

# Algoritmos e Estruturas de Dados I

## Arquivos

Pedro O.S. Vaz de Melo

# Problema 1

- Considere que um arquivo de dados contém os valores das dimensões (tam. max.: 100) e dos elementos de duas matrizes de números inteiros. Ex:

2 5

4 12 1 78 5

98 9 1994 0 52

79 12 458 2 29

47 19 784 12 8

- Implemente um programa que calcule e exiba a matriz resultante da soma dessas duas matrizes.

# Problema 1

- Implemente um programa que calcule e exiba a matriz resultante da soma de duas matrizes (tam. max.: 100) lidas de um arquivo.
- Algoritmo:
  - 1) Criar duas matrizes
  - 2) Ler o arquivo
  - 3) Armazenar os dados do arquivo nas matrizes
  - 4) fazer a soma das matrizes
  - 5) Imprimir a matriz resultante

# Problema 1

- Implemente um programa que calcule e exiba a matriz resultante da soma de duas matrizes (tam. max.: 100) lidas de um arquivo.
- Algoritmo:
  - 1) Criar duas matrizes :)
  - 2) Ler o arquivo :(
  - 3) Armazenar os dados do arquivo nas matrizes :(
  - 4) fazer a soma das matrizes :)
  - 5) Imprimir a matriz resultante :)

# Manipulação de arquivos

- Este problema apresenta uma novidade importante: a possibilidade dos dados necessários a um programa serem lidos diretamente a partir de um **arquivo**.
- Sabemos que o armazenamento de **dados** em **variáveis** é **temporário**, isto é, os dados se perdem quando o programa termina sua execução.
- **Arquivos**, por outro lado, são usados para manter grandes quantidades de dados de forma **permanente**.

# Aplicação de arquivos

- Aplicações empresariais como, por exemplo, controle de estoque, processam muitos dados e não seriam possíveis sem o uso de arquivos.
- Os arquivos são armazenados em dispositivos de memória secundária como fitas magnéticas, discos magnéticos e discos óticos.
- Estes dispositivos podem reter grandes volumes de dados por longos períodos de tempo.

# Vantagem do uso de arquivos

- Considere, por exemplo, que o problema recebesse duas matrizes de dimensões **10x10**.
- Se a entrada de dados fosse feita interativamente, o usuário teria que digitar **200 valores!!!**
- Estes valores deveriam ser digitados novamente a cada execução do programa.
- Portanto, é mais fácil armazenar estes valores em arquivo e, sempre que o programa for executado, efetuar a leitura dos valores partir do arquivo.

# Manipulação de arquivos

- A linguagem C não impõe estrutura alguma aos arquivos.
- Do ponto de vista físico, um arquivo é, simplesmente, uma sequência de bytes.
- Mas, do ponto de vista lógico, costuma-se imaginar que os dados armazenados em arquivos são organizados em registros.
- Considere, por exemplo, um sistema de folha de pagamento.



# Manipulação de arquivos

- Em um sistema como esse, cada **registro** corresponde a um **empregado** e pode conter, por exemplo, os seguintes dados:

Número de matrícula;  
Nome do empregado;  
Data de início na empresa;  
Departamento onde trabalha;  
Categoria funcional.

- Cada um dos dados que compõem um registro denomina-se **campo**.

# Manipulação de arquivos

A figura abaixo ilustra a organização lógica de dados em um arquivo:

Arquivo	1245	Pedro	12/03/1985	Recursos Humanos	A05
	1367	Henrique	03/01/1994	Financeiro	A01
	1380	Filipe	31/05/2002	Planejamento	B18
	1432	Bruno	28/11/1999	Administração	B11

Registro	1245	Pedro	12/03/1985	Recursos Humanos	A05
----------	------	-------	------------	------------------	-----

Campo

# Manipulação de arquivos

- Normalmente, a recuperação de dados em um arquivo é feita registro a registro.
- Para facilitar a recuperação, pelo menos um campo deve, univocamente, identificar o registro.
- Este campo especial, que identifica um registro específico, é conhecido como **chave de registro**.
- No exemplo anterior, o **número de matrícula** pode ser a chave do registro.

# Arquivos textuais

- Para utilizar um arquivo de dados, é preciso declarar um ponteiro para o tipo **FILE**.
- Isso é feito da seguinte forma:

```
FILE *arq;
```
- Com isso, a variável **arq** passa a representar, dentro do programa, um arquivo de dados.
- É importante observar que um arquivo de dados é reconhecido pelo sistema operacional como uma entidade externa ao programa.

