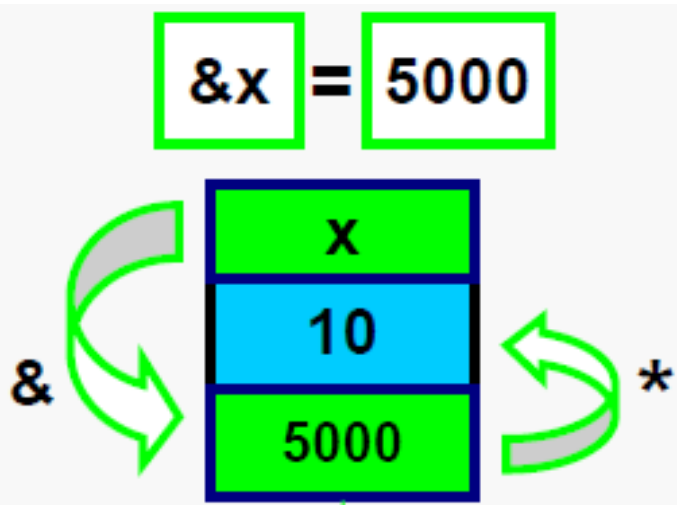
BCC 201 - Introdução à Programação I

Ponteiros

Guillermo Cámara-Chávez
UFOP

Ponteiros ... I

```c
int main()
{
    int x;
    x = 10
    printf("Conteudo de x: %d \n", x);
    printf("Endereco de x: %p \n", &x);
    return 0;
}
```

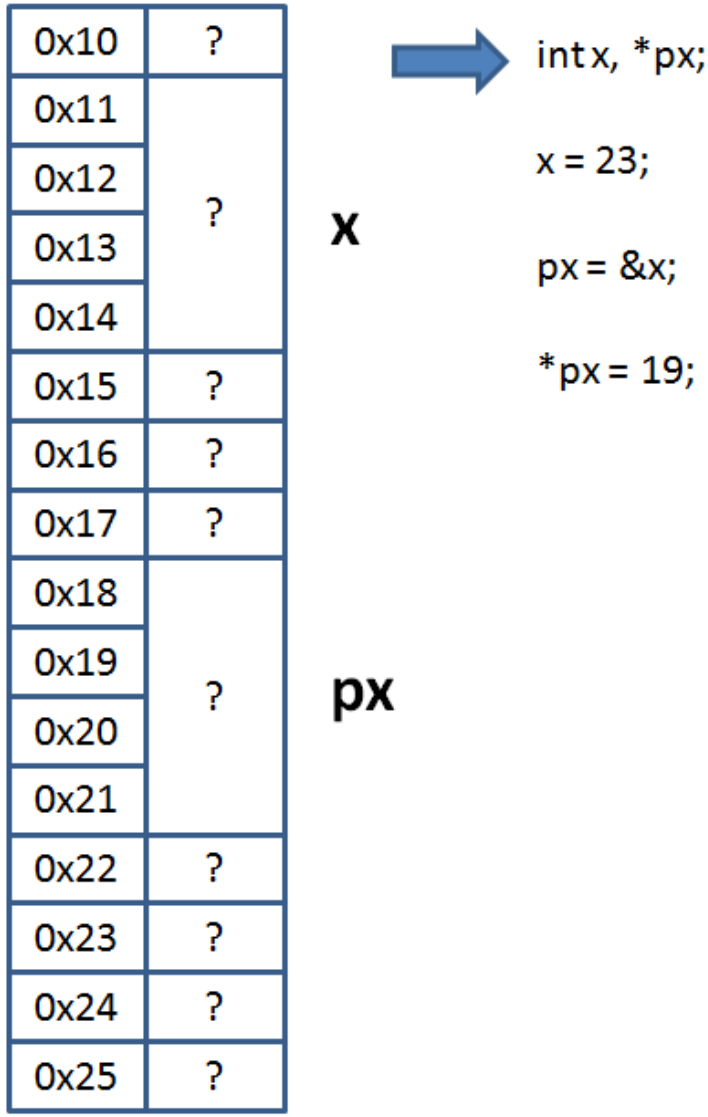


Ponteiros ... II

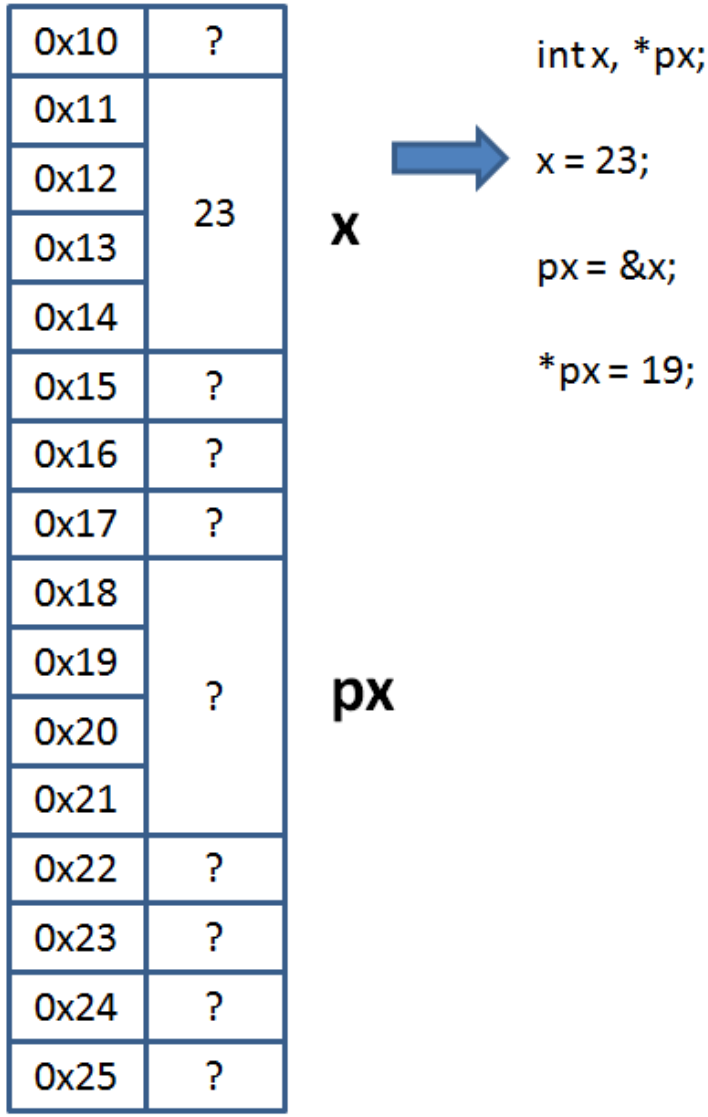
0x10	?
0x11	?
0x12	?
0x13	?
0x14	?
0x15	?
0x16	?
0x17	?
0x18	?
0x19	?
0x20	?
0x21	?
0x22	?
0x23	?
0x24	?
0x25	?

```
int x, *px;

x = 23;

px = &x;

*px = 19;
```

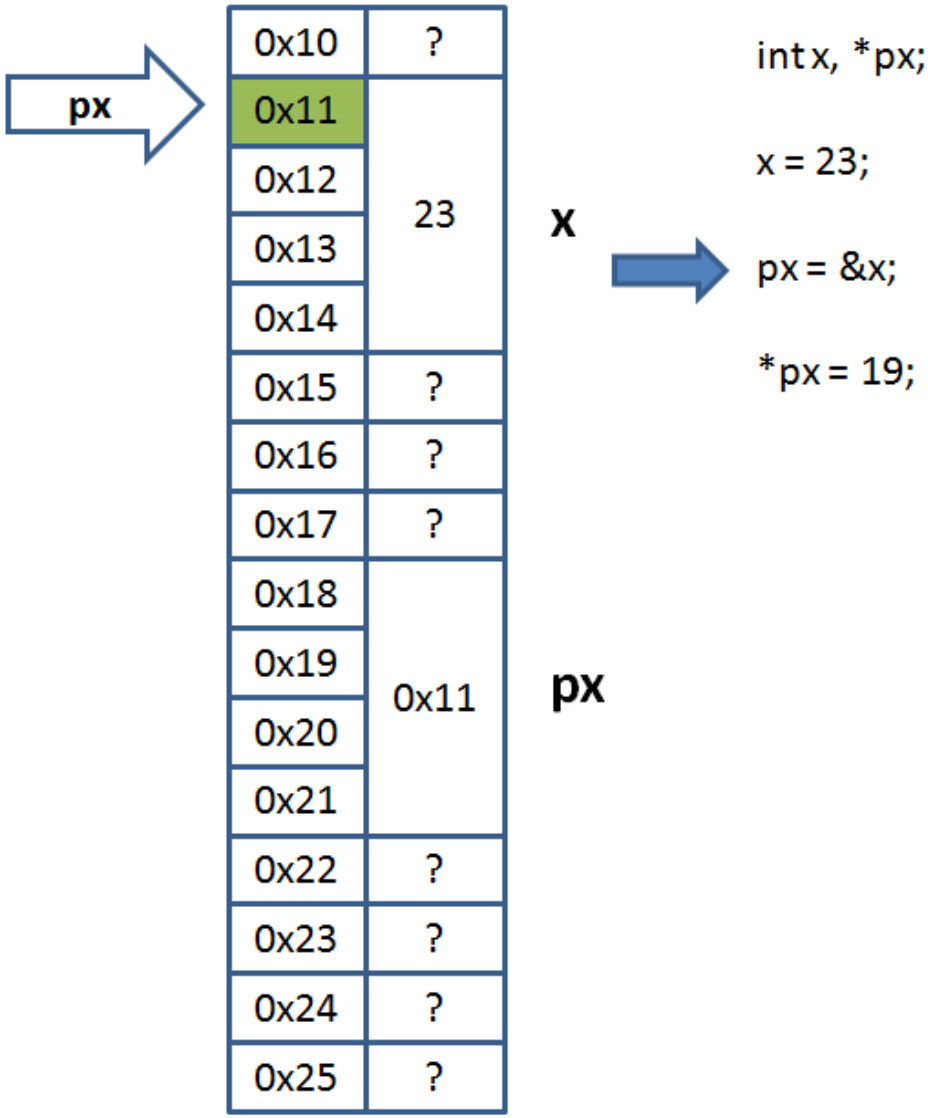
Ponteiros ... III



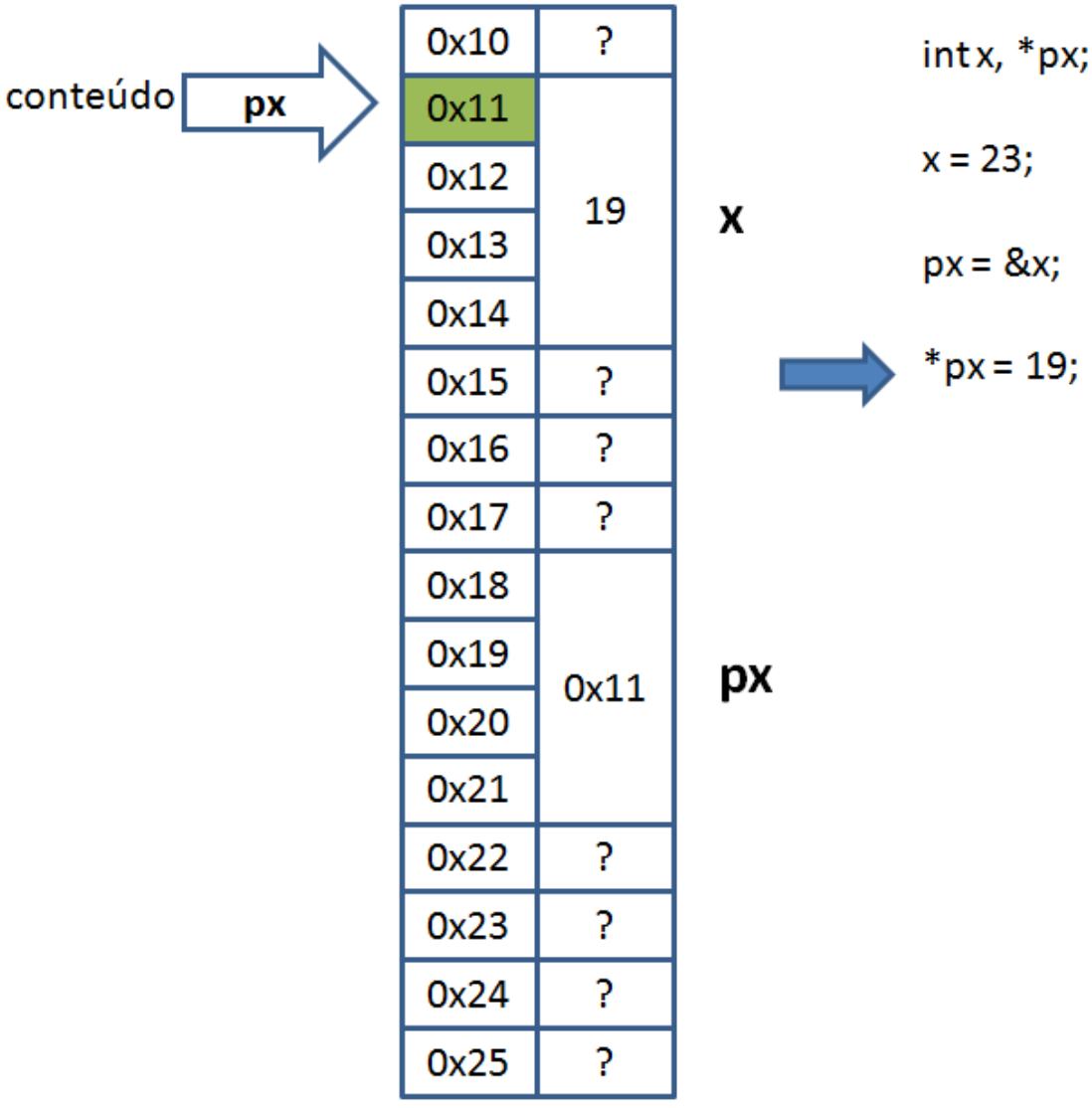
Ponteiros ... IV



Ponteiros ... V


px
0x10
?
0x11
0x12
0x13
0x14
23
x
0x15
?
0x16
?
0x17
?
0x18
0x19
0x20
0x21
0x11
px
0x22
?
0x23
?
0x24
?
0x25
?
int x, *px;
x = 23;
px = &x;
*px = 19;

Ponteiros ... VI



Ponteiros ... VII

0x12	?
0x16	?
0x20	?
0x24	?
0x28	?
0x32	?
0x36	?
0x40	?
0x44	?
0x48	?
0x52	?
0x56	?
0x60	?
0x64	?
0x68	?
0x72	?

