
Linguagem C: Ponteiros

Sumário

- Ponteiros;
- Variáveis ponteiros;
- Operadores de ponteiros;
- Expressões com ponteiros;
 - Atribuição de ponteiros;
 - Aritmética de ponteiros;
 - Comparação de ponteiros;
- Ponteiros e Matrizes;
 - Matrizes de Ponteiros;

Ponteiros

- A linguagem C é altamente **dependente de ponteiros**. Para ser um bom programador em C é fundamental que se tenha um bom domínio deles;
- Um ponteiro é uma variável que **contém um endereço de memória**;
- Assim como um int guarda inteiro, um float guarda ponto flutuante e um char guarda caracteres, o ponteiro guarda um endereço de memória;

Ponteiros

- Alguns motivos para se usar ponteiros:
 - **Passagem de parâmetros** para uma função por referência. Passamos o endereço (um ponteiro) da variável argumento;
 - Forma elegante de **passar matrizes e strings** como argumentos para funções;
 - A utilização sensata de ponteiros deixa o **programa mais rápido**;
 - Ponteiros são a base para a criação de **estruturas de dados mais avançadas**, como listas, pilhas, filas, árvores, etc(Estrutura de Dados I);

Declaração de ponteiros

- Para declarar um ponteiro temos a seguinte forma geral:

```
tipo_do_ponteiro *nome_da_variavel;
```

- O asterisco (*) que faz o compilador saber que aquela variável não vai guardar um valor mas um endereço para aquele tipo especificado;
- Vejamos exemplos de declaração de ponteiros:

```
int *a, *b, *c;
```

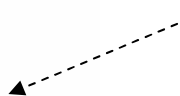
```
char *nome;
```

Operadores de ponteiros

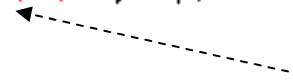
- Existem dois operadores especiais para ponteiros: * e & :
 - O & é um operador unário que devolve o endereço na memória do seu operando;
 - O * é um operador unário que devolve o valor da variável localizada no endereço que o segue;
- Por exemplo, o que seria na tela no programa a seguir?

```
#include<stdio.h>
int main()
{
    int *cont, m;
    m = &cont;
    printf("\n\nO valor de m eh: %p\n\n", m);
    system("pause");
    return 0;
}
```

O endereço de memória que contém a variável cont

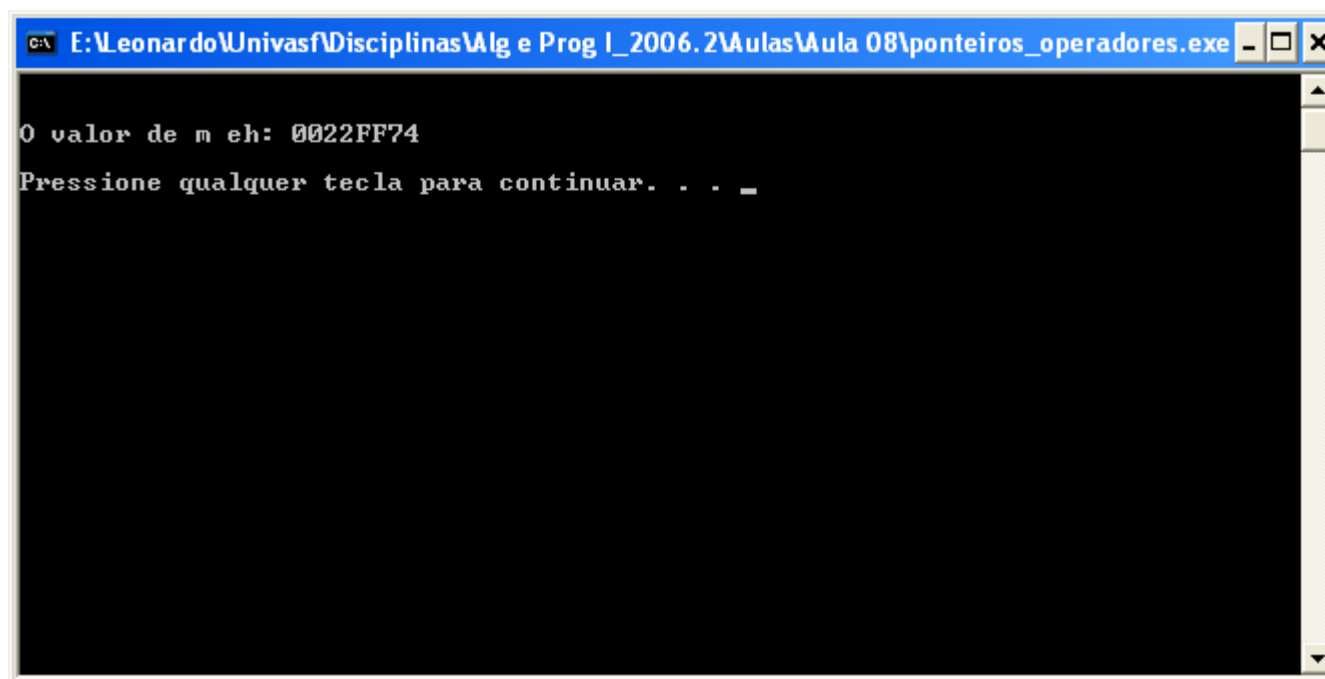


Código %p usado na na função printf() para indicar que ela deve imprimir um endereço em Hexa



