

Algoritmos e Programação
de Computadores I

Linguagem C

Vetores, Matrizes e Strings

Profa. Dra. Luanna Lopes Lobato

luanna@ufcat.edu.br

Departamento de Ciência da Computação
Universidade Federal de Catalão – Catalão, GO

Matrizes Unidimensionais (Vetor)

- Matriz: coleção de variáveis do mesmo tipo
 - Referenciada por um nome comum
 - Acessado por um índice
 - Pode ter 1 ou várias dimensões (**Vetor e Matriz**)
- Matrizes são posições contíguas na memória
 - Endereço mais baixo: 1º elemento
 - Endereço mais alto: Último elemento
- Matriz mais comum em C: **string**
 - Matriz de caracter terminada por um nulo

Relembrando... Vetor

- Forma geral de matriz unidimensional (Vetor):
 - *Tipo nome_var[tamanho];*
 - *char vetor[10]; //vetor[0] a vetor[9]*
- Vetor (como outras variáveis) devem ser explicitamente declaradas
 - Para que o compilador aloque espaço

```
void main(void)
{
    int x[100]; /* isto reserva 100 elementos inteiros */
    int t;

    for(t=0; t<100; ++t) x[t] = t;
}
```

Vetor

- Quantidade de armazenamento necessário para guardar uma matriz
 - Relacionada com seu tamanho e seu tipo
 - Matriz unidimensional, o tamanho em bytes é cálculo:
 - Total em bytes = `sizeof(tipo) * tamanho da matriz`
 - C não tem verificação de limites em matrizes
 - É possível ultrapassar o tamanho e escrever nos dados de algum outra variável ou mesmo no código do programa
 - Como programador vc TEM que fazer controle
 - Verificação dos limites onde for necessário

Vetor

- Exemplo de código com erros em código, mas que compilará sem erro.

```
int count[10], i;
```

```
/* isto faz com que count seja ultrapassada */  
for(i=0; i<100; i++) count[i]= i;
```

Vetor

- Matrizes Unidimensionais == Listas de informações de mesmo tipo
 - Armazenadas em posições contíguas da memória em uma ordem de índice

```
char a[7];
```

Elemento	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]
Endereço	1000	1001	1002	1003	1004	1005	1006

Figura 4.1 Uma matriz de sete elementos começando na posição 1000.

Inicialização de Vetor dimensionado

C permite a inicialização de matrizes no momento da declaração. A forma geral de uma inicialização de matriz é semelhante à de outras variáveis, como mostrado aqui:

```
especificador_de_tipo nome_da_matriz[tamanho1]...[tamanhoN] = {lista_valores};
```

A *lista_valores* é uma lista separada por vírgulas de constantes cujo tipo é compatível com *especificador_de_tipo*. A primeira constante é colocada na primeira posição, da matriz, a segunda, na segunda posição e assim por diante. Observe o ponto-e-vírgula que segue o }.

No exemplo seguinte, uma matriz inteira de dez elementos é inicializada com os números de 1 a 10:

```
■ int i[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

Isso significa que **i[0]** terá o valor 1 e **i[9]** terá o valor 10.

Inicialização de Vetor Não-Dimensionado

Imagine que você esteja usando inicialização de matrizes para construir uma tabela de mensagens de erro, como mostrado aqui:

```
char e1[17] = "erro de leitura\n";  
char e2[17] = "erro de escrita\n";  
char e3[29] = "arquivo não pode ser aberto\n";
```

Como você poderia supor, é tedioso contar os caracteres em cada mensagem, manualmente, para determinar a dimensão correta da matriz. Você pode deixar C calcular automaticamente as dimensões da matriz, usando matrizes não dimensionadas. Se, em um comando de inicialização de matriz, o tamanho da matriz não é especificado, o compilador C cria uma matriz grande o bastante para conter todos os inicializadores presentes. Isso é chamado de *matriz não-dimensionada*. Usando essa abordagem, a tabela de mensagens torna-se

Inicialização de Vetor Não-Dimensionado

```
char e1[] = "Erro de leitura\n";  
char e2[] = "Erro de escrita\n";  
char e3[] = "Arquivo não pode ser aberto\n";
```

Dadas essas inicializações, este comando

```
printf("%s tem comprimento %d\n", e2, sizeof e2);
```

mostrará

erro de escrita tem comprimento 17

Vetores e Ponteiros

(curiosidade... APCII)

```
int *p;  
int sample[10];  
  
p = sample;
```

- Matrizes e Ponteiros estão relacionados
 - Uma discussão sobre um deles, refere-se ao outro
 - Capítulo 5 -> Ponteiros
- Pode-se gerar um ponteiro para o 1º elemento simplesmente especificando o nome da Matriz
- Pode-se especificar o endereço do 1º elemento de uma Matriz
 - Operador &
 - *sample* e *&sample[0]* produzem os mesmos resultados
 - Porém, em códigos C, é difícil ver *&sample[0]*

Passando Vetor para Função

- Pode-se passar como argumento para uma função

```
int vet[10];  
func1(vet);
```

- Se a função recebe o vetor, o parâmetro formal pode ser declarado de 3 formas:
 - Como um ponteiro
 - especificando o nome da Matriz sem o índice
 - Como uma matriz dimensionada
 - Como uma matriz não-dimensionada

Passando Vetor para Função