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... VIII

0x12	?
0x16	?
0x20	?
0x24	?
0x28	?
0x32	?
0x36	?
0x40	?
0x44	?
0x48	?
0x52	?
0x56	?
0x60	100
0x64	200
0x68	?
0x72	?

x
y
main

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

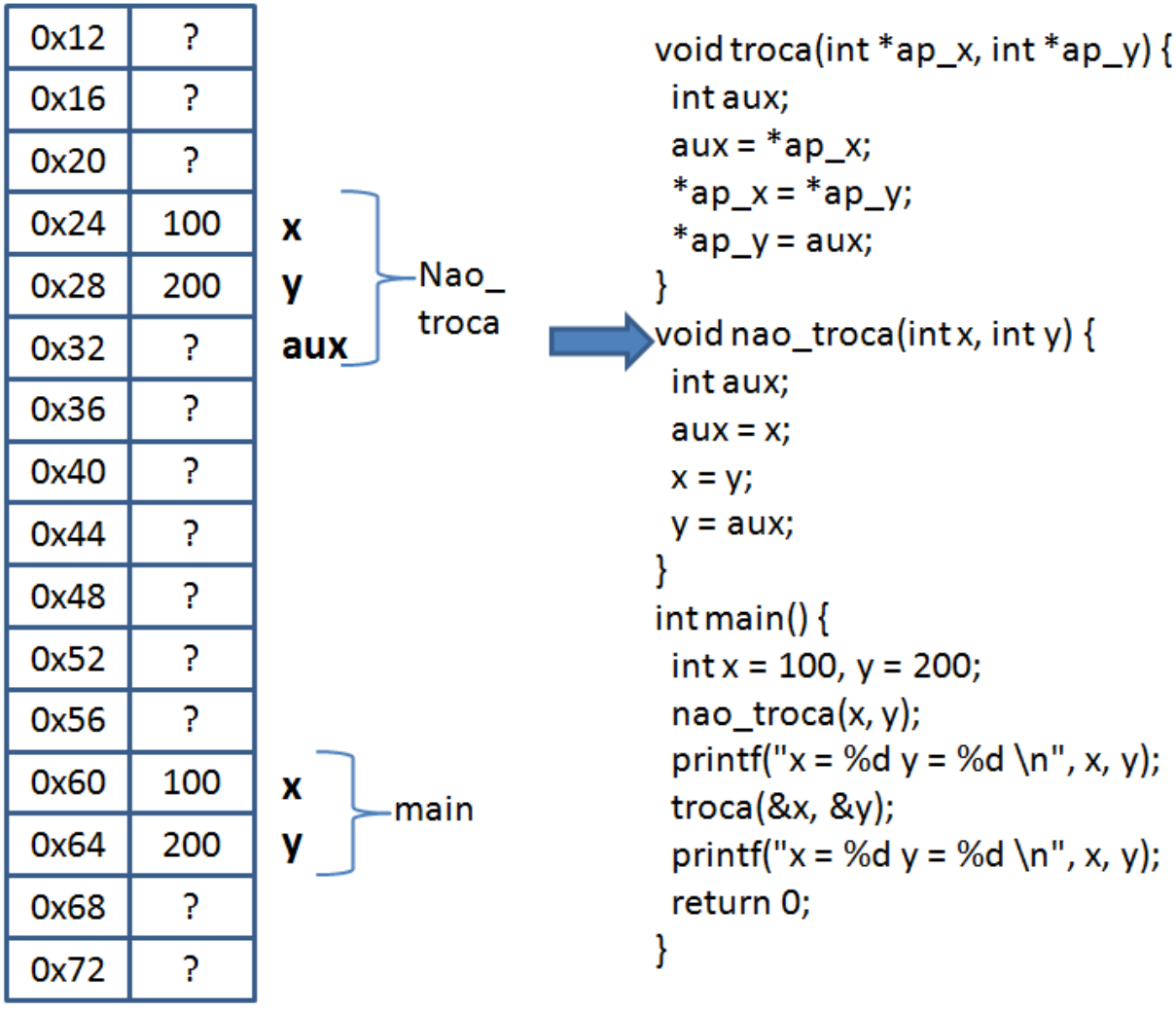
Ponteiros ... IX

0x12	?
0x16	?
0x20	?
0x24	?
0x28	?
0x32	?
0x36	?
0x40	?
0x44	?
0x48	?
0x52	?
0x56	?
0x60	100
0x64	200
0x68	?
0x72	?

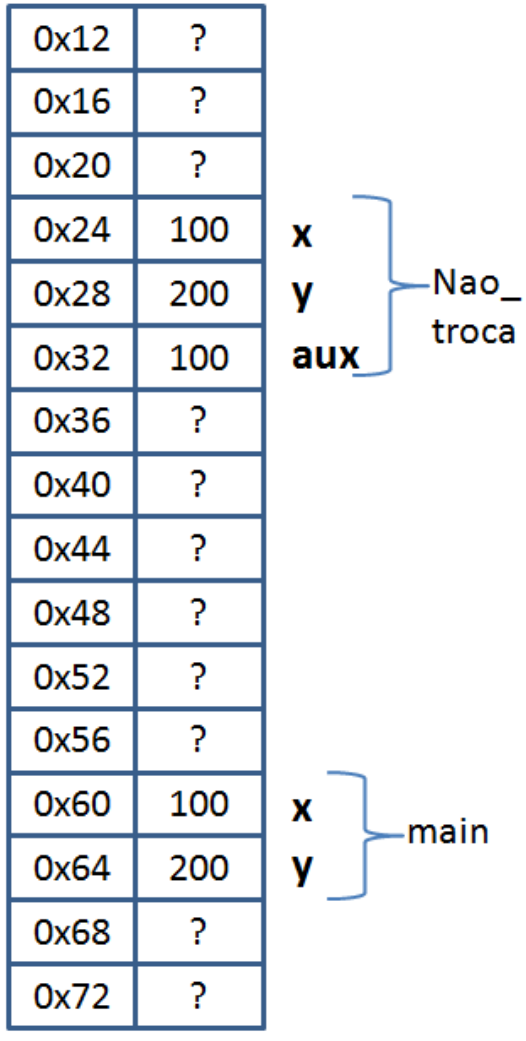
x
y
main

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... X

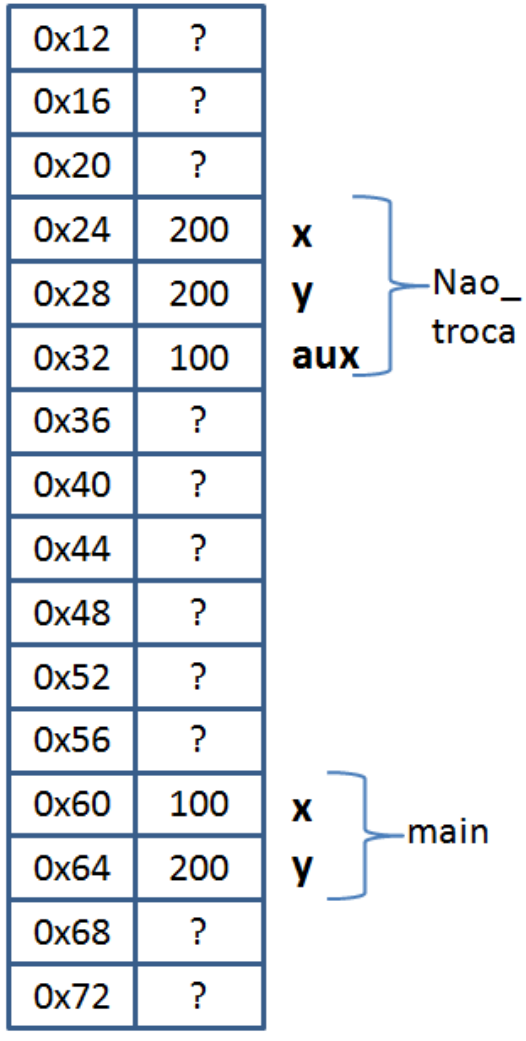

0x12 ?
0x16 ?
0x20 ?
0x24 100
0x28 200
0x32 ?
0x36 ?
0x40 ?
0x44 ?
0x48 ?
0x52 ?
0x56 ?
0x60 100
0x64 200
0x68 ?
0x72 ?
x
y
aux
Nao_ troca
x
y
main
void troca(int *ap_x, int *ap_y) {
int aux;
aux = *ap_x;
*ap_x = *ap_y;
*ap_y = aux;
}
void nao_troca(int x, int y) {
int aux;
aux = x;
x = y;
y = aux;
}
int main() {
int x = 100, y = 200;
nao_troca(x, y);
printf("x = %d y = %d \n", x, y);
troca(&x, &y);
printf("x = %d y = %d \n", x, y);
return 0;
}

