Apontadores

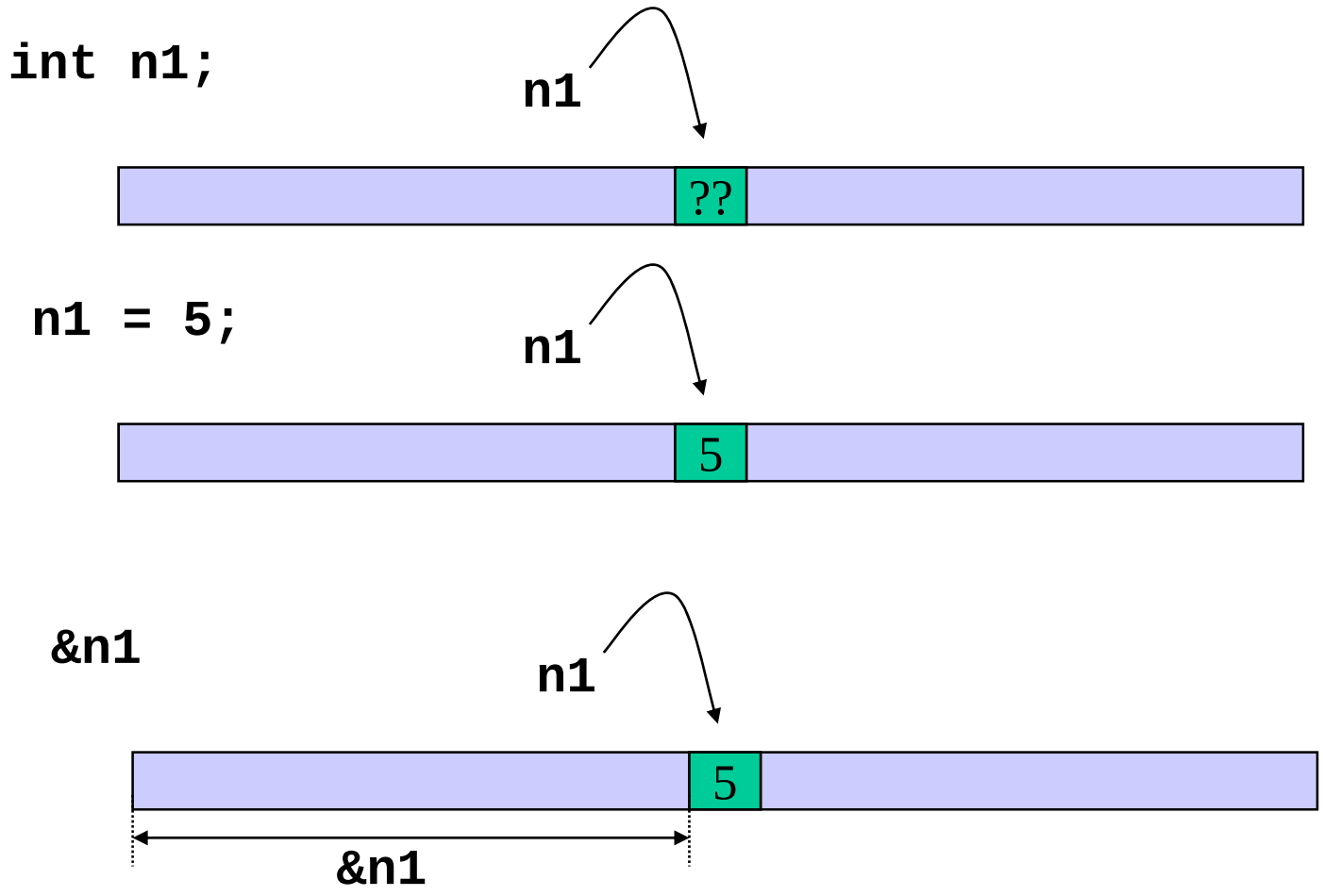
- A linguagem C permite aceder a memória através de apontadores.
- Para cada um dos tipos básicos é possível criar apontadores.
- Exemplo:

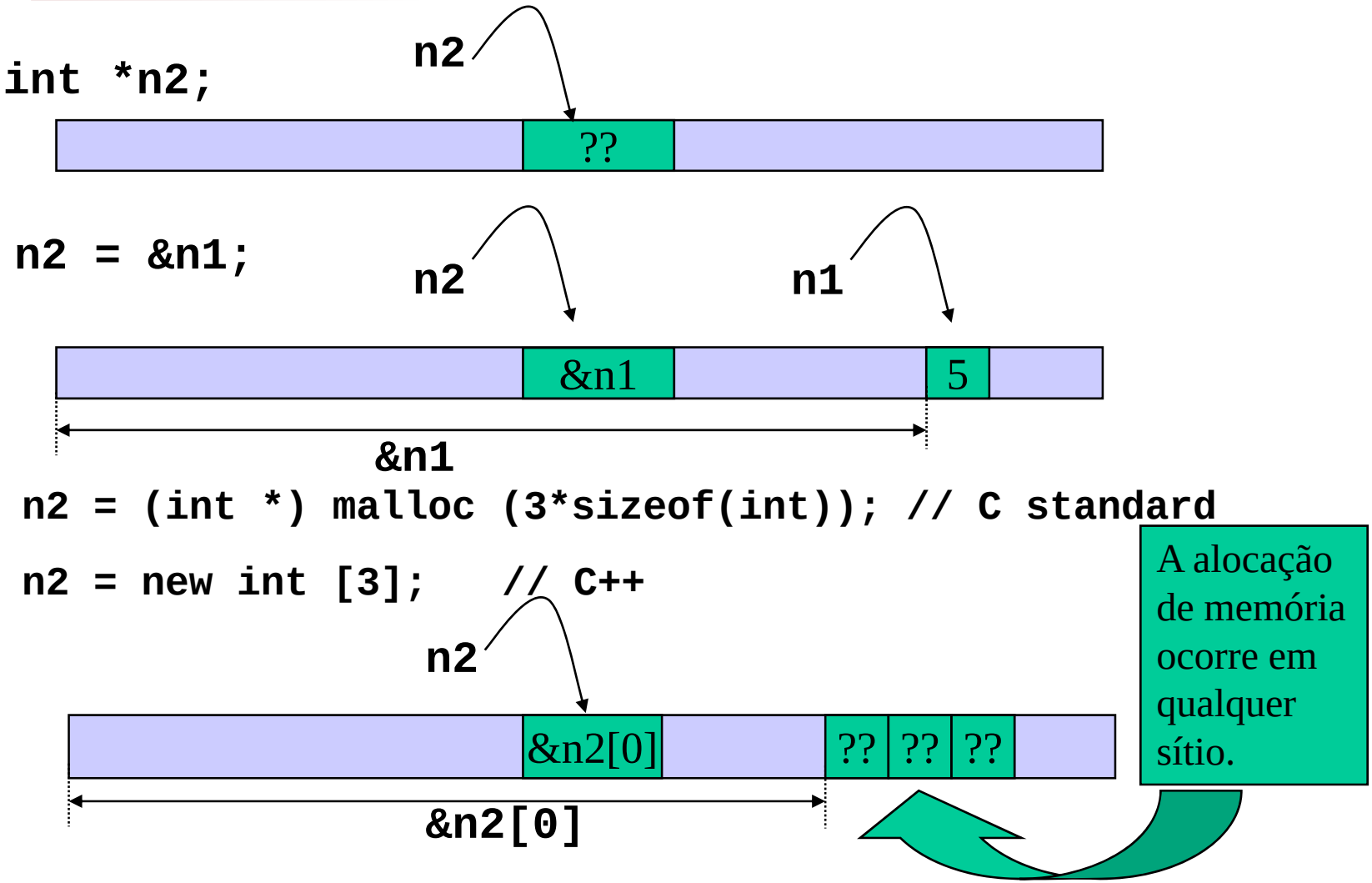
```
int n1;
```

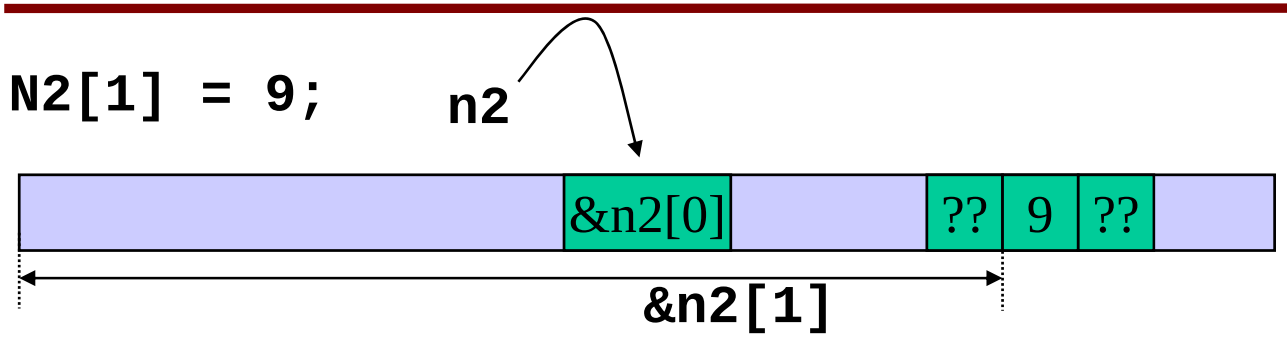
  - Reserva uma posição de memória para a variável "**n1**";
  - **&n1** é o endereço de memória onde o valor de n1 é armazenado;

```
int *n2;
```

  - Reserva uma posição de memória, de nome "**n2**", onde se poderá colocar o **endereço de um inteiro**;


int n1;
n1
??
n1 = 5;
n1
5
&n1
n1
5
&n1


int *n2;
n2
??
n2 = &n1;
n2
n1
&n1
5
&n1
n2 = (int *) malloc (3*sizeof(int)); // C standard
n2 = new int [3]; // C++
n2
&n2[0]
??
??
??
&n2[0]
A alocação de memória ocorre em qualquer sítio.




```
printf("%d", *(n2+1));
```

- 9
- Porque **n2+1** é igual **&n2[1]**

```
printf("%d", *n2+1);
```

- ??+1
- Porque ***n2** é o conteúdo de **n2[0]** que contém **??**

- Em C, existe uma correspondência entre tabelas e apontadores:
  - *array ⇔ array[0]
  - *(array + 1) ⇔ array[1]
  - *(array + idx) ⇔ array[idx]

```
int array[10];
int *ptr;
array[2] = 7;
ptr = array + 2;
printf("%d", *ptr);
```

- Com estas instruções aparece 7 escrito no écran

- Quando o tamanho da tabela é conhecido apenas na altura da execução deve-se alocar memória de forma dinâmica.