```
void func1(int *x) /* ponteiro */
```

```
{  
    .  
    .  
    .  
}
```

```
void func1(int x[10]) /* matriz dimensionada */
```

```
{  
    .  
    .  
    .  
}
```

```
void func1(int x[]) /* matriz não-dimensionada */
```

```
{  
    .  
    .  
    .  
}
```

Passando Vetor para Função

- Todos os três modos de declaração produzem resultados idênticos
 - Cada um diz ao compilador que um ponteiro inteiro será recebido
 - 1º declaração usa, de fato, um ponteiro
 - 2º emprega a declaração de Matriz padrão
 - 3º simplesmente especifica que uma Matriz do tipo **int**, de algum tamanho, será recebida
 - O comprimento da Matriz não importa à função
 - Pois C não verifica limites
 - No entanto, deve-se estar atento a isso!

Passando Vetor para Função

- No entanto, isso funciona

```
void func1(int x[32])  
{  
    .  
    .  
    .  
}
```

- O compilador C gera um código que instrui **func1()** a receber um ponteiro – não cria realmente uma matriz de 32 elementos

Vetor e Strings

- Uso mais comum de Vetor é como **string de caracteres**
 - Em C uma string é definida como uma matriz de caracter, terminada por nulo ('\0') e geralmente é zero
 - Importante declarar matrizes de caracteres como sendo um caracter mais longo que a maior string
- Embora C não tenha um tipo de dado string, ele permite constantes string
 - Uma *constante string* é uma lista de caracter entre ""
 - "Olá Mundo"
 - Compilador C adiciona '\0' automaticamente

Inicialização de Vetor

Matrizes de caracteres que contêm strings permitem uma inicialização abreviada que toma a forma:

```
char nome_da_matriz[tamanho] = "string";
```

Por exemplo, este fragmento de código inicializa **str** com a frase "Eu gosto de C".

```
■ char str[14] = "Eu gosto de C";
```

Isso é o mesmo que escrever

```
■ char str[14] = {'E', 'u', ' ', 'g', 'o', 's', 't', 'o', ' ', ' ', 'd', 'e', ' ',  
                 ' ', 'C', '\0'};
```

Como todas as strings em C terminam com um nulo, você deve ter certeza de que a matriz a ser declarada é longa o bastante para incluir o nulo. Isso explica porque **str** tem comprimento de 14 caracteres, muito embora "Eu gosto de C" tenha apenas 13. Quando você usa uma constante string, o compilador automaticamente fornece o terminador nulo.

Exemplo de código

```
#include <stdio.h>
#include <string.h>

void main(void) {
    char s1[80], s2[80];
    gets(s1);
    gets(s2);
    printf("Comprimentos: %d %d\n", strlen(s1), strlen(s2));
    if(!strcmp(s1, s2)) printf ("As strings s,,o iguais\n");

    strcat(s1, s2);
    printf("%s\n", s1);

    strcpy(s1, "Isto , um teste.\n");
    printf(s1);
    if(strchr("al'", "'")) printf("' est em al'\n");
    if(strstr("ol aqui", "ol ")) printf("ol encontrado");
}
```

Explicação das funções usadas no código

- Cabeçalho padrão **string.h**

Nome	Função
<code>strcpy(s1, s2)</code>	Copia s2 em s1.
<code>strcat(s1, s2)</code>	Concatena s2 ao final de s1.
<code>strlen(s1)</code>	Retorna o tamanho de s1.
<code>strcmp(s1, s2)</code>	Retorna 0 se s1 e s2 são iguais; menor que 0 se s1<s2; maior que 0 se s1>s2.
<code>strchr(s1, ch)</code>	Retorna um ponteiro para a primeira ocorrência de ch em s1.
<code>strstr(s1, s2)</code>	Retorna um ponteiro para a primeira ocorrência de s2 em s1.

Exemplo de execução

- Entrada:
 - *alo alo*
- Saída:
 - *comprimentos: 3 3*
 - *As strings são iguais*
 - *Aloalo*
 - *Isso é um teste.*
 - *o está em alo*
 - *ola encontrado*

Matrizes Bidimensionais (Matriz)

- C suporta matrizes multidimensionais
 - Bidimensionais é a forma mais simples
 - Uma matriz de matrizes unidimensionais

- Declaração