# Arquivos textuais

- Antes que um arquivo possa ser utilizado, é preciso informar ao sistema operacional **onde localizar o arquivo** e **como se pretende usar o arquivo**.
- Esta operação é conhecida como **abertura do arquivo** e é executada pela função **fopen**.
- Para o **Problema 1** em questão, a abertura do arquivo é feita assim: 

```
arq = fopen("matrizes.txt", "r");
```
- A função **fopen** tem dois parâmetros do tipo string. O primeiro parâmetro especifica o nome do arquivo e, eventualmente, onde localizá-lo.

# Arquivos textuais

- No exemplo anterior, o nome do arquivo é `matrizes.txt`.
- Como não foi fornecida informação sobre sua localização, este arquivo deve estar presente na mesma pasta que contém o arquivo executável gerado (ex: `a.exe`).
- Caso o arquivo não esteja na mesma pasta que o programa executável, é preciso informar a pasta que o contém. Por exemplo:

```
arq = fopen("c:\\ccn\\arquivos\\matrizes.txt", "r");
```

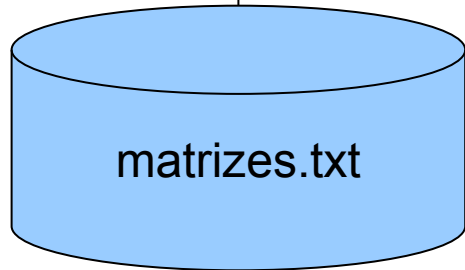
O símbolo ‘\’ é usado para indicar que o símbolo a seguir é um caractere especial.

# Arquivos textuais

- A abertura de um arquivo, se bem sucedida, retorna em uma estrutura do tipo **FILE** várias informações (**dados de ponteiros de arquivo**)
- É um ponteiro para permitir que seus campos sejam alterados por outras funções (ex: fscanf)
- Alguns exemplos dessas informações são:
  - um ponteiro para o buffer associado ao arquivo;
  - o tamanho do buffer;
  - a indicação se o buffer está cheio ou vazio;
  - um ponteiro para o dado atual no arquivo.
- Se, por alguma razão, o arquivo não puder ser aberto, a função **fopen** irá retornar a constante **NULL**

# Arquivos textuais

```
FILE *arq;  
arq = fopen("matrizes.txt", "r");
```



Endereço	Variável	Conteúdo
00FF12	arq.campo1	00FFA4
00FF13	arq.campo2	1024
00FF14	arq.campo3	0
00FF15	arq.campo4	00FFBB
...	...	...
00FF(n)	arq.campo(n)	48

Arq == 00FF12



# Arquivos textuais

- O segundo parâmetro da função `fopen` especifica o que deve ser feito com os dados.
- As possibilidades são:
  - Ler os dados existentes;
  - Gravar dados apagando (caso existam) os dados existentes;
  - Anexar novos dados aos já existentes.
- A instrução abaixo especifica o modo “r”, ou seja, “ler” (“read”) os dados existentes.

```
arq = fopen("matrizes.txt", "r");
```

# Arquivos textuais

- A tabela a seguir mostra os possíveis modos de abertura de um arquivo:

Modo	Significado
“r”	Abre um arquivo existente para leitura de dados; se o arquivo não existir, irá ocorrer um erro.
“w”	Abre um novo arquivo para gravação de dados; se o arquivo já existir, a gravação irá sobrescrever os dados existentes.
“a”	Abre um arquivo para operações de anexação de dados; se o arquivo não existir, será criado um novo arquivo.

# Problema 1

- Considere que um arquivo de dados contém os valores das dimensões (tam. max.: 100) e dos elementos de duas matrizes de números inteiros. Ex:

2 5

4 12 1 78 5

98 9 1994 0 52

79 12 458 2 29

47 19 784 12 8

- Implemente um programa que calcule e exiba a matriz resultante da soma dessas duas matrizes.

