COMPUTAÇÃO
UFBA

Universidade Federal
da Bahia

# Laboratório de Programação I (MATA57)

Prof.: Claudio Junior N. da Silva (claudiojns@ufba.br)

Ponteiros

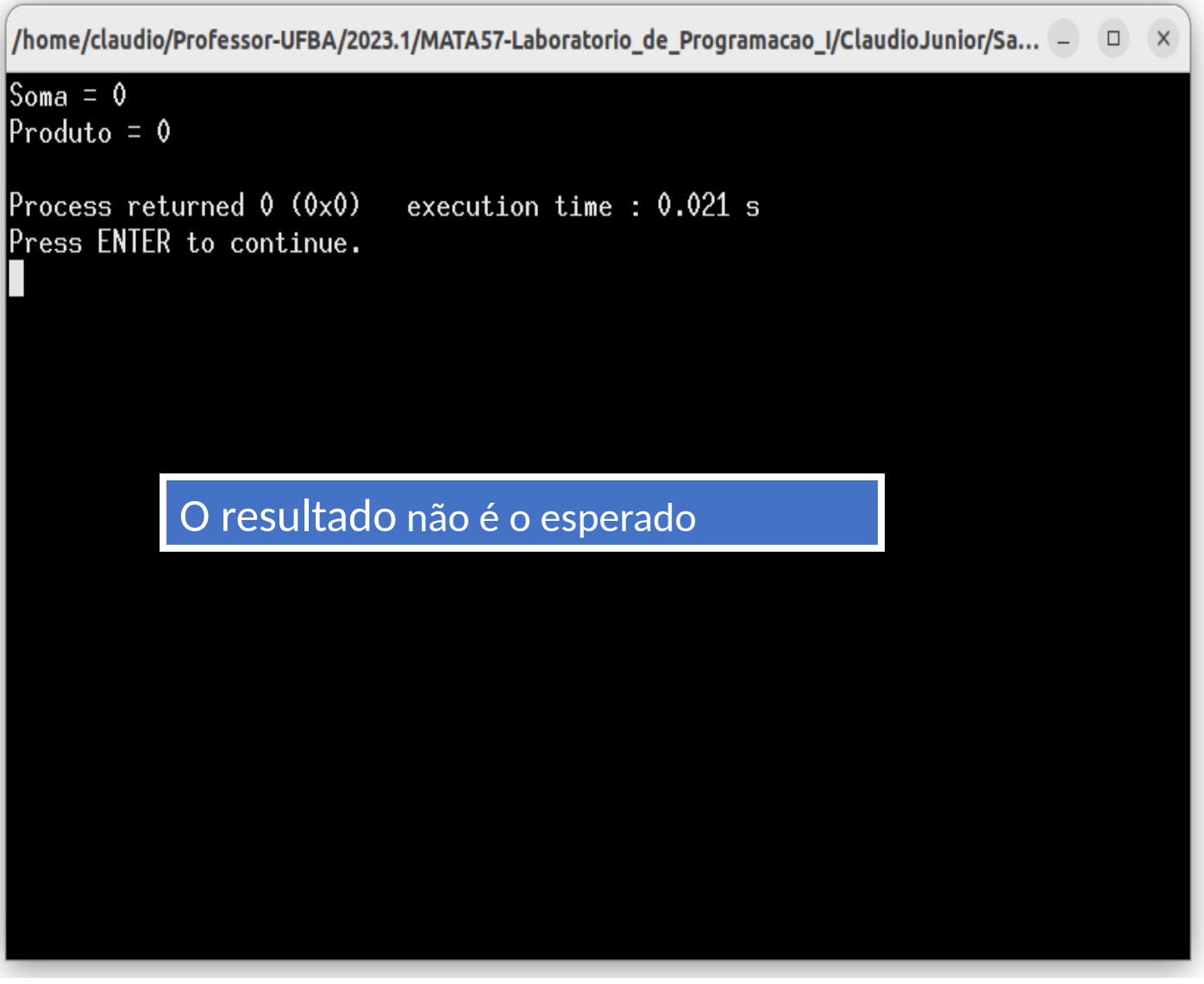
2023.1

# Caso somaprod:

- Com base no que estudamos até agora, escreva um programa em C++ que execute uma função chamada somaprod para calcular a soma e o produto de dois número inteiros (var1 e var2) e armazenar o resultado respectivamente em var3 e var4:
  - A função somaprod é executada com:
    - somaprod(var1, var2, var3, var4);
  - onde:
    - Var1 e var2 são dois números inteiros conhecidos quaisquer;
    - Var3 será o resultado de var1 + var2;
    - Var4 será o resultado de var1 * var2.
  - somaprod é uma função do tipo void;

