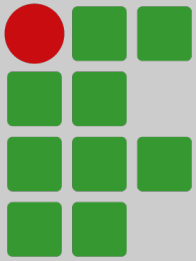


INSTITUTO FEDERAL

Campus Charqueadas

Programação

Ponteiros

Prof. Silvana Teodoro

Ponteiros

- Três propriedades que um programa deve manter quando armazena dados:
  1. onde a informação é armazenada;
  2. que valor é mantido lá;
  3. que tipo de informação é armazenada.

- A definição de uma variável simples obedece a estes três pontos. A declaração provê o tipo e um nome simbólico para o valor. Também faz com que o programa aloque memória para o valor.

Ponteiros

- Um ponteiro é uma variável que contém o endereço de um dado. Ou seja, variáveis que armazenam endereços ao invés dos próprios valores.

- Declaração: * indica que a variável é um ponteiro

  tipo_dado *nome_ponteiro;

- Exemplo:

```
int x;
int *pi;   /* compilador sabe que pi é um ponteiro */
           /* pi é um ponteiro para inteiro */
```

Ponteiros

Vantagens:

- Ponteiros fornecem uma vantagem de desempenho, permitindo que você acesse a memória do computador diretamente.

- Em um programa de computador, a maneira mais rápida de acessar e modificar um objeto é acessar diretamente a memória física onde o objeto está armazenado.

- Esta técnica é comumente usada para otimizar os algoritmos que requerem acesso freqüente ou repetitivo a grandes quantidades de dados.

Ponteiros

Desvantagens:

- Segurança: acesso direto à memória significa que você pode fazer coisas que talvez você não deveria.
- Você pode acidentalmente (ou intencionalmente) de memória de acesso que não é seu para acessar.
- Como resultado, você pode substituir a memória crítica, modificar o código de um aplicativo em execução , ou fazer com que seu aplicativo ou de outro aplicativo se comportar ou sair inesperadamente.

Ponteiros

- O operador **&** quando aplicado sobre uma variável retorna o seu endereço.

- Exemplo:

```
int x = 10, *pi;
pi = &x;
printf("&x: %p pi: %p", &x, pi);

=> &x: 0x03062fd8 pi: 0x03062fd8
```

Ponteiros

- Exemplo:

Aplique o operador de endereço, **&**, a uma variável para pegar sua posição; por exemplo, se **xicara** é uma variável, **&xicara** é seu endereço.

```
#include <stdio.h>
void main()
{
        int rosquinhas = 6;
        double xicaras = 4.5;
        printf("valor das rosquinhas = %d", rosquinhas);
        printf("e endereco das rosquinhas = %d\n", &rosquinhas);
        printf("valor das xicaras = %g", xicaras);
        printf("e endereco das xicaras = %d\n", &xicaras);
}
```

A saída do programa é:

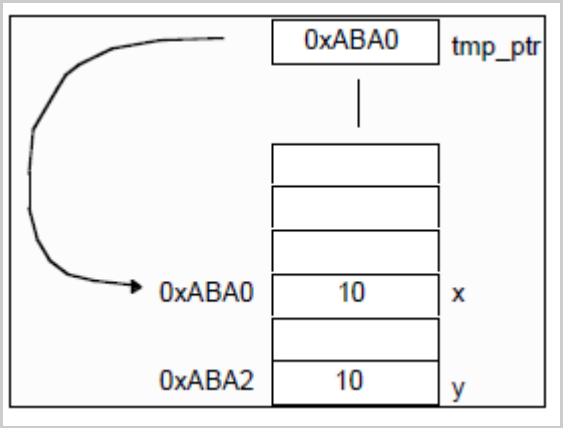
valor das rosquinhas = 6 e endereco das rosquinhas = 1245052
valor das xicaras = 4.5 e endereco das xicaras = 1245044

Ponteiros

- O operador * quando aplicado sobre um ponteiro retorna o dado apontado.

- Exemplo:

```
void main () {
        int *tmp_ptr;
        int x, y;
        x = 10;
        tmp_ptr = &x;
        y = *tmp_ptr;        /* (*tmp_ptr) = 10 */
}
```



Ponteiros

- O nome do ponteiro representa a posição.
- O operador * fornece o valor da posição.
- Suponha por exemplo, que **ordem** é um ponteiro. Então, **ordem** representa um endereço, e ***ordem** representa o valor naquele endereço.