```
int matriz[10][20];
```

- Manipulação de posições

```
matriz[1][2]=10;
```

- O exemplo a seguir carrega uma matriz bidimensional com os números de 1 a 12 e escreve-os linha por linha...

Matriz

Neste exemplo, **num[0][0]** tem o valor 1, **num[0][1]**, o valor 2, **num[0][2]**, o valor 3 e assim por diante. O valor de **num[2][3]** será 12. Você pode visualizar a matriz **num** como mostrada aqui:

num [t] [i]	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Matriz

- São armazenadas em uma matriz linha-coluna
- Veja Figura 4.2
 - **Pode-se pensar no primeiro índice como um “ponteiro” para a linha corrente.**
- N° de bytes de memória para armazenar
 - *bytes = tamanho do 1° índice * tamanho do 2° índice * sizeof (tipo base)*
 - Assumindo inteiros de 2 bytes, uma matriz 10x5:
 $10 * 5 * 2 = 100$ bytes alocados

Matriz

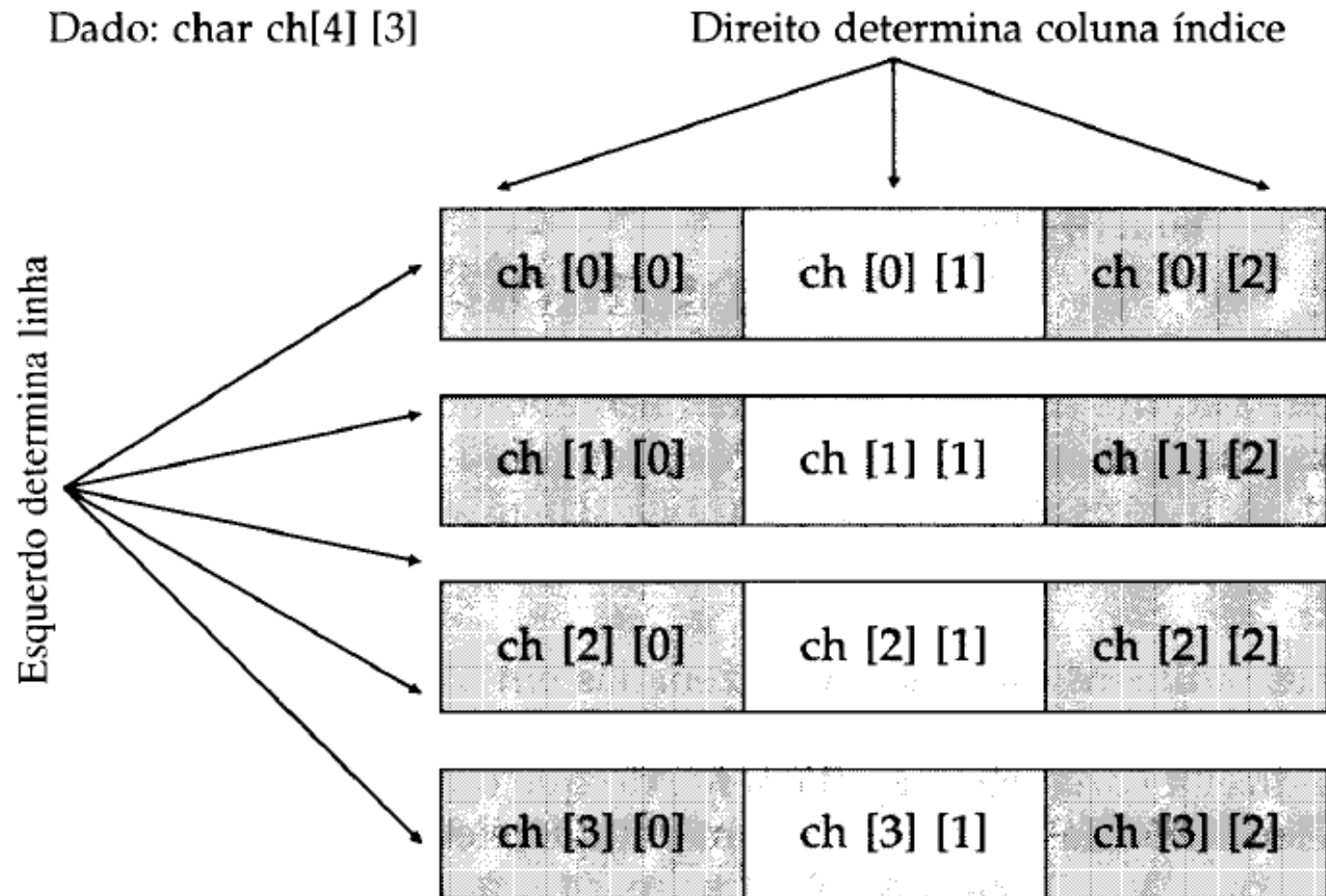


Figura 4.2 Uma matriz bidimensional na memória.

Inicialização de Matriz

Matrizes multidimensionais são inicializadas da mesma forma que matrizes unidimensionais. Por exemplo, o código seguinte inicializa `sqrs` com os números de 1 a 10 e seus quadrados.

```
int sqrs[10][2] = {  
    1,1,  
    2,4,  
    3,9,  
    4,16,  
    5,25,  
    6,36,  
    7,49,  
    8,64,  
    9,81,  
    10,100  
};
```

```
double tabela[3][2] = {  
    {1.0, 0.0}, /* linha 0 */  
    {-7.8, 1.3}, /* linha 1 */  
    {6.5, 0.0} /* linha 2 */  
};
```


Inicialização de Matrizes Não-Dimensionadas

Além de ser menos tediosa, a inicialização de matrizes *não-dimensionadas* permite a você alterar qualquer mensagem sem se preocupar em usar uma matriz de dimensões incorretas.

Inicializações de matrizes não-dimensionadas não estão restritas a matrizes unidimensionais. Para matrizes multidimensionais, você deve especificar todas, exceto a dimensão mais à esquerda, para que o compilador possa indexar a matriz de forma apropriada. Desta forma, você pode construir tabelas de comprimentos variáveis e o compilador aloca automaticamente armazenamento suficiente para guardá-las. Por exemplo, a declaração de `sqrs` como uma matriz não-dimensionada é mostrada aqui:

```
int sqrs[][2] = {  
    1,1,  
    2,4,  
    3,9,  
    4,16,  
    5,25,  
    6,36,  
    7,49,  
    8,64,  
    9,81,  
    10,100  
};
```

A vantagem dessa declaração sobre a versão que especifica o tamanho é que você pode aumentar ou diminuir a tabela sem alterar as dimensões da matriz.

Exemplo de código

```
#include <stdio.h>
```

```
void main(void) {  
    int t, i, num[3][4];
```

```
    for(t=0; t<3; ++t)  
        for(i=0; i<4; ++i)  
            num[t][i] = (t*4)+i+1;
```