Ponteiros ... XI



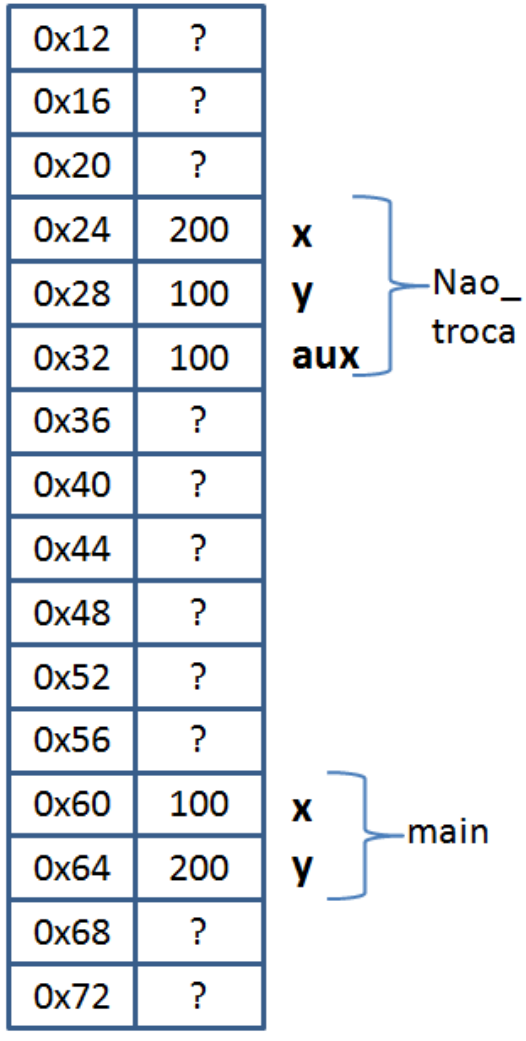

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... XII



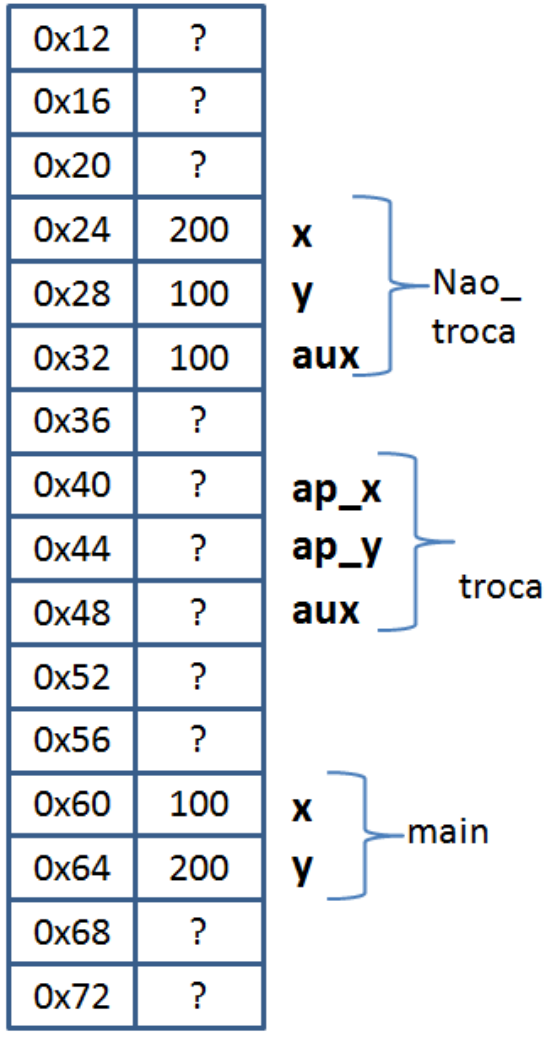

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... XIII



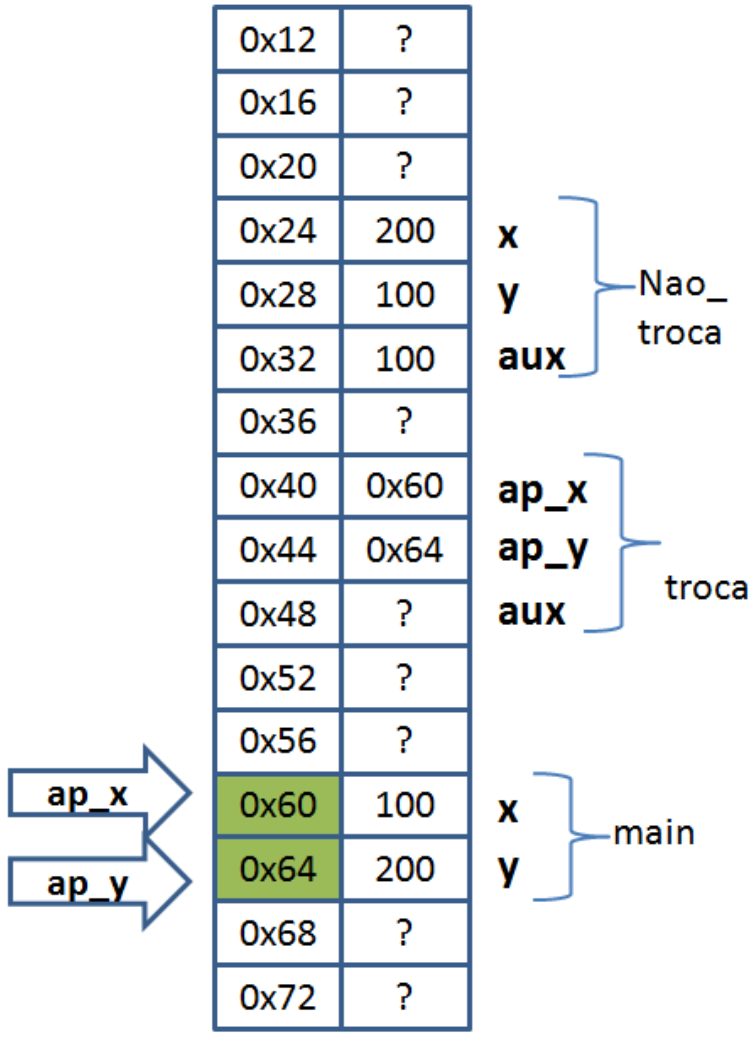

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... XIV



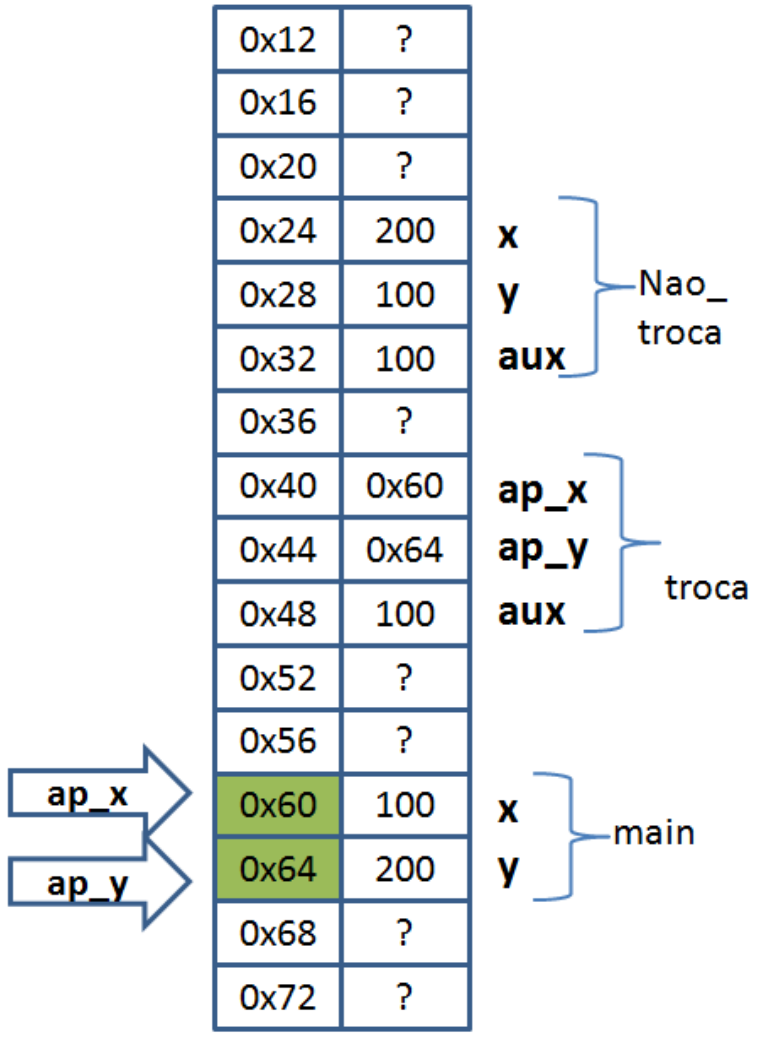

```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

Ponteiros ... XV



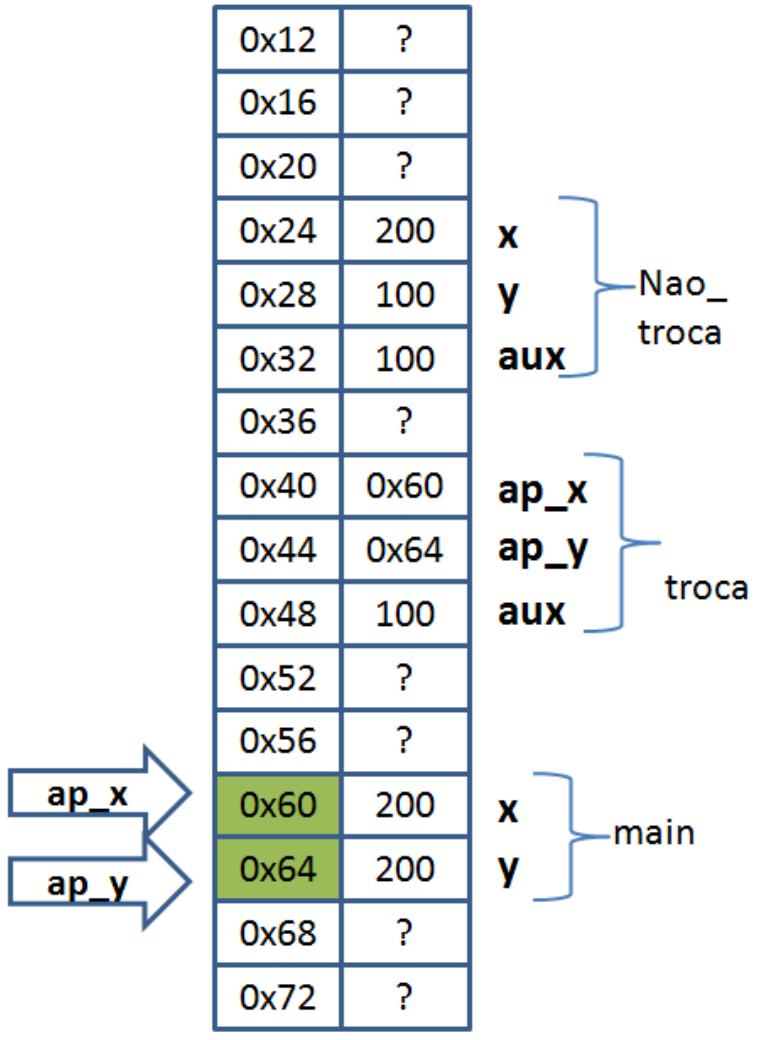

```
void troca(int *ap_x, int *ap_y) {
  int aux;
  aux = *ap_x;
  *ap_x = *ap_y;
  *ap_y = aux;
}
void nao_troca(int x, int y) {
  int aux;
  aux = x;
  x = y;
  y = aux;
}
int main() {
  int x = 100, y = 200;
  nao_troca(x, y);
  printf("x = %d y = %d \n", x, y);
  troca(&x, &y);
  printf("x = %d y = %d \n", x, y);
  return 0;
}
```

Ponteiros ... XVI



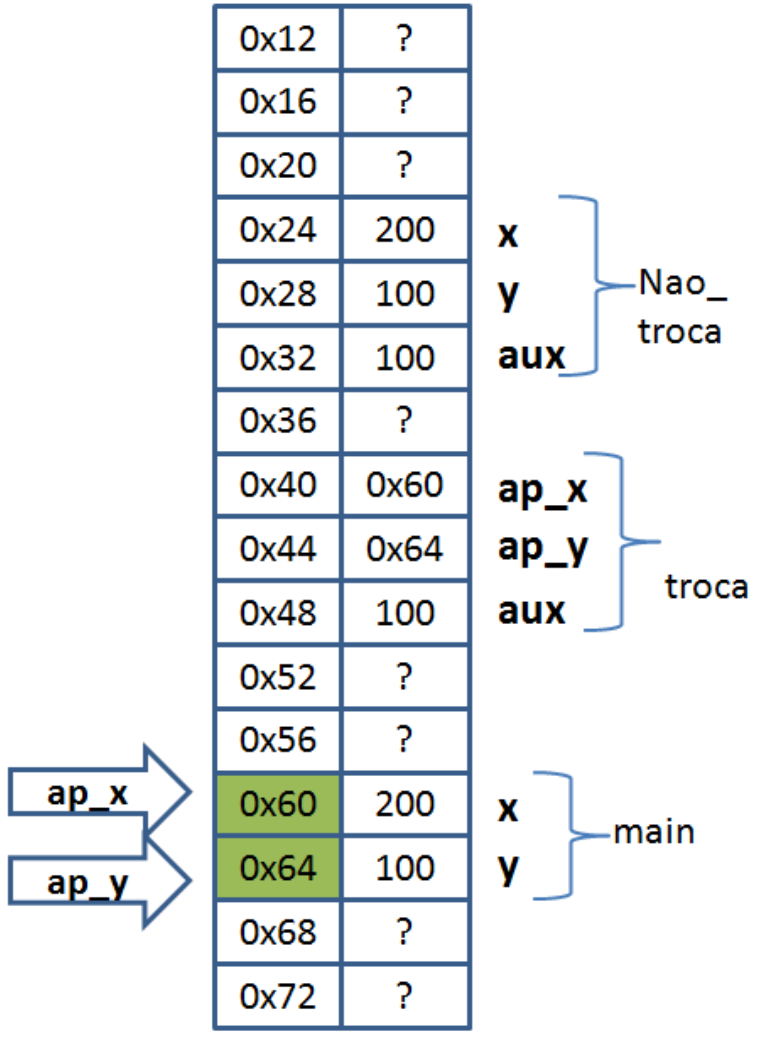

```
void troca(int *ap_x, int *ap_y) {
  int aux;
  aux = *ap_x;
  *ap_x = *ap_y;
  *ap_y = aux;
}
void nao_troca(int x, int y) {
  int aux;
  aux = x;
  x = y;
  y = aux;
}
int main() {
  int x = 100, y = 200;
  nao_troca(x, y);
  printf("x = %d y = %d \n", x, y);
  troca(&x, &y);
  printf("x = %d y = %d \n", x, y);
  return 0;
}
```

Ponteiros ... XVII




```
void troca(int *ap_x, int *ap_y) {
  int aux;
  aux = *ap_x;
  *ap_x = *ap_y;
  *ap_y = aux;
}
void nao_troca(int x, int y) {
  int aux;
  aux = x;
  x = y;
  y = aux;
}
int main() {
  int x = 100, y = 200;
  nao_troca(x, y);
  printf("x = %d y = %d \n", x, y);
  troca(&x, &y);
  printf("x = %d y = %d \n", x, y);
  return 0;
}
```

Ponteiros ... XVIII

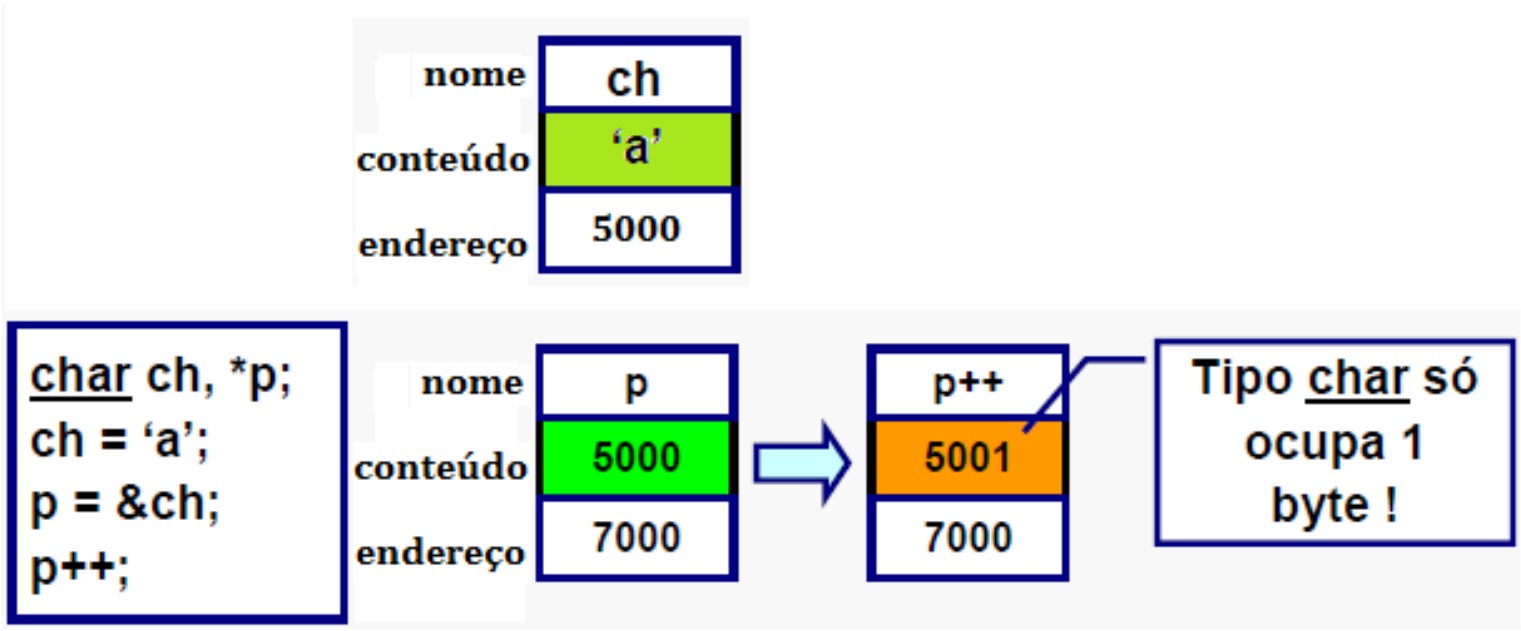