Ponteiros

```
#include <stdio.h>
void main()
{
        int atualiza = 6;          // declara uma variável
        int * p_atualiza;         // declara ponteiro para um int
        p_atualiza = &atualiza; // atribui endereço do int para o ponteiro
        // expressa o valor de duas formas
        printf("Valores: atualiza = %d", atualiza);
        printf(", *p_atualiza = %d\n", *p_atualiza);
        // expressa endereço de duas formas
        printf("Enderecos: &atualiza = %d", &atualiza);
        printf(", p_atualiza = %d\n", p_atualiza);
        // usa ponteiro para mudar valor
       *p_atualiza = *p_atualiza + 1;
        printf("Agora atualiza = %d\n", atualiza);
}
```

A saída do programa é:

Valores: atualiza = 6, *p_atualiza = 6

Enderecos: &atualiza = 1245052, p_atualiza = 1245052

Agora atualiza = 7

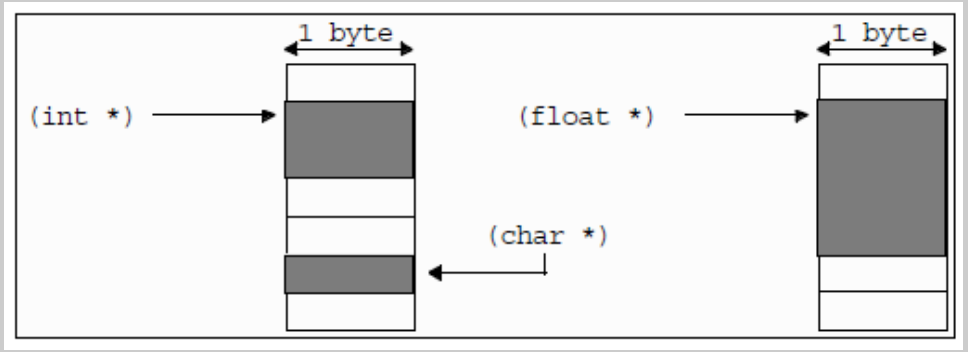
Ponteiros

- Ponteiros são variáveis tipadas:

  (int *) =! (float *) =! (char *)

- Exemplo:

```
main() {
int *ip, x;
float *fp, z;
   ip = &x; /* OK */
   fp = &z; /* OK */
   ip = &z; /* erro */
   fp = &x; /* erro */
}
```



Ponteiros

- Alterando valores com ponteiros:

```
void main()
{
    int x = 10;
    int *pi;
    pi = &x;    /* *pi == 10 */
   (*pi)++;    /* *pi == 11 */
   printf("%d", x);
}
```

imprime o valor 11

```
int main ( void )
{
    int a;
    int *p=&a;
    *p = 2;
    printf(" %d ", a);
    return 0;
}
```

imprime o valor 2

ao alterar *pi estamos alterando o conteúdo de x

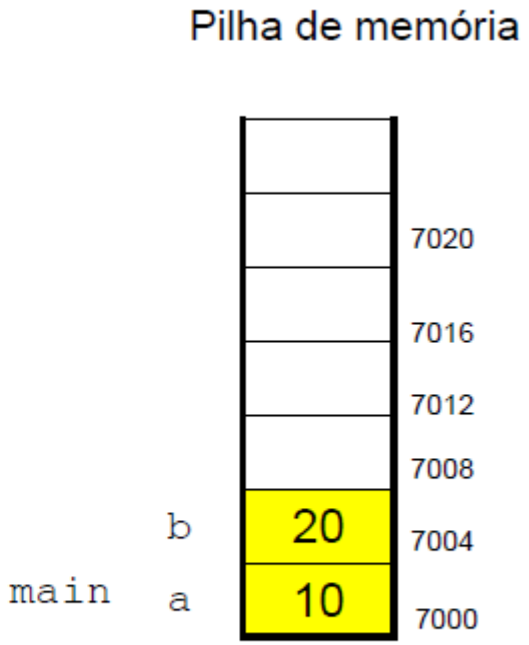
Ponteiros

```c
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
        b=a;
        a=tmp;
}
```

```
 a=10  b=20

Press any key to continue
```