# Código C++

```
#include <iostream>

using namespace std;

void somaprod(int a, int b, int c, int d){
    c = a + b;
    d = a * b;
}

int main(){
  int var1=5, var2=3, var3, var4;

  somaprod(var1, var2, var3, var4);

  cout << "Soma = " << var3 << endl;
  cout << "Produto = " << var4 << endl;

  return 0;
}
```



# Qual é o problema?

# Variáveis e Endereços

- Memória abstrata

| x | y | z | w | k |
|---|---|---|---|---|
| 5 | 7 | 9 | 1 | 3 |

Variáveis

Valor

- Memória concreta (associações):

| x | y | z | w | k |
|---|---|---|---|---|
| 13 | 72 | 91 | 100 | 15 |

Variáveis

Endereço

- Memória de fato:

| 13 | 72 | 91 | 100 | 15 |
|---|---|---|---|---|
| 5 | 7 | 9 | 1 | 3 |
| x | y | z | w | k |

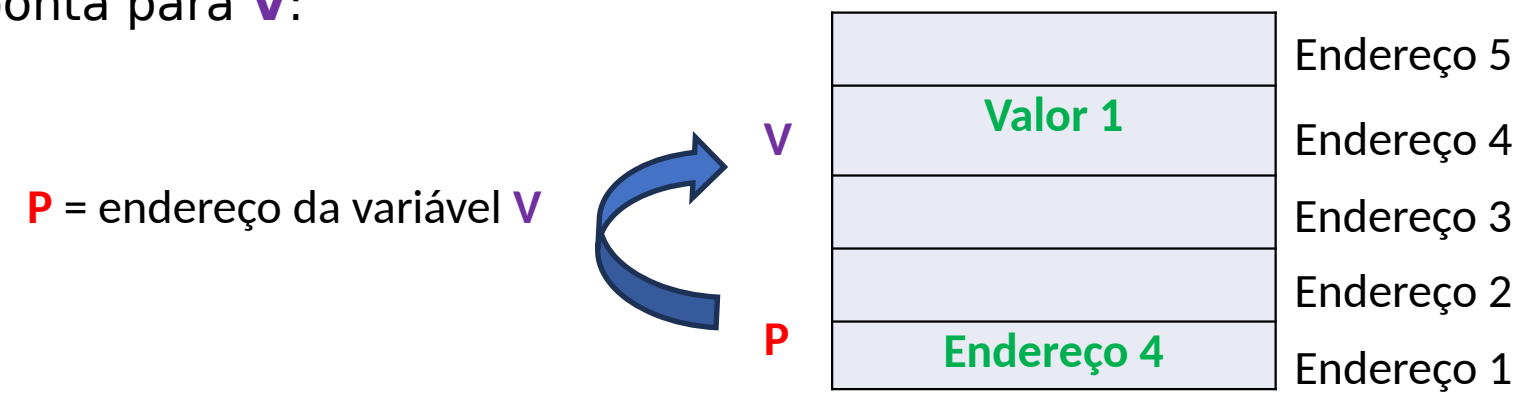
Endereço

Valor

Variáveis

# Ponteiros

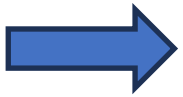
- Toda variável tem um endereço ou uma posição associados na memória;
- O endereço é visto como um ponteiro (ou apontador). É uma referência para a posição de memória de uma variável;
- Ponteiros fornecem um modo de acesso à variável sem referenciá-la diretamente;
- Um endereço pode ser armazenado em uma variável do tipo ponteiro (ponteiro variável):
  - Ponteiro variável é uma variável que contém o endereço de outra variável:
    - **P** aponta para **V**:



# Ponteiros - Declaração

- Declarando uma variável de nome var1 que ponde armazenar valores inteiros:
  - **int var1;**
- Para declarar uma variável do tipo ponteiro:
  - Forma Geral:
    - tipo* variável
    - tipo  *variavel
    - tipo * variavel

int* p;
int *p;
int * p;



Declara uma variável de nome **p** que pode armazenar um **endereço** de memória para um **inteiro**.

