



Universidade Federal do ABC
Centro de Matemática, Computação e Cognição

Linguagem de Programação C++

Monael Pinheiro Ribeiro, D.Sc.

Primeiro Programa em C++

Primeiro Programa em C++

```
int main()  
{  
    return 0;  
}
```

Primeiro Programa em C++

```
int main()  
{  
    return 0;  
}
```

Para “compilar” o programa C++ deve-se usar o seguinte comando, no seu console ou terminal:

```
g++ <nome_do_arquivo_fonte>.cpp -o <nome_do_arquivo_fonte>.exe
```

Para o programa do exemplo:

```
g++ primeiro.cpp -o primeiro.exe
```

Primeiro Programa em C++

```
int main()  
{  
    return 0;  
}
```

Então será gerado um arquivo objeto, que pode ser executado.
Ele é executado através do seguinte comando no console o terminal:

```
./<nome_do_arquivo>
```

Para o programa do exemplo:

```
./primeiro.exe
```

Comando de Saída

O comando para exibir uma mensagem no dispositivo de saída padrão (monitor):

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << "UFABC";
```



UFABC

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << 'M';
```



M

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << 97;
```



97

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << 3.1415;
```



3.1415

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores **<<**

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << "UFABC";  
std::cout << 'M';  
std::cout << 97;  
std::cout << 3.1415;
```



UFABCM973.1415

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << "UFABC" << std::endl;  
std::cout << 'M' << std::endl;  
std::cout << 97 << std::endl;  
std::cout << 3.1415 << std::endl;
```

```
UFABC  
M  
97  
3.1415
```

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores **<<**

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
std::cout << "UFABC\n";  
std::cout << 'M' << '\n';  
std::cout << 97 << '\n';  
std::cout << 3.1415 << '\n';
```

```
UFABC  
M  
97  
3.1415
```

Comando de Saída

- Sintaxe:

- `std::cout << literal ou variável;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para exibir mais de uma variável ou literal, separe-as por operadores <<

```
std::cout << variavel1 << literal << variavel2;
```

Exemplos:

```
int valor = 4;  
std::cout << valor << std::endl;
```



4

Segundo Programa em C++

```
#include <iostream>
int main()
{
    std::cout << "Ola Mundo!" << std::endl;
    return 0;
}
```

Comando para “compilar”:

Segundo Programa em C++

```
#include <iostream>
int main()
{
    std::cout << "Ola Mundo!" << std::endl;
    return 0;
}
```

Comando para “compilar”:

```
g++ segundo.cpp -o segundo.exe
```

Segundo Programa em C++

```
#include <iostream>
int main()
{
    std::cout << "Ola Mundo!" << std::endl;
    return 0;
}
```

Comando para “compilar”:

```
g++ segundo.cpp -o segundo.exe
```

Comando para “executar”:

Segundo Programa em C++

```
#include <iostream>
int main()
{
    std::cout << "Ola Mundo!" << std::endl;
    return 0;
}
```

Comando para “compilar”:

```
g++ segundo.cpp -o segundo.exe
```

Comando para “executar”:

```
./segundo.exe
```

Comando de Saída

Para exibir uma mensagem no dispositivo de saída padrão (monitor):

```
std::cout << literal ou variável;
```

O operador << concatena (junta, une) valores para serem exibidos.

Exemplo:

```
int tempo = 8;  
std::string universidade = "UFABC";  
std::cout << "Eu estou na " << universidade << " ha " << tempo <<  
" anos." << std::endl;
```

Comando de Saída

Para exibir uma mensagem no dispositivo de saída padrão (monitor):

```
std::cout << literal ou variável;
```

O operador << concatena (junta, une) valores para serem exibidos.

Exemplo:



```
Eu estou na UFABC ha 8 anos
```

```
int tempo = 8;
std::string universidade = "UFABC";
std::cout << "Eu estou na " << universidade << " ha " << tempo <<
" anos." << std::endl;
```

Lendo valores via teclado

- Sintaxe:

- `std::cin >> variavel;`

- Para omitir o **std**, deve-se acrescentar a linha **using namespace std;** no início do programa.
- Para ler mais de uma variável, separe-as por operadores `>>`
`std::cin >> variavel1 >> variavel2;`

Comando de Entrada

Exemplo para ler um inteiro:

```
int valor;  
std::cin >> valor;
```

Terceiro Programa em C++

```
#include <iostream>
int main()
{
    int tempo;
    std::string universidade;

    std::cout << "Onde voce estuda? ";
    std::cin >> universidade;
    std::cout << "Quanto tempo: ";
    std::cin >> tempo;

    std::cout << "Voce estuda na " << universidade << " ha " << tempo
                << " anos." << std::endl;
    return 0;
}
```

Comando para “compilar”: `g++ terceiro.cpp -o terceiro.exe`

Comando para “executar”: `./terceiro.exe`

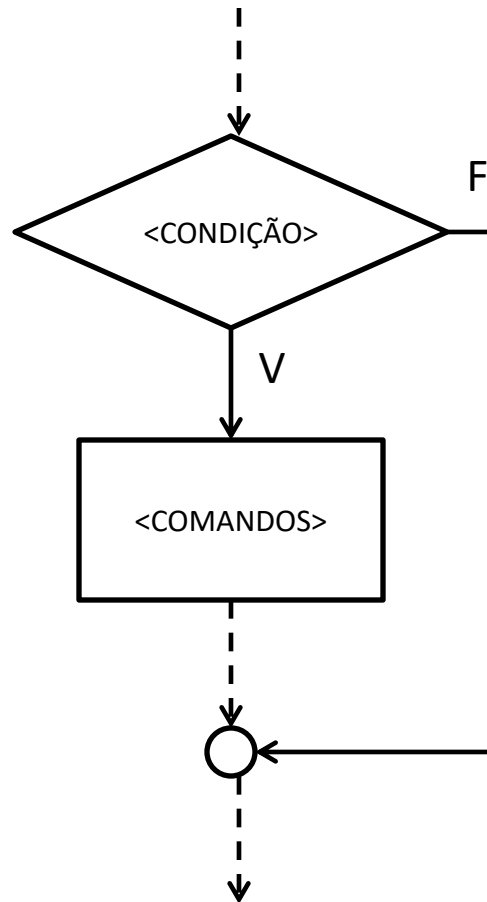
Operadores Aritméticos

- A tabela a seguir mostra os operadores aritméticos e suas funções.

| Operador | Função | Tipo | Resultado |
|----------|---------------------|---------|-----------------|
| + | Manutenção de sinal | Unário | |
| - | Inversão de sinal | Unário | |
| + | Adição | Binário | Inteiro ou Real |
| - | Subtração | Binário | Inteiro ou Real |
| * | Multiplicação | Binário | Inteiro ou Real |
| / | Divisão | Binário | Inteiro ou Real |
| % | Resto da Divisão | Binário | Inteiro |

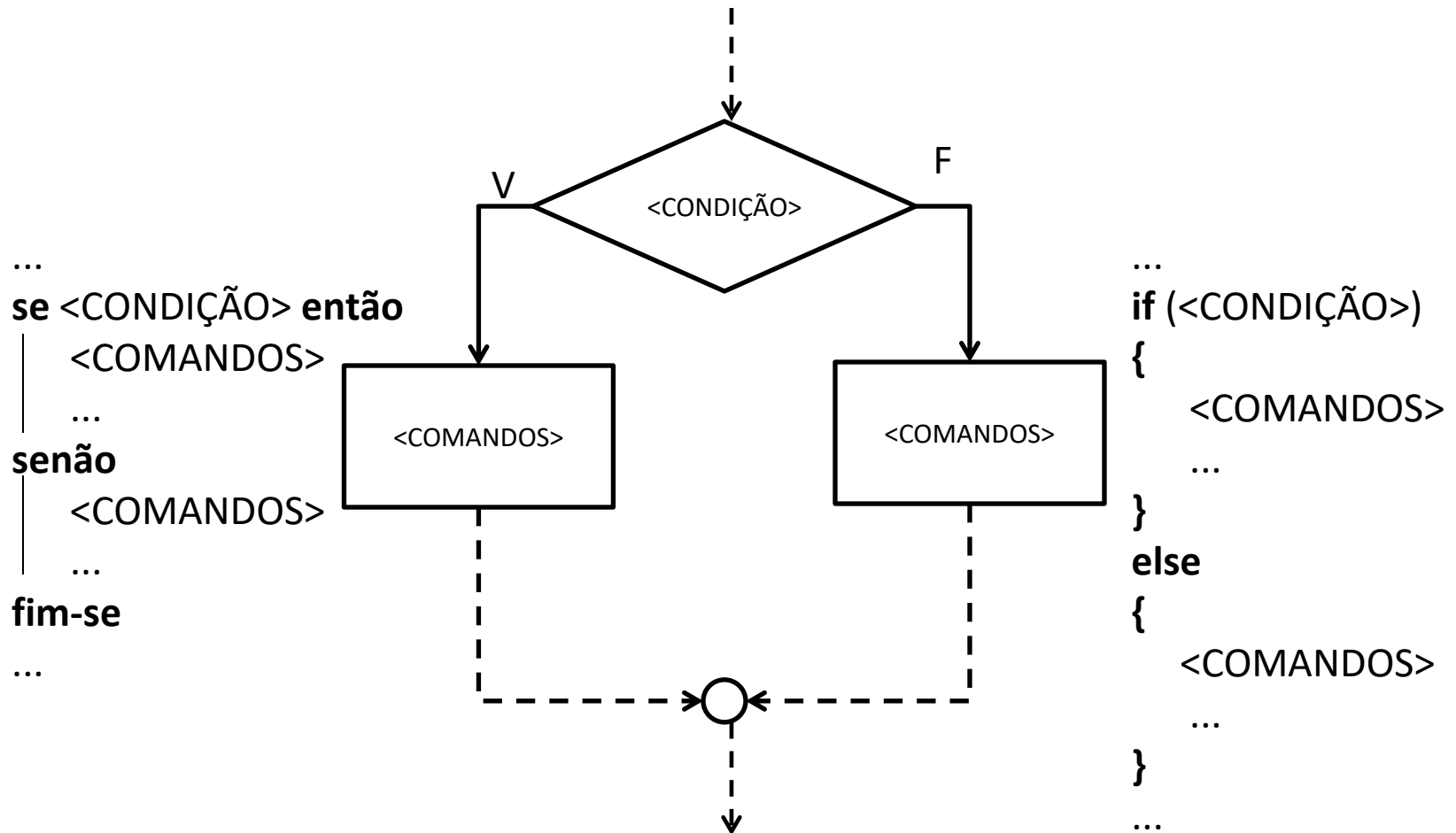
Estrutura Condicional Simples