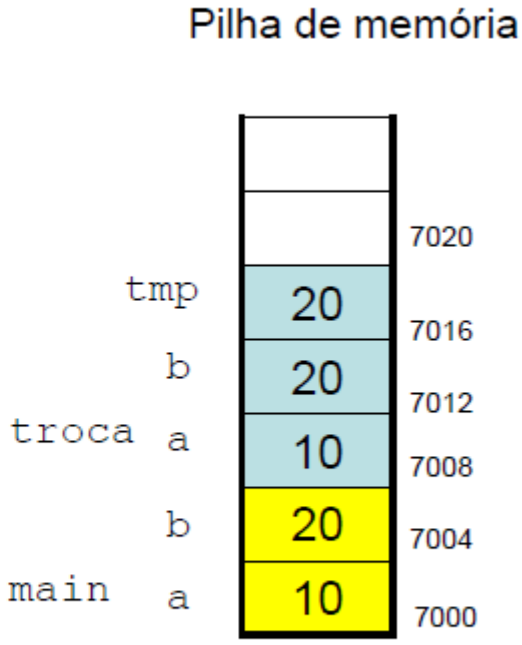
Ponteiros

```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
        b=a;
        a=tmp;
}
```



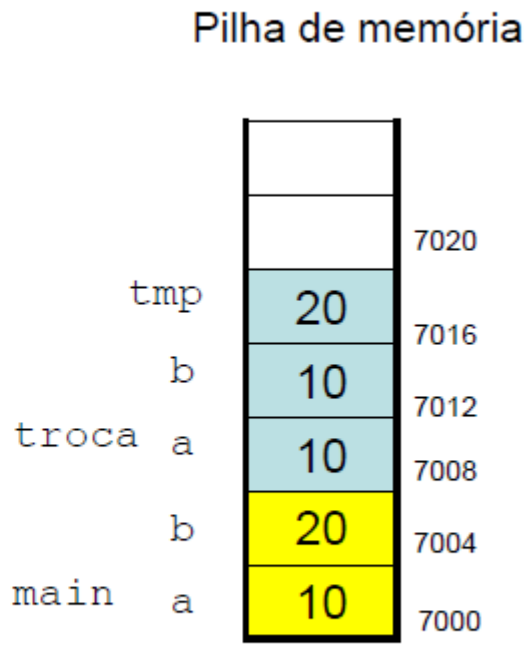
Ponteiros

```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
    ⇨   b=a;
        a=tmp;
}
```



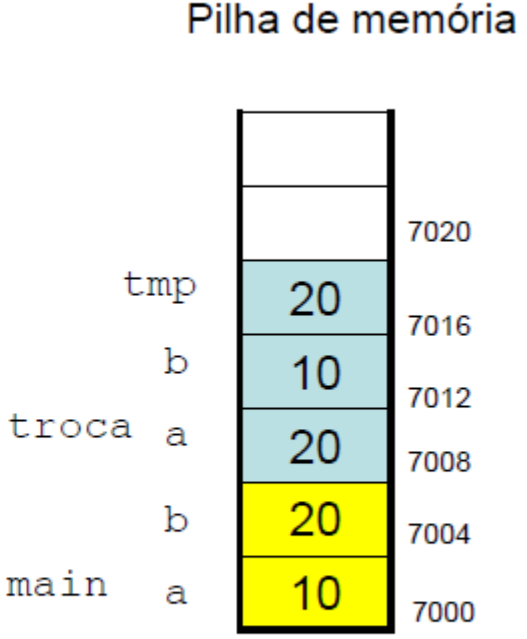
Ponteiros

```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
        b=a;
        a=tmp;
}
```