Operadores de ponteiros

- A saída no console para o programa anterior será:



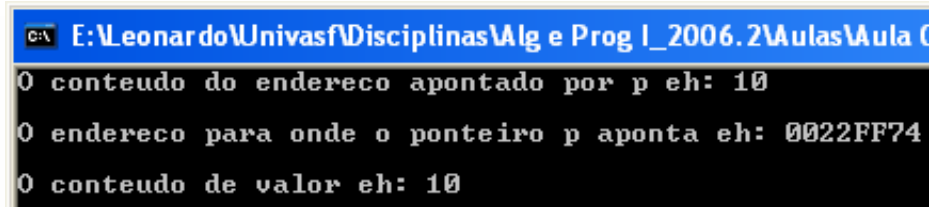
```
C:\ E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_operadores.exe
O valor de m eh: 0022FF74
Pressione qualquer tecla para continuar. . . _
```

Operadores de ponteiros

- Vejamos outro exemplo que manipula ponteiros:

Saída no console:

```
#include<stdio.h>
int main()
{
    int num, valor;
    int *p;
    num = 10;
    p = &num;
    valor = *p;
    printf("O conteudo do endereco apontado por p eh: %d\n", *p);
    printf("\nO endereco para onde o ponteiro p aponta eh: %p", p);
    printf("\n\nO conteudo de valor eh: %d\n\n", valor);
    system("pause");
    return 0;
}
```



```
C:\ E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 0
O conteudo do endereco apontado por p eh: 10
O endereco para onde o ponteiro p aponta eh: 0022FF74
O conteudo de valor eh: 10
```

Expressões com ponteiros

- Em geral, expressões envolvendo ponteiros concordam com as mesmas regras de qualquer outra expressão de C;
- Vejamos alguns poucos aspectos especiais de expressões com ponteiros:
 - Atribuição de ponteiros;
 - Aritmética de ponteiros;
 - Comparação de ponteiros;

Atribuição de ponteiros

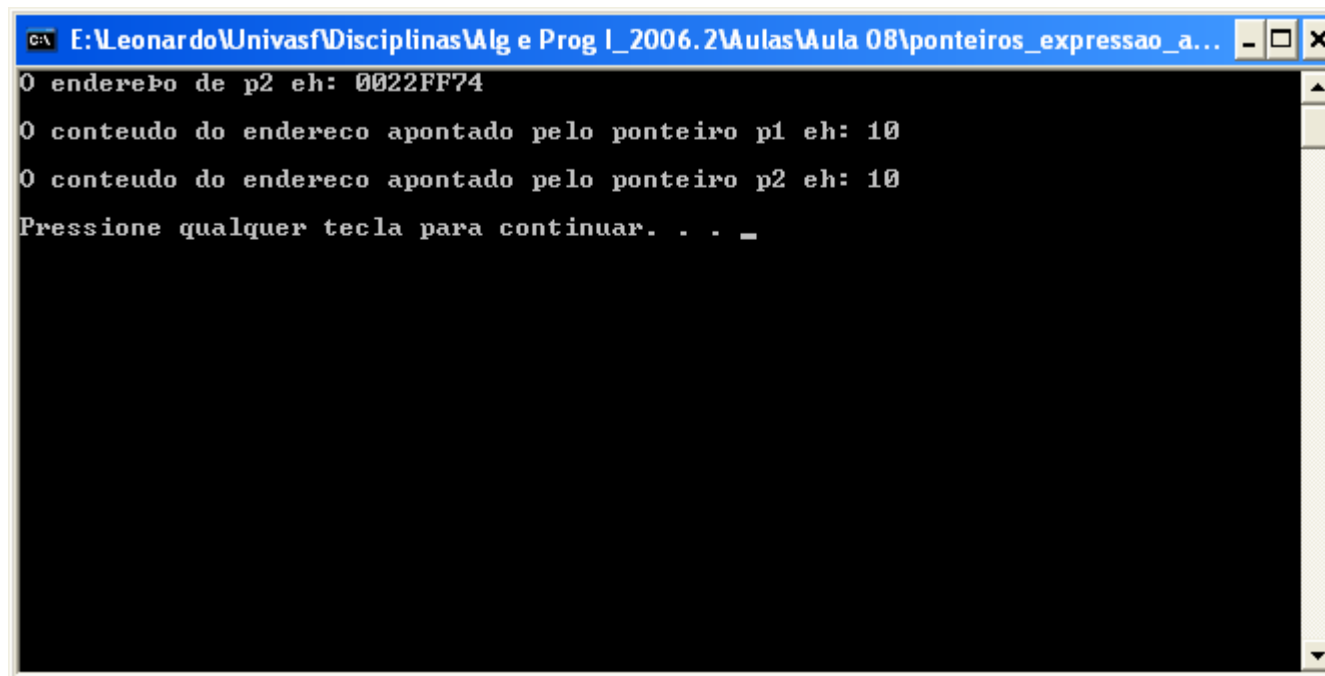
- Como é o caso com qualquer variável, um ponteiro pode ser usado no lado direito de um comando de atribuição para passar seu valor para outro ponteiro, vejamos:

```
#include<stdio.h>
int main()
{
    int x=10;
    int *p1, *p2;
    p1 = &x;
    p2 = p1;
    printf("O endereço de p2 eh: %p\n\n", p2);
    printf("O conteudo do endereco apontado pelo ponteiro p1 eh: %d\n\n", *p1);
    printf("O conteudo do endereco apontado pelo ponteiro p2 eh: %d\n\n", *p2);
    system("pause");
}
```