```
...  
se <CONDIÇÃO> então  
|   <COMANDOS>  
|   ...  
fim-se  
...
```



```
...  
if (<CONDIÇÃO>)  
{  
    <COMANDOS>  
    ...  
}  
...
```

Estrutura Condicional Composto



Operadores Relacionais

- São operadores que operam sobre variáveis numéricas ou expressões aritméticas e retornam valores lógicos.

| Operador | Operação | Descrição |
|----------|------------------|--|
| > | Maior que | Verifica se um valor é maior a outro. |
| < | Menor que | Verifica se um valor é menor a outro. |
| >= | Maior ou igual a | Verifica se um valor é maior ou igual a outro. |
| <= | Menor ou igual a | Verifica se um valor é menor ou igual a outro. |
| == | Igualdade | Verifica se dois valores são iguais. |
| != | Diferença | Verifica se dois valores são diferentes |

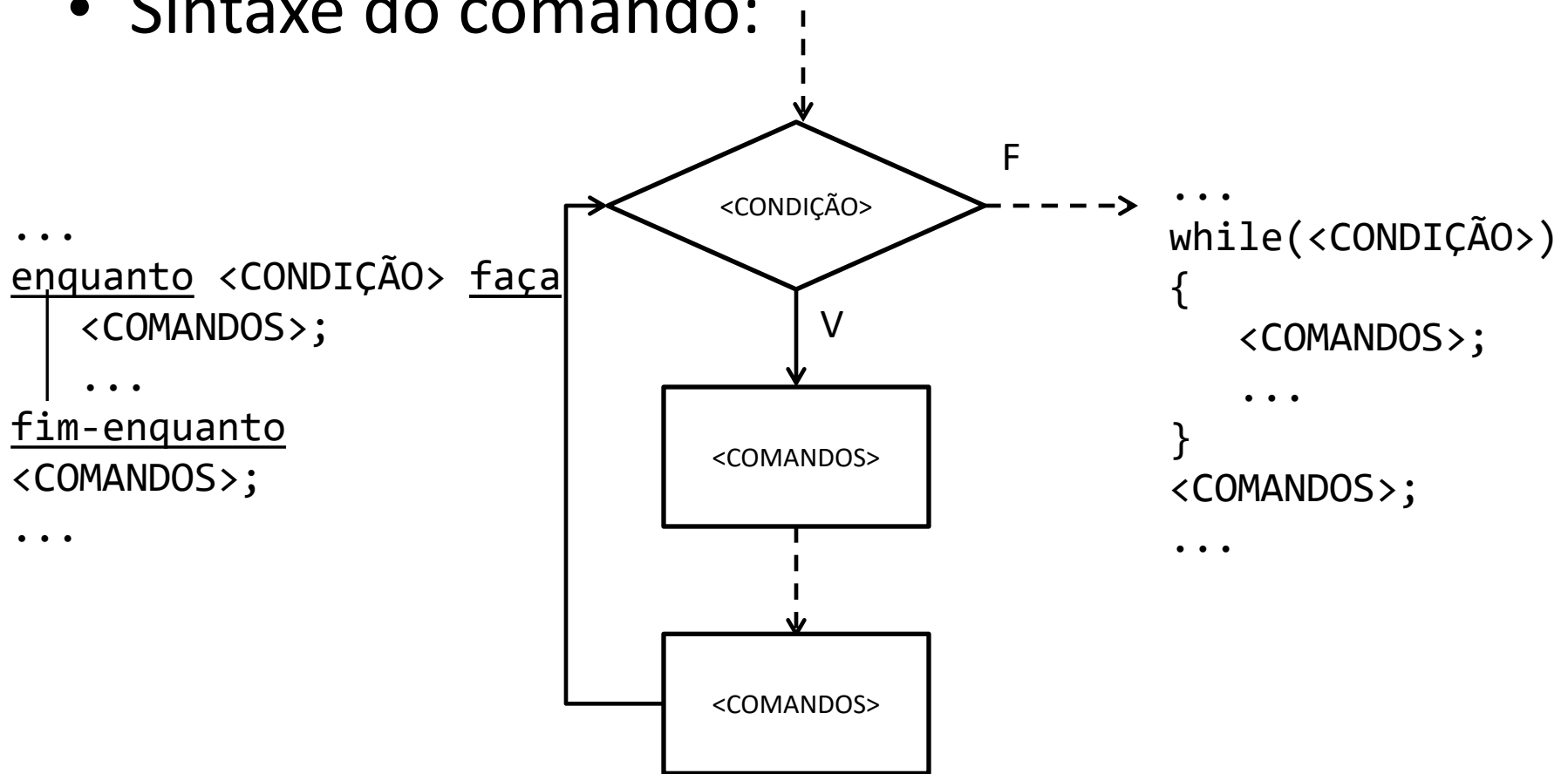
Operadores Lógicos

- Quando você precisa criar uma condição **com mais de uma** expressão relacional, então usa-se os operadores lógicos.

| Operador | Operação | Descrição |
|----------|-----------|--|
| && | E lógico | Retorna verdade apenas se os dois componentes forem verdadeiros. |
| | Ou Lógico | Retorna falso apenas se os dois componentes forem falsos. |
| ! | Negação | Contradiz o argumento. |

Estrutura de Repetição

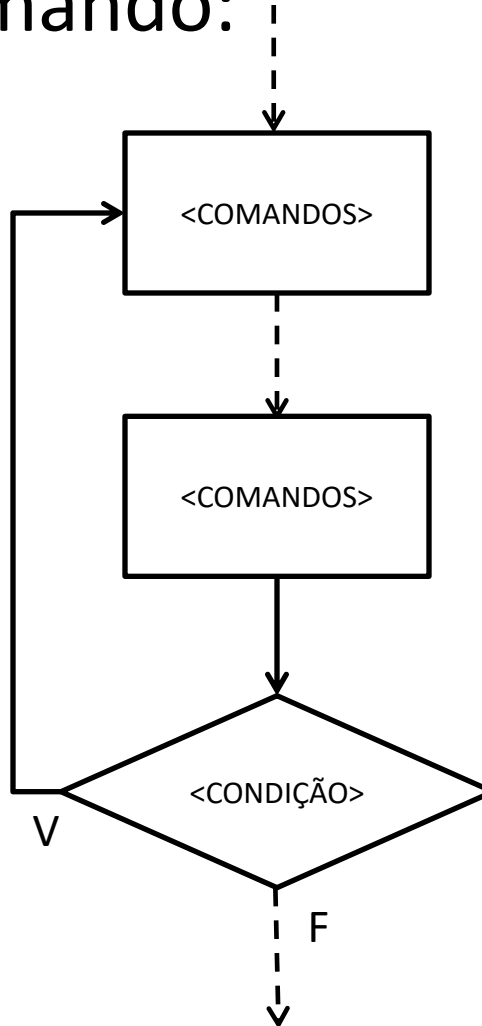
- Sintaxe do comando:



Estrutura de Repetição

- Sintaxe do comando:

...
faça <COMANDOS>
 ...
até que <CONDIÇÃO>
<COMANDOS>
...

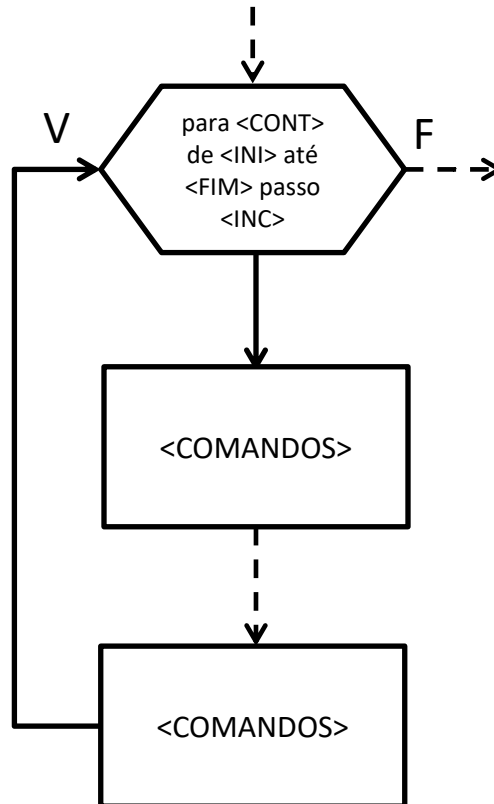


...
do
{
 <COMANDOS>;
 ...
}while(<CONDIÇÃO>);
<COMANDOS>;
...

Estrutura de Repetição

- Sintaxe do comando:

...
para <VAR> de <INI> até <FIM> passo <INC> faça
|
<COMANDOS>
|
...
fim-para
<COMANDOS>
...



...
for(<INI>;<COND>;<INC>)
{
 <COMANDOS>;
 ...
}
<COMANDOS>;
...

Variáveis Acumuladoras e Contadores

- São variáveis que acumulam valores
- Acumula seu valor anterior, mais outro valor
- No caso dos contadores o valor adicionado é sempre uma constante.
- Costumam ser usadas em ciclos para acumularem valores
- As variáveis acumuladoras devem SEMPRE ser inicializadas com um valor NULO.

Operadores de Incremento e Decremento

| Operador | Tipo | Descrição |
|----------|---------|------------------------------|
| ++ | Unário | Incremento Unitário |
| -- | Unário | Decremento Unitário |
| += | Binário | Atribuição por Adição |
| -= | Binário | Atribuição por Subtração |
| *= | Binário | Atribuição por Multiplicação |
| /= | Binário | Atribuição por Divisão |
| %= | Binário | Atribuição por Módulo |

Ponteiros

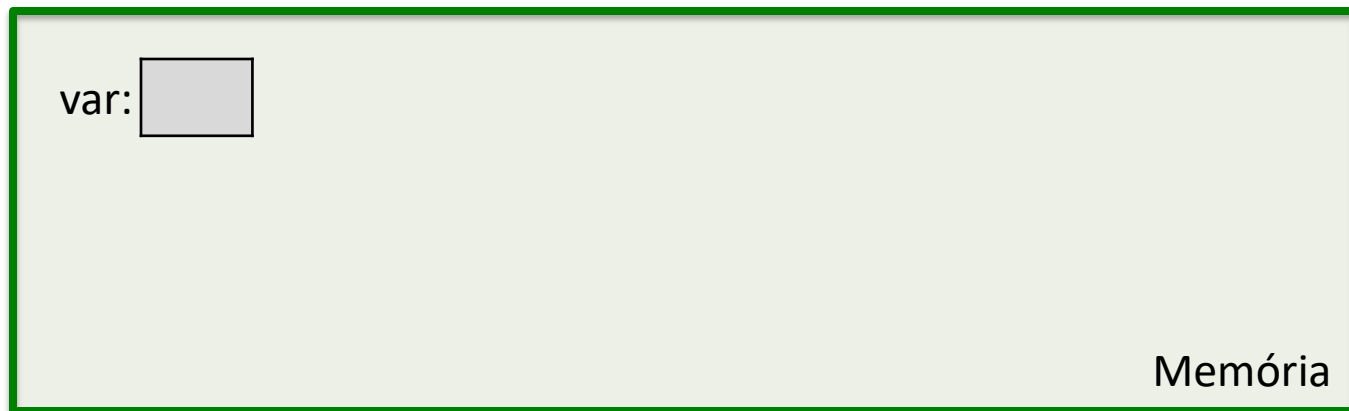
- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo

Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var;

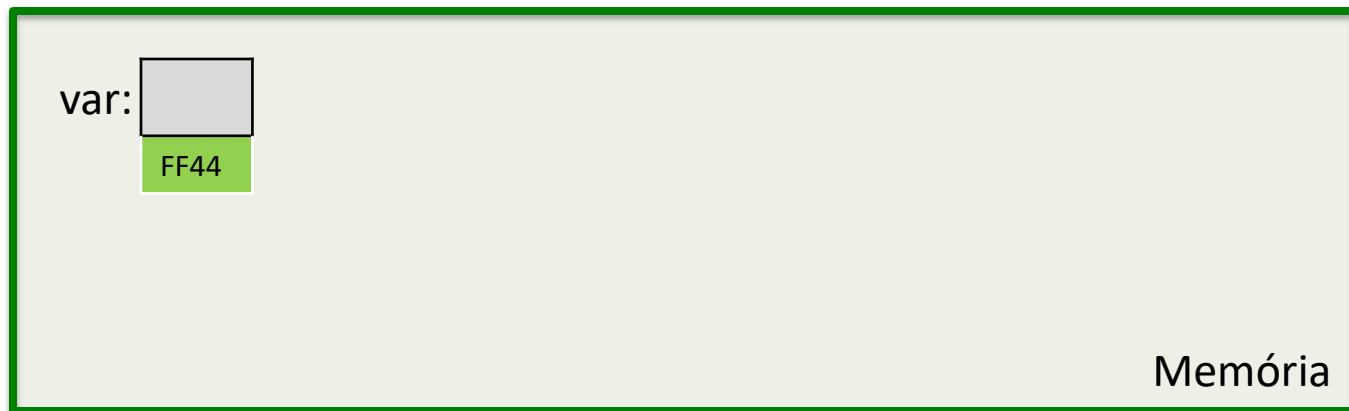
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** **var**;



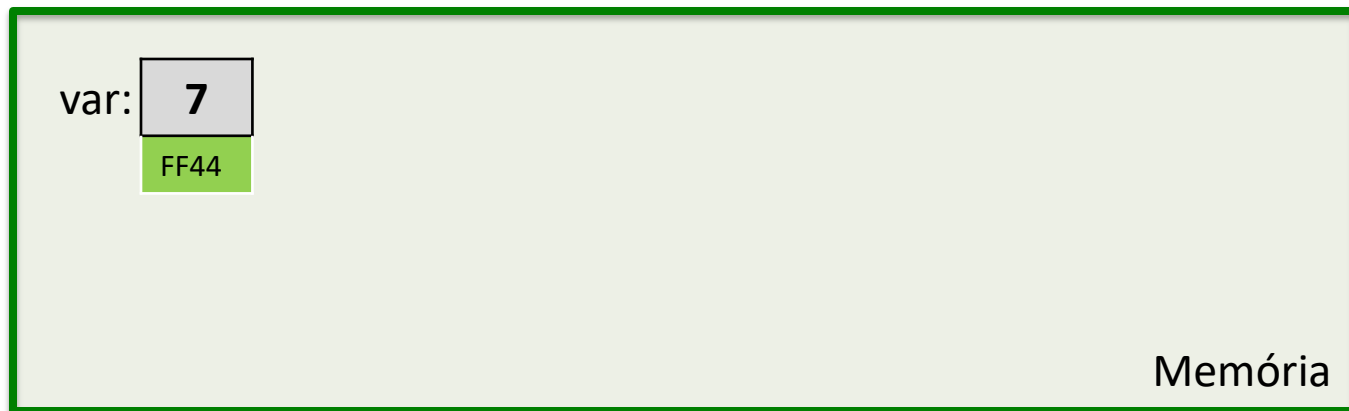
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var;



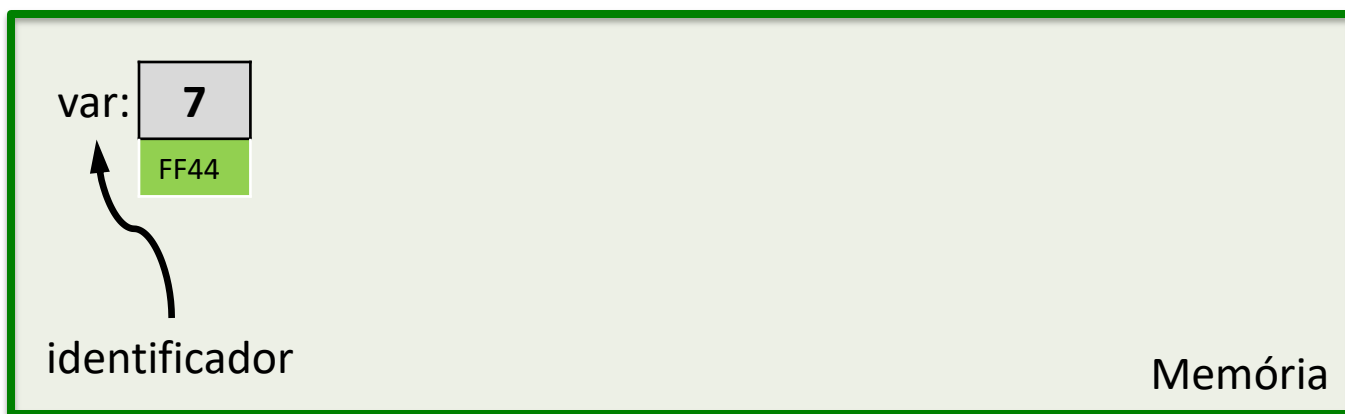
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var; var = 7;



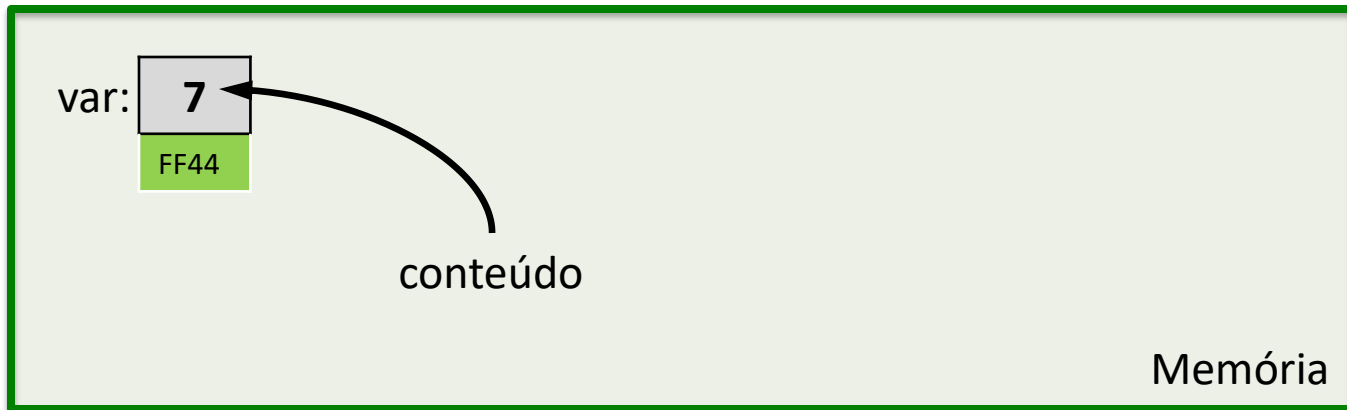
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var; var = 7;



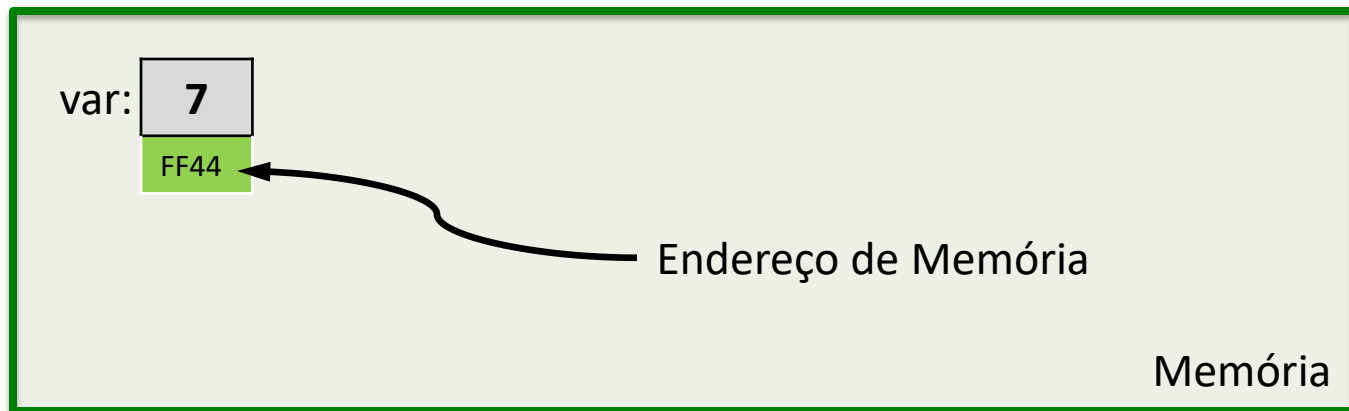
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var; var = 7;



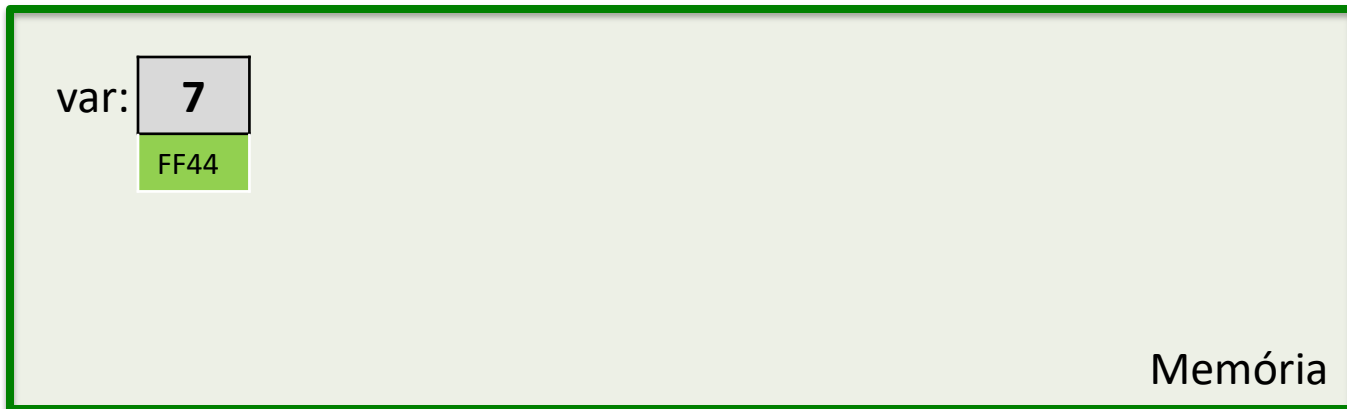
Ponteiros

- O que são ponteiros?
 - Toda variável tem um identificador (nome) e está alocada em uma região de memória.
 - Esta região é o ponteiro para essa variável.
 - Exemplo: **int** var; var = 7;



Variáveis Ponteiros

- Ponteiros como variáveis

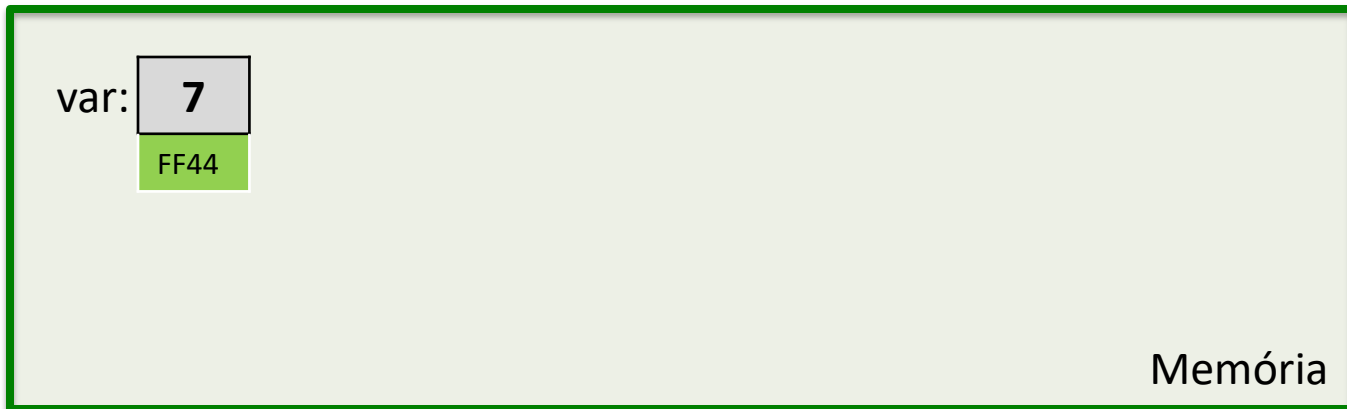


- Seria possível armazenar o endereço de uma variável em uma outra variável?



Variáveis Ponteiros

- Ponteiros como variáveis



- Seria possível armazenar o endereço de uma variável em uma outra variável?

SIM



Variáveis Ponteiros

- Ponteiros como variáveis
 - Ponteiros são variáveis que armazenam endereços de outras variáveis.
 - Declaração:
 - Para declarar uma variável do tipo ponteiro, você deve explicitar seu tipo e seu identificador. Da seguinte forma:

<tipo> * **<identificador>;**

- Exemplo:

int ***ptInteiro;**

float ***ptReal;**

char ***ptlCaractere;**

Endereço de uma variável

- Como obter o endereço de uma variável?
 - Através do operador unário &.
 - Para conhecer o endereço ocupado por uma variável usa-se o operador de endereço (&).
 - O resultado deste operador é um ponteiro constante.
 - Exemplo:

```
1. int main()    {  
2.     int var1, var2;  
3.     std::cout << "Endvar1 = " << &var1 << "\nEndvar2 = " << &var2 << std::endl;  
4.     return 0;  
5. }
```

– Saída:

```
Endvar1 = 0022FF44  
Endvar2 = 0022FF40
```

Entenda Melhor

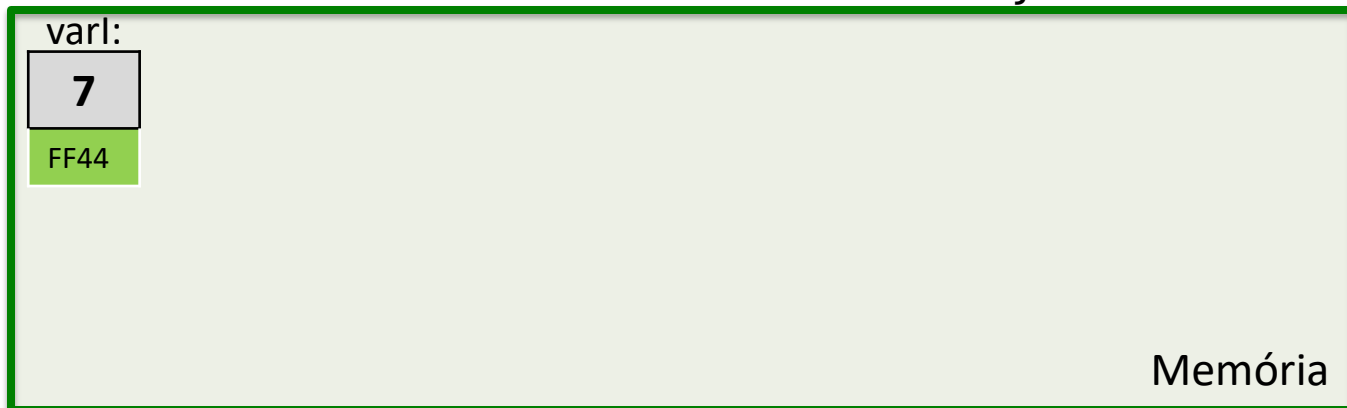
- Onde guardar um endereço de memória?
 - O retorno do operador & sobre uma variável, pode ser armazenado em um ponteiro desde que haja compatibilidade de tipos.
 - Ou seja, variáveis ponteiros para inteiro, guardam endereços de inteiros; assim como ponteiros para flutuantes armazenam endereços de flutuantes e assim para todo e qualquer tipo.
 - Exemplo:

```
1.  int main( )  {
2.      int varI;
3.      float varF;
4.      char varC;
5.      int *ptInt;
6.      float *ptFloat;
7.      char *ptChar;
8.      ptInt = &varI;
9.      ptFloat = &varF;
10.     ptChar = &varC;
11.     return 0;
12. }
```

Entenda Melhor

- Veja o código semelhante ao anterior:

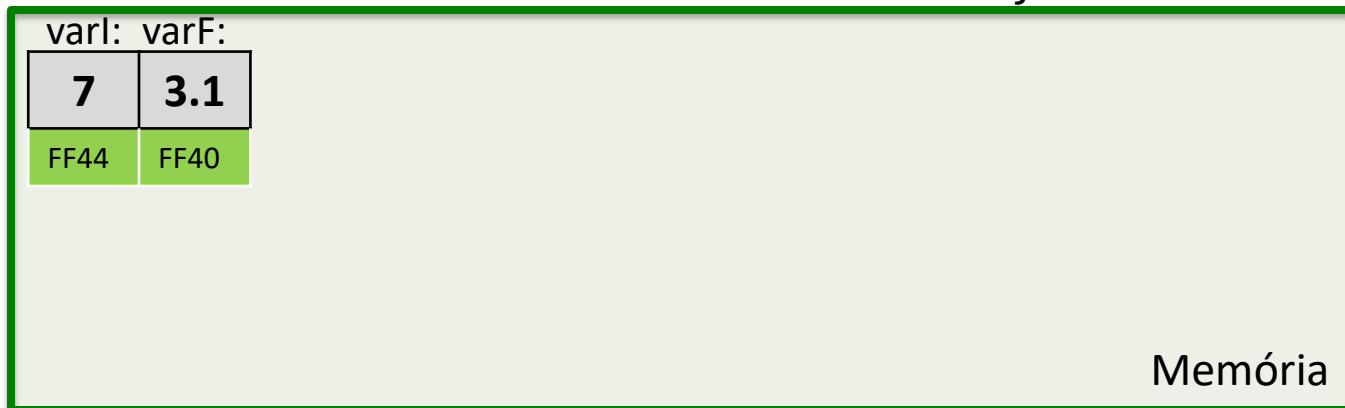
```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```



Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```



Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

varI: varF: varC:

| | | |
|------|------|------|
| 7 | 3.1 | 'M' |
| FF44 | FF40 | FF3F |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

varI: varF: varC: ptInt:

| | | | |
|------|------|------|------|
| 7 | 3.1 | 'M' | |
| FF44 | FF40 | FF3F | FF38 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

| varI: | varF: | varC: | ptInt: | ptFloat: |
|-------|-------|-------|--------|----------|
| 7 | 3.1 | 'M' | | |
| FF44 | FF40 | FF3F | FF38 | FF34 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

| varI: | varF: | varC: | ptInt: | ptFloat: | ptChar: |
|-------|-------|-------|--------|----------|---------|
| 7 | 3.1 | 'M' | | | |
| FF44 | FF40 | FF3F | FF38 | FF34 | FF30 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

| varI: | varF: | varC: | ptInt: | ptFloat: | ptChar: |
|-------|-------|-------|--------|----------|---------|
| 7 | 3.1 | 'M' | FF44 | | |
| FF44 | FF40 | FF3F | FF38 | FF34 | FF30 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

| varI: | varF: | varC: | ptInt: | ptFloat: | ptChar: |
|-------|-------|-------|--------|----------|---------|
| 7 | 3.1 | 'M' | FF44 | FF40 | |
| FF44 | FF40 | FF3F | FF38 | FF34 | FF30 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

| varI: | varF: | varC: | ptInt: | ptFloat: | ptChar: |
|-------|-------|-------|--------|----------|---------|
| 7 | 3.1 | 'M' | FF44 | FF40 | FF3F |
| FF44 | FF40 | FF3F | FF38 | FF34 | FF30 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

- Dizemos que:

- ptInt aponta para varI;
- ptFloat aponta para varF; e
- ptChar aponta para varC.

```
1.  int main( )  {  
2.      int varI = 7;  
3.      float varF = 3.1;  
4.      char varC = 'M';  
5.      int *ptInt;  
6.      float *ptFloat;  
7.      char *ptChar;  
8.      ptInt = &varI;  
9.      ptFloat = &varF;  
10.     ptChar = &varC;  
11.     return 0;  
12. }
```

varI: varF: varC: ptInt: ptFloat: ptChar:

| | | | | | |
|------|------|------|------|------|------|
| 7 | 3.1 | 'M' | FF44 | FF40 | FF3F |
| FF44 | FF40 | FF3F | FF38 | FF34 | FF30 |