```
void troca(int *ap_x, int *ap_y) {
 int aux;
 aux = *ap_x;
 *ap_x = *ap_y;
 *ap_y = aux;
}
void nao_troca(int x, int y) {
 int aux;
 aux = x;
 x = y;
 y = aux;
}
int main() {
 int x = 100, y = 200;
 nao_troca(x, y);
 printf("x = %d y = %d \n", x, y);
 troca(&x, &y);
 printf("x = %d y = %d \n", x, y);
 return 0;
}
```

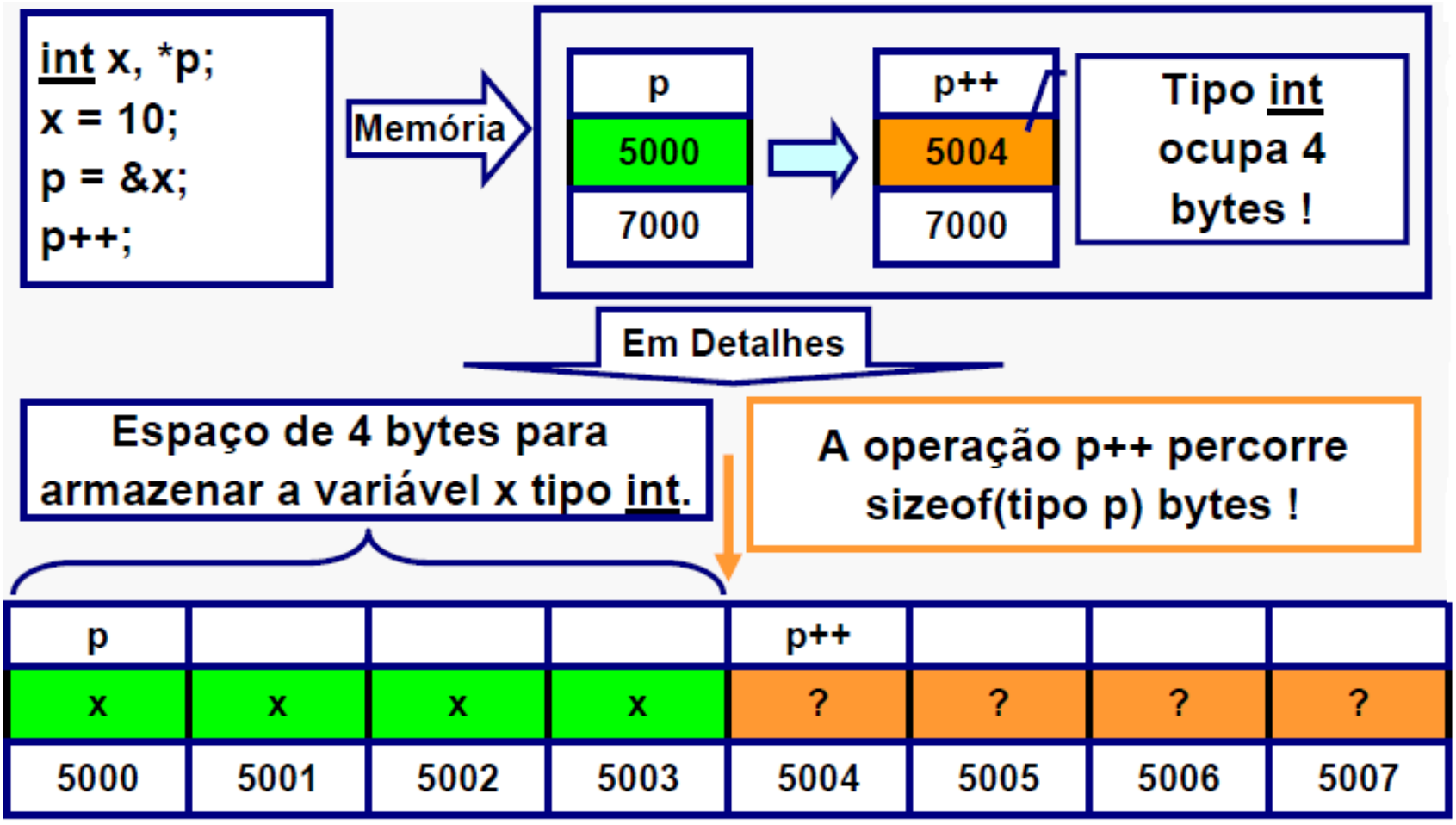
Aritmética de Ponteiros I

- Uma variável do tipo ponteiro está sempre associada a um tipo
- Um ponteiro para um dado tipo t endereça o número de bytes que esse tipo t ocupa na memória, i.e., endereça **sizeof(t)** bytes.
- Se um ponteiro para uma variável do tipo t for incrementada através do operador $++$, automáticamente este ponteiro passará a ter o valor **x + sizeof(t)**

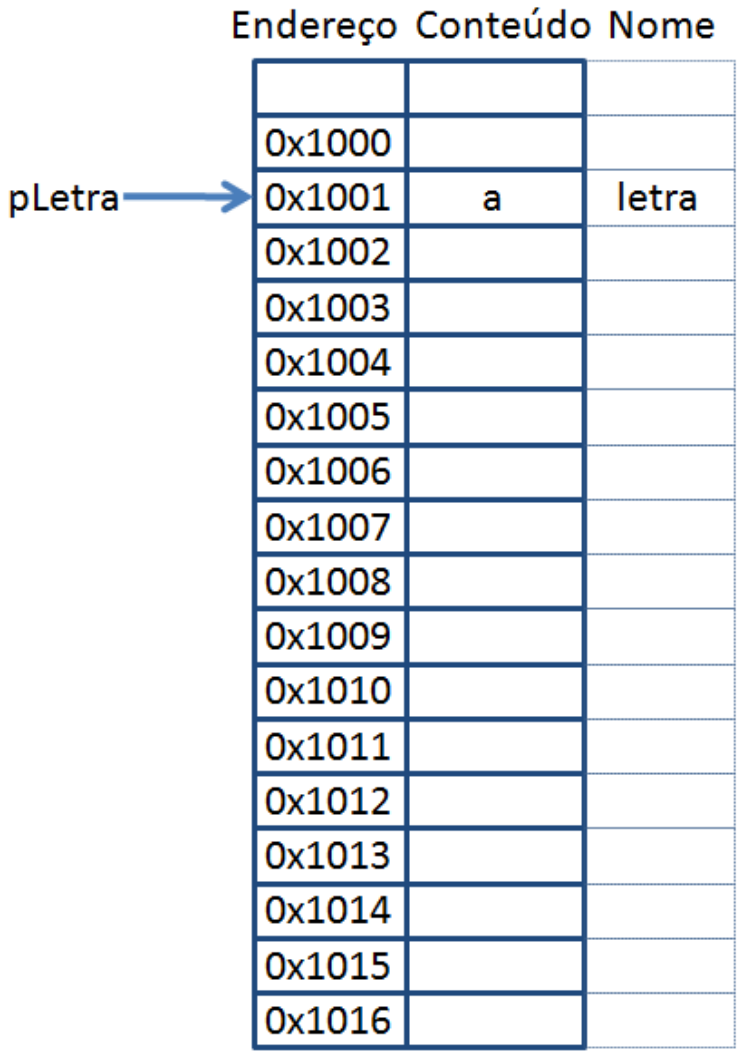
Aritmética de Ponteiros II



Aritmética de Ponteiros III

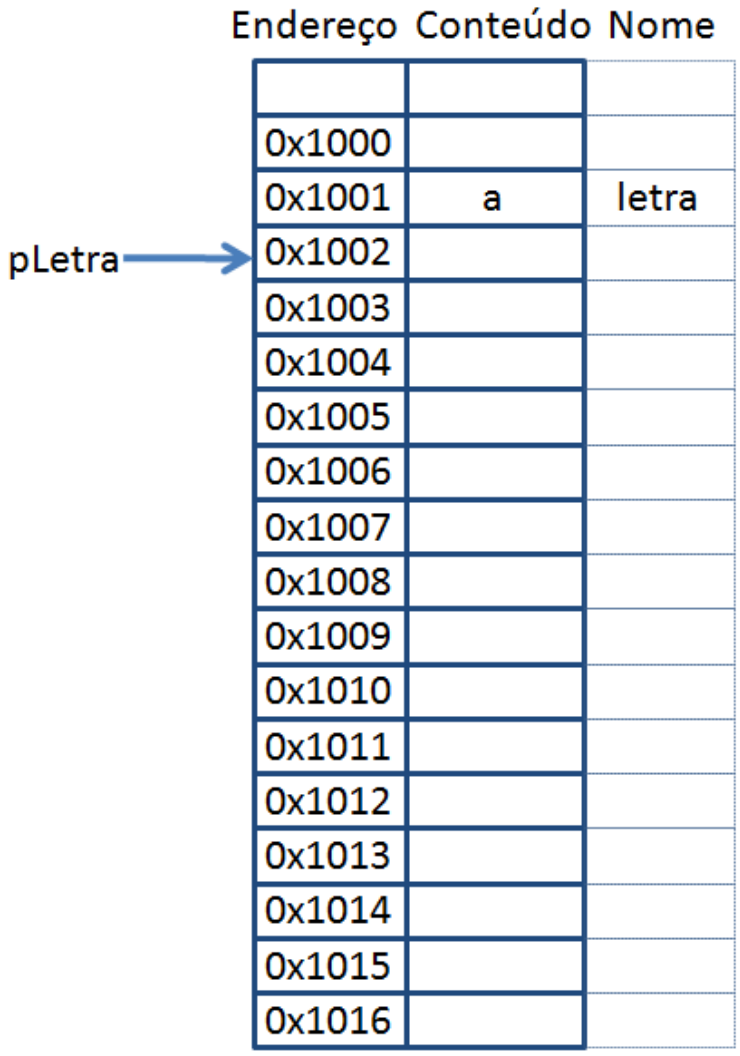


Aritmética de Ponteiros IV



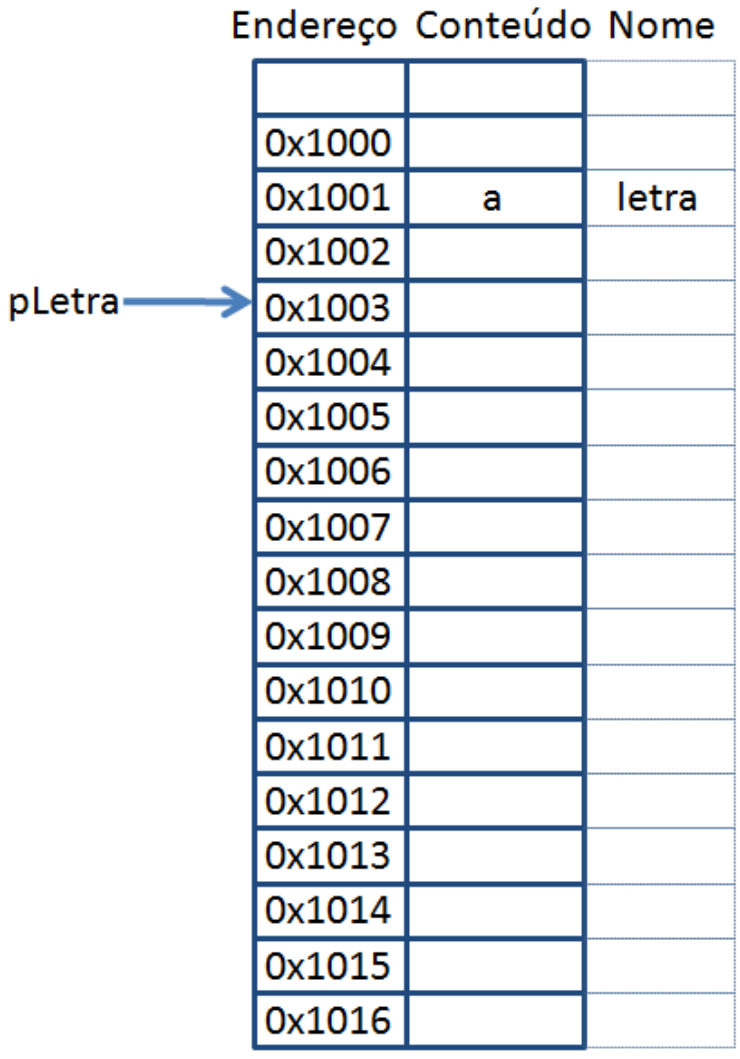

```
int main() {
  char* pChar, letra = 'a';
   pLetra = &letra;
   ...
   pLetra++;
  return 0;
}
```

Aritmética de Ponteiros V



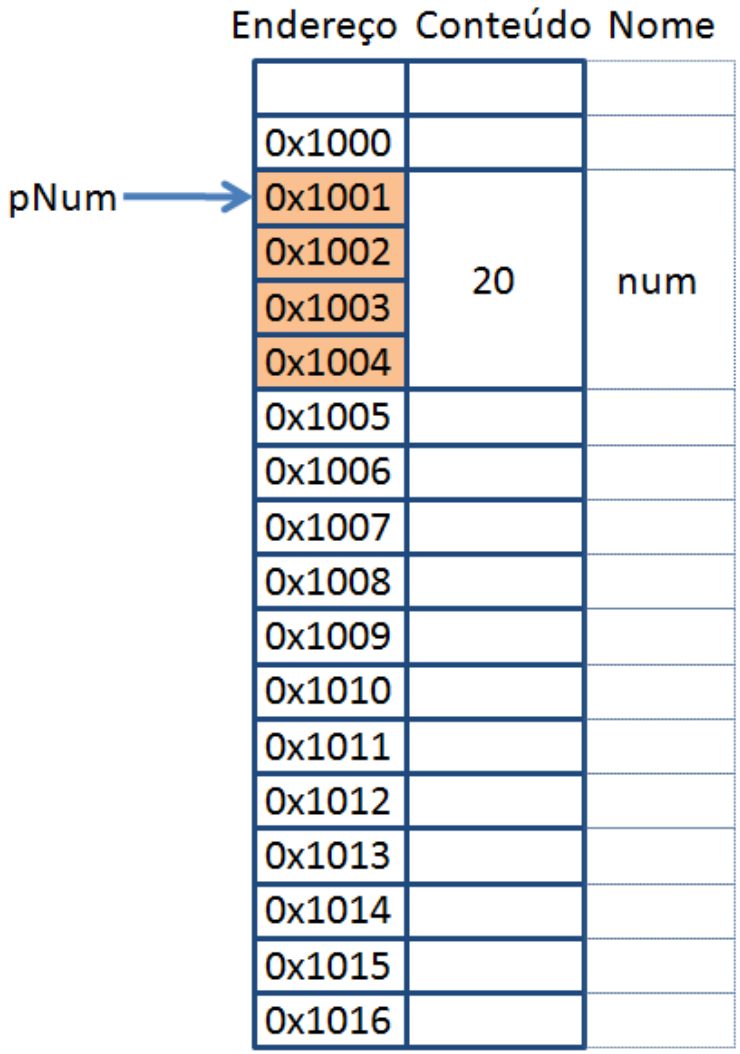

```
int main() {
  char* pChar, letra = 'a';
   pLetra = &letra;
   ...
   pLetra++;
   pLetra++;
  return 0;
}
```

Aritmética de Ponteiros VI



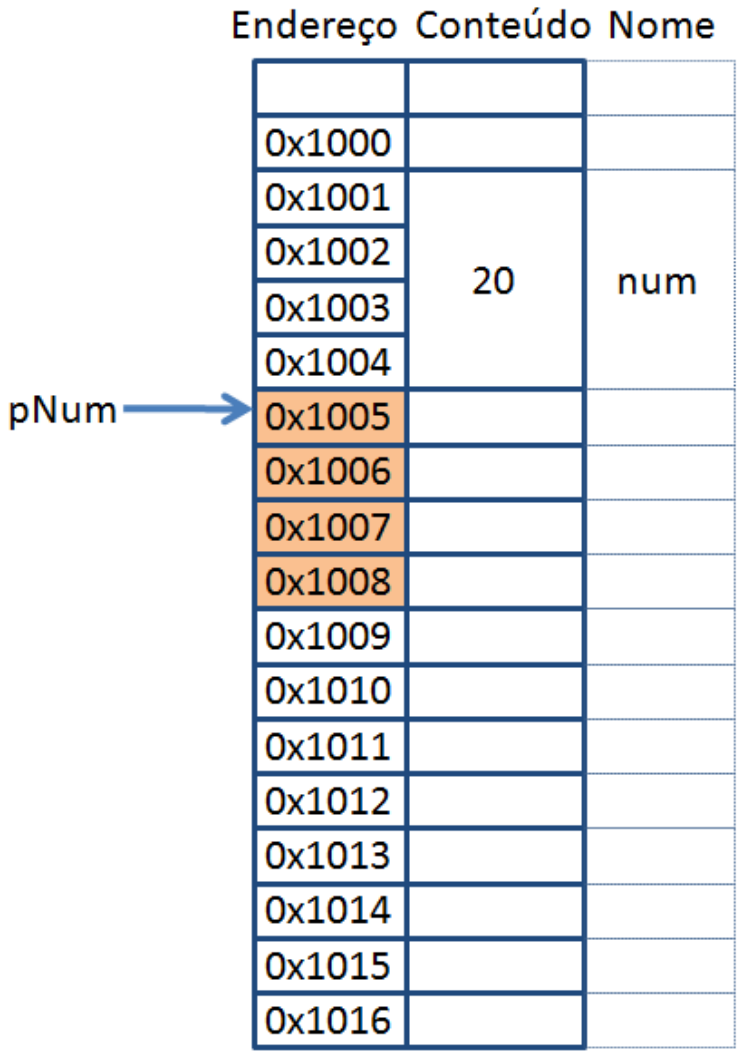