```
    /* agora imprima-os */  
    for(t=0; t<3; ++t) {  
        for(i=0; i<4; ++i)  
            printf("%3d ", num[t][i]);  
        printf("\n");  
    }  
}
```

Matrizes Bidimensionais

- Como argumento para a função
 - Apenas um ponteiro para o 1º elemento é passado
 - A função que recebe uma matriz como parâmetro deve definir ao menos o comprimento da 2ª dimensão
 - O compilador precisa saber o comprimento de cada linha para indexar a matriz corretamente

Matrizes Bidimensionais

- Exemplo:
 - Função recebe uma matriz bidimensional de inteiro com dimensões 10x10

```
void func1(int x[][10])  
{  
    .  
    .  
    .  
}
```

Exemplos com o uso de Matriz



```
#include <stdio.h>
```

```
#define ALTURA 5
```

```
#define LARGURA 5
```

```
int main(){
```

```
    int x, y;                /* numero da coluna e linha */
```

```
    int matriz [ALTURA] [LARGURA]; /* array 2-D [num_lins, num_cols] */
```

```
    /* preenche a matriz com zeros */
```

```
    y = 0;
```

```
    while(y < ALTURA){
```

```
        x = 0;
```

```
        while(x < LARGURA){
```

```
            matriz[y][x] = 0;
```

```
            x+=1;
```

```
        }
```

```
        y+=1;
```

```
    }
```

```
    /* Imprime a matriz com zeros e a coordenada escolhida com 1 */
```

```
    printf("\nEntre coordenadas na forma y,x (Ex.: 2,4).\n");
```

```
    printf("Use valores negativos para sair do programa.\n");
```

```
    printf("Coordenadas: ");
```

```
    scanf("%d,%d", &y, &x);
```

```
    while (x >= 0 && y >= 0){
```

```
        matriz[y][x] = 1; /*coloca 1 no elemento escolhido */
```

```
        y = 0;
```

```
        while (y < ALTURA){ /* imprime o array todo */
```

```
            x = 0;
```

```
            while (x < LARGURA){
```

```
                printf("%d ", matriz[y][x] );
```

```
                x += 1;
```

```
            }
```

```
            printf("\n\n");
```

```
            y += 1;
```

```
        }
```

```
    printf("\n");
```

```
    printf("Coordenadas: ");
```

```
    scanf("%d,%d", &y, &x);
```

```
    }
```

```
}
```

```
/* Preenche os elementos de um array bidimensional com os
valores que representam a taboada e imprime a matriz*/
```

```
#include <stdio.h>
```

```
#define LIN 11
```

```
#define COL 11
```

```
int main()
{
    int x;           /* numero da coluna */
    int y;           /* numero da linha */
    int tabela[LIN] [COL]; /* tabela de taboada */
```

```
/* preenche a tabela */
```

```
for(y=0; y < LIN; y+=1)
    for(x=0; x < COL; x+=1)
        tabela[y][x] = y*x;
```

```
printf("\n      Tabela de Multiplicacao\n");
```

```
/* Imprime o numero das colunas */
```

```
printf("%6d", 0);
for (x=1; x < COL; x+=1)
    printf("%3d", x);
printf("\n");
```

```
/* Imprime uma linha horizontal */
```

```
printf(" ");
for (x=0; x < 3*COL; x+=1)
    printf("-");
printf("\n");
```

```
/* Imprime as linhas da tablea.
```

```
    Cada linha a precedida pelo indice de linha e uma barra
vertical */
```

```
for (y=0; y < LIN; y+=1) {
    printf("%2d |", y);
    for(x=0; x < COL; x+=1)
        printf("%3d", tabela[y][x]);
    printf("\n");
}
```

```
}
```

```
#include <stdio.h>
#define DIM 3
// Programação em linguagem de programação C!

int main()
{
    int matriz[DIM][DIM];
    int linha, coluna;

    //escrevendo na Matriz
    for(linha = 0 ; linha < DIM ; linha++)
        for(coluna = 0 ; coluna < DIM ; coluna++)
        {
            printf("Elemento [%d][%d]: ", linha+1, coluna+1);
            scanf("%d", &matriz[linha][coluna]);
        }

    // imprimindo a matriz na tela
    for(linha = 0 ; linha < DIM ; linha++)
    {
        for(coluna = 0 ; coluna < DIM ; coluna++)
            printf("%3d", matriz[linha][coluna]);

        printf("\n"); //após cada linha ser impressa, um salto de linha
    }
}
```


Exemplos com o uso de Matriz e Funções



```
#include <stdio.h>
#include <stdlib.h>
```

```
int t, i;
```

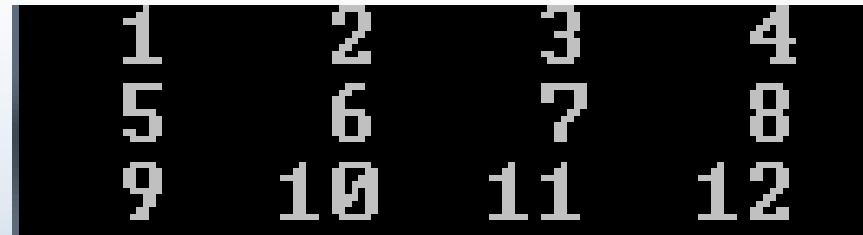
```
void carregaMatriz(int num[3][4]){
    for(t=0; t<3; ++t)
        for(i=0; i<4; ++i)
            num[t][i] = (t*4)+i+1;
}
```