Memória

Conteúdo de uma variável apontada

- Como ter acesso ao valor de uma variável apontada?
 - Usa-se o operador unário * (resolução de endereço).
 - O operador de resolução de endereço é aplicado sobre uma variável ponteiro.
 - Seu resultado é o valor da variável apontada.
 - Exemplo:

```
1. int main( ) {  
2.     int var=7, *ptrVar = &var;  
3.     std::cout << "ptrVar = " << ptrVar << "\n*ptrVar = " << *ptrVar << std::endl;  
4. }
```

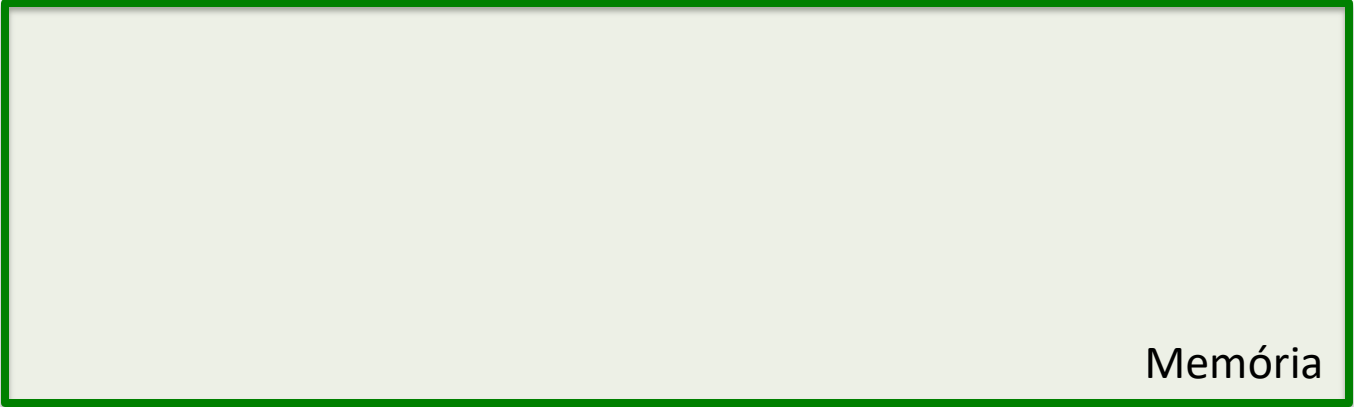
– Saída:

```
ptrVar = 0022FF44  
*ptrVar = 7
```

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

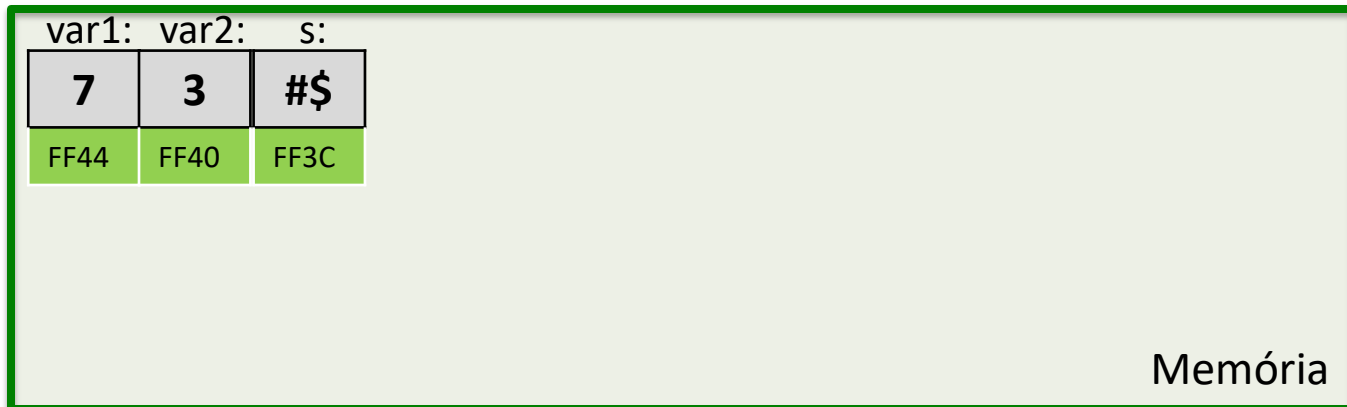


Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```



Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | #\$ | | |
| FF44 | FF40 | FF3C | FF38 | FF34 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | #\$ | FF44 | |
| FF44 | FF40 | FF3C | FF38 | FF34 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | #\$ | FF44 | FF40 |
| FF44 | FF40 | FF3C | FF38 | FF34 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | #\$ | FF44 | FF40 |
| FF44 | FF40 | FF3C | FF38 | FF34 |

Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

6. `s = (*ptV1) + (*ptV2);`

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | #\$ | FF44 | FF40 |
| FF44 | FF40 | FF3C | FF38 | FF34 |

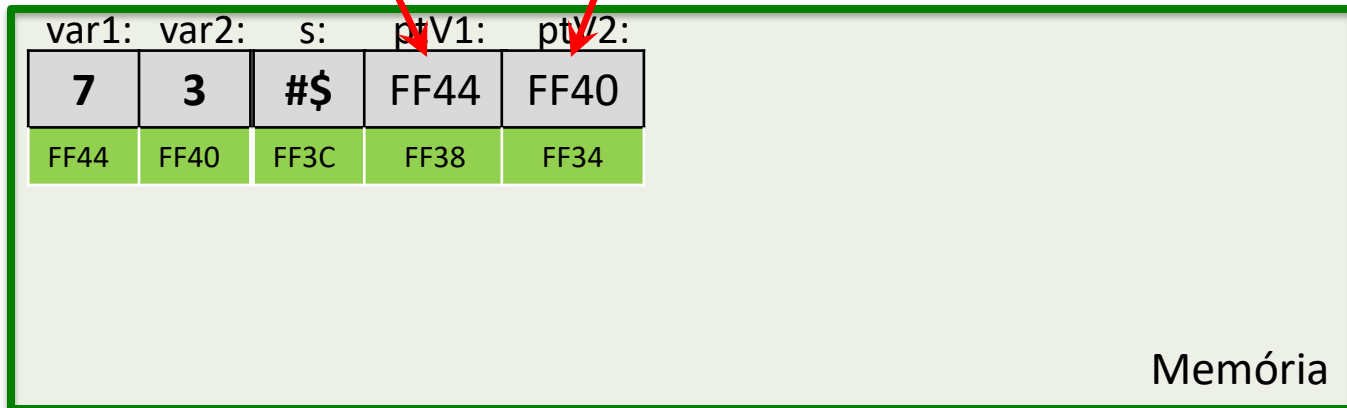
Memória

Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

6. $s = (*ptV1) + (*ptV2);$

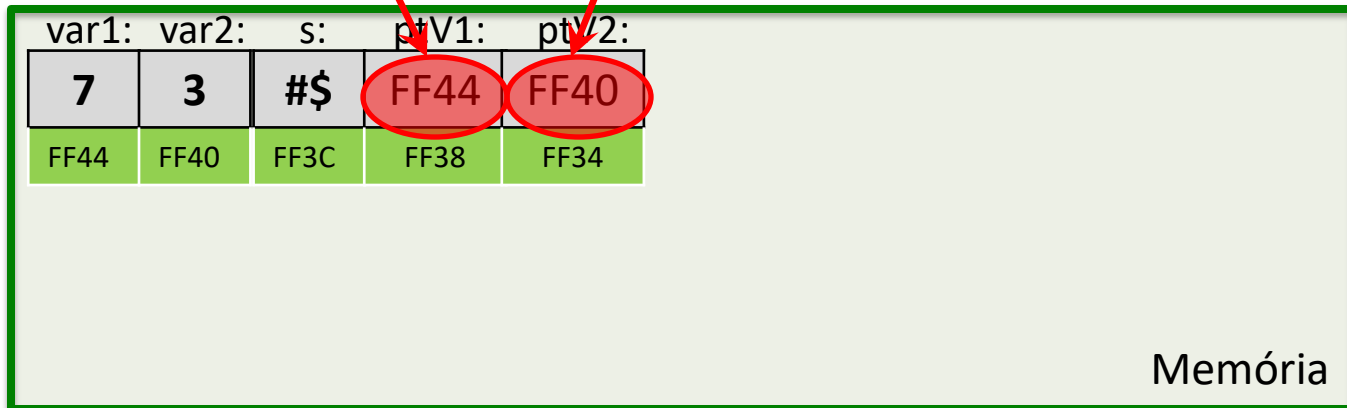


Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

6. $s = (*ptV1) + (*ptV2);$

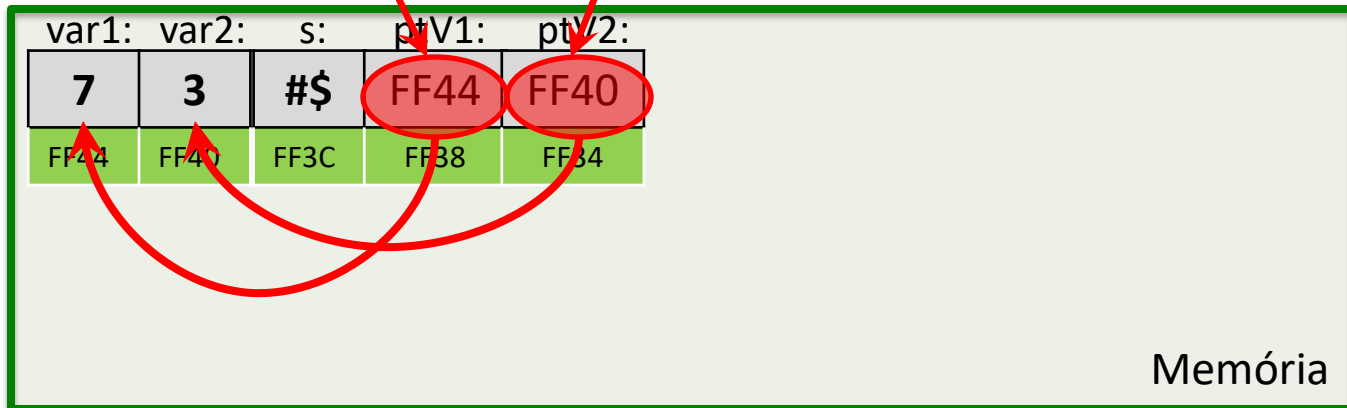


Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

6. $s = (*ptV1) + (*ptV2);$

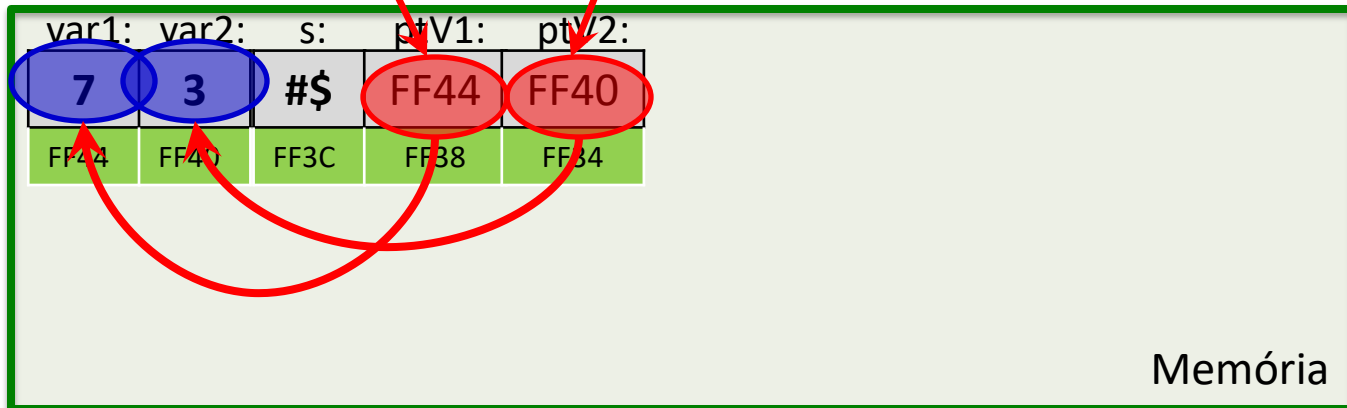


Entenda Melhor

- Veja o código semelhante ao anterior:

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

6. $s = (*ptV1) + (*ptV2);$



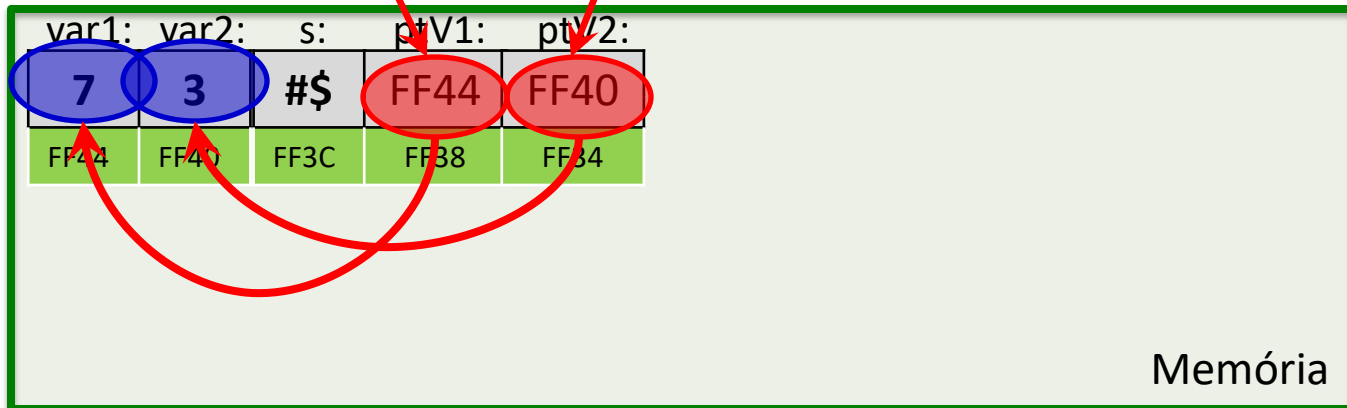
Entenda Melhor

- Veja o código semelhante ao anterior:

6.

