código:

```
void soma_um(long arr[], long pos) {
    arr[pos]+=1;
}

int main(int argc, char *argv[]) {
    long arr[] = {1, 2, 3, 4, 5, 10};
    soma_um(arr, 5);
    printf("arr[5] %ld\n", arr[5]);
}
```

arr[5] 11

Parâmetros: passagem por valor ou referência

Passagem por valor: `long n` é uma **cópia** do valor passado.

```
void soma_um(long n) {
    n += 1;
}
```

Passagem por referência: `long arr[]` é o **mesmo objeto** da função chamadora.

```
void soma_um(long arr[], long pos) {
    arr[pos]+=1;
}
```

Exercícios

- Faça um programa que lê um inteiro $n < 100$ da linha de comando, depois lê um vetor com n elementos e imprime 1 se o vetor estiver ordenado e 0 caso contrário.
- Faça uma função que encontra o menor elemento de um vetor e troca ele de lugar com o primeiro elemento.

  Entrada: {4, 5, 7, 1} Saída: {1, 5, 7, 4};

Strings

Sintaxe

Strings são vetores de `char` cuja última posição contém o caractere `'\0'`.

```
char string[] = "bla bla bla"; /* Inicializado */
char cmd[100];  /* 99 caracteres, string inválida! */
```

Funciona exatamente como um vetor, com a diferença que **uma string sabe seu tamanho**!.

Organização na memória

Os dados de uma string são organizados sequencialmente, *sem intervalos*, sendo que **o último elemento é sempre o caractere** `'\0'`.

```
char str[] = "INSPER";
```

Figure 2:

Tamanho de string

```
int string_length(char str[]) {
    int n = 0;
    while (str[n] != '\0') {
        n++;
    }
    return n;
}
```

A string não precisa ocupar todas as posições do array!

Funções da biblioteca padrão

```c
#include <string.h>

/* Tamanho da string */
size_t strlen(char *s);

/* Devolve 0 se forem iguais,
    <0 se str1 vier antes de str2 e
    >0 se str1 vier depois de str2 */
int strcmp(char *str1, char *str2);

/* Copia src para dest, sendo que dest deve ter
   espaço para fazer a cópia. */
char *strcpy(char *dest, char *src);
```

Exercícios

1. Faça um programa que lê uma linha da entrada padrão e converte a primeira letra de toda palavra para maiúsculas.
2. Faça uma função `int strfind(char msg[], char tk[]);` que devolve 1 se `tk` estiver contido em `msg` e 0 caso contrário.

Exercícios

1. Faça um programa que lê uma linha da entrada padrão e converte a primeira letra de toda palavra para maiúsculas.
2. Faça uma função `int strfind(char msg[], char tk[]);` que devolve 1 se `tk` estiver contido em `msg` e 0 caso contrário.

   ***Dica**: usar strcmp ajuda?*

Matrizes

Sintaxe

- Declaração:

```
long mat[100][200];
long mat2[2][3] = {{1, 2, 5}, {3, 4, 6}} ;
```

- Acesso a elemento:

```
printf("%ld\n", mat2[0][2]); /* 5 */
```

- Uso como parâmetro de função:

```
void funcao1(long vetor[100][200]);
```

É obrigatório colocar a última dimensão! A primeira é facultativa, mas recomendamos colocar também.

Organização na memória

Os dados de uma matriz são armazenados *linha a linha*

```
long mat[2][3];
```

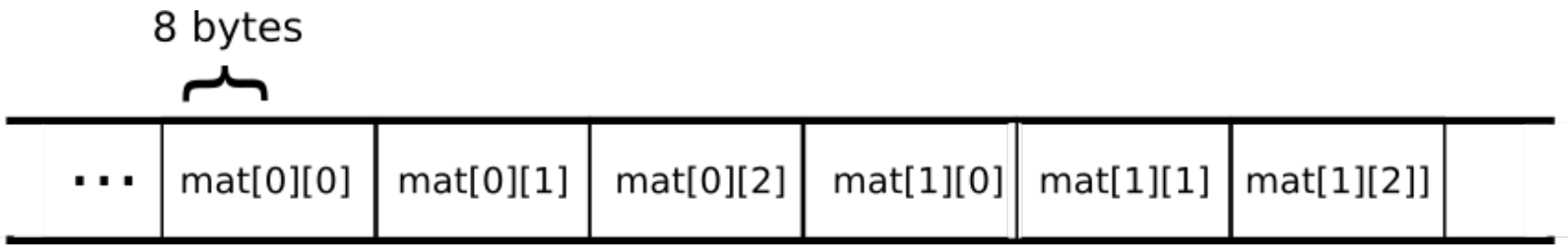

Figure 3:

Organização na memória

Os dados de uma matriz são armazenados *linha a linha*

```
long mat[2][3];
```

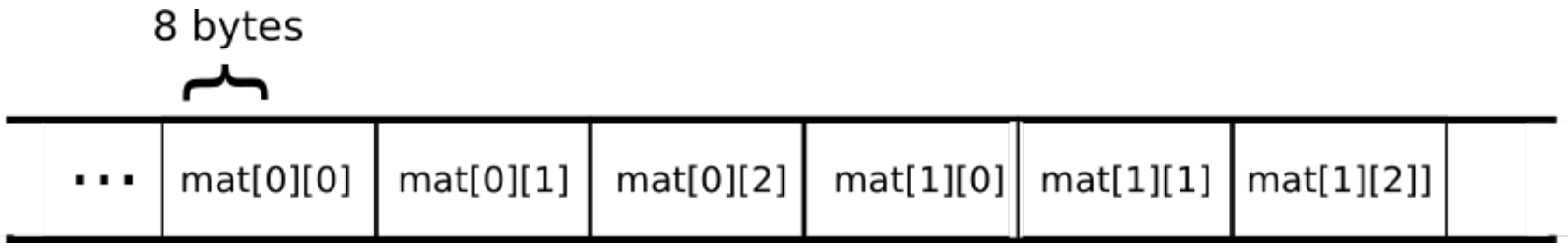

Figure 3:

É preciso conhecer a segunda dimensão para acessar os elementos!

Exercícios

Vamos voltar a trabalhar com o Embarcado agora!

1. Abra o projeto da aula de hoje.
2. Vamos programar vários processamentos de imagens interessantes.
3. Todas as funções já estão no projeto. Você só precisa completá-las.

Exercícios I

Limiar: faça a função `threshold` que converte uma imagem para binário.

Exercícios II

Normalização de histograma: calcule o menor e o maior nível de cinza da imagem e modifique-a de modo que estes valores sejam 0 e 255 na saída.

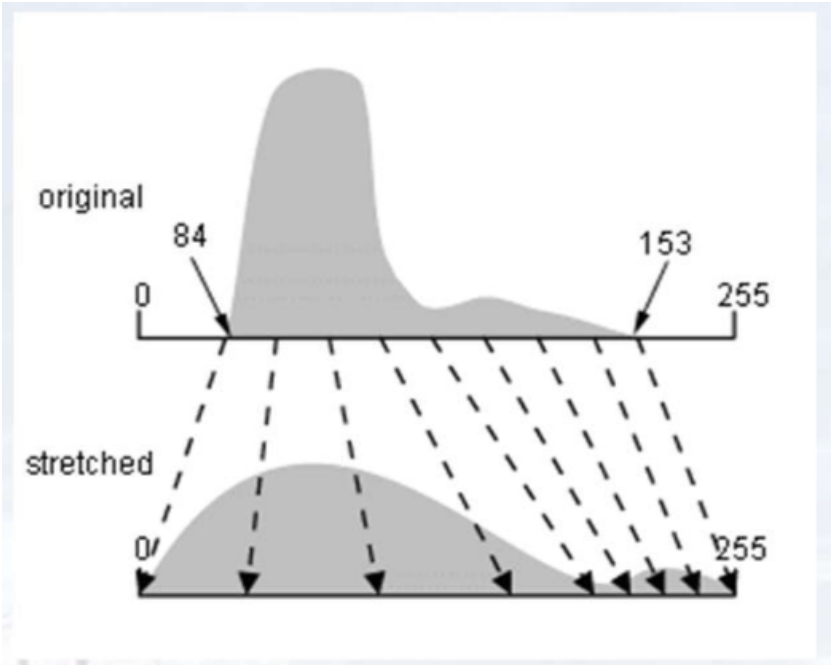

Figure 4: Constrast Stretching

fonte:
https://stackoverflow.com/questions/41118808/difference-between-contrast-stretching-and-histogram-equalization

Exercícios III

Convolução: cada pixel da imagem será substituído pelo valor absoluto da média ponderada de seus vizinhos em uma vizinhança 3×3.

```
void convolution(unsigned char imageIn[320][320],
unsigned char imageOut[320][320], double
kernel[3][3]);
```

Teste seu código com os seguintes kernels:

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \qquad \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}$$