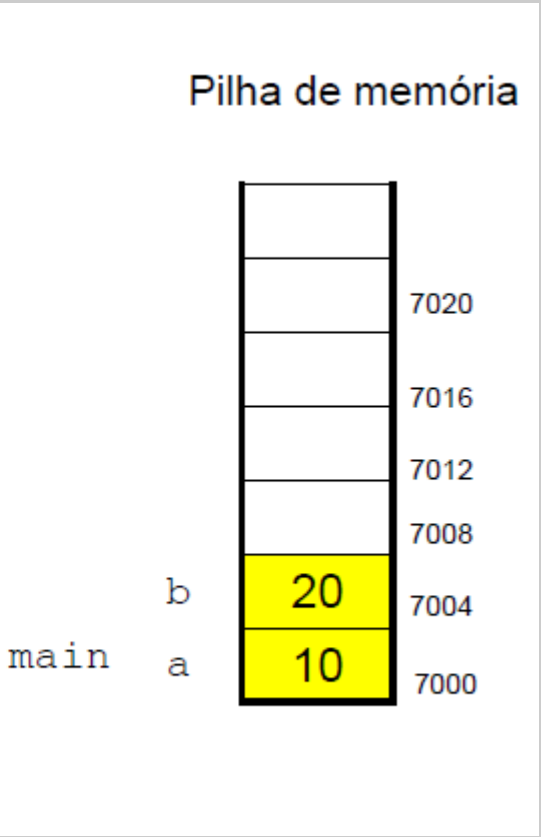
Ponteiros

```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
        b=a;
        a=tmp;
}
```



Ponteiros

```
#include <stdio.h>

void troca(int a, int b);

int main (void)
{
        int a=10, b=20;
        troca(a,b);
    ⇨   printf(" a=%d  b=%d\n",a,b);
}

void troca(int a, int b) {
        int tmp=b;
        b=a;
        a=tmp;
}
```



Ponteiros

- Passagem de ponteiros para funções:
  - função **g** chama função **f**
    - **f** não pode alterar diretamente valores de variáveis de **g**, porém
      - se **g** passar para **f** os valores dos endereços de memória onde as variáveis de **g** estão armazenadas, **f** pode alterar, indiretamente, os valores das variáveis de **g**.

Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

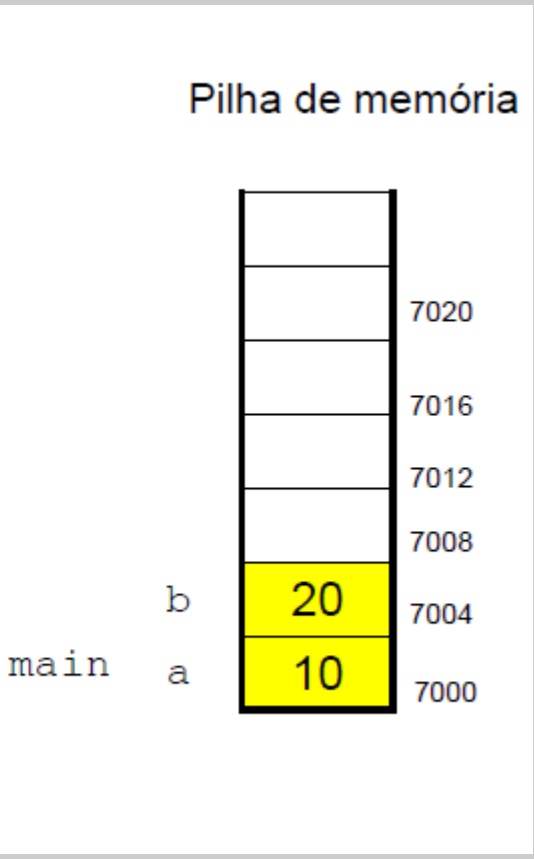
```
 a=20  b=10

Press any key to continue
```

Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

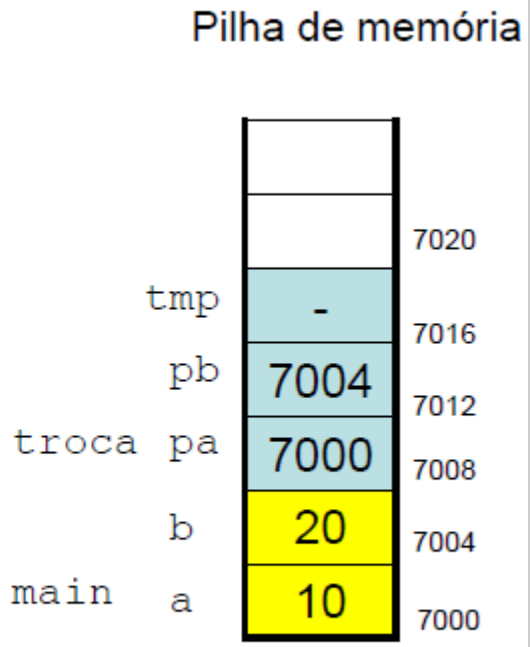


Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

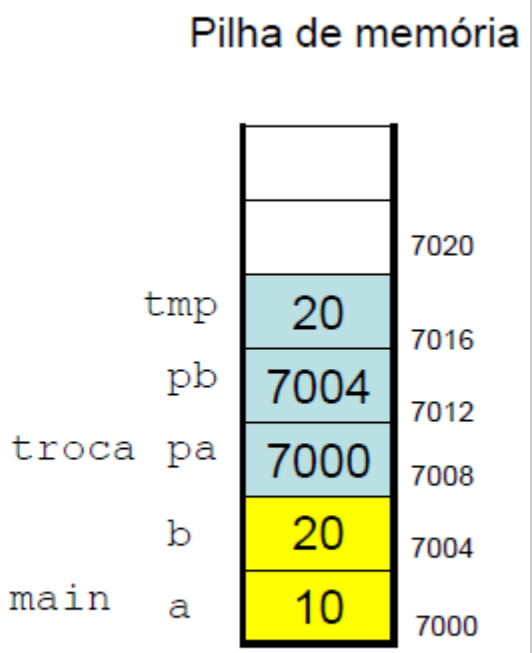


Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

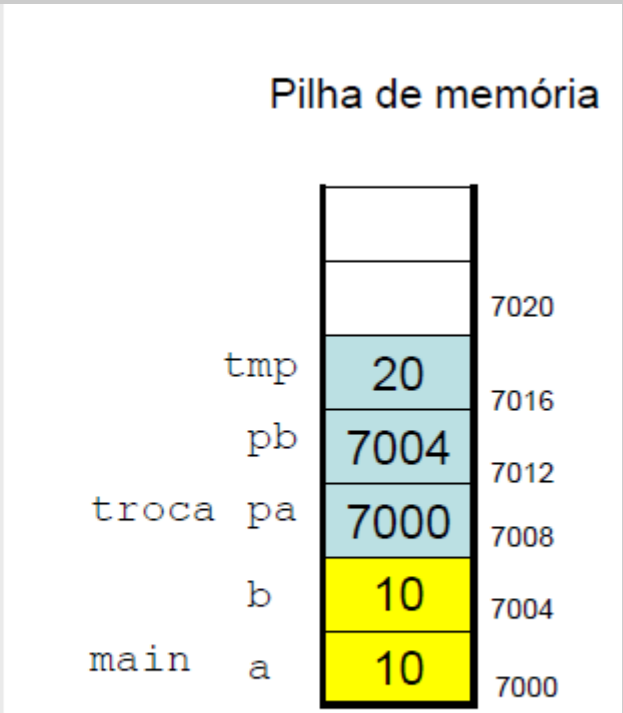


Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

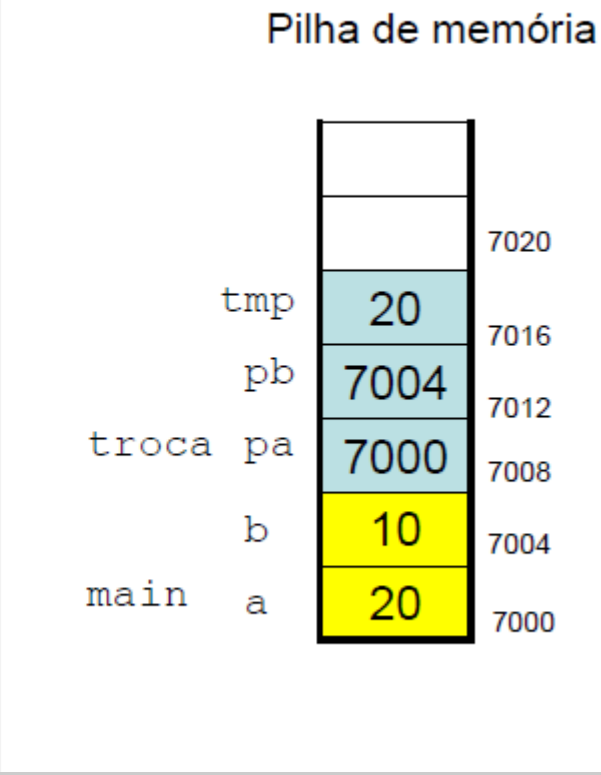


Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```

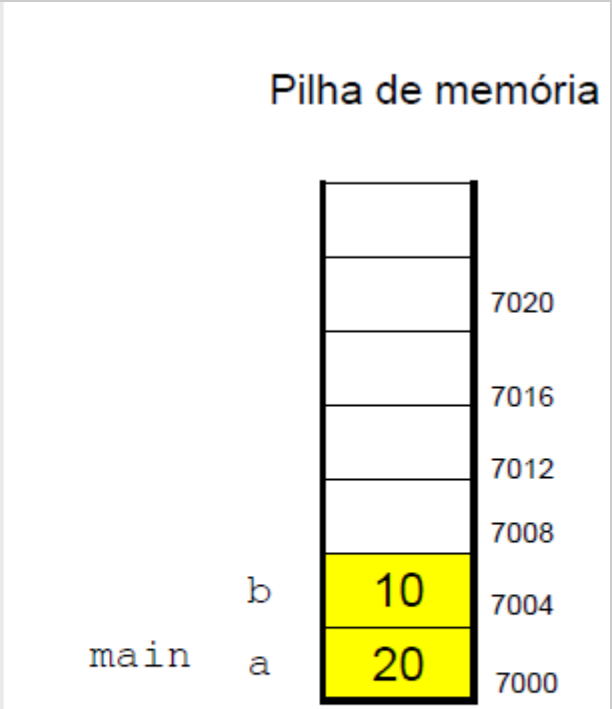


Ponteiros

Funções que mudam valores de variáveis de outras

- Passagem de ponteiros para funções:

```
#include <stdio.h>

void troca(int *pa, int *pb);

int main (void)
{
        int a=10, b=20;
        troca(&a,&b);
        printf(" a=%d  b=%d\n",a,b);
}

void troca(int *pa, int *pb) {
        int tmp=*pb;
        *pb=*pa;
        *pa=tmp;
}
```




```
 a=20  b=10