$s = 7 + 3 ;$
 $s = (*ptV1) + (*ptV2);$

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```



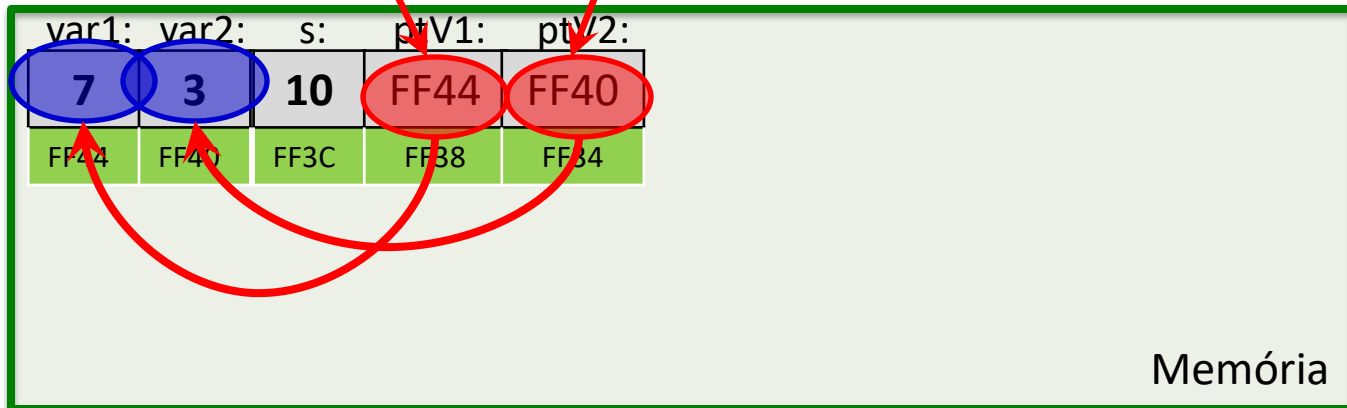
Entenda Melhor

- Veja o código semelhante ao anterior:

6.

$s = 7 + 3 ;$
 $s = (*ptV1) + (*ptV2);$

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```



Entenda Melhor

- Veja o código semelhante ao anterior:

- Na linha 6, dizemos que:

- As variáveis var1 e var 2 foram acessadas indiretamente.

```
1.  int main( )  {  
2.      int var1 = 7, var2 = 3, s;  
3.      int *ptV1, *ptV2;  
4.      ptV1 = &var1;  
5.      ptV2 = &var2;  
6.      s = (*ptV1) + (*ptV2);  
7.      return 0;  
8.  }
```

| var1: | var2: | s: | ptV1: | ptV2: |
|-------|-------|------|-------|-------|
| 7 | 3 | 10 | FF44 | FF40 |
| FF44 | FF40 | FF3C | FF38 | FF34 |

Memória

Permutação de Valores

- Analise o seguinte programa:

```
void troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
}

int main( )
{
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(a, b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

Permutação de Valores

- Analise o seguinte programa:

```
void troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
}
```

O que será impresso na tela?

```
int main( )
{
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(a, b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

Permutação de Valores

- Analise o seguinte programa:

```
void troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
}
```

```
a = 1
b = 3
Apos invocar a funcao troca:
```

```
int main( )
{
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(a, b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

Permutação de Valores

- Analise o seguinte programa:



```
void troca(int x, int y)
{
    int aux = x;
    x = y;
    y = aux;
}
```

```
a = 1
b = 3
Apos invocar a funcao troca:
a = 1
b = 3
```

```
int main( )
{
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(a, b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

Permutação de Valores

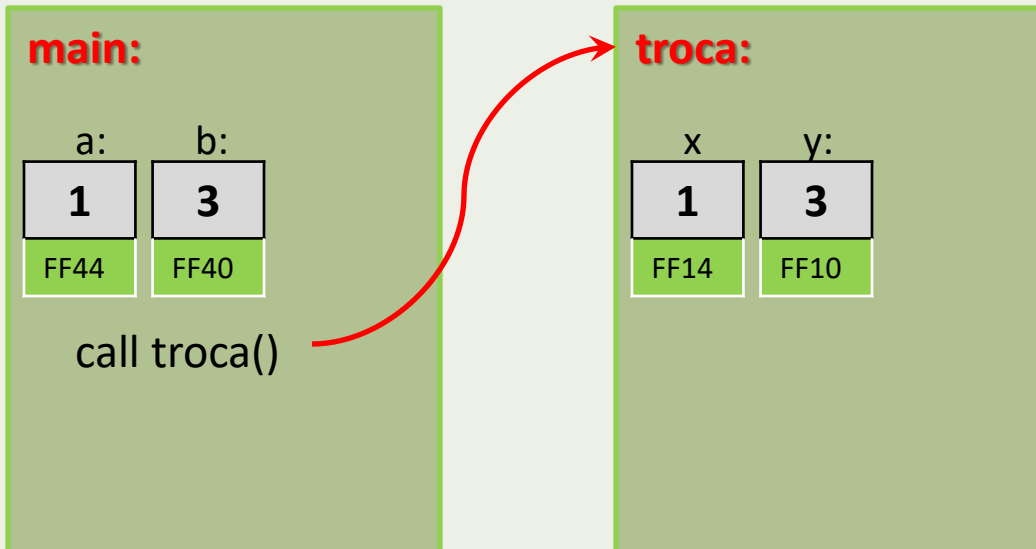
- Observe o que acontece...
 - Ao iniciar o programa, as variáveis a e b tem escopo local restrito à função principal.

main

| a: | b: |
|------|------|
| 1 | 3 |
| FF44 | FF40 |

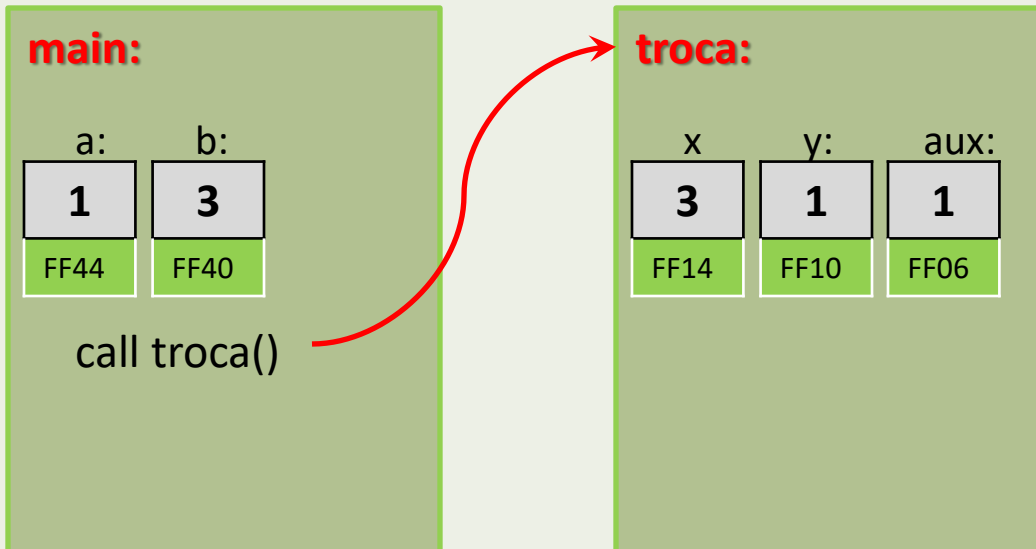
Permutação de Valores

- Observe o que acontece...
 - Então a função principal invoca a sub-rotina troca passando como argumentos os valores das variáveis a e b para x e y respectivamente.



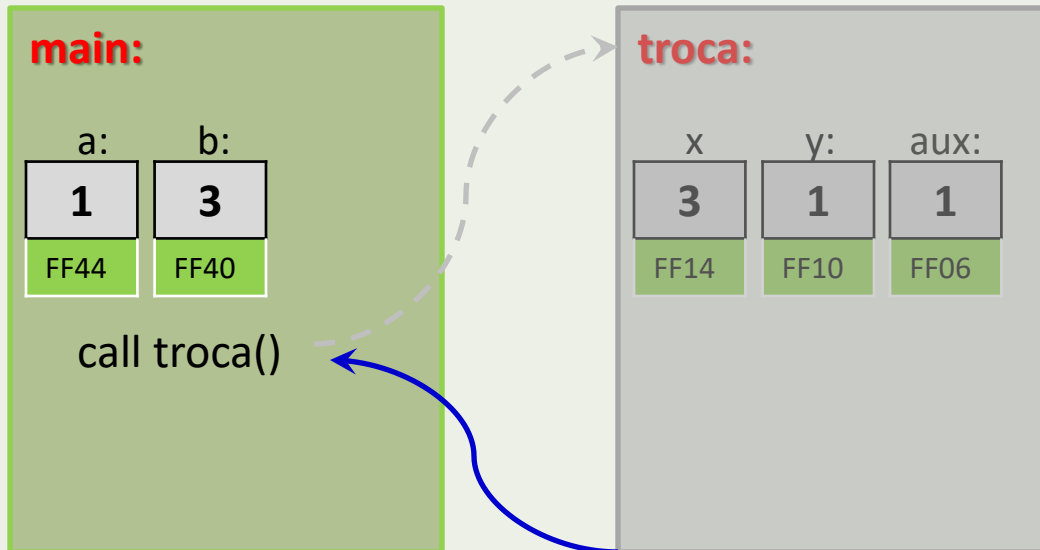
Permutação de Valores

- Observe o que acontece...
 - A sub-rotina troca efetua suas instruções promovendo através de uma variável auxiliar a troca de valores entre as variáveis x e y.



Permutação de Valores

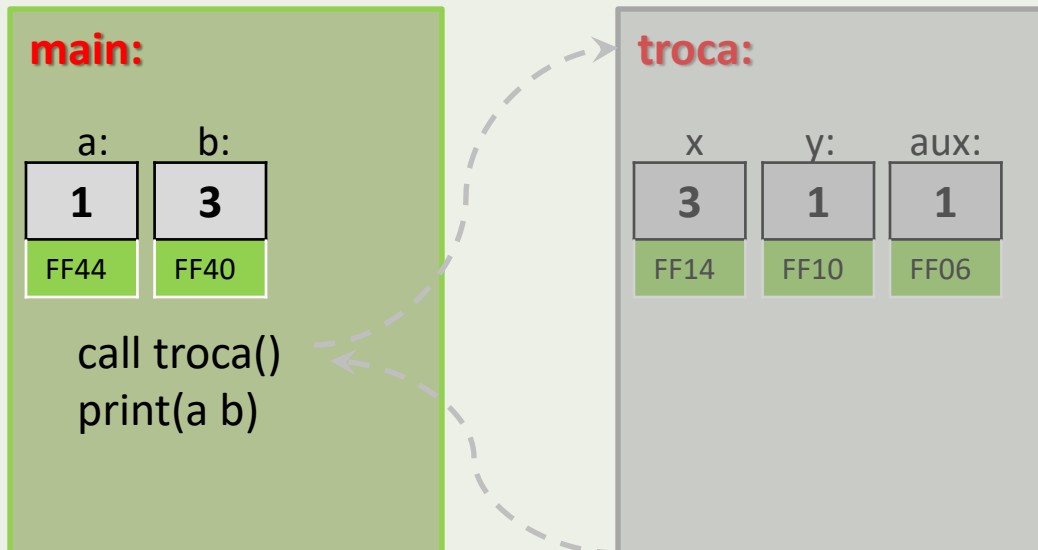
- Observe o que acontece...
 - A função troca retorna o controle para a função principal e seu escopo é extinto da memória.



Permutação de Valores



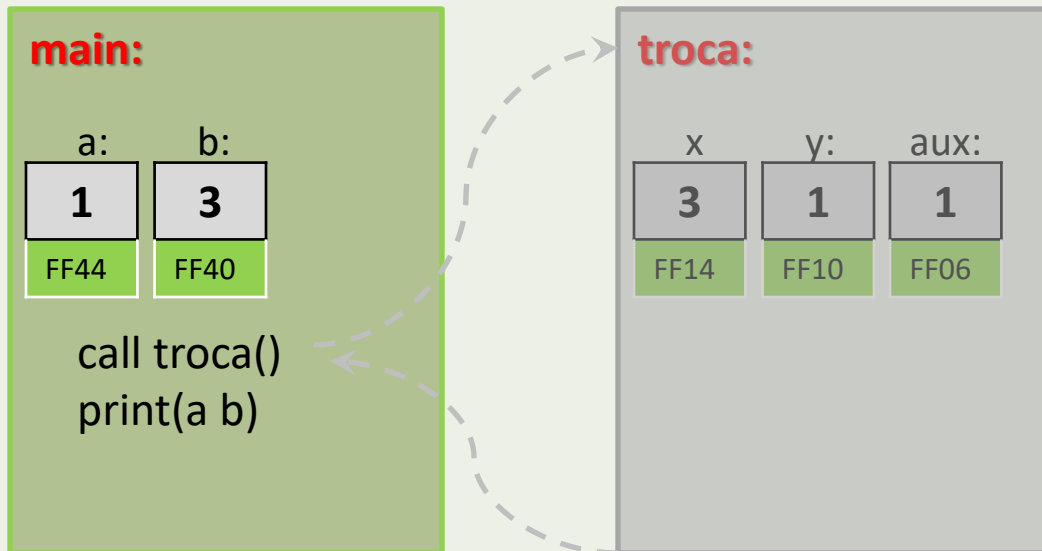
- Observe o que acontece...
 - Ao receber o controle a função principal imprime os valores de a e b na tela. Entretanto, observe que a troca aconteceu no escopo de troca, que já foi extinto...