# Arquivos textuais

Imagine que o arquivo `matrizes.txt` contenha os dados dispostos como a seguir:

2	5					← Dimensões das matrizes.
4	12	1	78	5	} ma	← Linhas das matrizes.
98	9	1994	0	52		
79	12	458	2	29	} mb	
47	19	784	12	8		

Para resolver o **Problema 1**, estes dados são lidos pela função `fscanf`. Por exemplo:

```
fscanf(arq, "%d %d", &numLinhas, &numColunas);
```

Após o uso de `fscanf`: `numLinhas = 2`, `numColunas = 5`

# Arquivos textuais

- A leitura das duas matrizes de **numLinhas** linhas e **numColunas** colunas é feita, linha por linha, pelas instruções:

```
for (i = 0; i < numLinhas; i++)  
    for (j = 0; j < numColunas; j++)  
        fscanf(arq, "%d", &M1[i][j]);
```

```
for (i = 0; i < numLinhas; i++)  
    for (j = 0; j < numColunas; j++)  
        fscanf(arq, "%d", &M2[i][j]);
```

- **Atenção!!!** Observe que a leitura deve ser feita de acordo com a disposição dos dados no arquivo.

# Solução do Problema 1

```
1  #include <stdio.h>
2  #include <ctype.h>
3  #include <stdlib.h>
4
5  #define MAX_TAM 100
6
7  int main() {
8
9  //criar duas matrizes
10 int M1[MAX_TAM][MAX_TAM], M2[MAX_TAM][MAX_TAM];
11 int numLinhas, numColunas;
12 int i,j;
```

# Solução do Problema 1

```
38 //fazer a soma das matrizes
39 for(i=0; i<numLinhas; i++)
40     for(j=0; j<numColunas; j++)
41         M1[i][j] = M1[i][j] + M2[i][j];
42
43 //imprimir as matrizes
44 for(i=0; i<numLinhas; i++) {
45     for(j=0; j<numColunas; j++)
46         printf("%d ", M1[i][j]);
47     printf("\n");
48 }
49 getch();
50 return 0;
51 }
```

# Solução do Problema 1

Matrizes.txt

2 5

4 12 1 78 5

98 9 1994 0 52

79 12 458 2 29

47 19 784 12 8

Matriz 1 [NxM]

Matriz 2 [NxM]

N = ?

M = ?



# Solução do Problema 1

Ler arquivo:

```
FILE *arq = fopen("matrizes.txt", "r")
```

Matrizes.txt

2 5

4 12 1 78 5

98 9 1994 0 52

79 12 458 2 29

47 19 784 12 8

Matriz 1 [NxM]

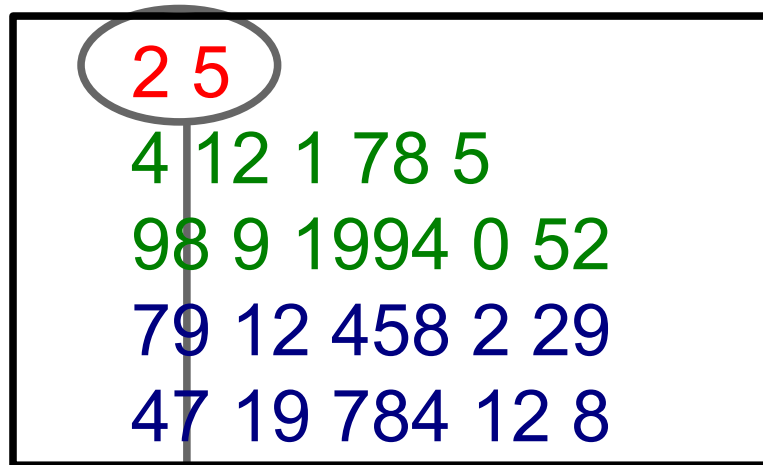
Matriz 2 [NxM]

N = ?

M = ?

# Solução do Problema 1

Matrizes.txt



```
2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8
```

`fscanf(arq, "%d%d", &N, &M)`

`N = 2`

`M = 5`

Matriz 1 [NxM]

Matriz 2 [NxM]

# Solução do Problema 1

Matrizes.txt

2 5  
4 12 1 78 5  
98 9 1994 0 52  
79 12 458 2 29  
47 19 784 12 8

N = 2

M = 5

Matriz 1 [2x5]


Matriz 2 [2x5]


# Solução do Problema 1

Matrizes.txt

```
2 5  
4 12 1 78 5  
98 9 1994 0 52  
79 12 458 2 29  
47 19 784 12 8
```

i = 0

j = 0

Matriz 1 [2x5]


Matriz 2 [2x5]


# Solução do Problema 1

Matrizes.txt

2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8

Matriz 1 [2x5]


Matriz 2 [2x5]


i = 0

j = 0

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8

Matriz 1 [2x5]

4				

Matriz 2 [2x5]


i = 0

j = 0

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5				
4	12	1	78	5	
98	9	1994	0	52	
79	12	458	2	29	
47	19	784	12	8	

Matriz 1 [2x5]