Tanto p1 quanto p2 apontam para o endereço de memória da variável x

Atribuição de ponteiros

- A saída no console para o programa anterior será:



```
C:\ E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_expressao_a... - □ ×
0 endereço de p2 eh: 0022FF74
0 conteúdo do endereço apontado pelo ponteiro p1 eh: 10
0 conteúdo do endereço apontado pelo ponteiro p2 eh: 10
Pressione qualquer tecla para continuar. . . _
```

Aritmética de ponteiros

- Existem apenas duas operações aritméticas que podem ser usadas com ponteiros:
 - Adição (incremento) e
 - Subtração (decremento).
- Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta;
 - Se tivermos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro;

Aritmética de ponteiros

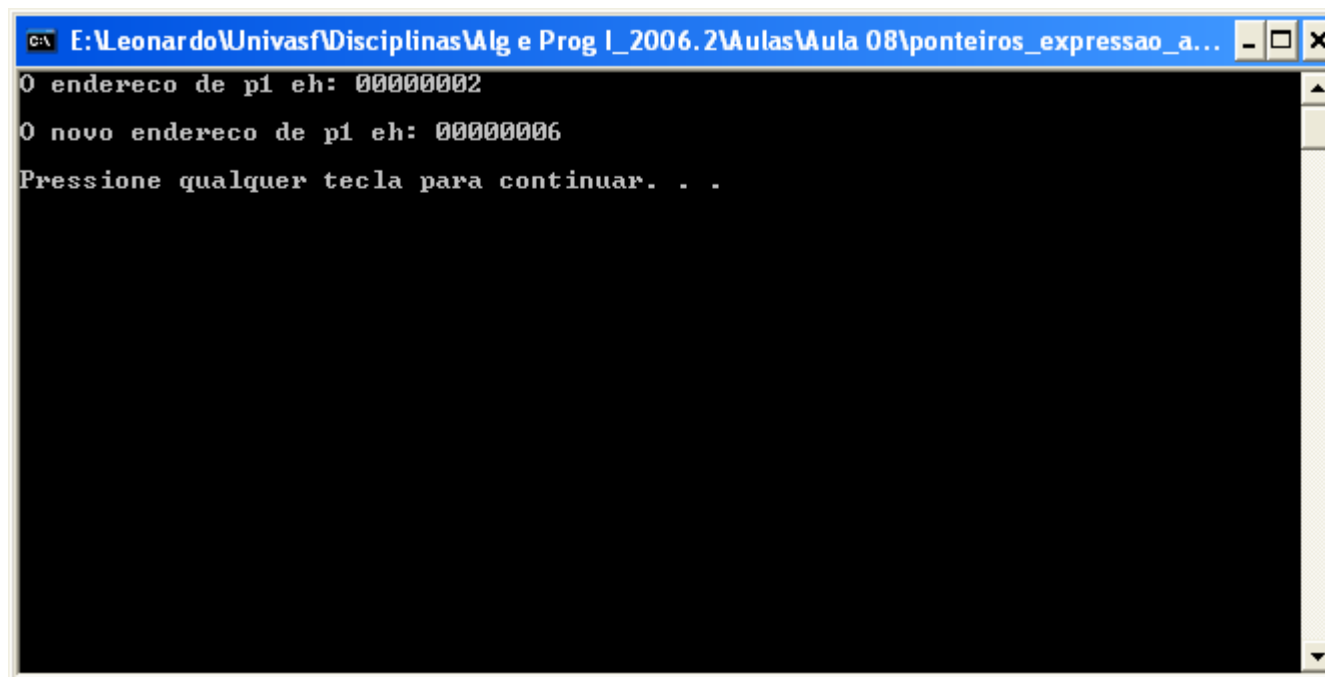
- Isso justifica a necessidade do compilador conhecer o tipo de um ponteiro;
 - Se incrementarmos um ponteiro `char *` ele anda 1 byte na memória;
 - Se incrementarmos um ponteiro `double *` ele anda 8 bytes na memória;

```
#include<stdio.h>
int main()
{
    float *p1;
    printf("O endereço de p1 eh: %p",p1);
    p1++;
    printf("\n\nO novo endereço de p1 eh: %p\n\n",p1);
    system("pause");
}
```

p1 apontará para o próximo elemento do tipo float (4 bytes adiante)

Aritmética de ponteiros

- A saída no console para o programa anterior será:



```

C:\> E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_expressao_a...
0 endereco de p1 eh: 00000002
0 novo endereco de p1 eh: 00000006
Pressione qualquer tecla para continuar. . .