Permutação de Valores



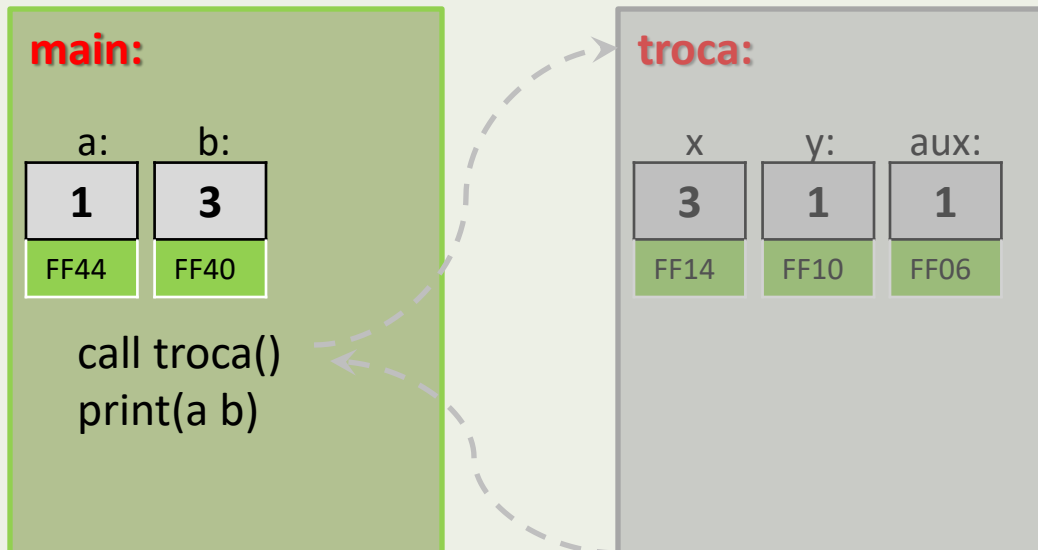
- Como resolver esse problema ???



Permutação de Valores



- Como resolver esse problema ???
 - Desejo que os valores de a e b que tem escopo na função principal, sejam acessados e modificados no escopo da função troca.



Permutação de Valores



- Como resolver esse problema ???
 - Passa-se como argumento para a função troca os endereços das variáveis e não o valor ...
 - Isso é o que chamamos de Passagem de Parâmetro por Referência.

```
void troca(int *x, int *y)
{
    int aux = *x;
    *x = *y;
    *y = aux;
}
```

O que será impresso na tela?

```
int main( )
{
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(&a, &b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

Permutação de Valores



- Como resolver esse problema ???
 - Passa-se como argumento para a função troca os endereços das variáveis e não o valor ...
 - Isso é o que chamamos de Passagem de Parâmetro por Referência.

```
void troca(int *x, int *y)
{
    int aux = *x;
    *x = *y;
    *y = aux;
}
```

```
int main( )
{
```

```
    int a=1, b=3;
    std::cout << "a = " << a << "b = " << b << std::endl;
    troca(&a, &b);
    std::cout << "Apos invocar a funcao troca" << std::endl;
    std::cout << "a = " << a << "b = " << b << std::endl;
}
```

```
a = 1
```

```
b = 3
```

```
Após invocar a funcao troca:
```

```
a = 3
```

```
b = 1
```

Alocação Dinâmica de Memória

- O operador que permite alocar memória é:
- **new** em Linguagem C++:
- **new** retorna um ponteiro para a região de memória alocado.

Alocação Dinâmica de Memória

- Em Linguagem C++:
- O endereço retornado deve ser devidamente armazenado em uma variável ponteiro.
 - `int *pi = new int;`
 - `float *pf = new float;`
 - `char *pc = new char;`

Alocação Dinâmica de Memória

- A memória requisitada pelo seu programa deve ser devolvida ao Sistema Operacional
 - Em Linguagem C++:
 - `delete variavel_ponteiro;`
 - Exemplo:
 - `delete vetFil;`
- Não devolver a memória alocada pode acarretar em erros de execução.
 - Stack Overflow

Vetores

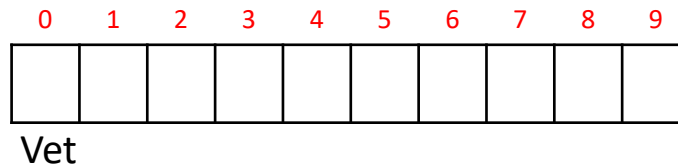
- Declaração

- Sintaxe:

<tipo> nome_da_variavel [tamanho];

- Exemplo: declaração de um vetor de inteiros chamado Vet com 10 elementos.

int vet[10];



- Exemplo: declaração de um vetor de flutuantes chamado pesos com 100 elementos.

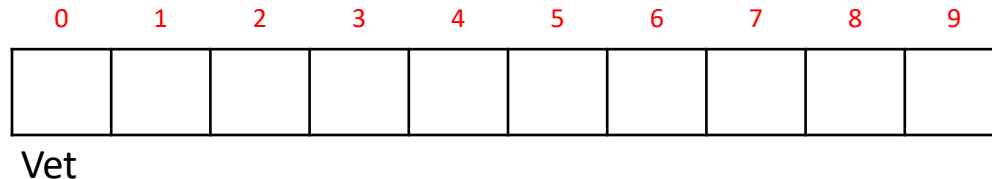
float pesos[100];

Vetores

- Acessando os índices de um vetor

– Sintaxe:

nome_da_variavel [**<indice>**] = valor;



Vetores

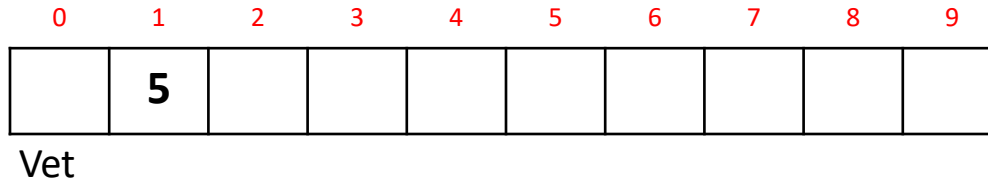
- Acessando os índices de um vetor

- Syntax:

nome_da_variavel [**<indice>**] = valor;

- **Exemplo:** atribuir 5 ao vetor Vet no índice 1:

Vet[1] = 5;



Vet

Vetores

- Acessando os índices de um vetor

- Syntax:

nome_da_variavel [**<indice>**] = valor;

- **Exemplo:** atribuir 5 ao vetor Vet no índice 1:

Vet[1] = 5;

- **Exemplo:** atribuir -7 ao vetor Vet no índice 0:

Vet[0] = -7;

| | | | | | | | | | |
|----|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| -7 | 5 | | | | | | | | |

Vet

Vetores

- Lendo e imprimindo valores a partir de um vetor:
 - Igual se faz com uma variável não vetor, porém jamais se esqueça de informar o índice.
 - Leitura:
 - Ler um flutuante via teclado e armazenar no índice 5 do vetor pesos.

```
std::cin >> pesos[5];
```

- Impressão:
 - Imprimir na tela o valor armazenado no índice 3 do vetor pesos.

```
std::cout << pesos[3];
```

Vetores

- Como manipular eficientemente vetores?

Vetores

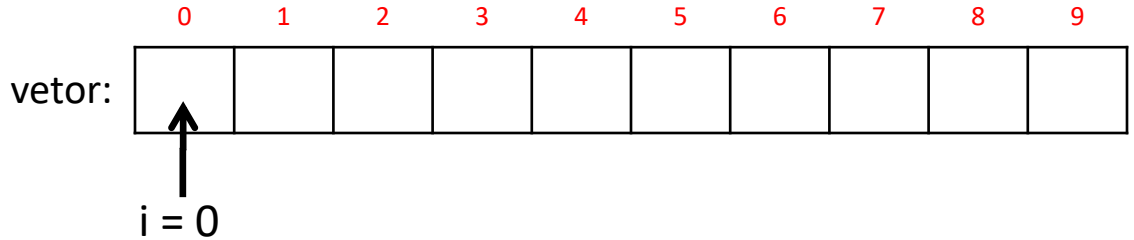
- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```


Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

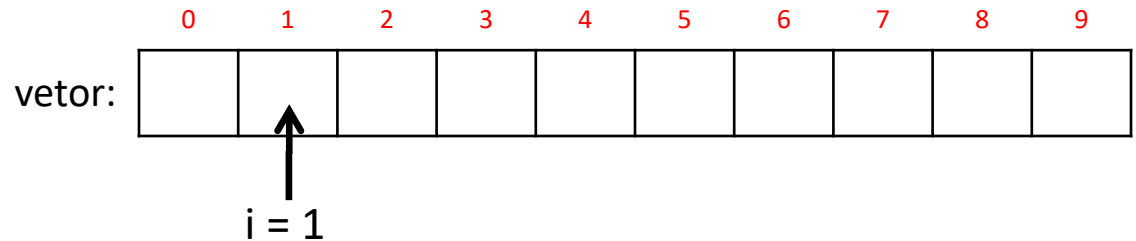
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

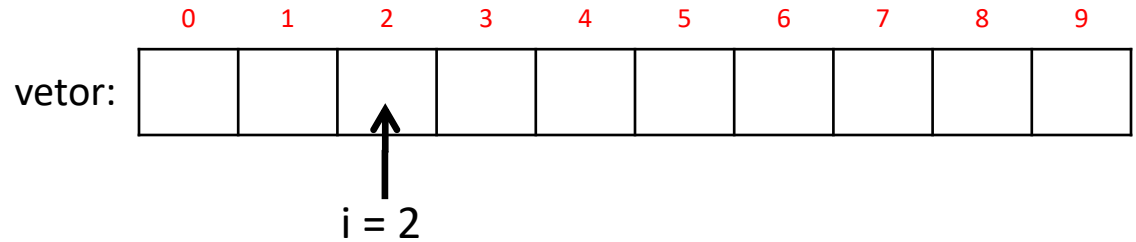
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

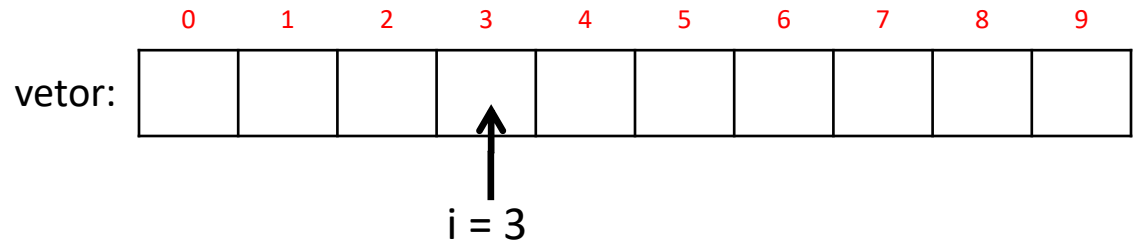
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

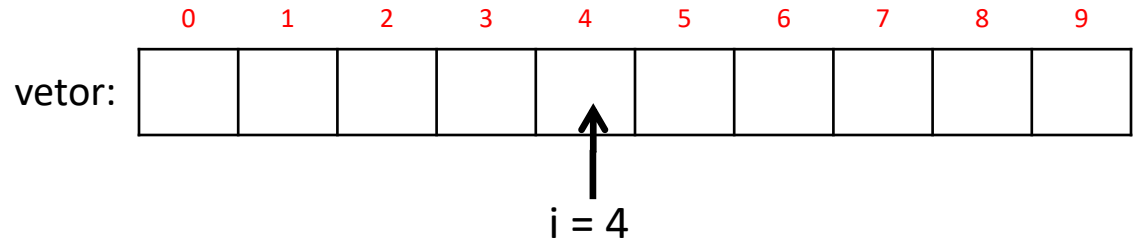
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

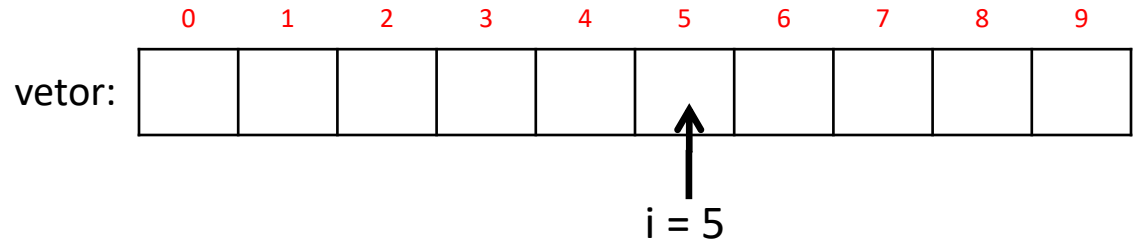
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

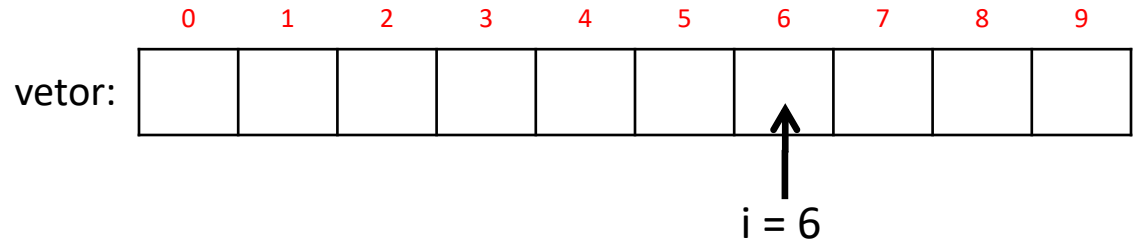
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

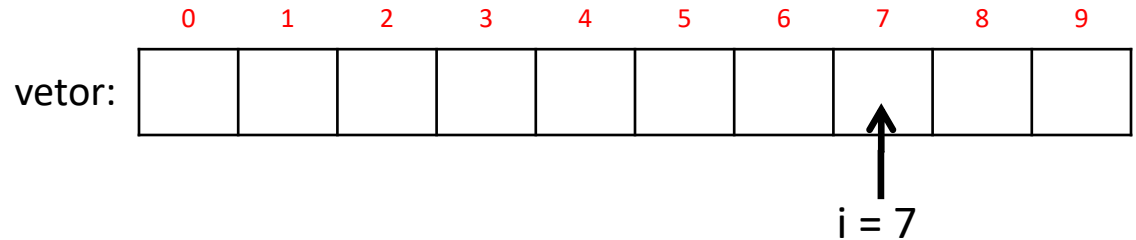
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

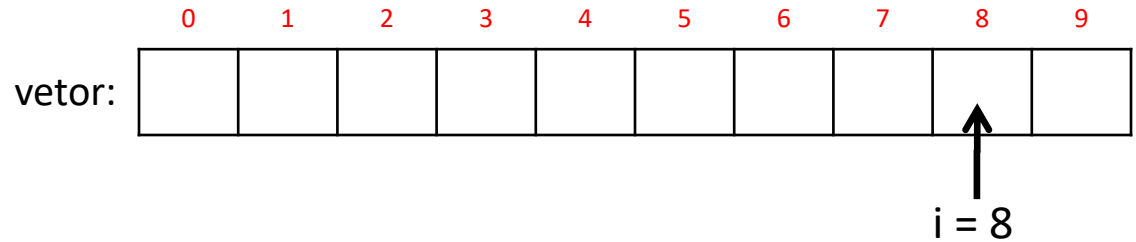
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

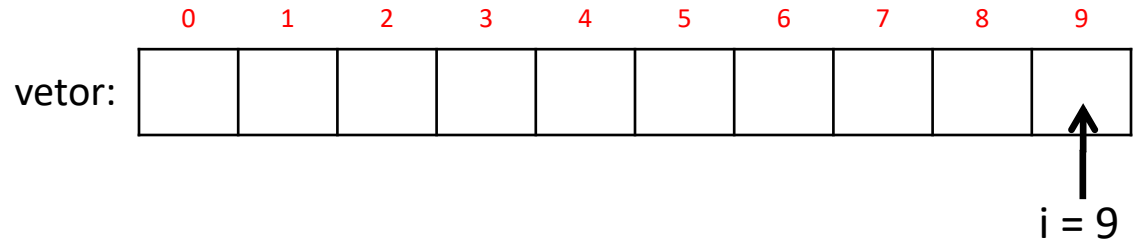
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

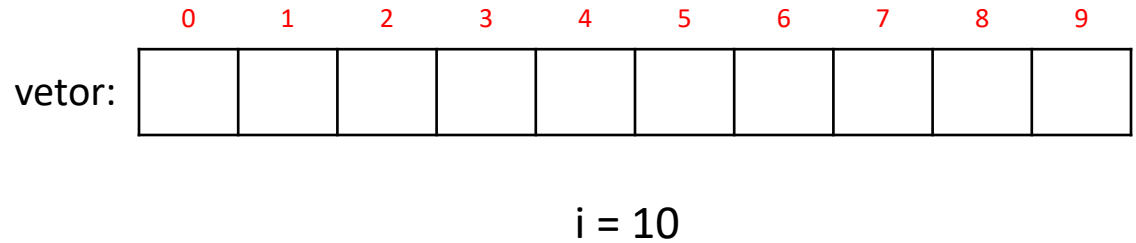
```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - O que faz o programa a seguir ?

```
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        vetor[i];
    }
    return 0;
}
```