E se a declaração for:

int* p, x, y;

Qual o resultado?

# Ponteiros – Operador &

- & é um operador unário que fornece o endereço de uma variável:
  - Forma Geral:
    - &variavel

```
int *p;          // declara p tipo ponteiro para inteiro
int v;           // declara v tipo inteiro
p = &v;          //  atribui o endereço de v ao ponteiro p
                 // ou seja, aponta-se p para v
```

A variável p de tipo ponteiro para inteiro recebe o endereço da variável v de tipo inteiro

- O operador & não pode ser aplicado à expressões ou constantes:
  - Exemplo: x = &178

# Ponteiros – Operador &

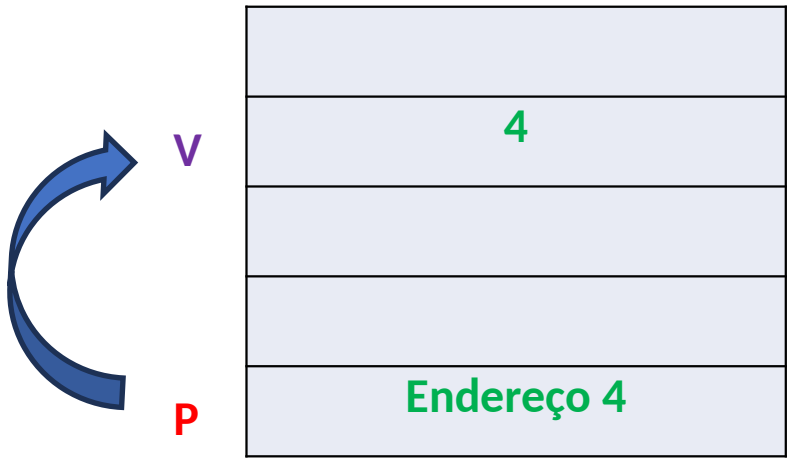
- A carga inicial de ponteiros se faz com o operador &;

```
int a = 15;
float fab = 8.2;

int *p = &a;          // declara p tipo ponteiro para inteiro iniciado com o endereço de a

float  *p_ab = &fab; // declara e incializa p_ab para float com o endereço de fab
```

# Ponteiros – Operador de Indireção *

- * é chamado de Operador de Indireção;
- Quando aplicado a uma variável do tipo ponteiro, acessa o conteúdo apontado por ela
  - Forma Geral:
    - *variavel

```
int *p;        // declara p tipo ponteiro para inteiro
int v = 3;     // declara v tipo inteiro com valor igual a 3
p = &v;        // atribui o endereço de v à variável p
*p = 4         // armazena no endereço v o valor 4
```



# Operações com Ponteiros

- Uma vez declarado um ponteiro, podem ser realizados sobre ele praticamente todos os tipos de operações que podem ser realizadas sobre um inteiro
- Ex:
  - p++; p--;  /* próximo endereço / endereço anterior */
  - p1 = p2 /* p1 aponta para o mesmo endereço de p2*/
  - *p1 = *p2; /* p1 terá o mesmo conteúdo que p2*/
- No entanto, reiterando, um ponteiro serve, sobretudo, para acessar outros objetos a partir de seus endereços

# Usando Ponteiros

```
int i, j;

int *ip;

i = 12;

ip = &i;

j = *ip;

*ip = 21;
```

A variável ip armazena um ponteiro para um inteiro

O endereço de i é armazenado em ip

O conteúdo da posição apontada por ip é armazenado em j

O conteúdo da posição apontada por ip passa a ser 21

# Cuidados

- Talvez o principal cuidado ao usar ponteiros seja saber para onde o ponteiro está apontando:
  - Nunca use um ponteiro que não foi inicializado

- Ex: Como **não** usar ponteiros

```
int main () /* Errado - Nao Execute */
{
        int x,*p;
        x=13;
        *p=x;
        return(0);
}
```

# Cuidados

```
#include <iostream>
int main    () {
    int    a , *p ;
     p = &a ;
    *p = 2 ;
    printf (“%d”,a);
    return    0 ;
}
```

```
#include <iostream>
int main () {
  int a,b,*p ;
  a = 2 ;
  *p = 3 ;
  b = a +( *p );
  printf("%d",b);
return 0;
}
```

Qual o resultado da execução destes códigos?