```

Aritmética de ponteiros

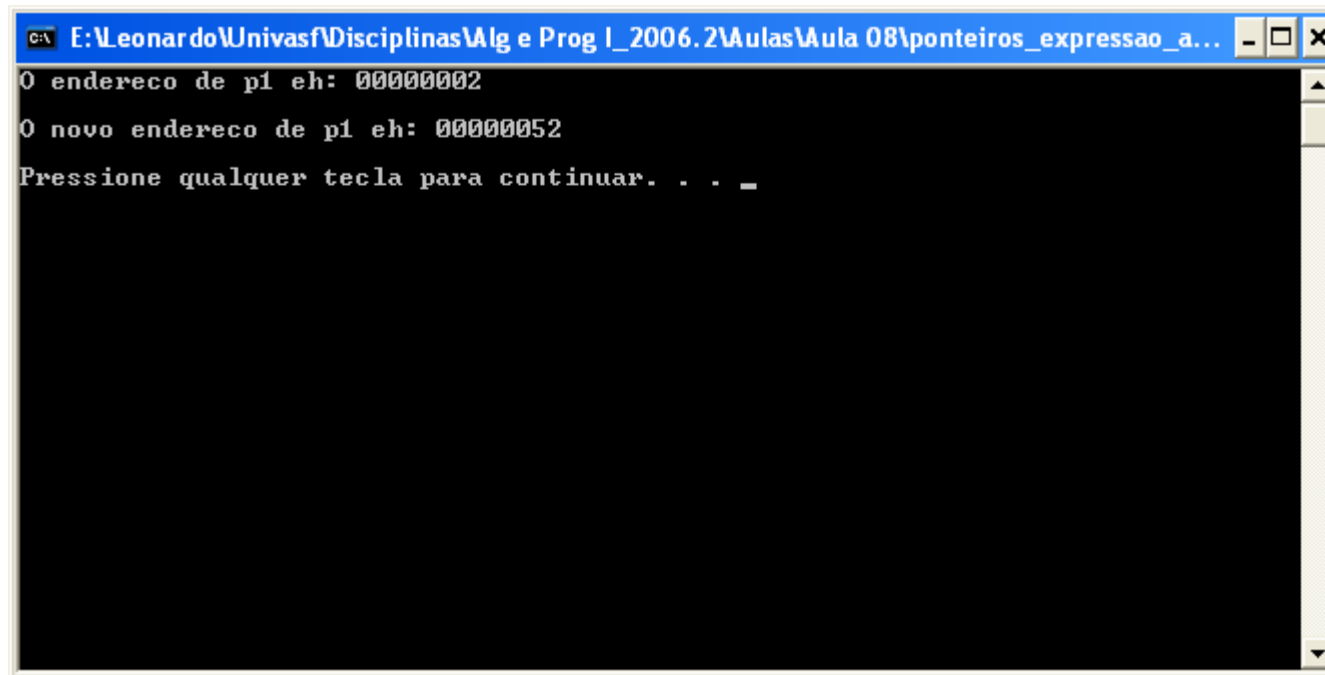
- A aritmética de ponteiros não se limita apenas ao incremento e decremento, podemos somar ou subtrair inteiros de ponteiros, vejamos:

```
#include<stdio.h>
int main()
{
    float *p1;
    printf("O endereço de p1 eh: %p",p1);
    p1 = p1+20;      //equivale à p1+=20;
    printf("\n\nO novo endereço de p1 eh: %p\n\n",p1);
    system("pause");
}
```

p1 apontará para o 20º elemento do tipo float adiante (20 x 4 bytes adiante)

Aritmética de ponteiros

- A saída no console para o programa anterior será:



```
C:\ E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_expressao_a... - □ ×
0 endereco de p1 eh: 00000002
0 novo endereco de p1 eh: 00000052
Pressione qualquer tecla para continuar. . . _
```


Aritmética de ponteiros

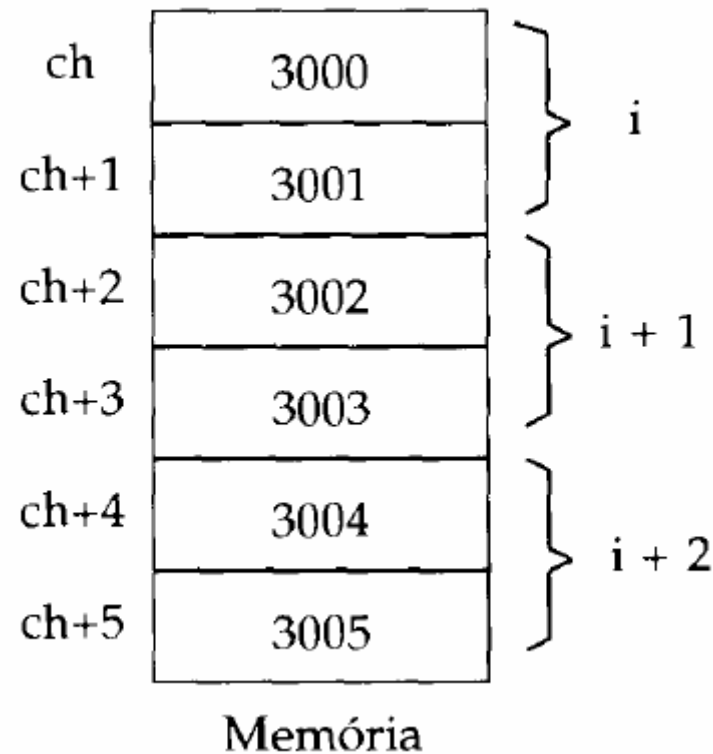
- Toda aritmética de ponteiros é relativa à seu tipo base:

```
char *ch;
```

```
int *i;
```

```
i = ch;
```

Os ponteiros `*ch` e `*i` apontam para a mesma posição de memória. Supondo esse endereço de memória igual a 3000.