Vetores

- Como manipular eficientemente vetores?
 - Lendo valores via teclado...

Vetores

- Como manipular eficientemente vetores?
 - Lendo valores via teclado...

```
#include <iostream>
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        std::cin >> vetor[i];
    }
    return 0;
}
```

Vetores

- Como manipular eficientemente vetores?
 - Imprimindo valores na tela ...

Vetores

- Como manipular eficientemente vetores?
 - Imprimindo valores na tela ...

```
#include <iostream>
int main()
{
    int i, vetor[10];
    for(i=0; i<10; i++)
    {
        std::cout << vetor[i] << std::endl;
    }
    return 0;
}
```

Vetores

- Como manipular eficientemente vetores?
 - Somando os valores de cada índice ...

Vetores

- Como manipular eficientemente vetores?
 - Somando os valores de cada índice ...

```
#include <iostream>
int main()
{
    int i, soma=0, vetor[10];
    for(i=0; i<10; i++)
    {
        soma = soma + vetor[i];
    }
    return 0;
}
```

Vetores

- Cuidados
 - **Jamais** ultrapasse os limites do vetor.
 - Linguagem C++ não trata a extrapolação dos limites do vetor. Deste modo, caso você ultrapasse os limites o comportamento do programa é inesperado.
 - O tamanho do vetor deve ser uma constante.
 - Não é previsto no C++ ANSI uma variável na alocação do vetor. Caso deseje escolher o tamanho do vetor em tempo de execução, deve-se usar alocação dinâmica.

Vetores

- Passagem de Parâmetros
 - Por padrão na Linguagem C++, todo vetor é automaticamente passado por referência.
 - Atenção:
 - É necessário explicitar o recebimento de um ponteiro no cabeçalho da função.

Vetores

- Passagem de Parâmetros (exemplo)
 - Suponha a função `exVet()` que recebe um vetor de inteiros e um inteiro.

- Invocando `exVet()`

`exVet (vetor, n);`

Não é necessário explicitar a passagem de um endereço.

- Cabeçalho da função `exVet()`

`void exVet (int * v, int n);`

Entretanto é necessário explicitar o recebimento de um ponteiro.

Matrizes

- Declaração:

- Sintaxe:

- `<tipo> nome_da_variavel[<tamanho>][<tamanho>;`

- Exemplo: declaração de uma matriz de inteiros chamado Mat com 12 elementos.

`int mat[3][4];`

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |

- Exemplo: declaração de uma matriz de flutuantes chamado dados com 1050 elementos, em 30 linhas e 35 colunas.

`float dados[30][35];`

Matrizes

- Acessando os valores de uma matriz

Mat:

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |

Matrizes

- Acessando os valores de uma matriz
 - Sintaxe:
nome_da_variavel [**<linha>**][**<coluna>**] = valor;

Mat:

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | | | |
| 1 | | | | |
| 2 | | | | |

Matrizes

- Acessando os valores de uma matriz
 - Sintaxe:
`nome_da_variavel [<linha>][<coluna>] = valor;`
 - Exemplo: atribuir 5 a matriz Mat na linha 1 coluna 2.
`Mat[1][2] = 5;`

Mat:

| | | | | |
|---|---|---|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | | | |
| 1 | | | 5 | |
| 2 | | | | |

Matrizes

- Acessando os valores de uma matriz

- Sintaxe:

- `nome_da_variavel [<linha>][<coluna>] = valor;`

- Exemplo: atribuir 5 a matriz Mat na linha 1 coluna 2.

- `Mat[1][2] = 5;`

- Exemplo: atribuir -7 a matriz Mat na linha 0 coluna 1:

- `Mat[0][1] = -7;`

Mat:

| | | | | |
|---|---|----|---|---|
| | 0 | 1 | 2 | 3 |
| 0 | | -7 | | |
| 1 | | | 5 | |
| 2 | | | | |

Matrizes

- Lendo e imprimindo valores a partir de uma matriz:
 - Igual se faz com um vetor, porém jamais se esqueça de informar os índices linha e coluna.
 - Leitura:
 - Ler um flutuante via teclado e armazenar na linha 3, coluna 4 da matriz valores.

```
std::cin >> valores[3][4];
```

- Impressão:
 - Imprimir na tela o valor armazenado na linha 5 coluna 7 da matriz val.

```
std::cout << val[5][7] << std::endl;
```

Matrizes

- Como manipular eficientemente matrizes?

Matrizes

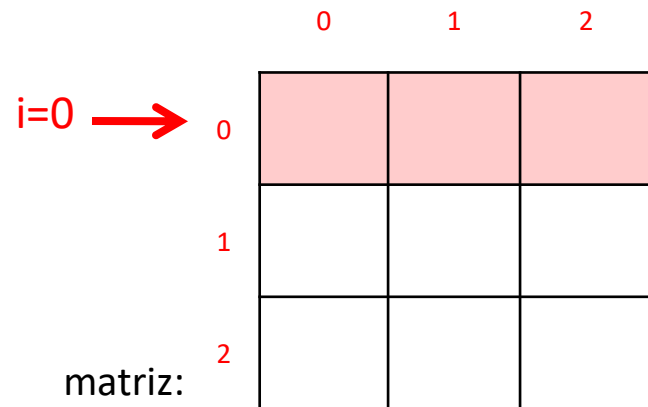
- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

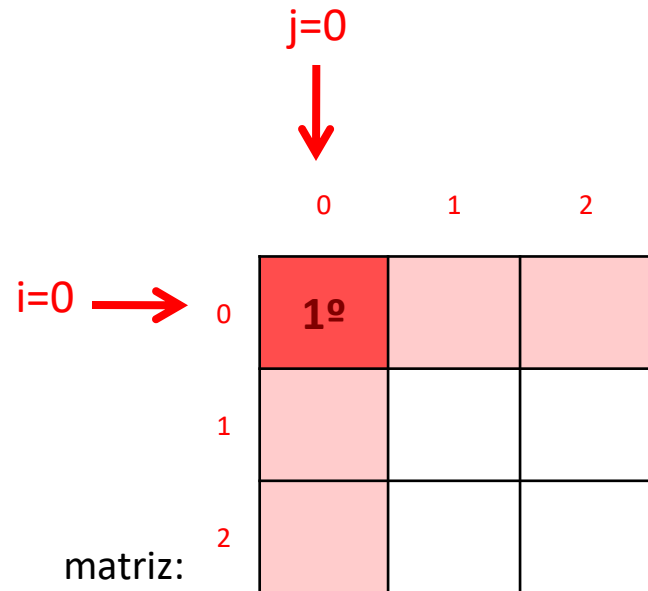
```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

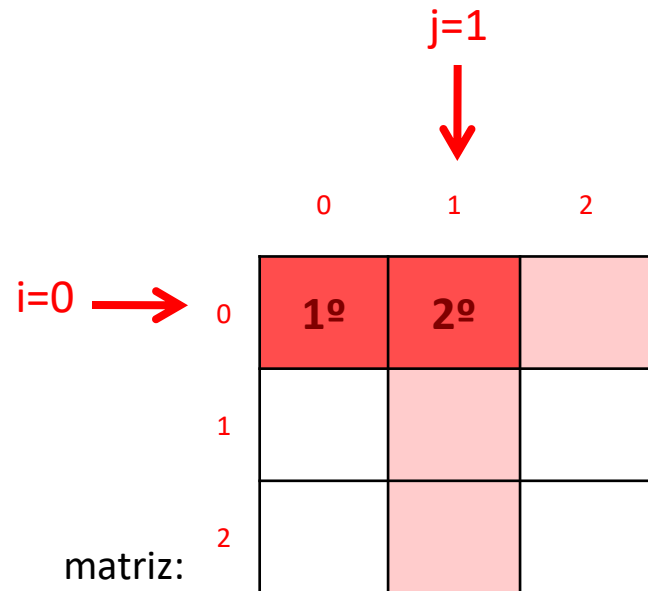
```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

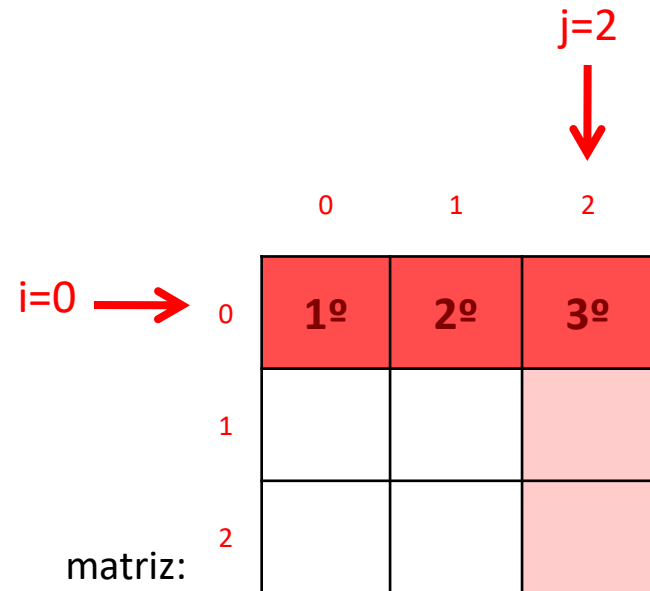
```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

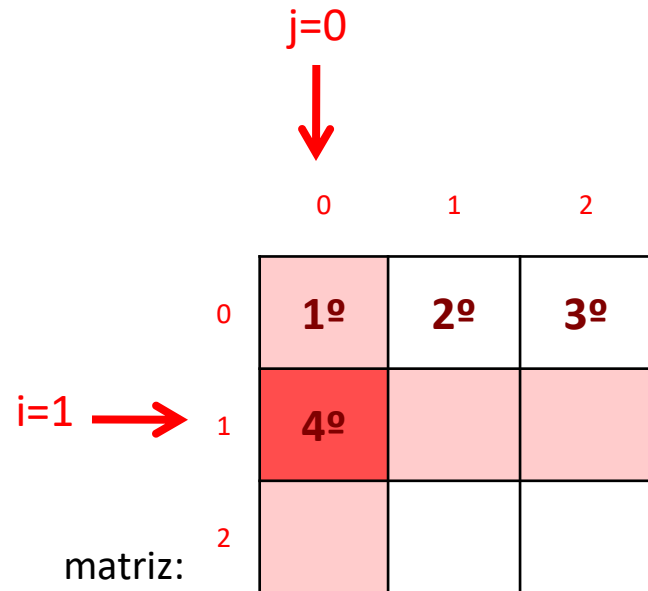
```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

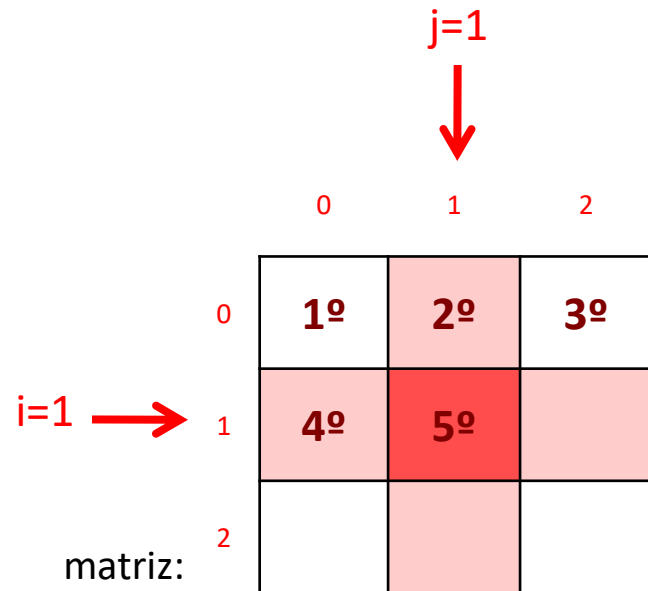
```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```



Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

Diagram illustrating a 3x3 matrix access. The matrix is shown with indices *i* and *j* ranging from 0 to 2. The current state is *i*=1 and *j*=2, indicated by red arrows. The matrix cells are labeled with their values (1º to 6º) and shaded in light red.

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | | | |

matriz:

Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

| | | | |
|---|----------|----|----|
| | j=0 ↓ | | |
| | 0 | 1 | 2 |
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | 7º | | |

i=2 →
matriz:

Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

A 3x3 matrix is shown with rows indexed 0 to 2 and columns indexed 0 to 2. The cell at row 2, column 1 (containing '8º') is highlighted in a darker red. The entire row 2 and column 1 are shaded in a lighter red. A red arrow labeled 'j=1' points down to column 1, and a red arrow labeled 'i=2' points right to row 2. The label 'matriz:' is placed below the row 2 arrow.

| | | | |
|---|----|----|----|
| | 0 | 1 | 2 |
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | 7º | 8º | |

Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | 7º | 8º | 9º |

Matrizes

- Como manipular eficientemente matrizes?
 - O que faz o programa a seguir ?