4	12			

Matriz 2 [2x5]


i = 0

j = 1

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5				
4	12	1	78	5	
98	9	1994	0	52	
79	12	458	2	29	
47	19	784	12	8	

Matriz 1 [2x5]

4	12	1		

Matriz 2 [2x5]


i = 0

j = 2



# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5			
4	12	1	78	5
98	9	1994	0	52
79	12	458	2	29
47	19	784	12	8

Matriz 1 [2x5]

4	12	1	78	

Matriz 2 [2x5]


i = 0

j = 3

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5				
4	12	1	78	5	
98	9	1994	0	52	
79	12	458	2	29	
47	19	784	12	8	

Matriz 1 [2x5]

4	12	1	78	5

Matriz 2 [2x5]


i = 0

j = 4

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5				
4	12	1	78	5	
98	9	1994	0	52	
79	12	458	2	29	
47	19	784	12	8	

Matriz 1 [2x5]

4	12	1	78	5
98				

Matriz 2 [2x5]


i = 1  
j = 0

# Solução do Problema 1

```
fscanf(arq, "%d", &Matriz1[i][j])
```

Matrizes.txt

2	5				
4	12	1	78	5	
98	9	1994	0	52	
79	12	458	2	29	
47	19	784	12	8	

Matriz 1 [2x5]

4	12	1	78	5
98	9	1994	0	52

Matriz 2 [2x5]

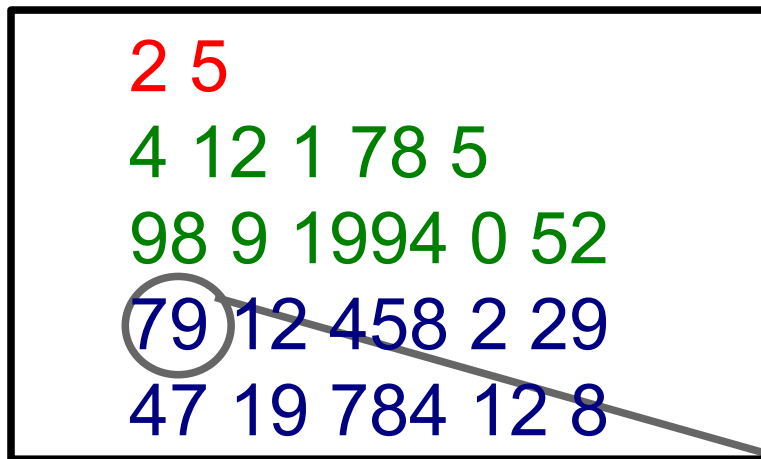

i = 1

j = 4

# Solução do Problema 1

Matrizes.txt

```
2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8
```



Matriz 1 [2x5]

4	12	1	78	5
98	9	1994	0	52

Matriz 2 [2x5]

79				

i = 0

j = 0

```
fscanf(arq, "%d", &Matriz2[i][j])
```

# Solução do Problema 1

Matrizes.txt

```
2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8
```

Matriz 1 [2x5]

4	12	1	78	5
98	9	1994	0	52

Matriz 2 [2x5]

79	12			

i = 0

j = 1

```
fscanf(arq, "%d", &Matriz2[i][j])
```

# Solução do Problema 1

Matrizes.txt

```
2 5
4 12 1 78 5
98 9 1994 0 52
79 12 458 2 29
47 19 784 12 8
```

Matriz 1 [2x5]

4	12	1	78	5
98	9	1994	0	52

Matriz 2 [2x5]

79	12	458	2	29
47	19	784	12	8

i = 1

j = 4

```
fscanf(arq, "%d", &Matriz2[i][j])
```

# Solução do Problema 1

```
15 // abrir o arquivo para leitura
16 FILE *arq;
17 arq = fopen("matrizes.txt","r");
18 if(arq == NULL) {
19     printf("\nErro ao ler o arquivo!");
20     getch();
21     return 1;
22 }
```



# Solução do Problema 1

```
24 //armazenar os dados do arquivo na matriz
25 fscanf(arq, "%d %d", &numLinhas, &numColunas);
26 //Matriz 1
27 for (i = 0; i < numLinhas; i++)
28     for (j = 0; j < numColunas; j++)
29         fscanf(arq, "%d", &M1[i][j]);
30 //Matriz2
31 for (i = 0; i < numLinhas; i++)
32     for (j = 0; j < numColunas; j++)
33         fscanf(arq, "%d", &M2[i][j]);
34
35 //fechar o arquivo
36 fclose(arq);
```

# Arquivos textuais

- Por exemplo, se os dados no arquivo estivessem dispostos como a seguir:

2	5											
4	12	1	78	5	79	12	458	2	29			
98	9	1994	0	52	47	19	784	12	8			

M1

M2

a leitura das matrizes seria como?