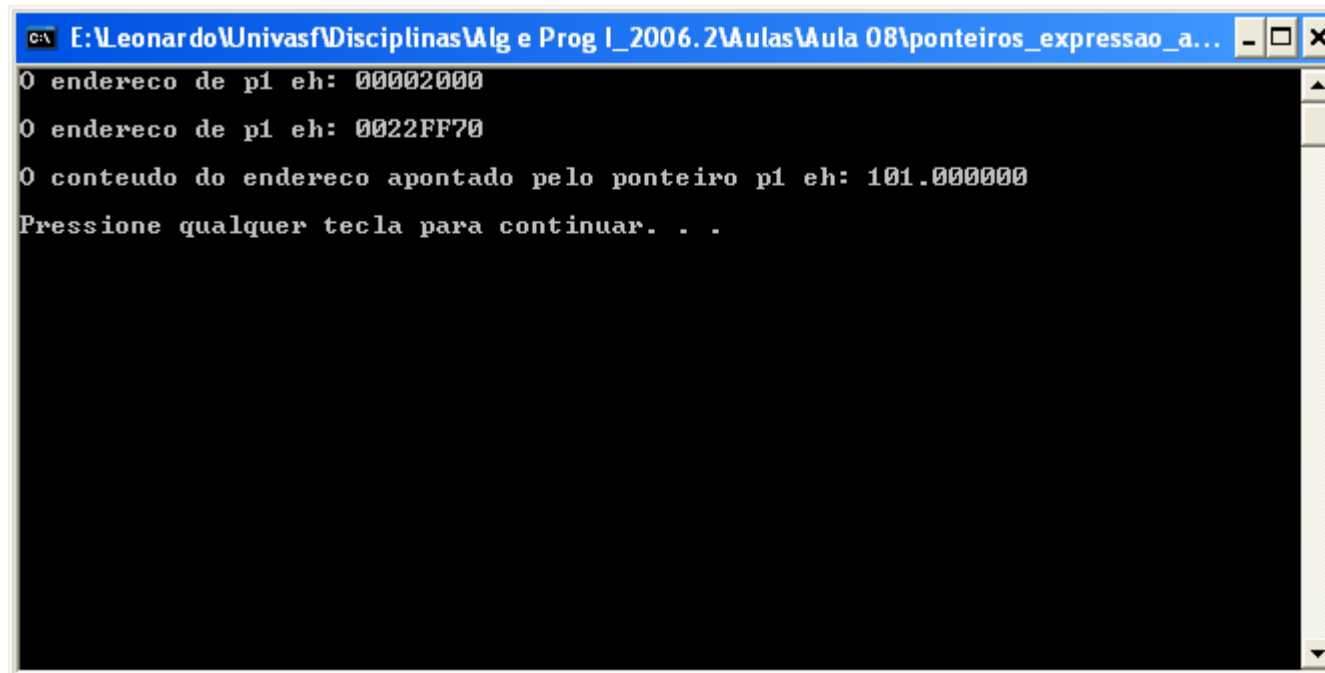
Aritmética de ponteiros

- Como faríamos para incrementar o conteúdo da variável apontada por um ponteiro p qualquer?
 - `(*p)++`

```
#include<stdio.h>
int main()
{
    float *p1, valor=100;
    p1 = 0x2000;
    printf("O endereço de p1 eh: %p",p1);
    p1 = &valor;
    printf("\n\nO endereço de p1 eh: %p",p1);
    (*p1)++;
    printf("\n\nO conteúdo do endereço apontado pelo ponteiro p1 eh: %f\n\n", *p1);
    system("pause");
}
```

Aritmética de ponteiros

- A saída no console para o programa anterior será:



```
C:\E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_expressao_a... - □ ×
0 endereco de p1 eh: 00002000
0 endereco de p1 eh: 0022FF70
0 conteudo do endereco apontado pelo ponteiro p1 eh: 101.000000
Pressione qualquer tecla para continuar. . .
```

Aritmética de ponteiros

- Além de adição e subtração entre um ponteiro e um inteiro, nenhuma outra operação aritmética pode ser efetuada com ponteiros;
 - Não podemos multiplicar ou dividir ponteiros;
 - Não podemos aplicar os operadores de deslocamento e de mascaramento bit a bit com ponteiros;
 - Não podemos adicionar ou subtrair o tipo float ou o tipo double a ponteiros;

Comparação de ponteiros

- É possível comparar dois ponteiros em uma expressão relacional (<, <=, > e >=) ou se eles são iguais ou diferentes (== e !=);
- A comparação entre dois ponteiros se escreve como a comparação entre outras duas variáveis quaisquer, vejamos:

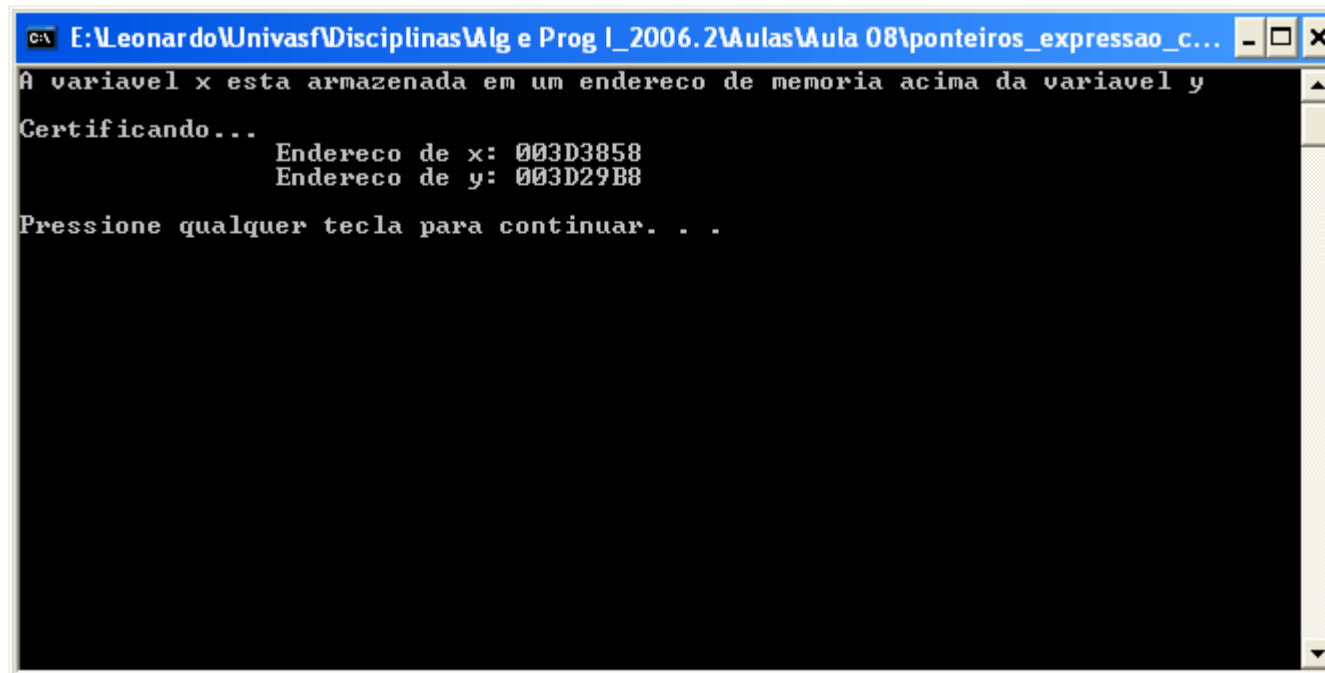
Comparação de ponteiros

- Vejamos:

```
#include<stdio.h>
int main()
{
    int x=10, y=10;
    int *p1, *p2;
    p1 = &x;
    p2 = &y;
    if(p1>p2)
        printf("A variavel x esta armazenada em um endereco de memoria acima da variavel y");
    else
        printf("\n\nA variavel y esta armazenada em um endereco de memoria acima da variavel x");
    printf("\n\nCertificando...\n\t\tEndereco de x: %p \n\t\tEndereco de y: %p\n\n");
    system("pause");
}
```

Comparação de ponteiros

- A saída no console para o programa anterior será:



```
E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_expressao_c... - □ ×
A variavel x esta armazenada em um endereco de memoria acima da variavel y
Certificando...
      Endereco de x: 003D3858
      Endereco de y: 003D29B8
Pressione qualquer tecla para continuar. . .
```

Ponteiros e Matrizes

- Há uma estreita relação entre ponteiros e matrizes;
- Quando declaramos uma matriz da seguinte forma, por exemplo:

```
int mat[20][30];
```

- O compilador *C* calcula o tamanho, em bytes necessário para armazenar esta matriz. Este tamanho é:

20 x 30 x 2 bytes

- O compilador então aloca este número de bytes em um espaço livre de memória;

Ponteiros e Matrizes

- O nome da variável que declaramos é na verdade um ponteiro para o tipo da variável da matriz;
- Consideremos o seguinte trecho de código:

```
char str[30], *p;
```

```
p = str;
```

- Neste caso p foi inicializado com o endereço do primeiro elemento da matriz st;
- Como faríamos para acessar o quinto elemento em str?

```
str[4];          ou          *(p + 4);
```

Ponteiros e Matrizes

- Vejamos um dos usos mais importantes dos ponteiros:
A varredura sequencial de uma matriz:

```
#include<stdio.h>
/* Este programa demonstra a varredura sequencial de uma matriz */
int main()
{
    char str[30], *p;
    p = str;
    printf("Este programa demonstra a varredura sequencial de uma matriz com ponteiros");
    printf("\n\nDigite um nome: ");
    gets(str);
    printf("\nO nome digitado foi: ");
    while(*p)
        printf("%c", *p++);
    printf("\n\n\n");
    system("pause");
}
```