```
int main() {
  char* pChar, letra = 'a';
   pLetra = &letra;
   ...
   pLetra++;
   pLetra++;
  return 0;
}
```

Aritmética de Ponteiros VII



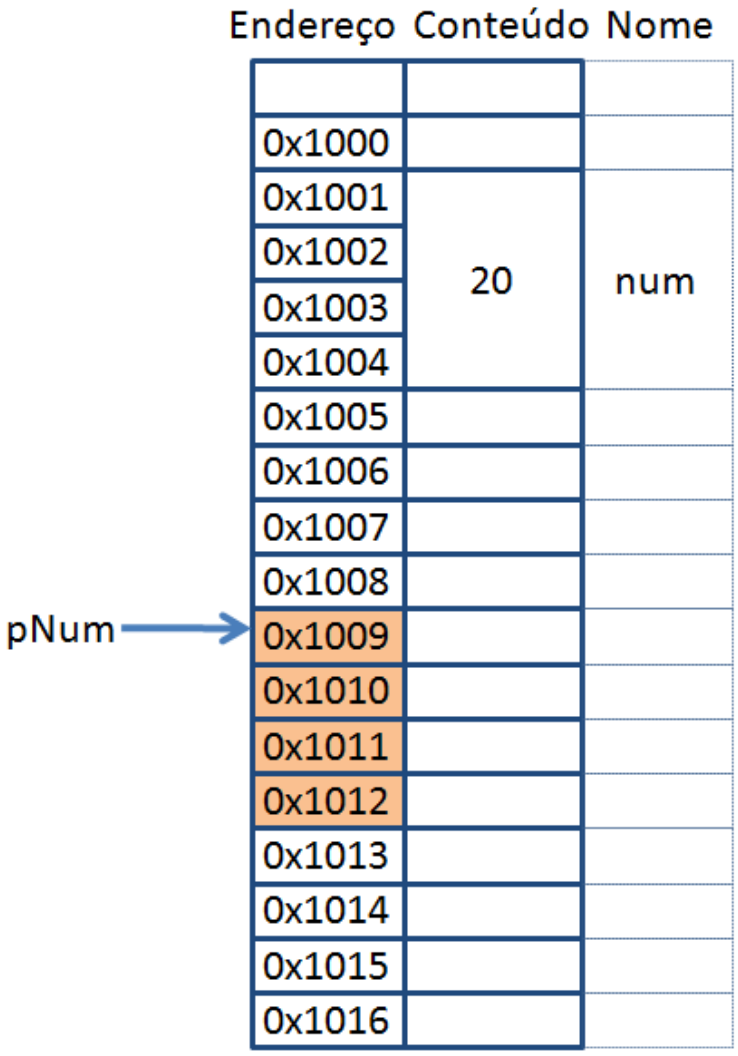

```
int main() {
  int* pNum, num = 20;
   pNum = &Num;
   ...
   pNum++;
  return 0;
}
```

Aritmética de Ponteiros VIII



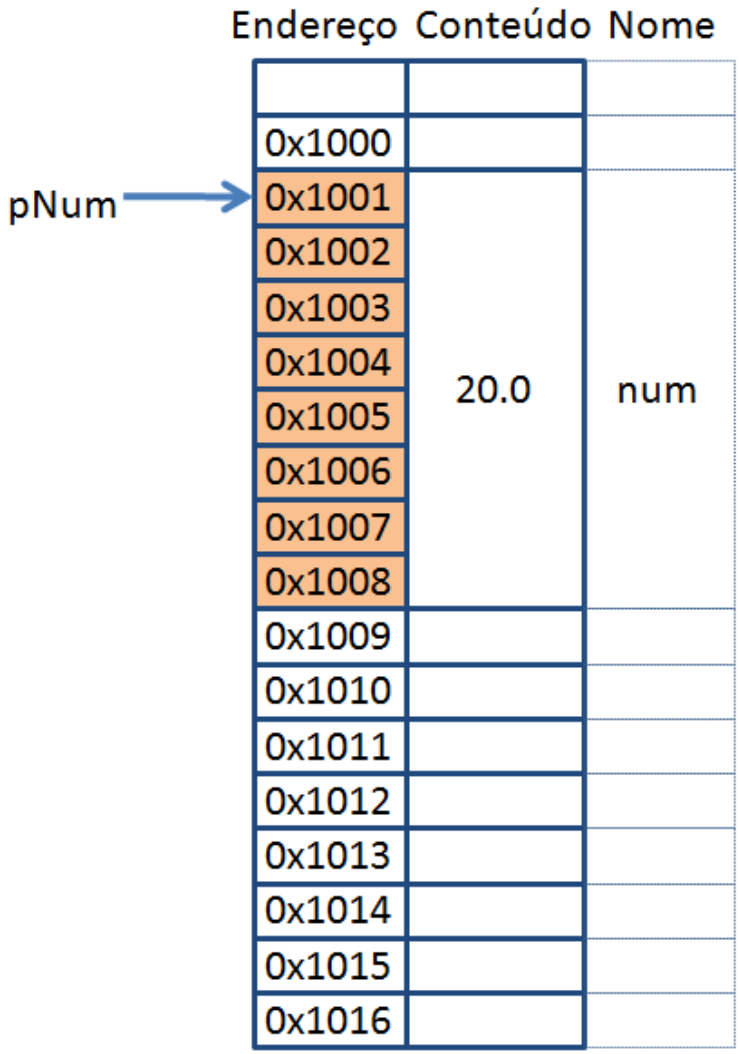

```
int main() {
  int* pNum, num = 20;
   pNum = &Num;
   ...
   pNum++;
  return 0;
}
```

Aritmética de Ponteiros IX



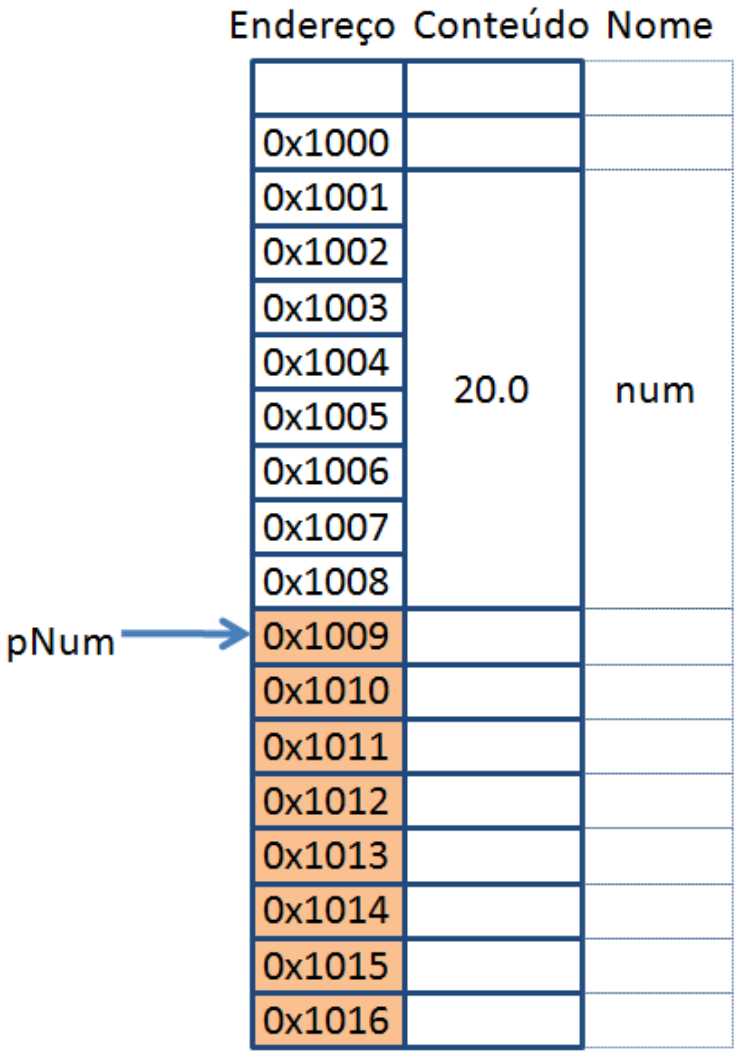

```
int main() {
  int* pNum, num = 20;
   pNum = &Num;
   ...
   pNum++;
   pNum++

  return 0;
}
```

Aritmética de Ponteiros X




```
int main() {
  double* pNum, num = 20.0;
   pNum = &Num;
   ...
   pNum++;
  return 0;
}
```

Aritmética de Ponteiros XI

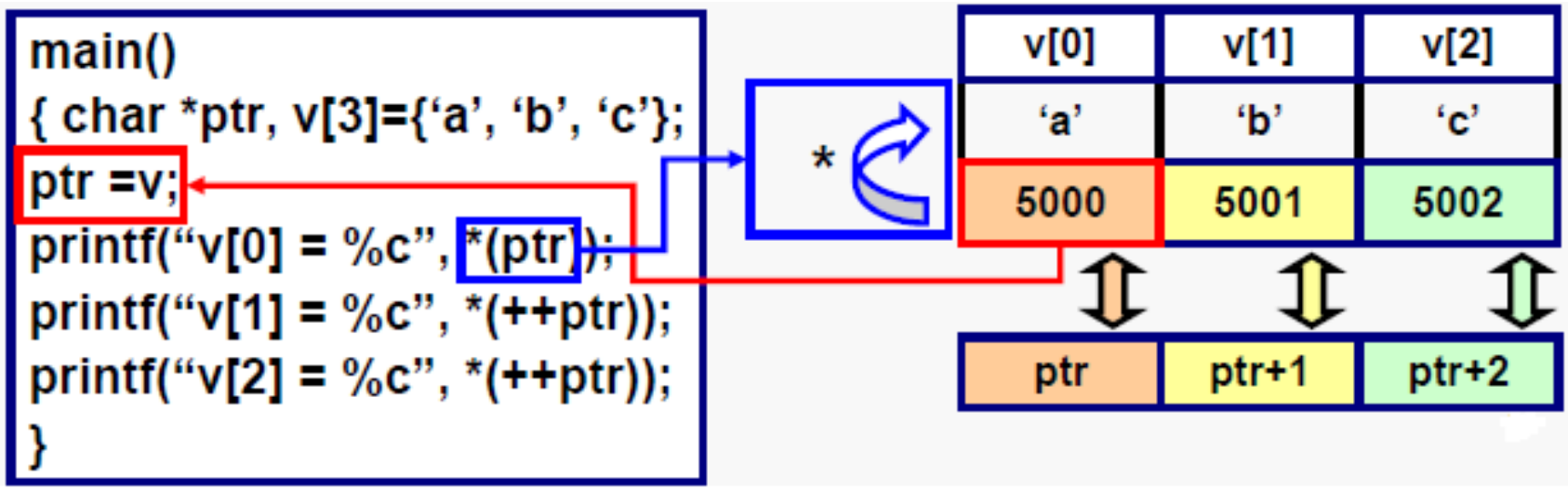



```
int main() {
  double* pNum, num = 20.0;
   pNum = &Num;
   ...
   pNum++;
  return 0;
}
```

Relação entre ponteiros e vetores I

- A aritmética de ponteiros é particularmente importante para manipulação de vetores e strings.
- Quando declaramos um vetor seus elementos são alocados em espaços de memória vizinhos.
- O nome de um vetor equivale ao endereço do primeiro elemento dele (se um vetor possui nome v, então, v equivale a $v[0]$.

Relação entre ponteiros e vetores II

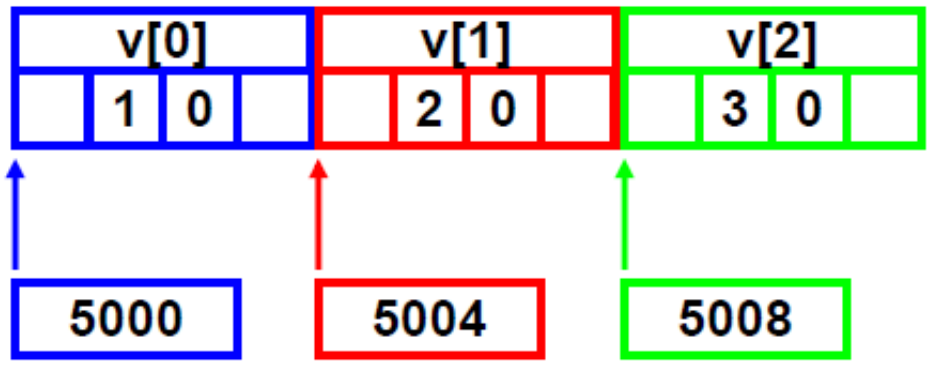


Vetores I

Exemplo – Vetor+Ponteiro

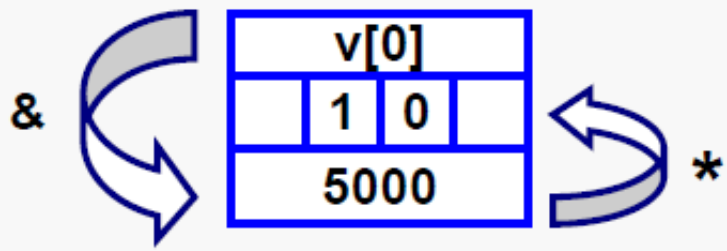
```
#include <stdio.h>
main()
{
  int i, *p, v[3];
  p = &v[0];  // ou p = &v;
  // Usando indices para acessar v.
  printf("Usando v[i]:");
  for (i=0; i < 3; i++)
  {
    scanf("%d", &v[i]);
    printf(" [%4d] ", v[i]);
  }
  // Usando ponteiros.
  printf("Usando v[i]:");
  for (i=0; i < 3; i++, ptr++)
  {
    scanf("%d", ptr);
    printf(" [%4d] ", *ptr);
  }
}
```

Em termos de memória



Observe que &v[i] equivale à ptr !

Observe, ainda, que:

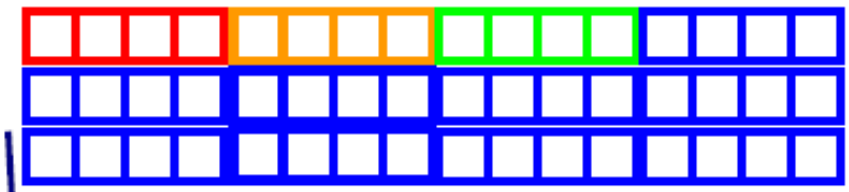


Alocação Dinâmica de Memória I

Exemplo – Alocação 1

```
#include <stdio.h>
main()
{
  int i, s, v[12];
  // Só usa 3 blocos de 4 bytes !
  s = 0;
  for (i=0; i < 3; i++)
  {
    printf(" Entre com v[%d]: ", i);
    scanf("%d", &v[i]);
    // Contando bytes usados.
    s = s + sizeof(v[i]);
  }
  for (i=0; i < 3; i++)
    printf("\n [%4d] \n", v[i]);
  // Mostra o uso da memória.
  printf("Bytes usados = %d \n",s);
  s = sizeof(v);
  printf("Bytes de v = %d \n",s);
}
```

Alocação Estática de Memória p/ v



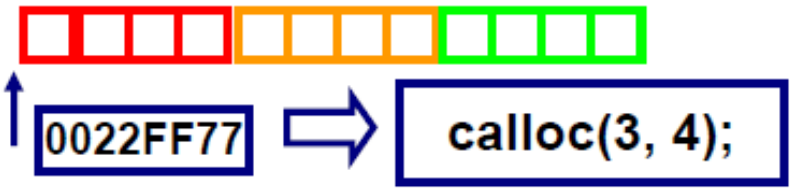
Quando o vetor v é declarado são reservados 12 espaços para elementos do tipo int. Como cada elemento de v é do tipo int e ocupa 4 bytes, então, é realizada a alocação de 12 * 4 = 48 bytes para o vetor v ! Observe, porém, que só são usados os elementos v[0], v[1] e v[2], ou seja, 3*4 = 12 bytes de um total de 48 bytes.

Alocação Dinâmica de Memória II

Exemplo – Alocação 2