```
1. int main()
2. {
3.     int matriz[3][3], i, j;
4.     for(i=0; i<3; i++)
5.     {
6.         for(j=0; j<3; j++)
7.         {
8.             matriz[i][j];
9.         }
10.    }
11.    return 0;
12. }
```

matriz:

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | 7º | 8º | 9º |

i=3

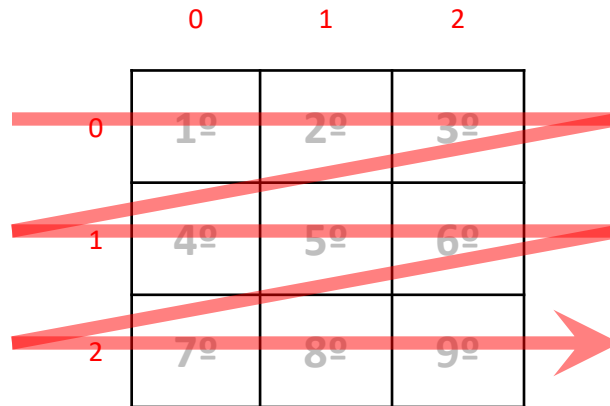
Matrizes

- Ordem de visita aos elementos
 - No código anterior a ordem de visitação foi essa.

| | 0 | 1 | 2 |
|---|----|----|----|
| 0 | 1º | 2º | 3º |
| 1 | 4º | 5º | 6º |
| 2 | 7º | 8º | 9º |

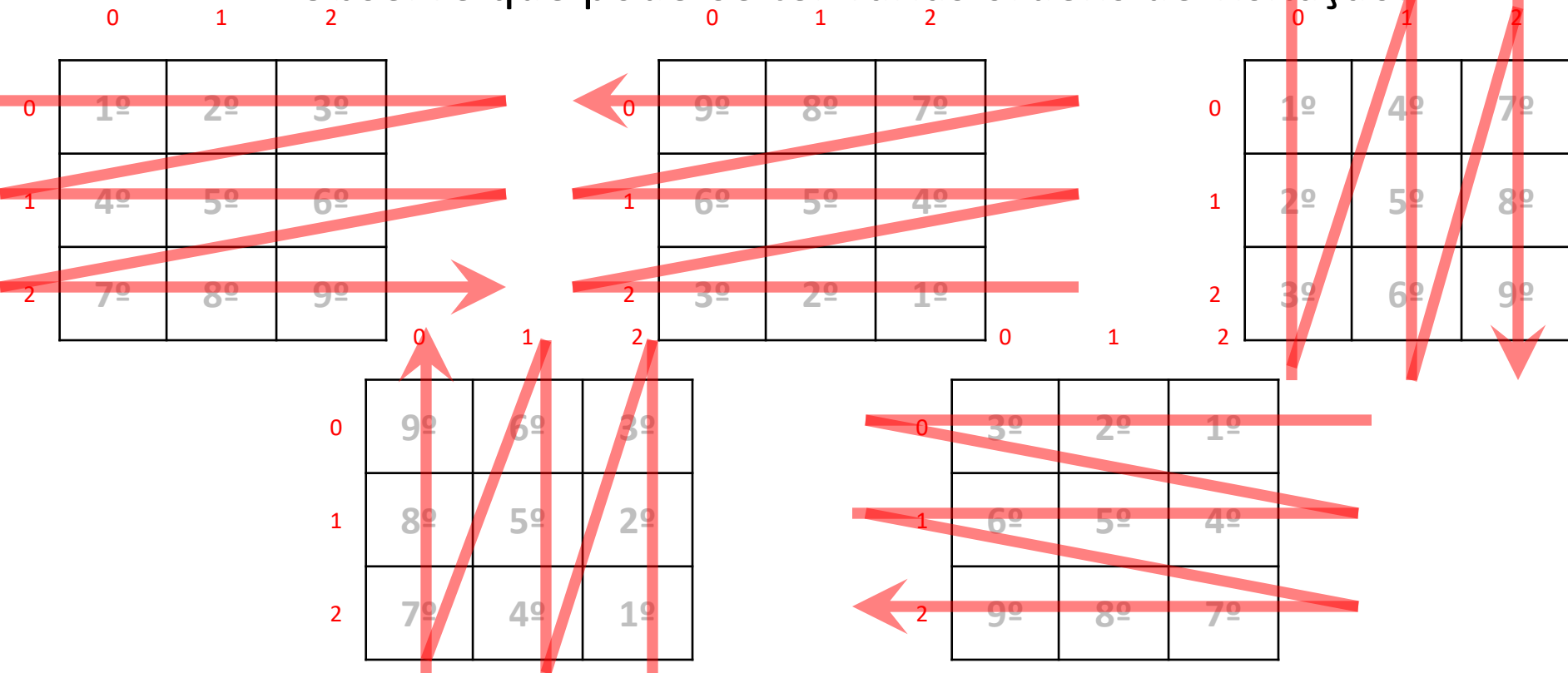
Matrizes

- Ordem de visita aos elementos
 - No código anterior a ordem de visitação foi essa.



Matrizes

- Ordem de visita aos elementos
 - No código anterior a ordem de visitação foi essa.
 - Observe que pode-se ter várias ordens de visitação.



Matrizes

- Cuidados
 - **Jamais** ultrapasse os limites da matriz.
 - Linguagem C++ não trata a extrapolação dos limites da matriz. Deste modo, caso você ultrapasse os limites o comportamento do programa é inesperado.
 - O tamanho da matriz devem ser constantes.
 - Não é previsto no C++ ANSI variáveis na alocação da matriz. Caso deseje escolher a quantidade de linhas e colunas da matriz em tempo de execução, deve-se usar alocação dinâmica.

Matrizes

- Passagem de Parâmetros (exemplo)
 - Suponha a função `exMat()` que recebe uma matriz de inteiros.

- Invocando `exMat()`

`exMat (matriz, n, m);`

Não é necessário explicitar a passagem de um endereço

- Cabeçalho da função `exMat()`

`void exMat (int **m, int n, int m);`

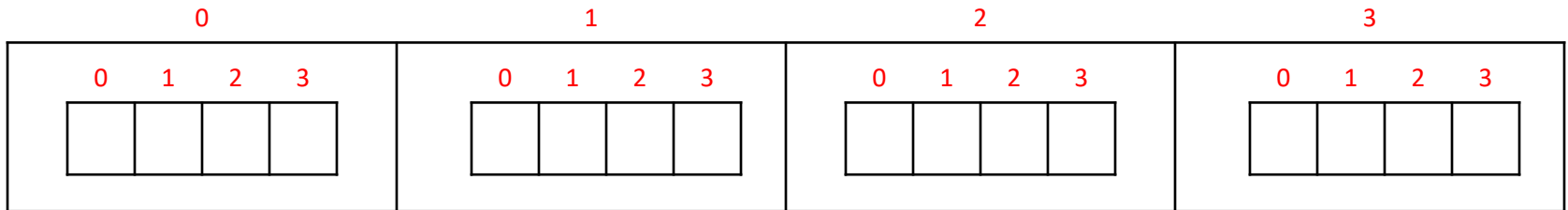
É necessário explicitar o recebimento de um ponteiro de ponteiro.

Matrizes

- Internamente
 - Uma matriz pode ser entendida como um vetor unidimensional, onde cada item deste vetor é outro vetor unidimensional.

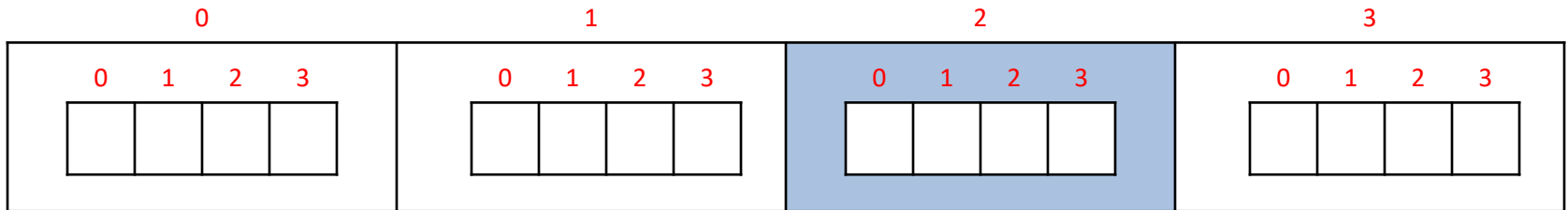
Matrizes

- Internamente
 - A matriz pode ser vista como um vetor de vetores.



Matrizes

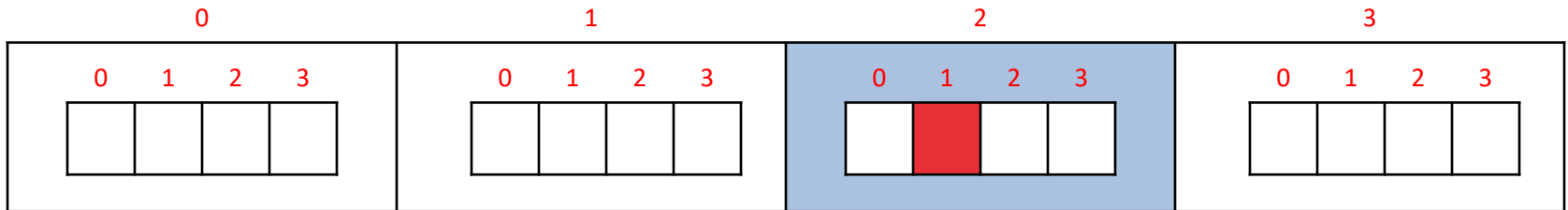
- Internamente
 - A matriz pode ser vista como um vetor de vetores.



- Ao referenciar a matriz na linha 2 coluna 1:
 - a linha resolve o índice do primeiro vetor.

Matrizes

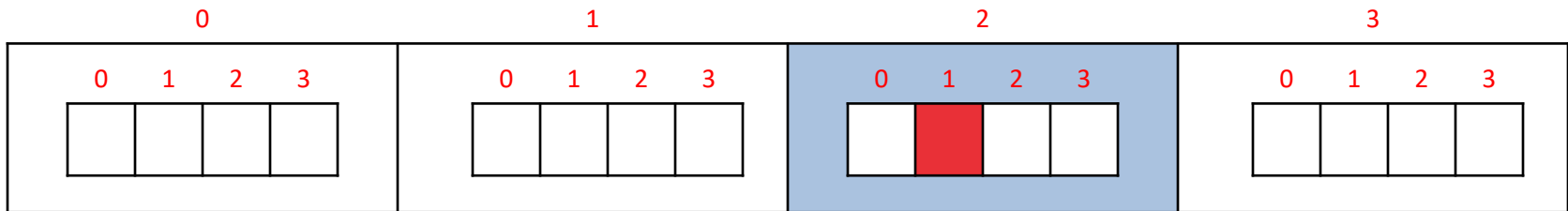
- Internamente
 - A matriz pode ser vista como um vetor de vetores.



- Ao referenciar a matriz na linha 2 coluna 1:
 - a linha resolve o índice do primeiro vetor.
 - Enquanto a coluna resolve o índice do segundo vetor.

Matrizes

- Internamente
 - A matriz pode ser vista como um vetor de vetores.



- Ao referenciar a matriz na linha 2 coluna 1:
 - a linha resolve o índice do primeiro vetor.
 - Enquanto a coluna resolve o índice do segundo vetor.
- Desta forma podemos tratar a matriz como um conjunto de vetores.

Matrizes

- Internamente
 - Desta forma podemos tratar a matriz como um conjunto de vetores.
 - Para isto, basta omitirmos a segunda dimensão da matriz e então teríamos um vetor.
 - Deste modo, ao referenciar `Mat[1]`, temos:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | | | | | | | |
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |

O vetor formado pela linha 1 da matriz Mat

Matrizes

- Internamente
 - Desta forma podemos tratar a matriz como um conjunto de vetores.
 - Para isto, basta omitirmos a segunda dimensão da matriz e então teríamos um vetor.
 - Deste modo, ao referenciar `Mat[3]`, temos:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | | | | | | | |
| 1 | | | | | | | |
| 2 | | | | | | | |
| 3 | | | | | | | |
| 4 | | | | | | | |

O vetor formado pela linha 3 da matriz Mat

Matrizes

- Internamente
 - Desta forma podemos tratar a matriz como um conjunto de vetores.
 - Para isto, basta omitirmos a segunda dimensão da matriz e então teríamos um vetor.
 - Desta maneira, pode-se obter este vetor e tratá-lo como um vetor tradicional.
 - Vejamos um exemplo ...

Matrizes

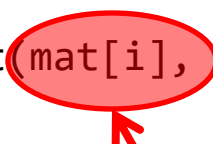
- Usando uma matriz como um conjunto de vetores.
 - Agora, usando a função somaVet() feita anteriormente, faça um programa que exiba na tela a somatória de cada linha da matriz a seguir:

```
#include<iostream>
int main()
{
    int mat[10][50], i;
    ...
    for(i=0; i<10; i++)
    {
        std::cout << "Soma linha " << i << ": " << somaVet(mat[i], 50) << std::endl;
    }
    return 0;
}
```

Matrizes

- Usando uma matriz como um conjunto de vetores.
 - Agora, usando a função somaVet() feita anteriormente, faça um programa que exiba na tela a somatória de cada linha da matriz a seguir:

```
#include<iostream>
int main()
{
    int mat[10][50], i;
    ...
    for(i=0; i<10; i++)
    {
        std::cout << "Soma linha " << i << ": " << somaVet(mat[i], 50) << std::endl;
    }
    return 0;
}
```



Aqui espera-se um vetor...
E isso é um vetor...