```
void imprimeMatriz(int num[3][4]){
    for(t=0; t<3; ++t) {
        for(i=0; i<4; ++i)
            printf("%3d ", num[t][i]);
        printf("\n");
    }
}
```

```
main() {
    int num[3][4];

    carregaMatriz(num);
    imprimeMatriz(num);

    system("pause");
}
```



1	2	3	4
5	6	7	8
9	10	11	12

```

/* Exemplo de array 2-D - tabela de multiplicacao */
#include <stdio.h>
#define LIN 11
#define COL 11

void inicializa_arr (int arr[][COL], int);
void imprime_arr (int arr[][COL], int);

int main(){
    int tabela[LIN] [COL];

    inicializa_arr(tabela, LIN);
    printf("\n      Tabela de Multiplicacao\n");
    imprime_arr(tabela, LIN);
}

/* Inicializa o array com a tabela de multiplicacao */
void inicializa_arr (int arr[][COL], int nLIN){
    int x, y;          /* numero da coluna e linha*/
    /* preenche o array */
    for (y=0; y < nLIN; y+=1)
        for(x=0; x < COL; x+=1)
            arr[y][x] = y*x;
}

```

```

/* imprime um array LIN x COL */
void imprime_arr(int arr[][COL], int nlin){
    int x, y;          /* numero da coluna e linha */
    /* imprime o numero das colunas */
    printf("%6d", 0);
    for (x=1; x < COL; x+=1)
        printf("%3d", x);
    printf("\n");

    /* imprime uma linha horizontal */
    printf(" ");
    for (x=0; x < 3*COL; x+=1)
        printf("_");
    printf("\n");

    /* imprime as linhas do array. cada linha e' precedida
    pelo numero da linha e uma barra vertical */

    for (y=0; y < nlin; y+=1) {
        printf("%2d|", y);
        for(x=0; x < COL; x+=1)
            printf("%3d", arr[y][x]);
        printf("\n");
    }
}

```

Exemplo de código

- Matriz para armazenar as notas numéricas de cada aluno de uma sala de aula
 - Professor tem 3 turmas
 - 5 alunos por turma
- Note a maneira como a matriz **grade** é acessada em cada uma das funções...

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <locale.h> //p setlocale

/* Um banco de dados simples para
notas de alunos */
#define CLASSES 3
#define GRADES 5
char ch, str[80];

int grade [CLASSES][GRADES];

void imprimeMenu(void);
void enter_grades(void);
int get_grade(int num);
void disp_grades(int g[][GRADES]);
```

```

main() {
    setlocale(LC_ALL, ""); //acentuação
    imprimeMenu();
    system("pause");
}

```

```

void imprimeMenu(void){
    for(;;) {
        do {
            printf("\n*****\n");
            printf("*** Escolha a opção ***\n");
            printf("*** (E)ntar notas ***\n");
            printf("*** (M)ostrar notas ***\n");
            printf("*** (S)air ***\n");
            printf("*****\n");
            gets(str);
            ch = toupper(*str);
        } while(ch!='E' && ch!='M' && ch!='S');

        switch(ch) {
            case 'E':
                enter_grades();    break;
            case 'M':
                disp_grades(grade);    break;
            case 'S':    exit(0);
        }
    }
}

```

```
/* Entra com a nota dos alunos. */
```

```
void enter_grades(void){
```

```
    int t, i;
```

```
    for(t=0; t<CLASSES; t++) {
```

```
        printf("\n**** Turma #%d ****\n", t+1);
```

```
        for(i=0; i<GRADES; ++i)
```

```
            grade[t][i]= get_grade(i);
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
/* Lê uma nota. */
```

```
int get_grade(int num){
```

```
    char s[80];
```

```
    printf("Entre a nota do aluno # %d:\n",num+1);
```

```
    gets(s);
```

```
    return(atoi(s));
```

```
}
```

```
/* Mostra as notas. */
```

```
void disp_grades(int g[][GRADES]){
```

```
    int t, i;
```

```
    for(t=0; t<CLASSES; ++t) {
```

```
        printf("\n**** Turma #%d ****\n", t+1);
```

```
        for(i=0; i<GRADES; ++i)
```

```
            printf("Aluno #%d e %d\n", i+1, g[t][i]);
```

```
    }
```

```
    printf("\n");
```

```
}
```

Exercício – em sala de aula

- Implementar o exemplo anterior:
 - Função para cálculo da média por turma
 - Informar qual turma teve maior média

Matrizes de String

- O processador de entrada de um BD pode verificar os comandos do usuário com base em uma string de comandos válidos
- Criar uma matriz de string
 - Matriz bidimensional de caracteres
- Matriz de 30 strings, com comprimento máximo de 79 caracteres

```
char str_array[30][80];
```

Matrizes de String

- Acessar uma string individual

- Especifica apenas o índice esquerdo

- ```
//gets p a 3º string em str_array
gets(str_array[2]);
```