```
#include <stdio.h>
main()
{int i, n; float *nota = NULL;
printf("Entre com tamanho n: ");
scanf("%d",&n);
// Aloca 4*n bytes de memória em
// tempo de execução com
// ponteiro!
nota=(float *)calloc(n,sizeof(float));
for (i=0; i < n; i++)
{
   printf(" Entre a nota %d:", i+1);
   scanf("%f", nota+i);
}
for (i=0; i < n; i++)
    printf("\n [%4f] \n", *(nota+i));
// Mostra o uso da memória.
s = n*sizeof(float);
printf("Bytes de nota = %d \n",s);
}
```

Alocação Dinâmica de Memória



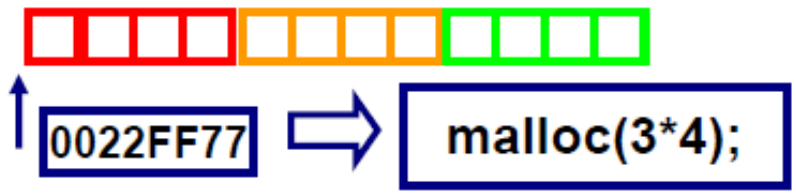
A alocação dinâmica de memória consiste em determinar em tempo de execução o quanto de memória será utilizado. A função calloc(n, size_t) fornece o primeiro endereço de memória de um bloco de n espaços com size_t bytes. No exemplo acima são alocados 3*sizeof(float)=3*4 = 12 bytes.

Alocação Dinâmica de Memória III

Exemplo – Alocação 3

```
#include <stdio.h>
main()
{int i, n; float *nota = NULL;
printf("Entre com tamanho n: ");
scanf("%d",&n);
// Aloca 4*n bytes de memória em
// tempo de execução com
// ponteiro!
nota=(float *)malloc(n*sizeof(float));
for (i=0; i < n; i++)
{
   printf(" Entre a nota %d:", i+1);
   scanf("%f", nota+i);
}
for (i=0; i < n; i++)
    printf("\n [%4f] \n", *(nota+i));
// Mostra o uso da memória.
s = n*sizeof(float);
printf("Bytes de nota = %d \n",s);
}
```

Alocação Dinâmica de Memória



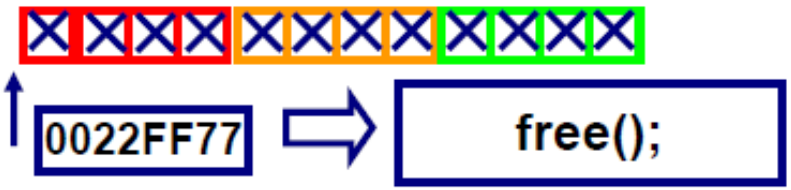
A função malloc(n*size_t) fornece o primeiro endereço de memória de um bloco de n espaços com size_t bytes. No exemplo acima são alocados 3*sizeof(float) = 3*4 = 12 bytes. A função malloc possui a mesma utilidade da função calloc, apenas sua sintaxe é diferente.

Alocação Dinâmica de Memória IV

Exemplo – Alocação 4

```
#include <stdio.h>
main()
{int i, n; float *nota = NULL;
printf("Entre com tamanho n: ");
scanf("%d",&n);
// Aloca 4*n bytes de memória em
// tempo de execução com
// ponteiro!
nota=(float *)malloc(n*sizeof(float));
for (i=0; i < n; i++)
{
   printf(" Entre a nota %d:", i+1);
   scanf("%f", nota+i);
}
for (i=0; i < n; i++)
    printf("\n [%4f] \n", *(nota+i));
// Libera memória alocada cujo
// endereço inicial está em nota.
free(nota);
}
```

Alocação Dinâmica de Memória



Quando é realizada alocação estática de memória o programa em C se encarrega de liberar a memória utilizada. Mas, quando a alocação dinâmica de memória é empregada, através de malloc ou calloc, é necessário, antes de terminar o uso do programa, liberar a memória através do comando free().

Alocação Dinâmica de Memória V

	Comando	Liberação Memória
Alocação Estática de Memória	`int v[3];` Reserva 3 espaços de 4 bytes em **v**	O propio programa ao ser encerrado, se encarrega de liberar a memória alocada.
Alocação Dinâmica de Memória	`v = (int*) malloc (n*sizeof(int));` Reserva *n* espaços de 4 bytes em **v**	Se não for usar mais a variável *v*, então, é necessário empregar o comando free().

Relação entre ponteiros, vetores e matrizes I

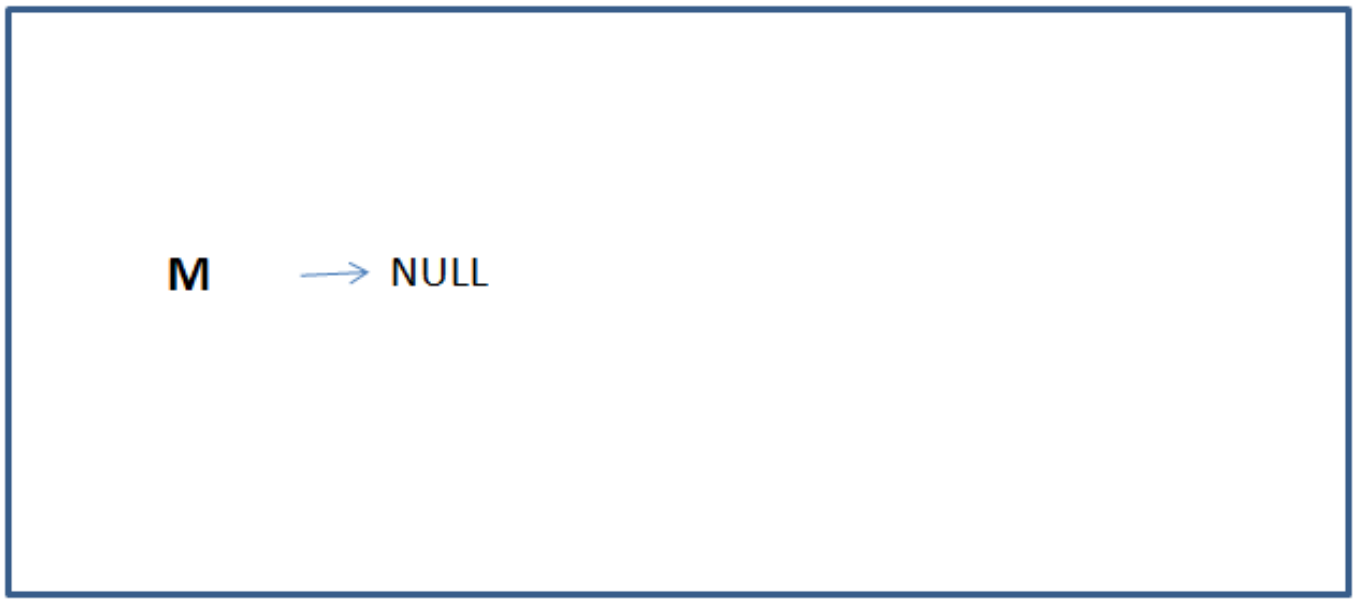
- Assim como é possível alocar memória em tempo de execução para armazenar um vetor, também, é possível construir uma matriz M com m linhas e n colunas. Os comandos para tal tarefa são como dados a seguir:

```
int main()
{
    char **M;
    int m, n, i;
    printf("Entre com m e n");
    scanf("%d %d", &m, &n);
    // Vetor de endereços (os elementos são do tipo char*)
    M = (char**) calloc (m, sizeof(char*));
    // Cria para cada endereço um vetor de elementos int
    for (i = 0; i < m; i++)
        M[i] = (char*) calloc (n, sizeof(char))
    return 0;
}
```

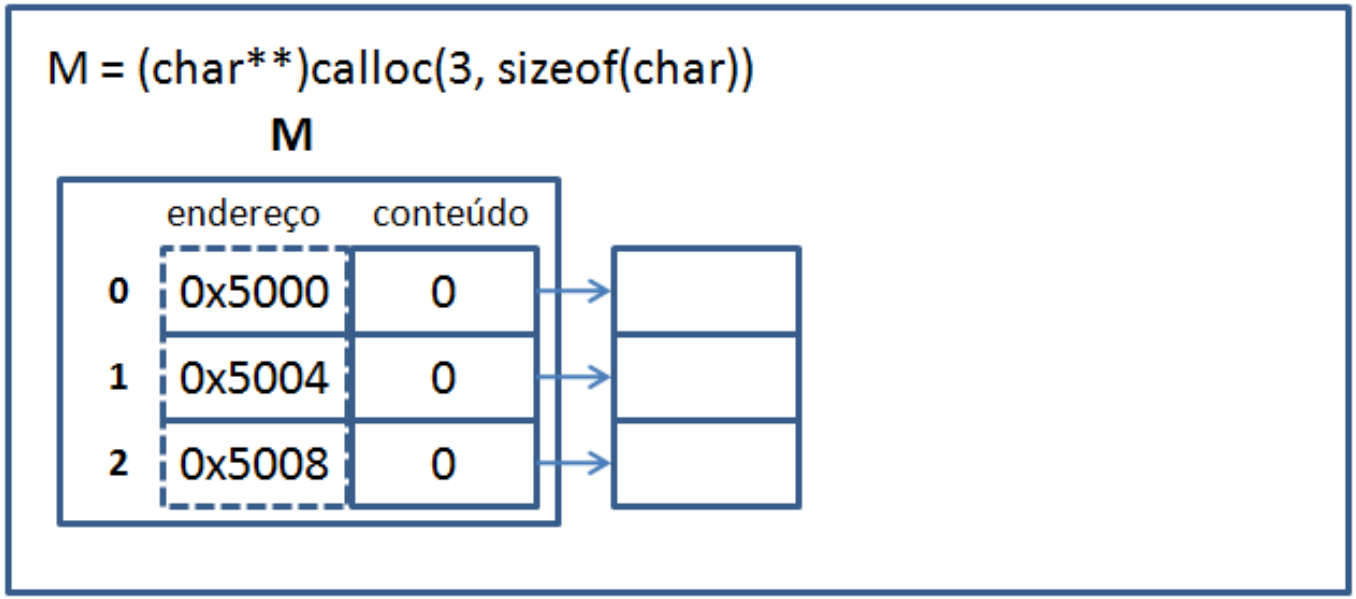
Relação entre ponteiros, vetores e matrizes II

- Vejamos um exemplo:

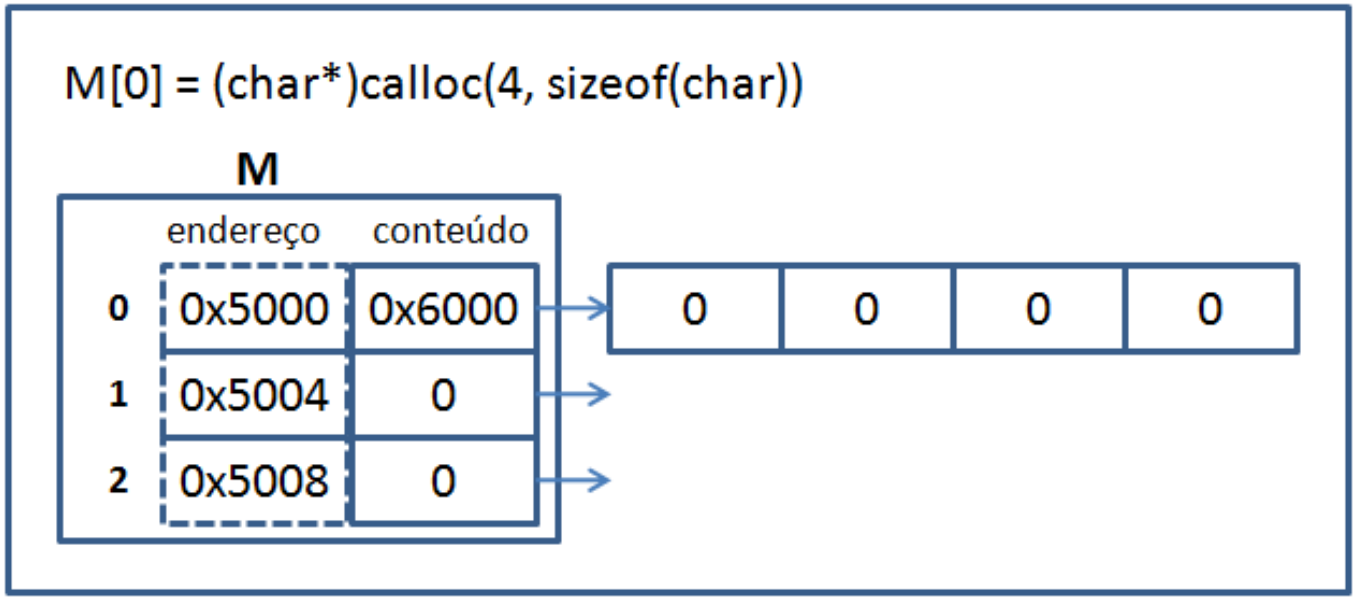
```
M = (char**) calloc (3, sizeof(char*));
```



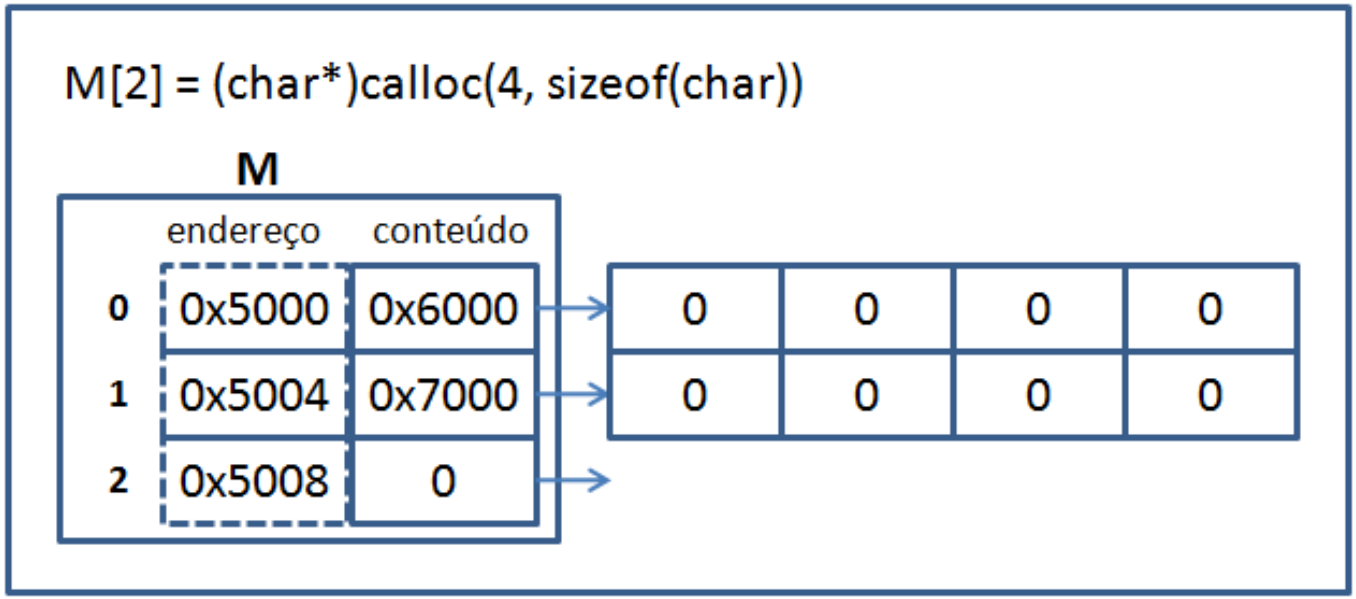
Relação entre ponteiros, vetores e matrizes III



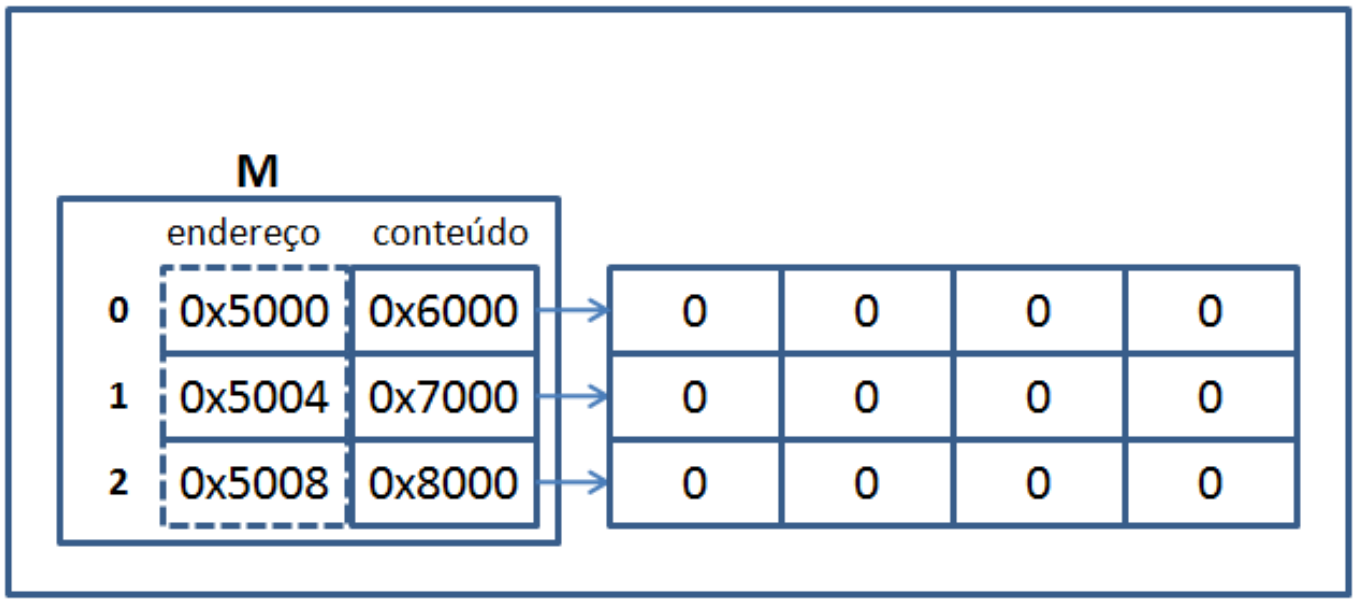
Relação entre ponteiros, vetores e matrizes IV



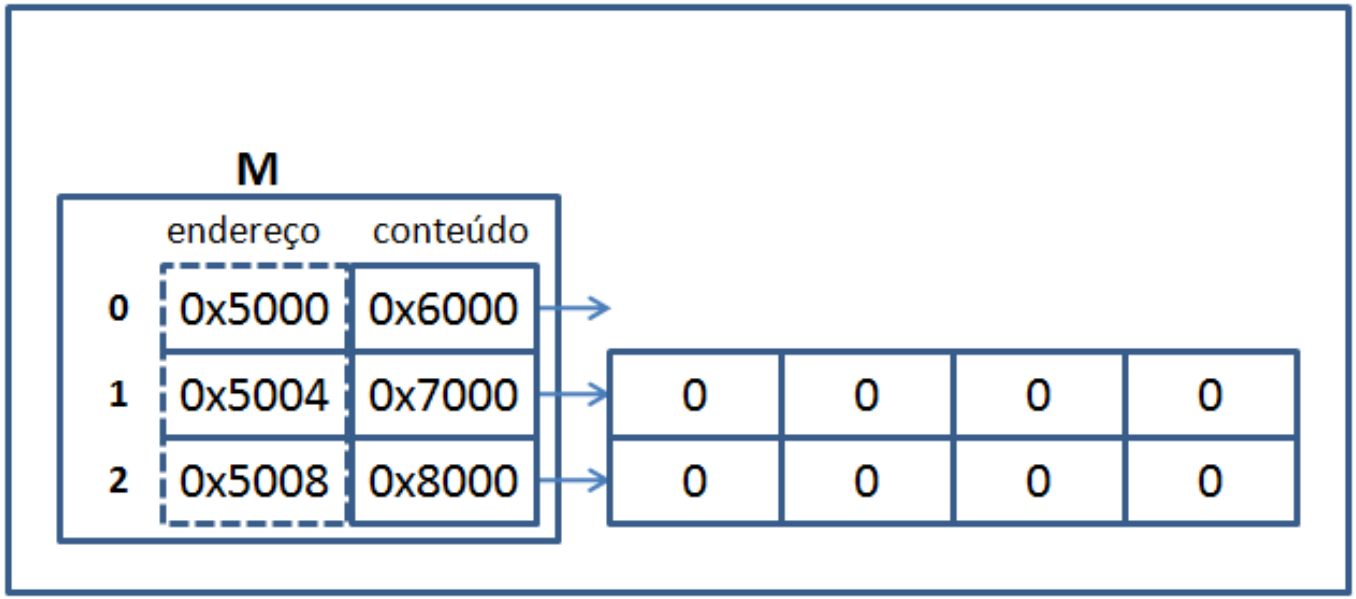
Relação entre ponteiros, vetores e matrizes V



Relação entre ponteiros, vetores e matrizes VI

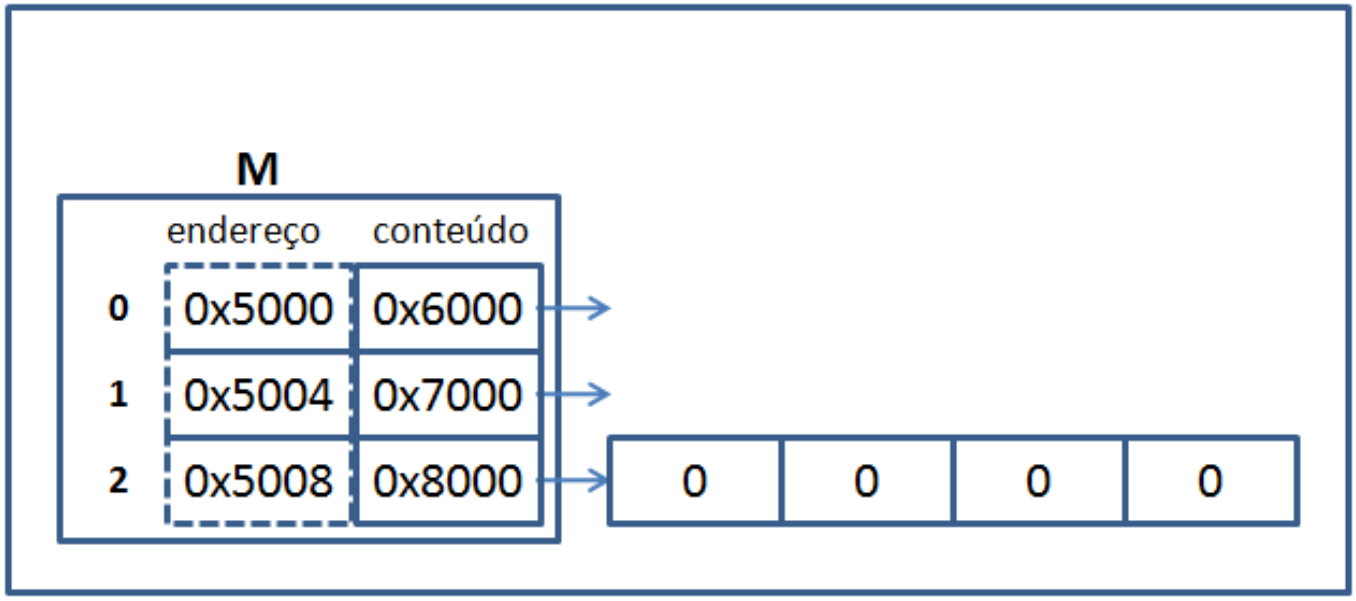


Liberação de memória I



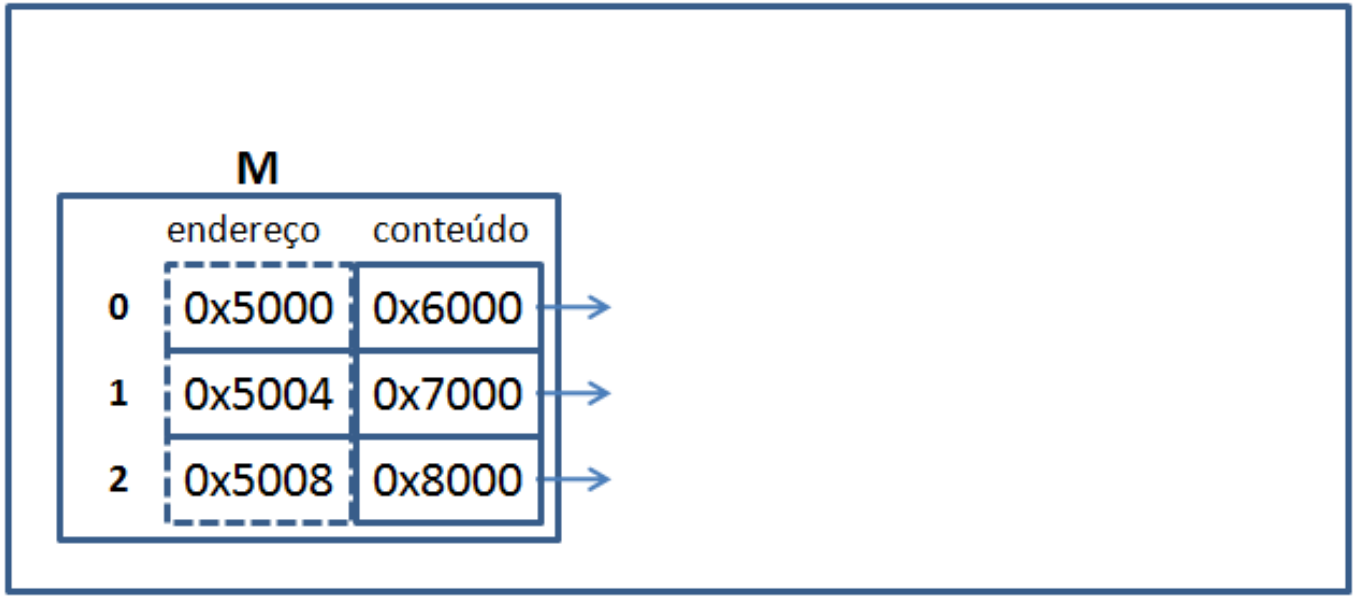
free(M[0])

Liberação de memória II



free(M[1])

Liberação de memória III



free(M[2])

Liberação de memória IV

Exercicios I

Inserir n notas de um total de m alunos

Exercicios II

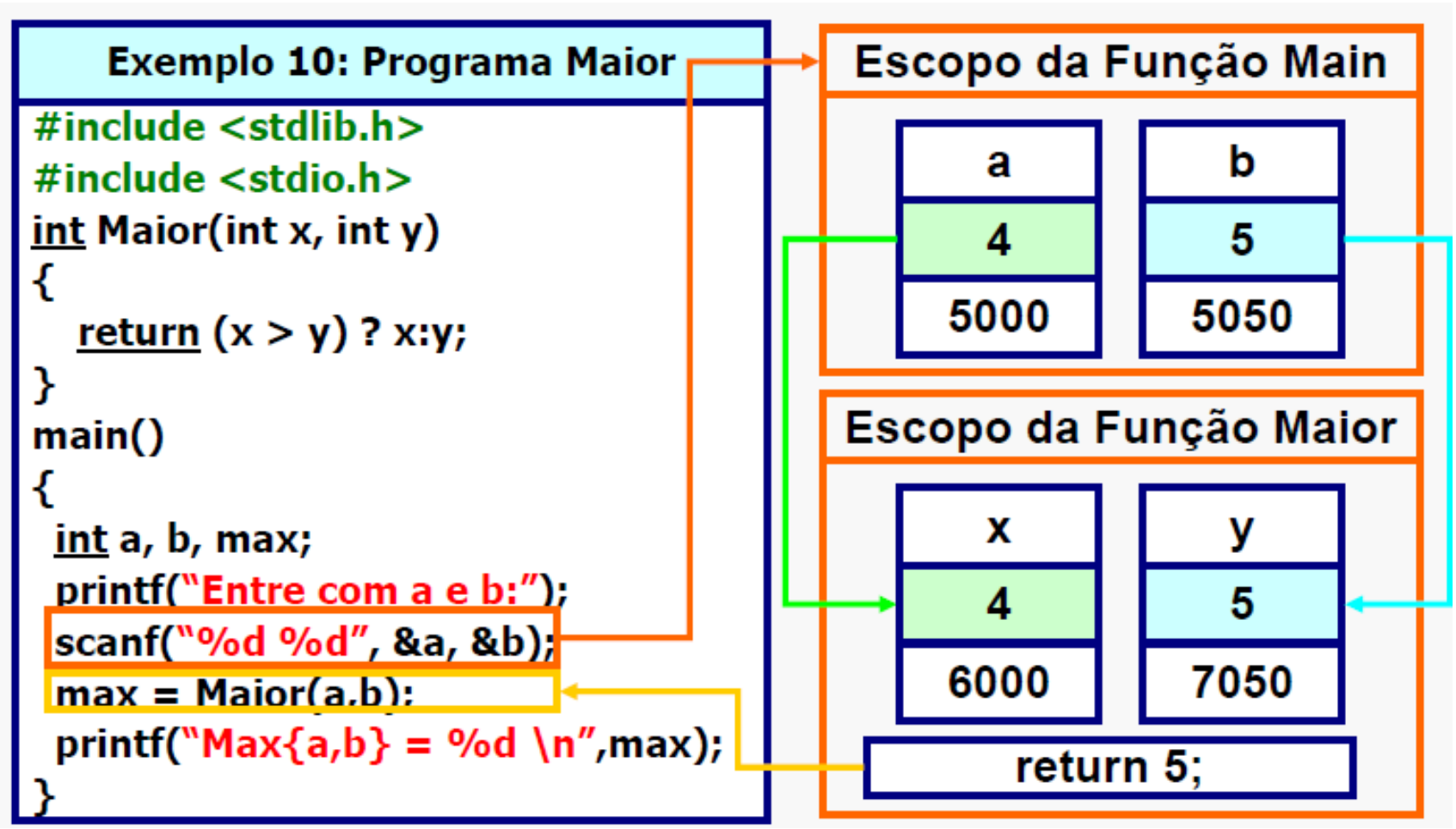
```c
#include <stdio.h>
#include <stdlib.h>
int main()
{   int i, j, m, n; float **M = NULL;
    printf("Entre com m e n: ");
    scanf("%d %d", &m, &n);
    // Aloca m espaços tipo float *.
    M = (float **)calloc(m, sizeof(float *));
    // Aloca n espaços tipo float, cada M[i].
    for (i = 0; i < m; i++)
        M[i] = (float *)calloc(n, sizeof(float));
    // Preenchendo a matriz M usando índices: M[i][j].
    for (i=0; i < m; i++)
    {   printf("Aluno %d: ",i+1);
        for (j=0; j < n; j++)
        {   printf(" Nota %d:", j+1);
            scanf("%f", &M[i][j]);
        }
    }
    ...
}
```

Exercicios III

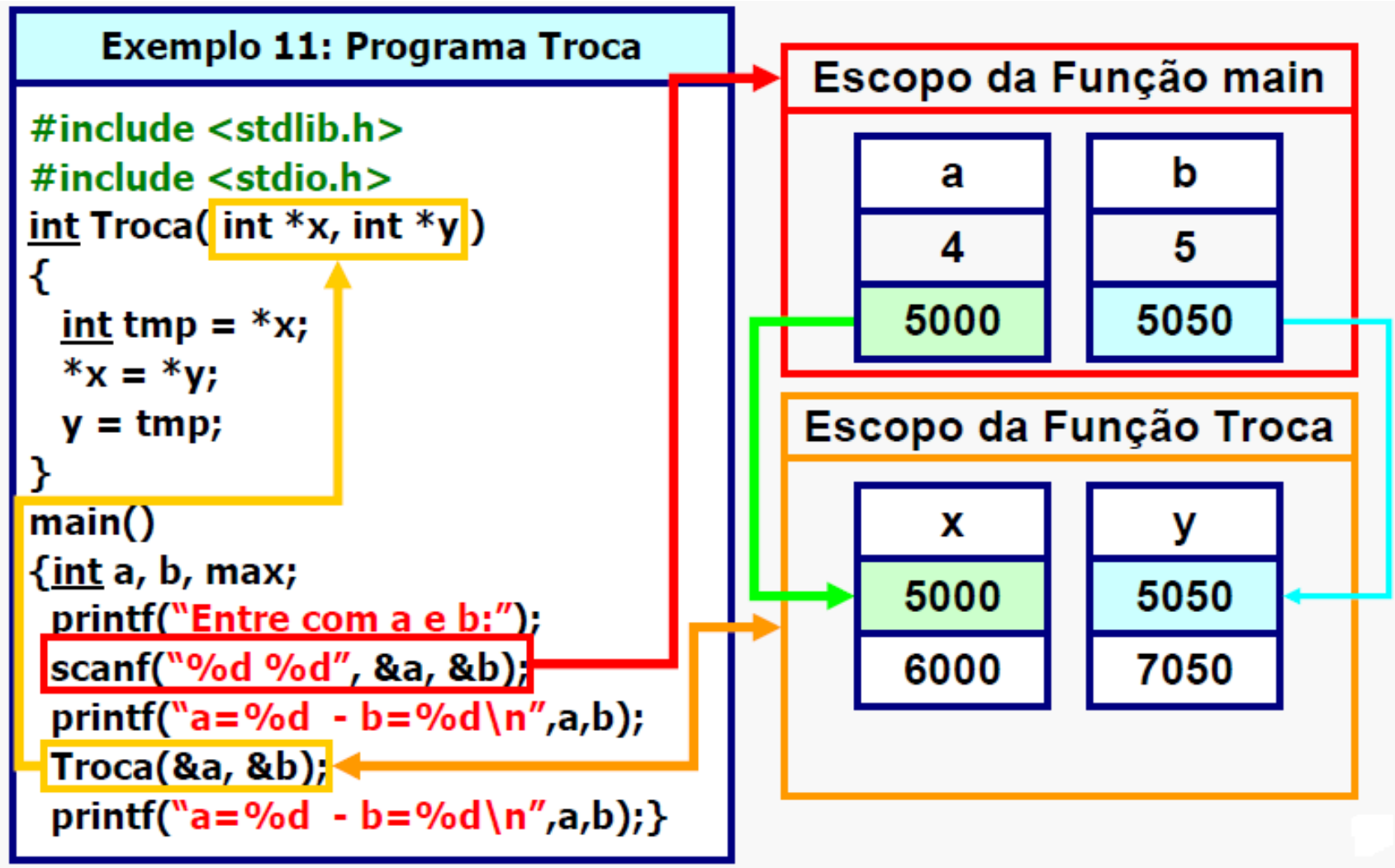
```
    . . .
    // Impressão dos elementos de M, empregando ponteiros.
    for (i=0; i < m; i++)
    {    for (j=0; j < n; j++)
         printf(" [%4f] ", M[i][j]);
         printf(" \n ");
    }
    // Liberação de memória.
    // Liberando m vetores de tamanho n.
    if (M != NULL){
         for (i=0; i < m; i++)
              if (M[i] != NULL)free(M[i]);
         // Liberando o vetor de ponteiros
         // de tamanho m.
         free(M);
    }
    return 0;
}
```

Ponteiros: Passagem por valor e por referência I

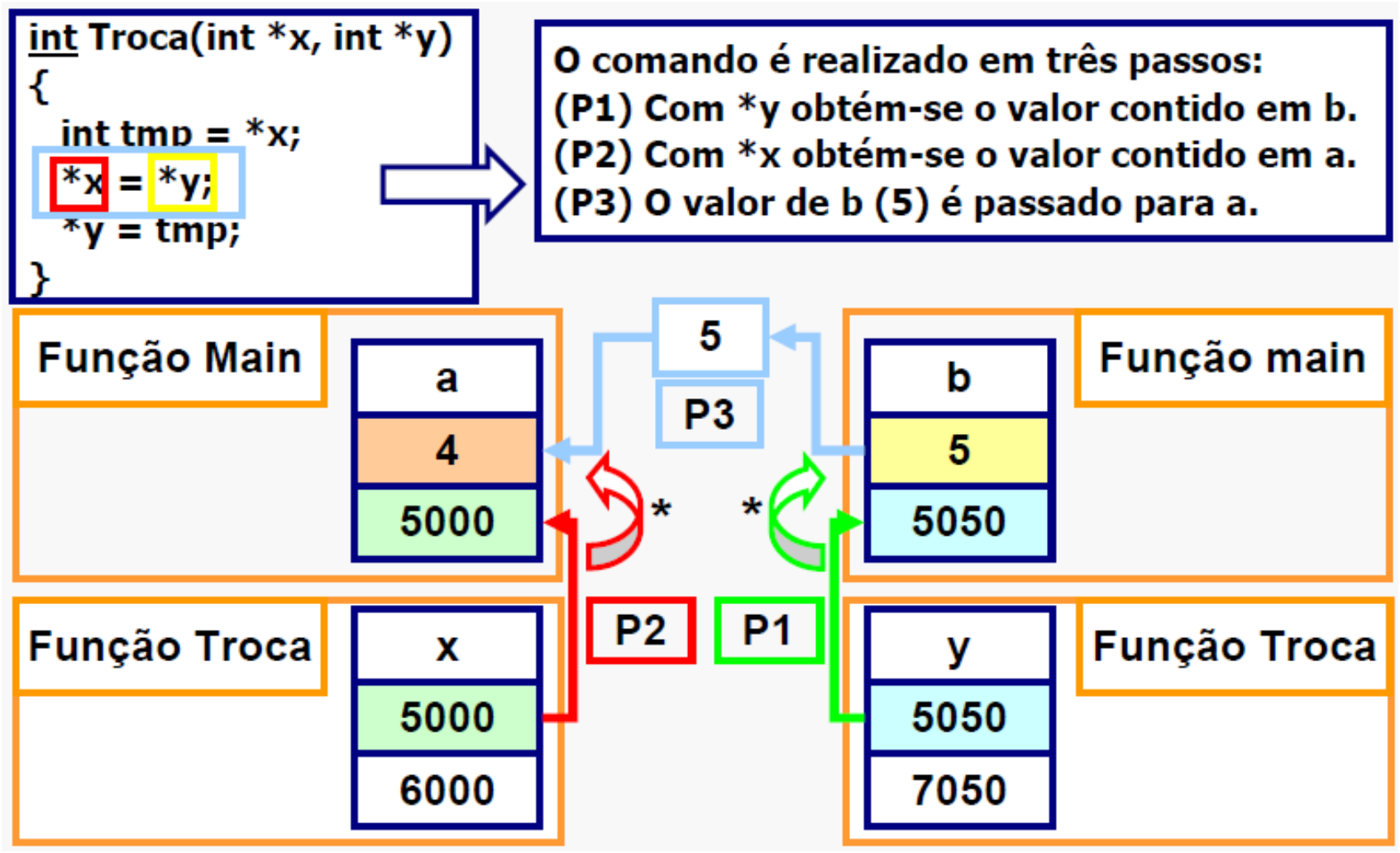
Implementar uma função que encontre o maior de dois números



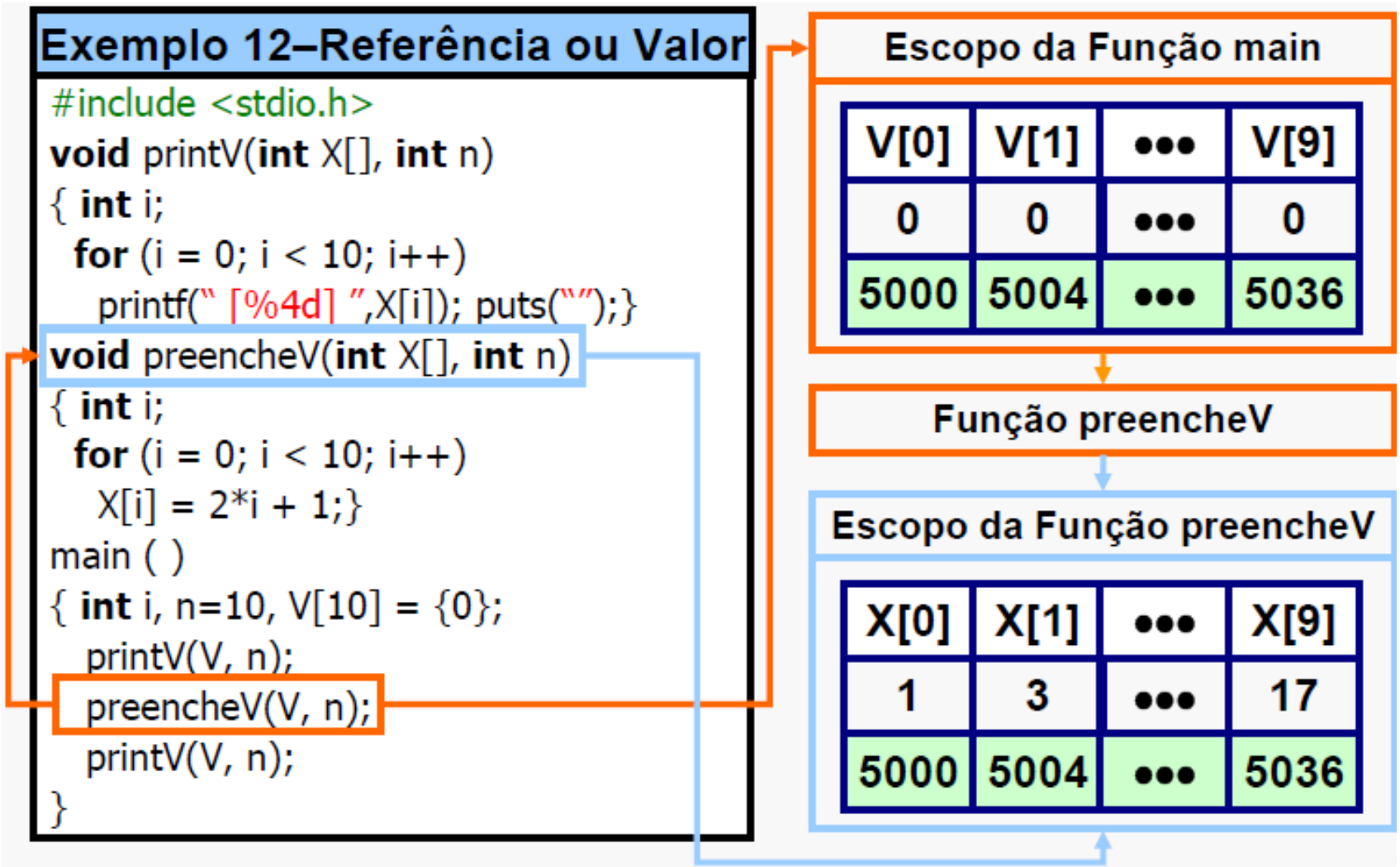
Ponteiros: Passagem por valor e por referência II



Ponteiros: Passagem por valor e por referência III



Ponteiros: Passagem por valor e por referência IV



Ponteiros: Passagem por valor e por referência V

Criar uma estrutura empregrado com os seguintes campos:

- nome
- salario
- sexo

Inserir *n* empregados (criar um vetor dinâmico)

Ponteiros: Passagem por valor e por referência VI