# Vamos ao Compilador C++

```
#include <iostream>
using namespace std;
int main(){
    int a, *c;
    cin >> a;
    c = &a;
    cout << "o endereco de a: " << &a << endl;
    cout << "o endereco de c: " << &c << endl;
    cout << "o valor de a: " << a << endl;
    cout << "o valor de c: " << c << endl;
    cout << "o valor de onde c aponta: " << *c << endl;
    return 0;
}
```

```
11
o endereco de a: 0x7ffce50f010c
o endereco de c: 0x7ffce50f0110

o valor de a: 11
o valor de c: 0x7ffce50f010c

o valor de onde c aponta: 11
```

# Caso SWAP

```
#include <iostream>
using namespace std;

void swap(int a, int b);            // protótipo da função

int main(){
  int a=8, b=12;                    // inicializa variáveis a e b
  swap(a,b);                        // esta função troca os valores entre a e b
  cout << endl;
  cout << "No programa principal: " << endl; // print dos valores de a e b
  cout << "O valor de a: " << a <<endl;      // após o retorno da função
  cout << "O valor de b: " << b <<endl;      // swap
  return 0;
}
void swap(int a, int b){
  int temp = a;                     // variável temporária iniciada com o valor de a
  a = b;                            // troca de valores a = b
  b = temp;                         // b recebe o valor de temp que foi inicilizada com a
  cout << "Na funcao swap o valor de a: " << a <<endl;
  cout << "Na funcao swap o valor de b: " << b <<endl;
}
```

```
Na funcao swap o valor de a: 12
Na funcao swap o valor de b: 8

No programa principal:
O valor de a: 8
O valor de b: 12

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

# Caso SWAP

```
#include <iostream>
using namespace std;
void swap(int* pa, int* pb);  // protótipo da função
int main(){
  int a=8, b=12;  // inicializa variáveis a e b
  int *pta, *ptb;
  cout << "================" << endl;
  cout << "Valores inciais: " << endl;
  cout << "Valor de a     : " << a << "  Endereco de a: " << &a << endl;
  cout << "Valor de b     : " << b << " Endereco de b: " << &b << endl << endl;
  swap(&a,&b);      // esta função troca os valores a por b / b por a
  cout << "------------------ " << endl;
  cout << "Valores apos SWAP: " << endl;
  cout << "Valor de a      : " << a << " Endereco de a: " << &a << endl;
  cout << "Valor de b      : " << b << "  Endereco de b: " << &b << endl << endl;
  return 0;
}
void swap(int* pa, int* pb){
  int temp = *pa; // variável temporária iniciada com o valor de a
  *pa = *pb;       // troca de valores a = b
  *pb = temp;     // b recebe o valor de temp que foi inicilizada com a
}
```

```
================
Valores inciais:
Valor de a     : 8  Endereco de a: 0x7ffcf1693f10
Valor de b     : 12 Endereco de b: 0x7ffcf1693f14


------------------
Valores apos SWAP:
Valor de a      : 12 Endereco de a: 0x7ffcf1693f10
Valor de b      : 8  Endereco de b: 0x7ffcf1693f14
```

/home/claudio/Professor-UFBA/2023.1/MATA57-Laboratorio_de_Programacao_I/Clau...

```
================
Valores inciais:
Valor de a      : 8  Endereco de a: 0x7fff79ef68d0
Valor de b      : 12 Endereco de b: 0x7fff79ef68d4

-----------------
Na funcao SWAP:
O endereco de pa  : 0x7fff79ef68a8
O endereco de pb  : 0x7fff79ef68a0
O endereco de temp: 0x7fff79ef68b4

O conteudo de pa: 0x7fff79ef68d0
O conteudo de pb: 0x7fff79ef68d4

O valor de pa: 12
O valor de pb: 8
O valor de temp: 8

-----------------
Valores apos SWAP:
Valor de a        : 12 Endereco de a: 0x7fff79ef68d0
Valor de b        : 8  Endereco de b: 0x7fff79ef68d4


Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

# Caso somaprod:

- Com base no que estudamos até agora, escreva um programa em C++ que execute uma função chamada somaprod para calcular a soma e o produto de dois número inteiros (var1 e var2) e armazenar o resultado respectivamente em var3 e var4:
  - A função somaprod é executada com:
    - somaprod(var1, var2, var3, var4);
  - onde:
    - Var1 e var2 são dois números inteiros conhecidos quaisquer;
    - Var3 será o resultado de var1 + var2;
    - Var4 será o resultado de var1 * var2.
  - somaprod é uma função do tipo void;

# Código C++

```
#include <iostream>

using namespace std;

void somaprod(int a, int b, int c, int d){
    c = a + b;
    d = a * b;
}

int main(){
  int var1=5, var2=3, var3, var4;

  somaprod(var1, var2, var3, var4);

  cout << "Soma = " << var3 << endl;
  cout << "Produto = " << var4 << endl;

  return 0;
}
```

Qual correção deve ser feita?

# Ponteiros e Funções

- O retorno explícito de valores não permite transferir mais de um valor para a função que a chama;
- Como uma função pode alterar variáveis de quem a chamou?
  - função chamadora passa os endereços dos valores que devem ser modificados;
  - função chamada deve declarar os endereços recebidos como ponteiros;
  - Passagem por referência.

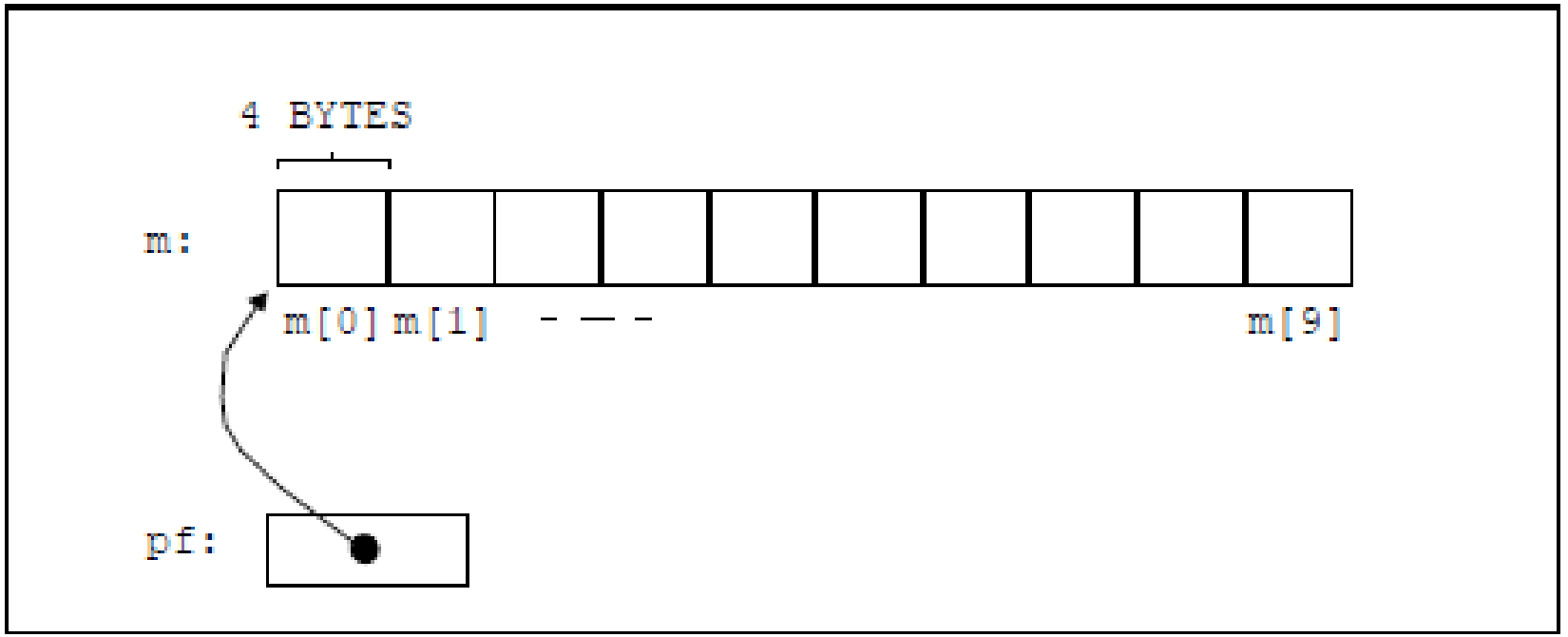
# Arrays

- Arrays são agrupamentos de dados adjacentes na memória
- Declaração:
  - **`tipo_dado nome_array[<tamanho>];`**
    - define um arranjo de <*tamanho*> *elementos* adjacentes na memória do tipo *tipo_dado*