Press any key to continue
```

Exercício

1) Pratique a declaração e utilização de ponteiros.

   – defina e inicialize uma variável inteira
   – defina um ponteiro para inteiro
   – modifique o valor da variável através do ponteiro
   – verifique os novos valores da variável usando printf()

Ponteiros
Vetores

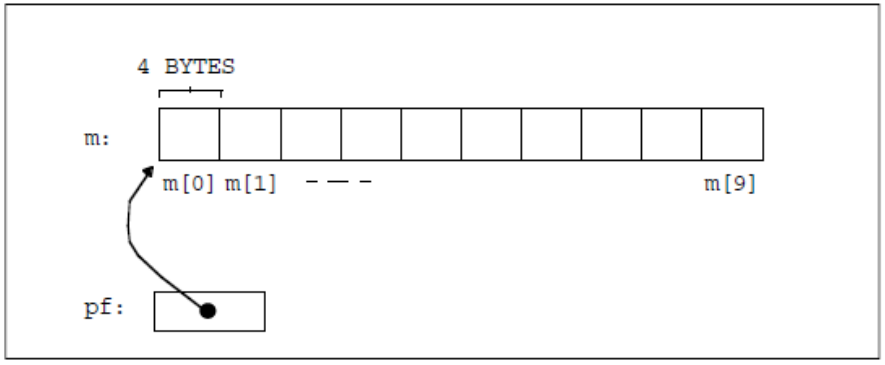
- Vetores são agrupamentos de dados adjacentes na memória.

  tipo_dado nome_vetor[<*tamanho*>];

- Define um arranjo de <*tamanho*> elementos adjacentes na memória do tipo *tipo_dado.*

- Ex.:

```
float m[10], *pf;
pf = m;
```



em **float** m[10] *m* é uma constante que endereça o primeiro elemento do vetor

Ponteiros
Vetores

- Referenciando Elementos

  – Pode-se referenciar os elementos do vetor através do seu nome e colchetes:

```
m[5] = 5.5;
if (m[5] == 5.5)
    printf(“Exito”);
else
    printf(“Falha”);
```