```
#include <stdlib.h>
#include <stdio.h>
void main( )
{
  int num, *ptabela;
  printf (“Digite o número de registos da tabela: “);
  scanf(“%d”, &num);
  ptabela = (int *)malloc(num*sizeof(int));
  if (ptabela == NULL)
  {
    printf (“Memória insuficiente.\n”);
    return;
  }
....
```

Exemplo (Cálculo da média de N valores)

```
#include <stdio.h>
// Cabeçalho da função que calcula a média
float media(int *valores, int nelems);

void main()
{
  int *vector;
  int nelems;
  float med;
  int i;

  // Introdução dados
  printf("Quantos valores para calcular a média? ");
  scanf ("%d", &nelems);

  // Alocação dinâmica da memória
  vector = new int[nelems];
  printf ("Introduza %d elementos inteiros\n", nelems);
  for (i=0; i<nelems; i++)
    scanf("%d", &vector[i]);

  // Calcular Média
  med = media(vector, nelems);

  // Output média
  printf ("Média: %f\n", med);

  // Libertar memória quando não é mais necessária
  delete[] vector;
}
```

Exemplo (cont)

```
float media(int *valores, int nelems)
{
  float avg;
  avg = 0;
  for (int i=0; i<nelems; i++)
    avg += valores[i];
  if (nelems != 0)
    avg /= nelems;
  return avg;
}
```