# Arquivos textuais

- Por exemplo, se os dados no arquivo estivessem dispostos como a seguir:

2 5

4 12 1 78 5 79 12 458 2 29

98 9 1994 0 52 47 19 784 12 8

M1

M2

a leitura das matrizes seria:

```
for (i = 0; i < numLinhas; i++)  
{  
    for (j = 0; j < numColunas; j++)  
        fscanf(arq, "%d", &M1[i][j]);  
    for (j = 0; j < numColunas; j++)  
        fscanf(arq, "%d", &M2[i][j]);  
}
```

# Arquivos textuais

- Da mesma forma que um programa deve abrir um arquivo antes de usá-lo, deve também fechar o arquivo quando não precisar mais dele.
- Na solução do **Problema 1**, o fechamento do arquivo é feito logo após a leitura das duas matrizes:

```
fclose(arq);
```

- O fechamento de um arquivo instrui o sistema operacional a:
  - esvaziar os buffers associados ao arquivo;
  - liberar os recursos do sistema que o arquivo consumiu.

# Arquivos textuais

- Se o fechamento do arquivo for bem sucedido, a função `fclose` retorna o valor 0.
- Se ocorrer um erro, `fclose` retorna a constante `EOF`, definida em `stdio.h` (igual a -1).
- Se não forem fechados pela função `fclose`, todos os arquivos abertos serão fechados quando o programa terminar sua execução.

# Problema 2

- Considere que uma matriz de distâncias é gerada aleatoriamente.
- Implemente um programa que, dependendo da escolha do usuário, simplesmente exiba a matriz ou, então, escreva a matriz gerada em um arquivo.

# Solução do Problema 2

```
// Exibir ou gravar?
do
{
    printf("\n(E)xibir ou (G)ravar a matriz gerada? ");
    op = toupper(getche());
    printf("\n");
    if (op == 'E')
        arq = stdout; ←
    else
        if (op == 'G')
            arq = fopen("dados33.txt", "w");
}
while ((op != 'E') && (op != 'G'));

// Exibir a matriz no arquivo
fprintf(arq, "%d\n", n);
for (i = 0; i < n; i++)
{
    for (j = 0; j < n; j++)
        fprintf(arq, "%2d ", D[i][j]);
    fprintf(arq, "\n");
}
system("pause");
return 0;
```

**stdout** é o dispositivo padrão de saída: o vídeo. Note que este dispositivo é também tratado como um arquivo.

# Dispositivos padrões de entrada e saída

- Dispositivos padrões de entrada e saída de dados são reconhecidos como arquivos pela linguagem C. Eles são designados por constantes predefinidas:

Constante	Significado	Dispositivo
stdin	Dispositivo padrão de entrada	Teclado
stdout	Dispositivo padrão de saída	Vídeo
stderr	Dispositivo padrão de erro	Vídeo
stdaux	Dispositivo padrão auxiliar	Porta serial
stdprn	Dispositivo padrão de impressão	Porta paralela



# Dispositivos padrões de entrada e saída

- Na solução do **Problema 2**, a matriz de distâncias é gravada no arquivo representado pelo ponteiro **arq**.
- Dependendo da escolha, o arquivo será o dispositivo padrão de saída (**stdout**) ou o arquivo **dados33.txt**.
- A gravação dos dados é feita pela função **fprintf**:

```
fprintf(arq, "%d\n", n);  
for (i = 0; i < n; i++)  
{  
    for (j = 0; j < n; j++)  
        fprintf(arq, "%2d ", D[i][j]);  
    fprintf(arq, "\n");  
}
```

# Dispositivos padrões de entrada e saída

- Considere, por exemplo, a matriz de distâncias:

$$D = \begin{bmatrix} 0 & 10 & 15 & 45 \\ 10 & 0 & 41 & 93 \\ 15 & 41 & 0 & 74 \\ 45 & 93 & 74 & 0 \end{bmatrix}$$

- Usando-se o trecho de código anteriormente exibido, será escrito no arquivo (`stdout` ou `dados33.txt`):

```
4
0 10 15 45
10 0 41 93
15 41 0 74
45 93 74 0
```