Equivalente a:

```
gets(&str_array[2][0]);
```

A 1º é mais comum em códigos profissionais

# Exemplo de código

- Exemplo:
  - Programa que usa uma matriz de string como base para um editor de texto simples:
    - Recebe linhas de texto até que uma linha em branco seja inserida (**lê por linha**)
    - Então mostra novamente cada linha um caracter por vez (**escreve por linha e coluna**)

```
#include <stdio.h>
#define MAX 100
#define LEN 80

char text[MAX][LEN];

/* Um editor de texto muito simples*/
main() {
 register int t, i, j;
 printf("Entre com as frases para seu texto.\n");
 printf("Entre com uma linha vazia para sair.\n");

 for(t=0; t<MAX; t++) {
 printf("%d: ", t);
 gets(text[t]);
 if(!*text[t]) break; /* sai com linha em branco */
 }

 for(i=0; i<t; i++) {
 for(j=0; text[i][j]; j++)
 putchar(text[i][j]);
 putchar('\n');
 }
}
```

# Matrizes Multidimensionais

- C permite matriz com **mais de 2 dimensões**
- O limite exato é determinado pelo compilador
- Forma geral de Matriz Multidimensional:  
*tipo nome[tamanho1][tamanho2][tamanho3]...[tamanhoN];*
- Não são frequentemente usadas
  - Devido a quantidade de memória necessária
  - Uma matriz tamanhos 10x6x9
    - Se guardasse char de 2 bytes  $10 \times 6 \times 9 \times 2 = 1.080$  bytes
    - Se guardasse inteiros de 4 bytes  $10 \times 6 \times 9 \times 4 = 2.160$  bytes

# Matrizes Multidimensionais

- O **armazenamento cresce exponencialmente** com o número de dimensões
- Grandes matrizes multidimensionais são alocadas, geralmente, dinamicamente
  - Uma parte por vez, com as funções de **alocação dinâmica** e **ponteiros**
  - Chamada Matriz esparsa, discutida no Capítulo 21
- Acessar um elemento nessa matriz é **mais lento** do que acessar um elemento em uma Unidimensional
  - Toma-se tempo do computador para calcular cada índice
- Passando para função deve-se declarar todas, menos a 1ª, dimensão (veremos isto mais a frente...)

# Matrizes Multidimensionais

```
int Mat[4][3][6][5];
```

- Uma função, que recebe a matriz Mat:

```
void func1(int d[][3][6][5])
{
 .
 .
 .
}
```

- Pode incluir a 1ª dimensão se quiser!
- Veremos esse assunto mais a frente... ;)

# Indexando Ponteiros para uso em Matriz (detalhes em APCII)

- Ponteiros e Matrizes estão relacionados em C
  - Nome de matriz, sem índice, é ponteiro para o 1º elemento

```
char p[10];
```

As seguintes sentenças são iguais:

```
p
&p[0]
```

Colocando de outra forma,

```
p == &p[0]
```

é avaliado como verdadeiro, porque o endereço do primeiro elemento de uma matriz é o mesmo que o da matriz.



# Indexando Ponteiros

Reciprocamente, qualquer ponteiro pode ser indexado como se uma matriz fosse declarada. Por exemplo, considere este fragmento de programa:

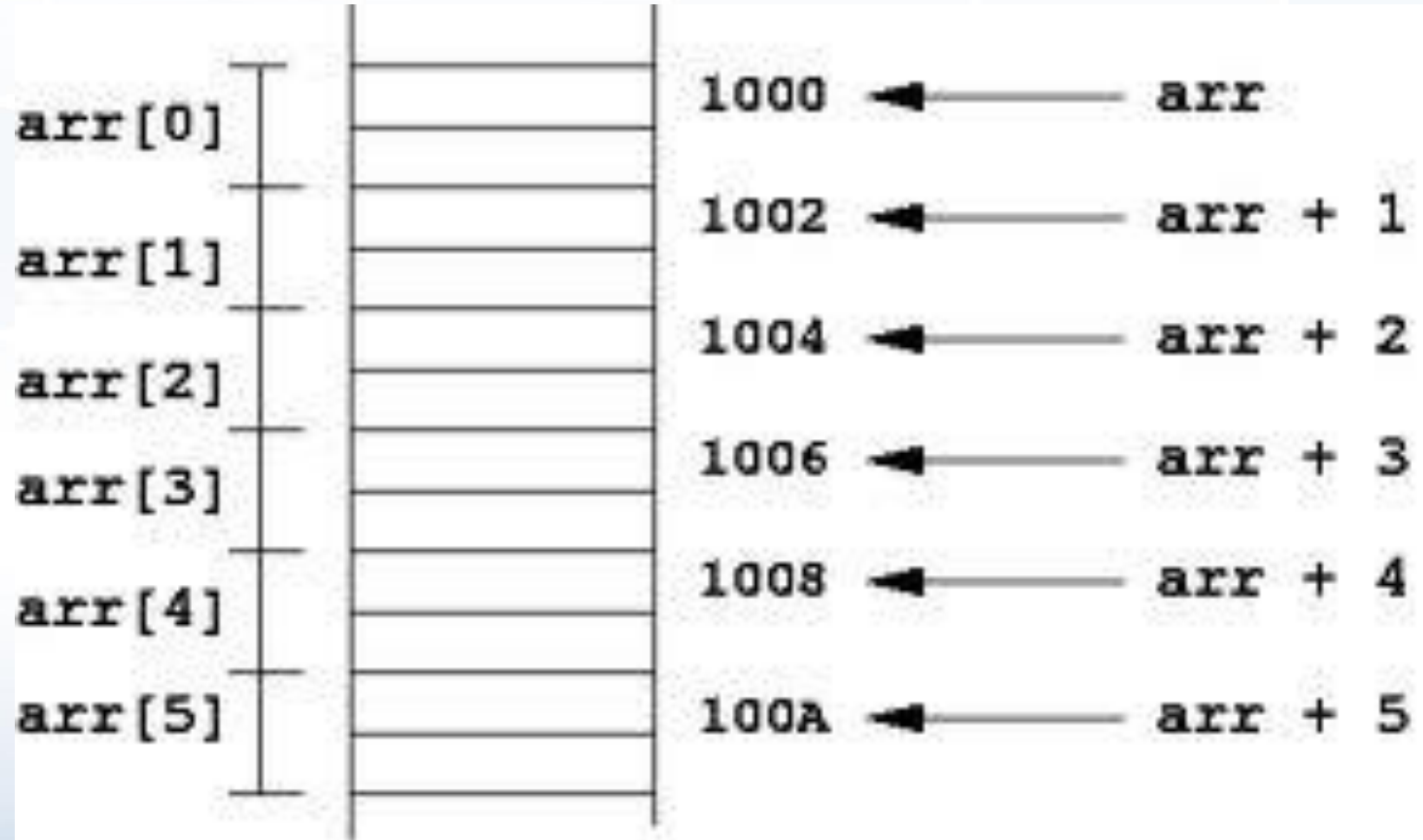
```
int *p, i[10];
p = i;
p[5] = 100; /* atribui usando o índice */
*(p+5) = 100; /*atribui usando aritmética de ponteiros */
```