Nº de elementos do array

Endereço de um array

- Declaração estática :

```
int matriz [10][20];
```

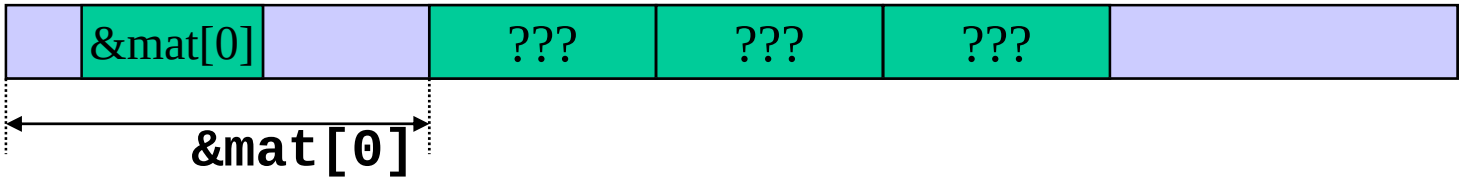
- Alocação dinâmica:

```
int ** matriz_d;
...
matriz_d = new int* [nlinhas];
for (i=0; i < nlinhas; i++)
  matriz_d[i] = new int [ncolunas];
```

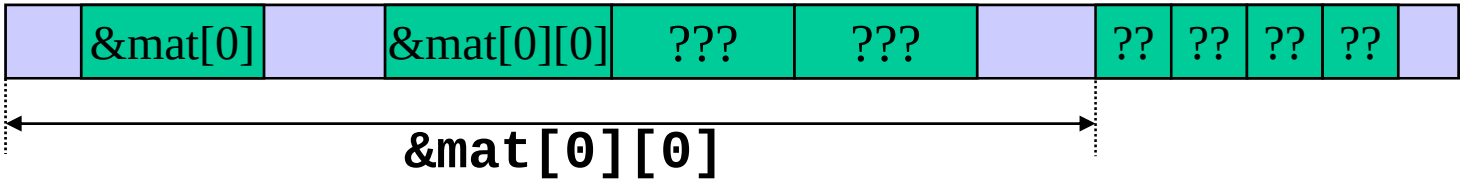
- Se nlinhas=10 e ncolunas=20 a **`matriz_d`** tem o mesmo tamanho que a matriz anterior

```
int ** mat;
```


```
mat = new int* [3];
```




```
mat[0] = new int[4];
```



- Para alocar o resto da matriz há que fazer o mesmo malloc para mat[1] e mat[2]

- Depois de se usar a matriz **`mat`** é **necessário libertar** a memória.
- Deve-se libertar a memória fazer pela ordem inversa da alocação.
- Liberta-se **`mat[0]`**, **`mat[1]`** e **`mat[2]`** e só depois se liberta **`mat`** através das seguintes instruções

```
free (mat[0]);   ou   delete [] mat[0];
free (mat[1]);   ou   delete [] mat[1];
free (mat[2]);   ou   delete [] mat[2];
free (mat);      ou   delete [] mat;
```

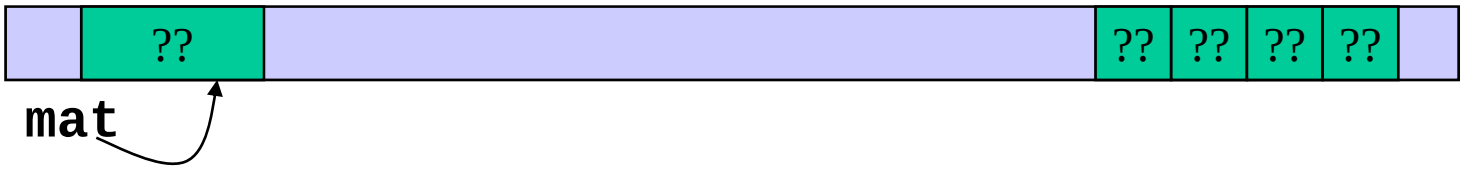
- O que aconteceria se depois de alocar "**`mat[0]`**"se fizesse a libertação de "**`mat`**" ?

```
mat = new int* [3];
mat[0] = new int [4];
delete [] mat;
```

ou

```
mat = (int **) malloc(3*sizeof(int *));
mat[0] = (int *) malloc(4*sizeof(int));
free(mat);
```

- Como fica a memória ?



  - O espaço de memória que foi reservado para "**`mat[0][0-3]`**" permanece, mas nunca mais pode ser usado para outras alocações

Cadeias de caracteres

- Uma cadeia de caracteres (“string”) é um vector de caracteres, em o último carácter tem que ser obrigatoriamente o ‘\0’ (carácter nulo).
  - Uma cadeia pode ser indicada entre aspas (ex: **“EDA”** é uma cadeia de 4 caracteres)
  - Numa cadeia de caracteres inicializada, o compilador de C determina a dimensão que é sempre igual ao número de caracteres + 1

    char msg[] = “texto”; // Cadeia com 6 caracteres

- Os apontadores e a memória dinâmica facilitam o tratamento de cadeias de caracteres.