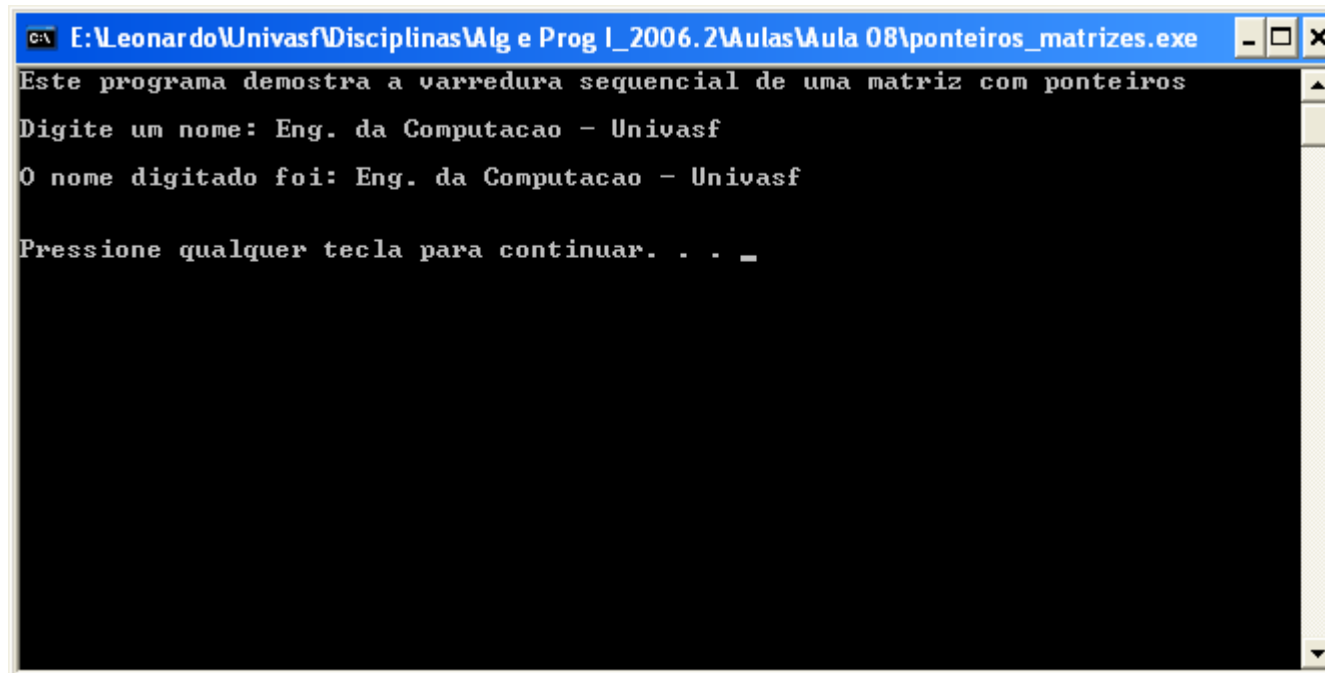
O ponteiro p aponta para a posição 0 (zero) da string str

O ponteiro p é incrementado e passa a apontar para próxima posição da string str

O ponteiro p é testado até atingir o delimitador /0

Ponteiros e Matrizes

- A saída no console para o programa anterior será:



```

E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_matrizes.exe
Este programa demonstra a varredura sequencial de uma matriz com ponteiros
Digite um nome: Eng. da Computacao - Univasf
O nome digitado foi: Eng. da Computacao - Univasf
Pressione qualquer tecla para continuar. . . _

```

Ponteiros e Matrizes

- Vejamos outro exemplo de varredura sequencial de uma matriz, onde se torna ainda mais evidente a eficácia dos ponteiros:

```
#include<stdio.h>
/* Este programa demonstra a varredura sequencial de uma matriz sem ponteiros*/
int main()
{
    float mat [50][50];
    int i, j, cont=0;
    for(i=0; i<50; i++)
        for(j=0; j<50; j++)
            mat[i][j]=cont++;
    return 0;
}
```

Cálculo de 2500 deslocamentos

```
#include<stdio.h>
/* Este programa demonstra a varredura sequencial de uma matriz com ponteiros*/
int main()
{
    float mat [50][50], *f, cont;
    f = mat;
    for(cont=0; cont<2500; cont++)
    {
        *f = cont;
        f++;
    }
    return 0;
}
```