# Dispositivos padrões de entrada e saída

- As funções `fscanf` e `fprintf` são generalizações das funções `scanf` e `printf`, podendo ser usadas para quaisquer tipos de arquivos.
- Já as funções `scanf` e `printf` só podem ser usadas para os dispositivos padrões (`stdin` e `stdout`).

```
scanf( ... ) é equivalente a fscanf(stdin, ... )  
printf( ... ) é equivalente a fprintf(stdout, ...)
```

# Dispositivos padrões de entrada e saída

## Atenção!

- Quando se usa `fscanf` para ler informações numéricas em um arquivo, os dados devem estar separados uns dos outros por, pelo menos, um espaço em branco.
- O espaço em branco é considerado como um delimitador para dados numéricos.
- Mas e quando um arquivo contém dados não numéricos como, por exemplo, “`Sao Paulo`”?
- Neste caso, é preciso usar outra forma de leitura, pois os espaços podem fazer parte do próprio dado.

# Problema 3

- Um professor armazena em um arquivo as seguintes informações sobre seus alunos: número (`int`), nome (`string`), notas de duas avaliações (`float`).
- Implemente um programa para listar o conteúdo deste arquivo e exibir a média das notas das avaliações.

# Solução do Problema 3

```
1  #include <stdio.h>
2  #include <ctype.h>
3  #include <stdlib.h>
4  #include <string.h>
5
6  #define MAX_TAM 100
7
8  int main() {
9
10     float notas, media;
11     char buf[MAX_TAM];
12     int num;
13     char *nome;
14     float nota1, nota2;
```

# Solução do Problema 3

```
16 //abrir o arquivo
17 FILE *arq;
18 arq = fopen("alunos.txt","r");
19 if(arq == NULL) {
20     printf("\nErro ao abrir o arquivo!");
21     getch();
22     return 1;
23 }
24
25 printf("\nmatricula\t nome\t\t nota1\t nota2\t");
```

# Solução do Problema 3

```
27 notas=0;
28 media=0;
29 fgets(buf, MAX_TAM, arq);
30 while(!feof(arq)) {
31     num = atoi(strtok(buf, ","));
32     nome = strtok(NULL, ",");
33     nota1 = atof(strtok(NULL, ","));
34     nota2 = atof(strtok(NULL, ","));
35     printf("\n%d \t %s \t %4.1f \t %4.1f", num, nome, nota1, nota2);
36     notas = notas + 2;
37     media = media + nota1 + nota2;
38     fgets(buf, MAX_TAM, arq);
39 }
40 printf("\nmedia = %f", media/notas);
41 fclose(arq);
```



# Leitura de dados não-numéricos

- Quando um arquivo contém informações não numéricas, a leitura dos dados deve ser feita do seguinte modo:

Ler toda uma linha do arquivo como um **string**;  
Extrair do string lido **substrings** que correspondem aos dados;  
Converter cada **substring** para o tipo de dado correspondente.

- A função **fgets** permite ler toda uma linha de um arquivo como um string.

```
fgets(buf, MAX, arq);
```

Vetor que armazenará o string.

Tamanho do string.  
(ou termina quando  
encontra uma quebra  
de linha '\n')

Ponteiro para o arquivo.

# Leitura de dados não-numéricos

- Para extrair os **substrings**, é preciso que, no arquivo, os dados estejam separados uns dos outros por um delimitador, diferente de espaço.
- Considere, por exemplo, que os dados do arquivo **alunos.txt** estejam separados por **vírgulas**:

```
2,Maria da Silva,8.5,4.8  
13,Joao de Almeida,7.5,6.1  
15,Pedro de Souza,5.0,4.4  
21,Jose de Carvalho,7.2,7.1  
30,Silvia Santos,5.9,6.2
```

# Leitura de dados não-numéricos

## Alunos.txt

```
2,Maria da Silva,8.5,4.8  
13,Joao Melo,7.5,6.1  
15,Pedro de Souza,5.0,4.4  
21,Jose de Carvalho,7.2,7.1  
30,Silvia Santos,5.9,6.2
```

## Variáveis

```
#define MAX 1000  
  
File *arq;  
char buf[MAX];
```

# Leitura de dados não-numéricos

Ler arquivo:

```
arq = fopen("Alunos.txt", "r")
```

Alunos.txt

```
2,Maria da Silva,8.5,4.8  
13,Joao Melo,7.5,6.1  
15,Pedro de Souza,5.0,4.4  
21,Jose de Carvalho,7.2,7.1  
30,Silvia Santos,5.9,6.2
```