Cadeias de caracteres

- A biblioteca de C **`string.h`** oferece um conjunto de funções para manipulação de cadeias de caracteres:
  - Função para copiar o conteúdo de uma cadeia de caracteres
    ```
    char *strcpy (char *dest, char *src);
    ```
  - O apontador **dest** deve ter a memória alocada suficiente para conter a cadeia **src**.
  - Função que devolve uma cópia da cadeia de caracteres:
    ```
    char *strdup (char *s);
    ```
  - Função que compara duas cadeias de caracteres:
    ```
    int strcmp(char *s1, char * s2);
    ```
    devolve um número inteiro maior, igual ou menor que 0 consoante a cadeia **`s1`** for menor, igual ou maior que **`s2`**.

Exemplo: produto de matrizes

- Matrizes iniciais
  - `int** mat_a;`
  - `int** mat_b;`
- Matriz resultado
  - `int** mat_c;`
- Alocação dinâmica das matrizes

```
mat_a = new int*[n];
mat_b = new int*[m];
mat_c = new int*[n];
```

Exemplo (cont.)

```
for (i=0; i<n; i++)  // n linhas da matriz A
{
   mat_a[i] = new int[m]; // m colunas da matriz A
   mat_c[i] = new int[p]; // p colunas da matriz C
}
for (j=0; j<m; j++)  // m linhas da matriz B
   mat_b[j] = new int[p]; // p colunas da matriz B
```

Exemplo (cont.)

- Algoritmo cálculo

```
for (k=0; k<m; k++)
   mat_c[i][j] +=mat_a[i][k]*mat_b[k][j];
          (Para cada célula da matriz)

for (i=0; i<n; i++)
  for (j=0; j<p; j++)
  {
   mat_c[i][j] = 0;
   for (k=0; k<m; k++)
     mat_c[i][j] += mat_a[i][k]*mat_b[k][j];
  }
```

Exemplo (cont.)

- Libertação da memória

```
for (i=0; i<n; i++) {
  delete [] mat_a[i];
  delete [] mat_c[i];
}
for (j=0; i<m; j++)
  delete [] mat_b[j];
delete [] mat_a;
delete [] mat_b;
delete [] mat_c;
```

```
void main()
{
  int **mat_a, **mat_b, **mat_c;
  int n, m, p;
  int i, j, k;

  printf ("Introduza as dimensões da matriz A (n x m): ");
  scanf_s ("%d %d", &n, &m);

  printf ("Introduza o número de colunas da matriz B: ");
  scanf_s ("%d", &p);

  // Alocação dinâmica das matrizes
  mat_a = new int*[n];
  mat_b = new int*[m];
  mat_c = new int*[n];

  for (i=0; i<n; i++)
  {
    mat_a[i] = new int[m];
    mat_c[i] = new int[p];
  }
  for (j=0; j<m; j++)
    mat_b[j] = new int[p];
```

```
    printf ("Introduza os valores da matriz A (%d x %d) por linhas: ", n, m);
    for (i=0; i<n; i++)
      for (j=0; j<m; j++)
        scanf_s("%d", &mat_a[i][j]);
    printf ("Introduza os valores da matriz B (%d x %d) por linhas: ", m, p);
    for (i=0; i<m; i++)
      for (j=0; j<p; j++)
        scanf_s ("%d", &mat_b[i][j]);
    for (i=0; i<n; i++)
      for (j=0; j<p; j++)
      {
        mat_c[i][j] = 0;
        for (k=0; k<m; k++)
          mat_c[i][j] += mat_a[i][k]*mat_b[k][j];
      }
    for (i=0; i<n; i++)
    {
      for (j=0; j<p; j++)
        printf ("%d   ", mat_c[i][j]);
      printf("\n");
    }
    for (i=0; i<n; i++)
    {
      delete [] mat_a[i];
      delete [] mat_c[i];
    }
    for (j=0; j<m; j++)
      delete [] mat_b[j];
    delete [] mat_a;
    delete [] mat_b;
    delete [] mat_c;
}
```