Os dois comandos de atribuição colocam o valor 100 no sexto elemento de **i**. O primeiro elemento indexa **p**; o segundo usa aritmética de ponteiro. De qualquer forma, o resultado é o mesmo. (O Capítulo 5 discute ponteiros e aritmética de ponteiros.)

# Indexando Ponteiros

- Esse processo pode ser aplicado a matrizes de 2 ou mais dimensões:
  - Matriz 10x10, as duas sentenças são equivalentes
    - `a` equivalente a `&a[0][0]`
  - O elemento `[0][4]` pode ser referenciado de duas formas:
    - `a[0][4]` equivalente a `* (a+4)`
    - `a[1][2]` equivalente a `* (a+12)`
    - `a[j][k]` equivalente a `* (a+(comprimento das linhas)+k)`

# Indexando Ponteiros



# Acessando Matriz por Ponteiros

- Ponteiros são usados para acessar matrizes pois a **aritmética de ponteiros é geralmente mais fácil**
- Matriz bidimensional é semelhante **matriz de ponteiros**
  - Que apontam para **matrizes de linhas**
- Usar uma variável de ponteiro torna-se uma maneira fácil de utilizar ponteiros para acessar os elementos de uma matriz bidimensional

# Acessando Matriz por Ponteiros

- Imprime o conteúdo da linha especificada da matriz

```
int num[10][10];
.
.
.
void pr_row(int j)
{
 int *p, t;

 p = &num[j][0]; /* obtém o endereço do primeiro
 elemento na linha j */

 for(t=0; t<10; ++t) printf("%d ", *(p+t));
}
```

# Acessando Matriz por Ponteiros

Você pode generalizar essa rotina usando como argumentos de chamada a linha, o comprimento das linhas e um ponteiro para o primeiro elemento da matriz, como mostrado aqui:

```
void pr_row(int j, int row_dimension, int *p)
{
 int t;

 p = p + (j * row_dimension);

 for(t=0; t<row_dimension; ++t)
 printf("%d ", *(p+t));
}
```

# Acessando Matriz por Ponteiros

Matrizes de dimensões maiores que dois podem ser reduzidas de forma semelhante. Por exemplo, uma matriz tridimensional pode ser reduzida a um ponteiro para uma matriz bidimensional, que pode ser reduzida a um ponteiro para uma matriz unidimensional. Genericamente, um matriz  $n$ -dimensional pode ser reduzida a um ponteiro para uma matriz  $(n-1)$ -dimensional. Essa nova matriz pode ser reduzida novamente com o mesmo método. O processo termina quando uma matriz unidimensional é produzida.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#define MAX 3
#define LEN 10
```

```
main(){
 char text[MAX][LEN];
 int t, i, j;
 printf("Entre com uma linha vazia para sair.\n");

 for(t=0; t<MAX; t++) {
 printf("%d: ", t);
 gets(text[t]);
 //printf("\n\n endereço: %p - Conteúdo: %s \n\n", *text[t], text[t]);
 if(!*text[t])
 break;//sai com linha em branco
 }
```

```
//system("cls"); //LIMPAR TELA
```

```
for(i=0; i<t; i++) {
 for(j=0; text[i][j]; j++)
 putchar(text[i][j]);
 putchar('\n');
}
system("pause");
}
```



# Exemplo: Passagem de Vetor com ponteiro

```
#include <stdio.h>
#include <stdlib.h>
```

```
void funcao(int *p){
 printf("\nEndereço=%p\n", p);
 printf("\nEndereço=%d\n", p);
 printf("\nOutro exemplo=%d\n", p[0]);
 printf("\n vet[0]=%d\n", *p);
 printf("\n vet[1]=%d\n", *(++p));
 p++;
 printf("\n vet[2]=%d\n", *p);
 printf("\nOutro exemplo=%d\n", p[2]);
}
```