Ponteiros
Vetores

- Referenciando Elementos

  – Pode-se referenciar os elementos de um vetor através de ponteiros:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
float *pf;
pf = &m[2];
printf("%f", *pf);                    /* Saída ==> 5.75 */
```

Ponteiros
Vetores

- Referenciando Elementos
  - Pode-se utilizar ponteiros e colchetes:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
float *pf;
pf = &m[2];
printf("%f", pf[0]);            /* Saída ==> 5.75 */
```

  - Note que o valor entre colchetes é o deslocamento a ser considerado a partir do endereço de referência

pf[n] => indica enésimo elemento a partir de pf

Ponteiros
Aritmética de Ponteiros

É possível fazer operações aritméticas e relacionais entre ponteiros e inteiros:

– Soma: ao somar-se um inteiro *n* a um ponteiro, endereçamos *n* elementos a mais (*n* positivo) ou a menos (*n* negativo):

- pf[2] equivale a *(pf+2)
- *(pf + n) endereça n elementos a frente
- *(pf - n) endereça n elementos atrás
- pf++ endereça próximo elemento vetor
- pf-- endereça elemento anterior vetor

Ponteiros
Aritmética de Ponteiros

Exemplo 1

```
void main ()
{
        int vetor[] = {1,2,3,4,5,6,7};
        int size = 7;          /* tamanho do vetor */
        int i, *pi;
        pi = vetor;
        for ( i = 0; i < size; i++){
            printf("%d ",  *pi++);
        }
}
```

Saída ==> 1 2 3 4 5 6 7

Ponteiros
Aritmética de Ponteiros

Exemplo 2

```
void main () {
        int vetor[] = { 1,2,3,4,5,6,7 };
        int size = 7;       /* tamanho do vetor */
        int i, *pi;
        pi = vetor;
        printf(" %d ", *pi);
        pi += 2;
        printf(" %d ", *pi);
        pi += 2;
        printf(" %d ", *pi);
        pi += 2;
        printf(" %d ", *pi);
}
```

Saída ==> 1 3 5 7

Ponteiros
Operações Válidas Sobre Ponteiros

- É válido:
  - *somar* ou *subtrair* um inteiro a um ponteiro: (pi ± int)
  - *incrementar* ou *decrementar* ponteiros: (pi++, pi--)
  - *subtrair* ponteiros (produz um inteiro): (pf - pi)
  - *comparar* ponteiros: (>, >=, <, <=, ==)

- Não é válido:
  - somar ponteiros: (pi + pf)
  - multiplicar ou dividir ponteiros: (pi*pf, pi/pf)
  - operar ponteiros com *double* ou *float:* (pi ± 2.0)

Ponteiros Genéricos

- Um ponteiro genérico é um ponteiro que pode apontar para qualquer tipo de dado.
- Define-se um ponteiro genérico utilizando-se o tipo *void*:

```
void *pv;
int x=10;
float f=3.5;
pv = &x;      /* aqui pv aponta para um inteiro */
pv = &f;      /* aqui, para um float */
```

Ponteiros e Strings

- *strings* são vetores de caracteres e podem ser acessados através de *char **

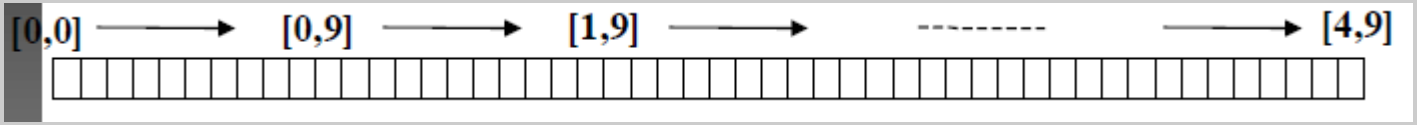
```
#include<stdio.h>

  void main ()
  {
              char str[]= "abcdef",  *pc;
              int i;
               pc = str;
              for (i =0; i<6; i++){
                   putchar(*pc++);
              }
   }
  Saída ==> abcdef
```

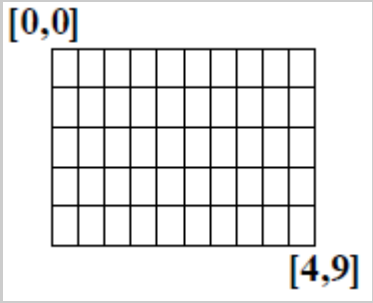
- o incremento de **pc** o posiciona sobre o próximo caracter.