```c
typedef struct Pessoa
{
    char nome[100];
    double salario;
    char sexo;
}PE;

void Insere(PE*, int);
void Print(PE*, int);
int main()
{
    int n;
    PE *trab = NULL;
    printf("Quantidade de pessoas");
    scanf("%d%*c", &n);
    trab = (PE*) calloc(n, sizeof(PE));
    Insere(trab, n);
    Print(trab, n);
    return 0;
}
```

Ponteiros: Passagem por valor e por referência VII

```
void Insere(PE* vet, int n)
{
    int i, valores;
    for (i = 0; i < n; i++)
    {
        printf("Cadastro numero %d\n", i+1);
        printf("Insere nome: ");
        fgets(vet[i].nome, 100, stdin);
        do{
            printf("Insere salario: ");
            valores = scanf("%lf", &vet[i].salario);
            if (valores == 0) scanf("%*s");
        }while(valores == 0);

        printf("Insere sexo: ");
        scanf("%c%*c", &vet[i].sexo);
        //vet[i].sexo = getchar();
    }
}
```

Ponteiros: Passagem por valor e por referência VIII

```
void Print(PE* vet, int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        printf("%s %lf %c \n",
               vet[i].nome, vet[i].salario, vet[i].sexo);
    }
}
```

Exercicios em Propostos I

1. Defina uma função que retorne a transposta de uma matriz x de dimensão $lin \times col$.

```
int** Transposta(int** M, int lin, int col)
```

2. Elaborar um programa para calcular a méda aritmética de dois valores reais utilizando apenas variáveis do tipo ponteiro.

FIM