```
main(){
 int i, vet[]={1,2,3};
 printf("\nValor de vet[0]=%d\n", vet[0]);
 funcao(vet);
 system("pause");
}
```



# Lista 11 de Exercícios – APC1 e LAB1

## Entrega pelo SIGAA

Data: 09/12/2021 até as 14:00hrs

Obs.: Enviar por email, para [luanna@ufcat.edu.br](mailto:luanna@ufcat.edu.br)

# Assunto: **Lista 11 de Exercícios APC1 e LAB1**

# Exercícios 1)

## Cap 3: Comandos de Controle do Programa

- Implementar exercícios do livro, Capítulo 3.
- |                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                                                                                                                             |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"><li>• Números mágicos #1 e #2 pág 63</li><li>• Números mágicos #3 pág 64</li><li>• Números mágicos #4 pág 65</li><li>• 2 Programas pág 67</li><li>• Números mágicos #5 pág 68</li><li>• Programa pág 69</li><li>• Programa pág 71</li></ul> | <ul style="list-style-type: none"><li>• Programa pág 72</li><li>• Programa pág 74</li><li>• Último programa pág 76</li><li>• Programa pág 77</li><li>• 2 Programas pág 78</li><li>• Programa pág 82</li><li>• Programa pág 84</li><li>• 1º programa pág 87</li><li>• 2 programas pág 89</li><li>• Programa pág 91</li></ul> |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## Exercícios 2)

- Implementar exercícios do livro C Completo e Total.
- Programa pág 97
- Programa pág 98
- Programa pág 100
- Programa pág 103
- Dois programas pág 106



# Trabalhos Finais – APC1 e LAB1

Entrega pelo SIGAA

Data: 14/12/2021 até as 14:00hrs

Obs.: Enviar por email, para

[luanna@ufcat.edu.br](mailto:luanna@ufcat.edu.br)

Assunto: **Trabalhos Finais de APC1 e LAB1**

# Sistema 1)

- Implementar o Jogo da Velha
  - **Não pode ser igual ao do livro**
    - Crie com suas características
  - Entrega: 4 pessoas
- Exemplo de Jogo da Velha
  - pág 109 C Completo e Total



## Sistema 2)

- Em uma cidade do Sul do país, sabe-se que de Janeiro a Abril (assumindo 121 dias), não ocorreu temperatura inferior a 15 C nem superior a 30 C. As temperaturas registradas em cada dia estão armazenadas em documentos da central do tempo. Assim, sua empresa foi contratada para desenvolver um sistema que calcule e imprima:
  - a. A menor temperatura ocorrida
  - b. A maior temperatura ocorrida
  - c. A temperatura média
  - d. O número de dias no qual a temperatura foi menor e maior do que a temperatura média.
- Os dados devem estar “amarrados” no código já que é obtido pelo documento da central do tempo.
  - Use vetor ou matriz para armazenar os dados!!!
- Use funções para verificar cada uma das opções acima, passando o vetor como ponteiro para essas funções.

## Sistema 3)

- Sua empresa foi contratada para o desenvolver um sistema que gerencie reservas de passagens aéreas da companhia.
  - O sistema deve permitir a leitura do número dos vôos e a respectiva quantidade de lugares disponíveis, além de pedidos de reserva, constituídos do número da carteira de identidade do cliente e do vôo desejado.
  - Para cada cliente, o sistema deve verificar se há disponibilidade no vôo desejado. Em caso afirmativo, o sistema deve imprimir o número da identidade do cliente e o número do vôo, atualizando o número de lugares disponíveis.
  - Caso contrário, o sistema deve avisar ao cliente da inexistência de lugares. A companhia área tem 30 números de vôos.
  - Use vetor ou matriz para armazenar os dados
  - Use funções para verificar cada uma das opções acima, passando o vetor como ponteiro para essas funções.



## Sistema 4)

- Sua empresa foi contratada para o desenvolvimento de um sistema que corrige as provas de múltipla escolha (com 05 opções de resposta e apenas uma correta).
- Cada **prova tem 10 questões, cada uma valendo 1.0**. O primeiro conjunto de dados a ser lido será o gabarito para a correção da prova. Os outros dados consistem da matrícula dos alunos e suas respectivas respostas, e o último número, do aluno fictício, será 9999.
- O sistema deve calcular e imprimir:
  - a. para cada aluno, o seu número de matrícula e a sua nota.
  - b. a porcentagem de aprovação, sabendo-se que a nota mínima para aprovação é 7.0;
  - c. a nota que teve a maior frequência absoluta.
- Use vetor ou matriz para armazenar as informações.
- Use funções para desenvolver o sistema, passando o vetor como ponteiro para essas funções.

# Sistema 5)

- Em um supermercado, há um equipamento que efetua a leitura do estoque de mercadorias, lendo o código das mercadorias e as quantidades. Em seguida, são feitos os pedidos dos clientes.
- Sua empresa deve desenvolver um sistema que permita a atualização do estoque de tal modo:
  - a. seja lido e mostrado o estoque inicial (máximo de 100 mercadorias)
  - b. sejam lidos os pedidos dos clientes, constituídos, cada um, do número do cliente, código da mercadoria e quantidade desejada
  - c. seja verificado, para cada pedido, se ele pode ser integralmente atendido. Em caso negativo, mostrar o número do cliente e uma mensagem de aviso que não possui a mercadoria em estoque de forma suficiente. O sistema deve informar quantidade atual e o cliente pode optar por fazer um novo pedido com a quantidade suficiente
  - d. seja atualizado o estoque após cada operação
  - e. seja listado o estoque final.
- Use vetor ou matriz para armazenar os dados!
- Use funções para verificar cada uma das opções acima, passando o vetor como ponteiro para essas funções.