Variáveis

```
#define MAX 1000  
  
File *arq;  
char buf[MAX];
```

# Leitura de dados não-numéricos

Ler toda uma linha do arquivo como um **string**:

```
fgets(buf, MAX, arq);
```

buf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	,	M	a	r	i	a		d	a		S	i	l	v	a	,	8	.	5	,	4	.	8

Alunos.txt

```
2,Maria da Silva,8.5,4.8
13,Joao Melo,7.5,6.1
15,Pedro de Souza,5.0,4.4
21,Jose de Carvalho,7.2,7.1
30,Silvia Santos,5.9,6.2
```

Variáveis

```
#define MAX 1000

File *arq;
char buf[MAX];
```

# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	,	M	a	r	i	a		d	a		S	i	l	v	a	,	8	.	5	,	4	.	8

Extrair do string lido substrings que correspondem aos dados  
//strtok(buf, ",",") faz isso  
e converter cada **substring** para o tipo de dado correspondente  
num = **atoi**(strtok(buf, ",",") );

Resultado

num = 2

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	\0	M	a	r	i	a		d	a		S	i	l	v	a	,	8	.	5	,	4	.	8

Internamente, a função strtok troca o delimitador já lido por um '\0', para indicar o fim da string

Extrair do string lido substrings que correspondem aos dados  
//strtok(buf, ",") faz isso  
e converter cada **substring** para o tipo de dado correspondente  
num = **atoi**(strtok(buf, ","));

Resultado

num = 2

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	10	M	a	r	i	a		d	a		S	i	l	v	a	,	8	.	5	,	4	.	8

- Extrair do string lido substrings que correspondem aos dados  
`//strtok(NULL, ",")` faz isso
- e converter cada **substring** para o tipo de dado correspondente  
`nome = strtok(NULL, ",");` //nao precisa conv.

Resultado

```
num = 2  
nome = "Maria da Silva"
```

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```



# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	10	M	a	r	i	a		d	a		S	i	l	v	a	,	8	.	5	,	4	.	8

Extrair do string lido substrings que correspondem aos dados  
//strtok(**NULL**, ",", ") faz isso  
e converter cada **substring** para o tipo de dado correspondente  
`nome = strtok(NULL, ",");` //nao precisa conv.

Resultado

```
num = 2  
nome = "Maria da Silva"
```

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	\0	M	a	r	i	a		d	a		S	i	l	v	a	\0	8	.	5	,	4	.	8

Extrair do string lido substrings que correspondem aos dados

```
//strtok(NULL, ",", ") == "8.5"
```

e converter cada **substring** para o tipo de dado correspondente

```
nota1 = atof(strtok(NULL, ",", ") );
```

Resultado

```
num = 2  
nome = "Maria da Silva"  
Nota1 = 8.5
```

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

buf																							
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	\0	M	a	r	i	a		d	a		S	i	l	v	a	\0	8	.	5	\0	4	.	8

Extrair do string lido substrings que correspondem aos dados

```
//strtok(NULL, ",", ") == "4.8"
```

e converter cada **substring** para o tipo de dado correspondente

```
nota2 = atof(strtok(NULL, ",", ") );
```

Resultado

```
num = 2  
nome = "Maria da Silva"  
Nota1 = 8.5  
Nota2 = 4.8
```

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

Ler toda uma linha do arquivo como um **string**:

```
fgets(buf, MAX, arq);
```

buf

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	3	,	J	o	a	o		M	e	l	o	,	7	.	5	,	6	.	1

Alunos.txt

```
2,Maria da Silva,8.5,4.8
13,Joao Melo,7.5,6.1
15,Pedro de Souza,5.0,4.4
21,Jose de Carvalho,7.2,7.1
30,Silvia Santos,5.9,6.2
```

Variáveis

```
#define MAX 1000

File *arq;
char buf[MAX];
```

# Leitura de dados não-numéricos

buf																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	3	,	J	o	a	o		M	e	l	o	,	7	.	5	,	6	.	1

Extrair do string lido substrings que correspondem aos dados  
//strtok(buf, ",",") faz isso  
e converter cada **substring** para o tipo de dado correspondente  
num = **atoi**(strtok(buf, ",",") );

Resultado

num = 13

Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

buf																			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	3	\0	J	o	a	o		M	e	l	o	\0	7	.	5	\0	6	.	1

Extrair do string lido substrings que correspondem aos dados  
`//strtok(buf, ",", ")") == "6.1"`  
e converter cada **substring** para o tipo de dado correspondente  
`nota2 = atof(strtok(buf, ",", ")");`