# Arrays

- Ex:

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



# Referenciando Arrays

- Em **float m[10]**, *m* é uma constante que endereça o primeiro elemento do array
- Portanto, não é possível mudar o valor de *m*
- Ex:

```
float m[10], n[10];  float *pf;
m = n;  /* erro: m é constante ! */
pf = m;  /* ok */
```

# Referenciando Elementos de Arrays

- Pode-se referenciar os elementos de um array através de ponteiros:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
float *pf;  pf = &m[2];

cout << *pf; /* ==> 5.75 */
```

# Referenciando Elementos de Arrays

- Pode-se utilizar ponteiros e colchetes:

```
float m[] = { 1.0, 3.0, 5.75, 2.345 };
float *pf;  pf = &m[2];

cout << pf[1]; /* ==> 2.345 */
```

- Note que o valor entre colchetes é o deslocamento a ser considerado a partir do endereço de referencia
  - pf[n] => indica enésimo elemento a partir de pf

# Vamos ao código C++

- Crie um programa C++;
- Parte I:
  - No programa principal defina um array (vetor) de 5 posições;
  - Inicialize os elementos desse vetor com a seguinte regra:
    - vetor[i] = i + 2;
    - Utilize a estrutura *for*;
  - Crie uma estrutura de repetição (for) para imprimir esses 5 elemento do vetor;
  - Teste o seu programa.
- Parte II:
  - Crie uma função chamada init_vetor para iniciar esse vetor:
    - O código criado para inicializar o vetor deve ser movido para essa função;
  - Crie uma função chamada list_vetor para listar os elementos desse vetor:
    - O código criado para imprimir os elementos desse vetor deve ser movido para essa função;
  - Teste o seu programa.
- Analise e reflita sobre as alterações.

# 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);

| Operação | Resultado |
|---|---|
| pf[2] | Equivale a *(pf+2) |
| *(pf + n) | Acessa o conteúdo de n posições à frente |
| *(pf – n) | Acessa o conteúdo de n posições anteriores |
| pf++ | Endereça o ponteiro uma posição à frente |
| pf-- | Endereça o ponteiro uma posição anterior |

# Operações válidas

| Válido | Não é válido |
|---|---|
| Subtrair ponteiros:<br>(pf – pi: produz inteiro) | Somar ponteiros:<br>(pi + pf) |
| Incrementar ou decrementar ponteiros:<br>(pi++, pi--) | Multiplicar ou dividir ponteiros:<br>(pi * pf) |
| Somar ou subtrair um inteiro a um ponteiro :<br>(pi +/- int) | Operar ponteiros com *double* ou *float*:<br>(pi + 2.0) |
| Comparar ponteiros | |

# Cuidados

- C/C++ não controla os limites dos arrays, o programador deve fazê-lo;
- Um ponteiro deve sempre apontar para um local válido antes de ser utilizado;

# Ponteiros genéricos

- Ponteiro que pode apontar para qualquer tipo de dado, sendo definido como 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 */
```

- Deve ser controlado pelo programados usando *typecast* (conversão de tipo:

```
pv = &x;
cout << "Inteiro: " << *(int*)pv); /*=> 10*/
pv = &f;
cout << "Real: " << *(float*)pv); /*=> 3.5*/
```

# Ponteiros e Strings

- Strings são arrays de caracteres e podem ser acessadas usando char *;

```
int main(){
   char str[] = "abcdef", *pc;
   for (pc = str; *pc != '\0'; pc++)
      cout << *pc;
}
```

Resultado:

- abcdef

- O incremento de pc o posiciona sobre o próximo caractere (byte a byte).

# Ponteiros e String

- Com base na função StrCopyC abaixo, você deverá construir um programa em C++ com duas variáveis chamadas frase1 e frase2. A variável frase1 deverá conter uma frase qualquer e deverá ser copiada para a variável frase2 utilizando a função abaixo:

```
void StrCopyC(char *destino, char *origem){
   while (*origem){
     *destino = *origem;
     origem++;
     destino++;
   }
   *destino = '\0';
}
```