Ponteiros
Matrizes

char matriz[5][10];

- declara uma matriz de 5 linhas e 10 colunas:
- na memória, entretanto, os caracteres são armazenados linearmente:





Ponteiros
Matriz de Caracteres

- Percorrendo *matriz* com ponteiro:

```
void main () {
        char matriz[5][10];
        char *pc;
        int i;
         pc = matriz;
         for (i=0; i < 50; i++, pc++){
              *pc = 'a';
               printf("%d - %c\n", i, *pc);
         }
}
```

Qual a saída?

Ponteiros
Matriz de Caracteres

- Percorrendo *matriz* com índices:

```
void main () {
        char matriz[5][10];
        int i, j;
        for (i=0; i<5; i++){
                for (j=0; j<10; j++){
                    matriz[i][j] = ‘a’;
                    printf("[%d] [%d] - %c\n", i, j, matriz[i][j]);
                   }
        }
 }
```

- as colunas mudam mais rápido

Ponteiros
Matriz de Strings

- Neste caso, cada elemento da matriz é um ponteiro para um caracter:

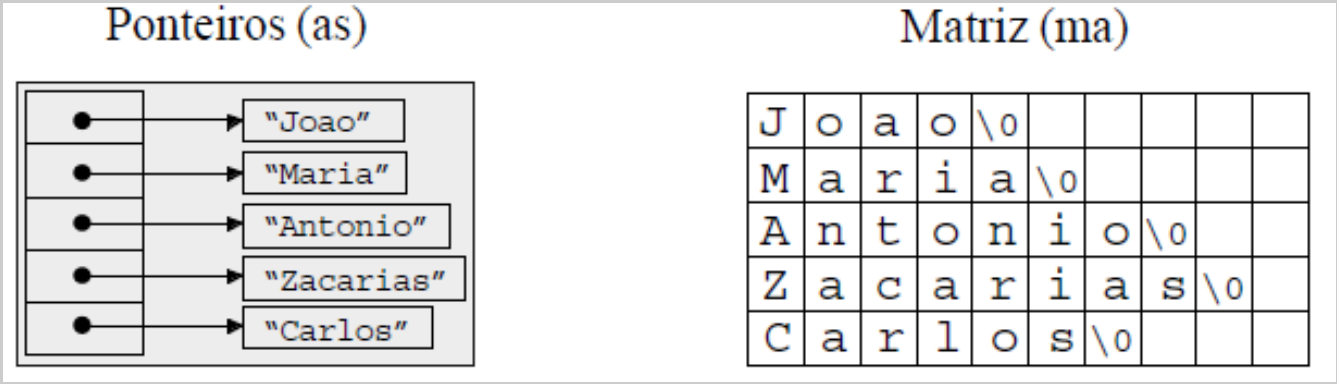
  char *exemplo[] = {“Joao”, “Maria”, “Antonio”, “Zacarias”, “Carlos”};

- *exemplo é* uma *matriz* de ponteiros para *char*, iniciado com as *strings* indicadas.

- Comparando *matriz* de *string* com matriz de *char :*

  char *as[] = {“Joao”, “Maria”, “Antonio”, “Zacarias”, “Carlos”};

  char ma[5][10] = {“Joao”, “Maria”, “Antonio”, “Zacarias”, “Carlos”};



Ponteiros para Ponteiros

- É possÍvel definir ponteiros para ponteiros até um nível arbitrário de indireção

- **Exemplo:**

```
char *pc;          /* ponteiro para char */
char **ppc;        /* ponteiro para ponteiro para char */
pc = “teste”;
ppc = &pc;
printf("%c", **ppc);      /* Saída ==> t */
```

Exercício

1) O que será impresso no programa abaixo?

```c
#include <stdio.h>

void main(){
    int a = 3, b = 2, *p = NULL, *q = NULL;
    p = &a;
    q = p;
    *q = *q +1;
    q = &b;
    b = b + 1;
    printf("%d\n", *q);
    printf("%d\n", *p);
}
```