Apenas o incremento de ponteiro

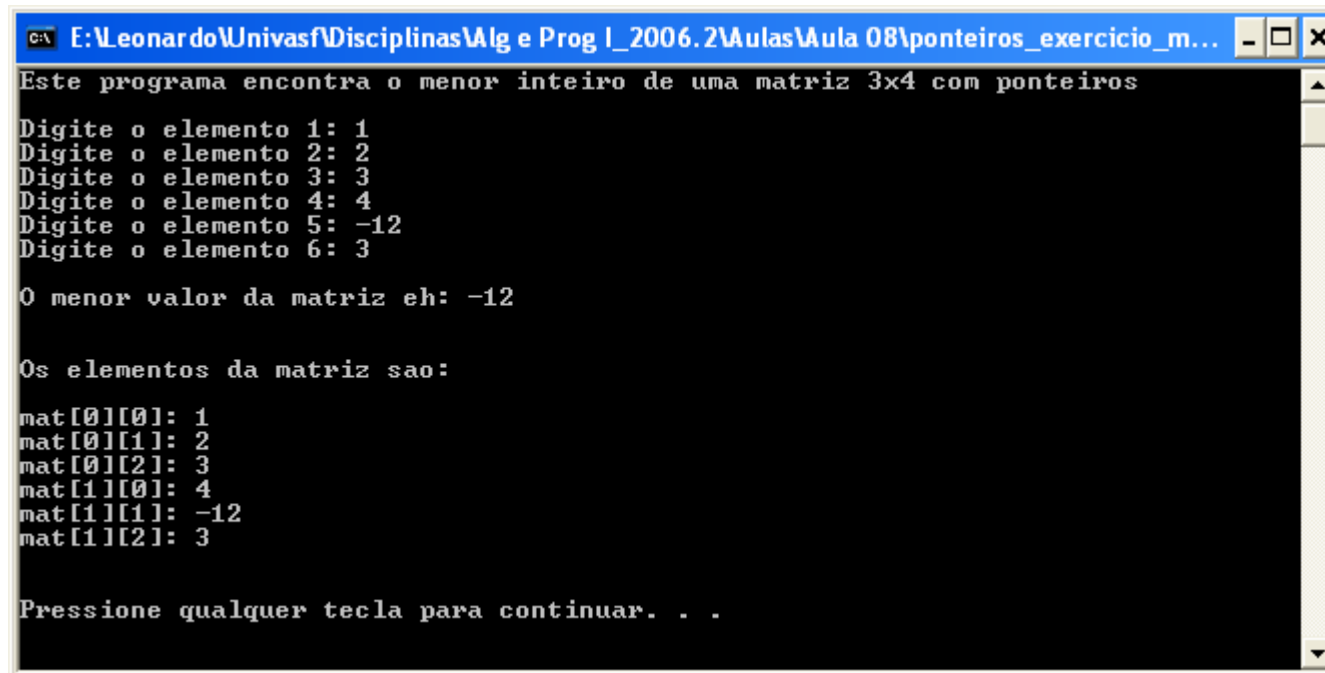
Ponteiros e Matrizes

- Faça um programa em C que leia uma matriz `mat` 2 x 3 de inteiros e encontre o menor elemento da matriz.

```
#include<stdio.h>
/* Este programa encontra o menor inteiro de uma matriz 2x3 com ponteiros */
int main()
{
    int mat [2][3], *p, cont, var, menor=32676, i, j;
    p = mat;
    printf("Este programa encontra o menor inteiro de uma matriz 2x3 com ponteiros\n\n");
    for(cont=0; cont<6; cont++, p++)
    {
        printf("Digite o elemento %d: ", cont+1);
        scanf("%d", &var);
        *p = var;
        if( *p <= menor )
            menor = *p;
    }
    printf("\nO menor valor da matriz eh: %d", menor);
    printf("\n\nOs elementos da matriz sao: \n");
    for(i=0; i<2; i++)
        for(j=0; j<3; j++)
            printf("\nmat[%d][%d]: %d", i, j, mat[i][j]);
    printf("\n\n\n");
    system("pause");
    return 0;
}
```

Ponteiros e Matrizes

- A saída no console para o programa anterior será:



```

E:\Leonardo\Univasf\Disciplinas\Alg e Prog I_2006.2\Aulas\Aula 08\ponteiros_exercicio_m...
Este programa encontra o menor inteiro de uma matriz 3x4 com ponteiros
Digite o elemento 1: 1
Digite o elemento 2: 2
Digite o elemento 3: 3
Digite o elemento 4: 4
Digite o elemento 5: -12
Digite o elemento 6: 3

O menor valor da matriz eh: -12

Os elementos da matriz sao:
mat[0][0]: 1
mat[0][1]: 2
mat[0][2]: 3
mat[1][0]: 4
mat[1][1]: -12
mat[1][2]: 3

Pressione qualquer tecla para continuar. . .

```

Ponteiros e Matrizes

- Há uma diferença entre o nome de uma matriz e um ponteiro que deve ser frisada:
 - Um ponteiro é uma variável, mas o nome de uma matriz não é uma variável;
- Supondo as declarações a seguir, vejamos a corretude das atribuições:

```
int vetor[10];
```

```
int *ponteiro;
```

```
vetor = vetor + 2;
```

```
vetor = ponteiro;
```

```
ponteiro = vetor + 2;
```

```
ponteiro = vetor;
```

Errado!

Certo!

Matrizes de Ponteiros

- Ponteiros podem ser organizados em matrizes como qualquer outro tipo de dado;
 - A declaração de uma matriz de ponteiros int, de tamanho 10, é:

```
int *x[10];
```

- Para atribuir o endereço de uma variável inteira, chamada var, ao terceiro elemento da matriz de ponteiros, deve-se escrever:

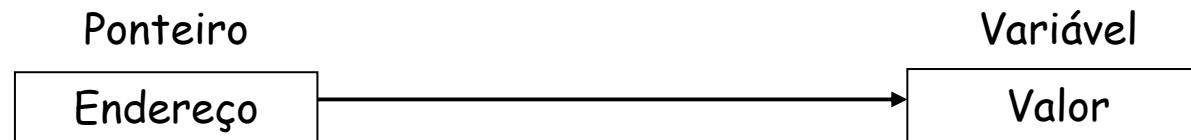
```
x[2] = &var;
```

- Para encontrar o valor de var, escreve-se:

```
*x[2];
```


Indireção Múltipla

- Podemos também, ter um ponteiro apontando para outro ponteiro que aponta para o valor final. Situação chamada indireção múltipla, ou ponteiros para ponteiros;



Bibliografia

- SCHILDT H. "*C Completo e Total*", Makron Books. SP, 1997.
- UFMG "Curso de Linguagem C", Universidade Federal de Minas Gerais.