## Resultado

```
num = 13  
nome = "Joao Melo"  
nota1 = 7.5  
nota2 = 6.1
```

## Variáveis

```
int num;  
char *nome;  
float nota1, nota2;
```

# Leitura de dados não-numéricos

- Assim, após a leitura da primeira linha do arquivo `alunos.txt`, o vetor `buf` irá conter:

```
2 , M a r i a   d a   S i l v a , 8 . 5 , 4 . 8 \0
```

- É preciso agora extrair do vetor `buf` os `substrings` correspondentes aos dados demandados.
- A função `strtok` permite extrair um `substring` (“`token`”) de um vetor de caracteres.
- Requer dois parâmetros: o nome do vetor de caracteres e um string contendo os delimitadores.

# Leitura de dados não-numéricos

- Para extrair o substring referente ao número do aluno, utilizamos a seguinte chamada:

```
strtok(buf, ",");
```

- A função `strtok` retorna o substring “2”. Para converter este `token` para o tipo `int`, usa-se a função `atoi`:

```
num = atoi(strtok(buf, ","));
```

- Para extrair do vetor `buf` os demais `tokens`, usou-se:

```
nome = strtok(NULL, ",");  
nota1 = atof(strtok(NULL, ","));  
nota2 = atof(strtok(NULL, ","));
```



# Leitura de dados não-numéricos

- Observe que, na função `strtok`, o nome do vetor é usado apenas para extrair o primeiro `token`.
- Para extrair os demais `tokens`, usa-se a constante `NULL` como primeiro parâmetro da função `strtok`.
- **Atenção!!** Qualquer caractere diferente do espaço em branco pode ser usado como delimitador.

```
2#Maria da Silva/8.5/4.8  
13#Joao de Almeida/7.5/6.1  
15#Pedro de Souza/5.0/4.4  
21#Jose de Carvalho/7.2/7.1
```

```
num = atoi(strtok(buf, "#/"));  
nome = strtok(NULL, "#/");  
nota1 = atof(strtok(NULL, "#/"));  
nota2 = atof(strtok(NULL, "#/"));
```

# Problema 4

Modifique o programa anterior de modo a permitir que novos alunos sejam incluídos no arquivo.

# Solução: leAluno()

```
8 void leAluno() {
9     int num;
10    char* nome;
11    float nota1, nota2;
12    //abrir o arquivo
13    FILE *arq;
14    arq = fopen("alunos.txt", "a");
15    if(arq == NULL) {
16        printf("\nErro ao abrir o arquivo!");
17        getch();
18        return;
19    }
```

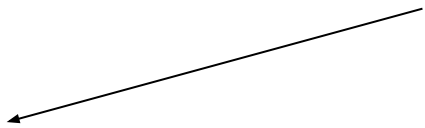
# Solução: leAluno()

```
21     printf("\nDigite os dados do novo aluno");
22     printf("\nNum: ");
23     scanf("%d", &num);
24     printf("\nNome: ");
25     fflush(stdin);
26     gets(nome);
27     printf("\nNota1: ");
28     scanf("%f", &nota1);
29     printf("\nNota2: ");
30     scanf("%f", &nota2);
31
32     fprintf(arq, "%d,%s,%4.1f,%4.1f\n", num, nome, nota1, nota2);
33     fclose(arq);
34
35 }
```

# Solução: leAluno()

```
21     printf("\nDigite os dados do novo aluno");
22     printf("\nNum: ");
23     scanf("%d", &num);
24     printf("\nNome: ");
25     fflush(stdin);
26     fgets(nome, MAX_TAM, stdin);
27     printf("\nNota1: ");
28     scanf("%f", &nota1);
29     printf("\nNota2: ");
30     scanf("%f", &nota2);
31
32     fprintf(arq, "%d,%s,%4.1f,%4.1f\n", num, nome, nota1, nota2);
33     fclose(arq);
34
35 }
```

Mais seguro!



# Solução: leAluno()

- No programa anterior incluiu-se a função `leAluno()`. Nesta função, o arquivo `alunos.txt` é aberto como:

```
arq = fopen("alunos.txt", "a");
```

- Ou seja, o arquivo é aberto no modo “a” para permitir operações de **anexação de dados**.
- Outra observação interessante é o uso da função `gets` para ler o nome do aluno que pode conter espaços em branco:

Limpa o buffer do dispositivo padrão de entrada.

```
printf(" Nome ..... ");  
fflush(stdin);  
gets(nome);
```