

# Programmazione e Laboratorio di Programmazione

## Lezione XIII

### Le matrici

# Matrici

- **Matrice (bidimensionale) di  $n \times m$  elementi:**  
definisce una corrispondenza biunivoca tra un insieme omogeneo di  $n \times m$  elementi e l'insieme di coppie di interi  $\{(0,0), (0,1), \dots, (n-1, m-1)\}$
- **Esempio:**

Matricedi 5 x 2 interi

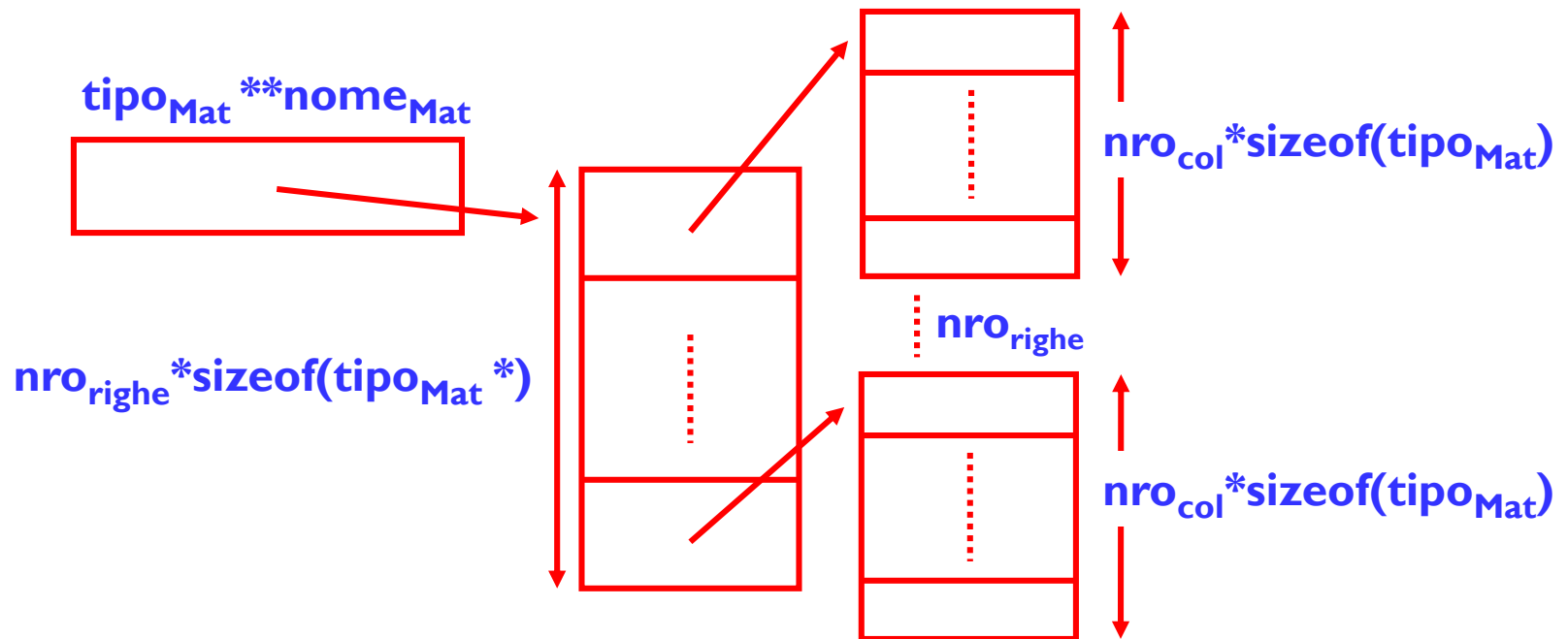
|       |    |    |       |
|-------|----|----|-------|
| (0,0) | 8  | 4  | (0,1) |
| (1,0) | 7  | -1 | (1,1) |
| (2,0) | 15 | 12 | (2,1) |
| (3,0) | 4  | -9 | (3,1) |
| (4,0) | 7  | 4  | (4,1) |

# Definizione statica di una matrice

- Definizione:

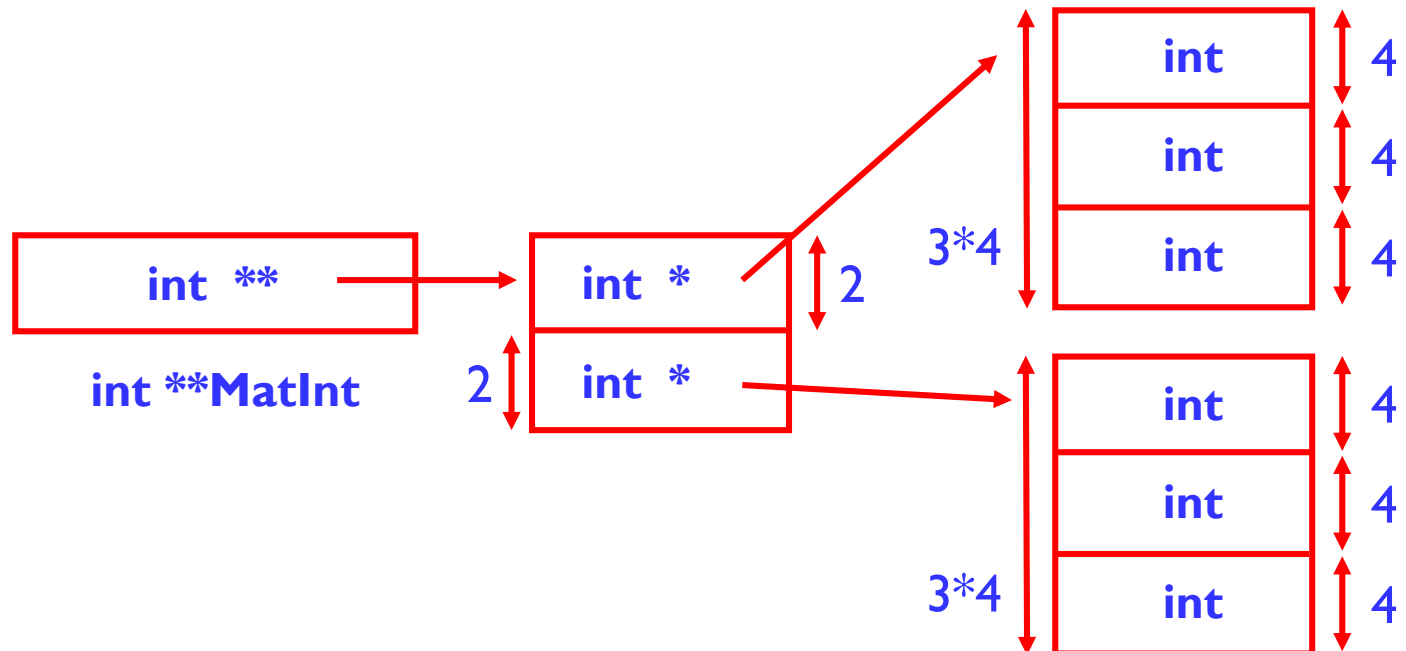
$\text{tipo}_{\text{Mat}} \text{ nome}_{\text{Mat}} [\text{nro}_{\text{righe}}] [\text{nro}_{\text{col}}]$

- Modifiche allo stato della memoria:



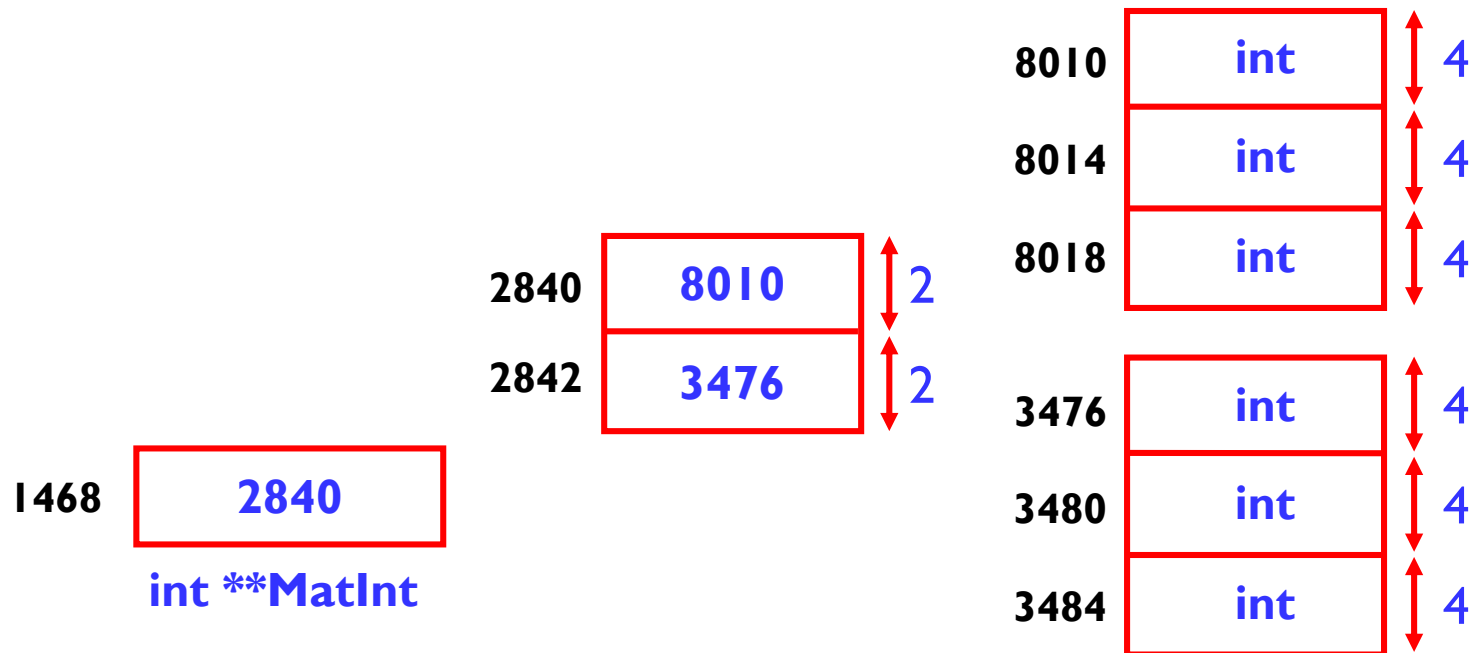
# Definizione statica di una matrice

- **Definizione:** `int MatInt[2][3]`
- **Assumiamo:** `sizeof(int) = 4` e `sizeof (int *) = 2`
- **Modifiche allo stato della memoria:**



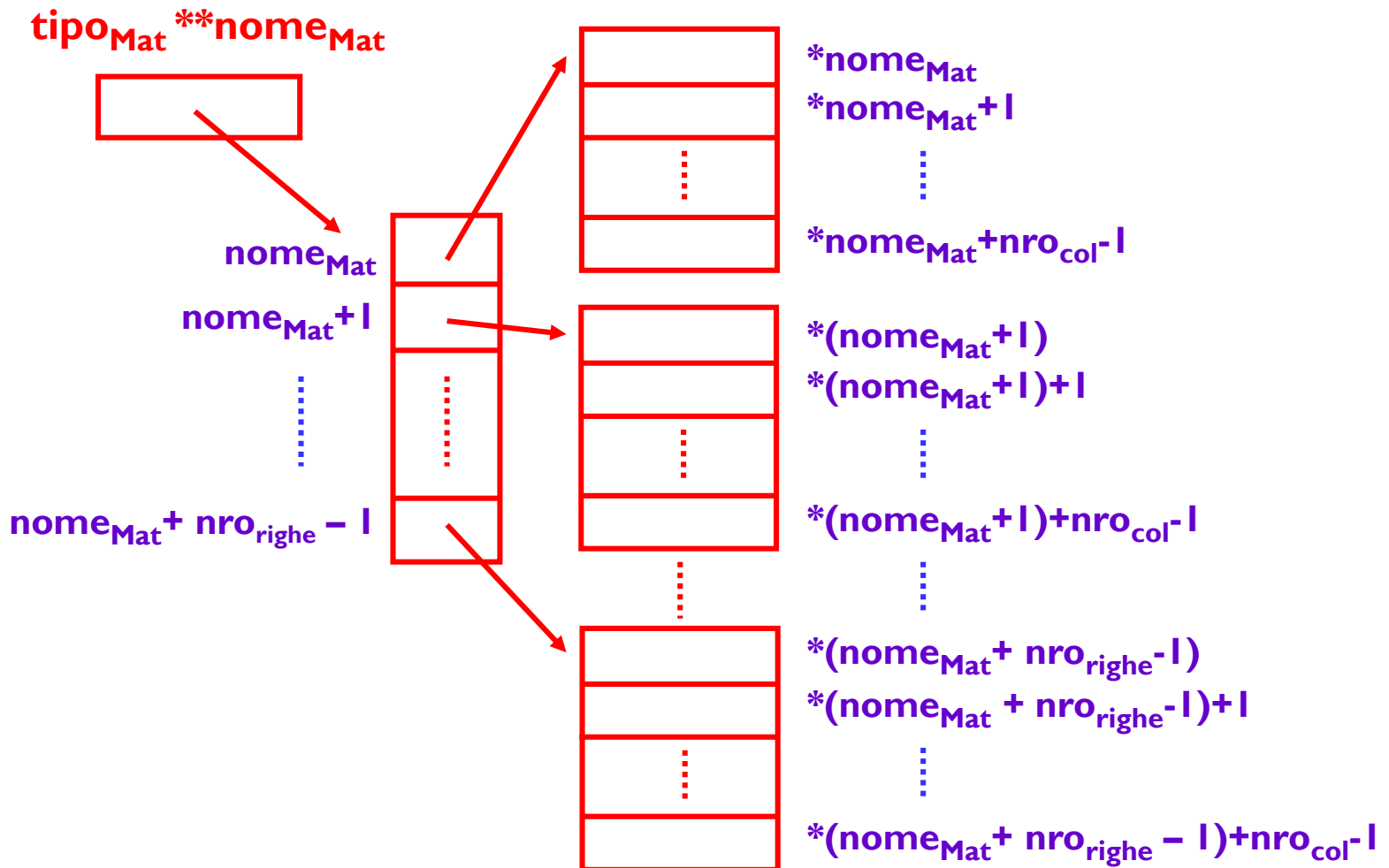
# Definizione statica di una matrice

- **Modifiche allo stato della memoria:**



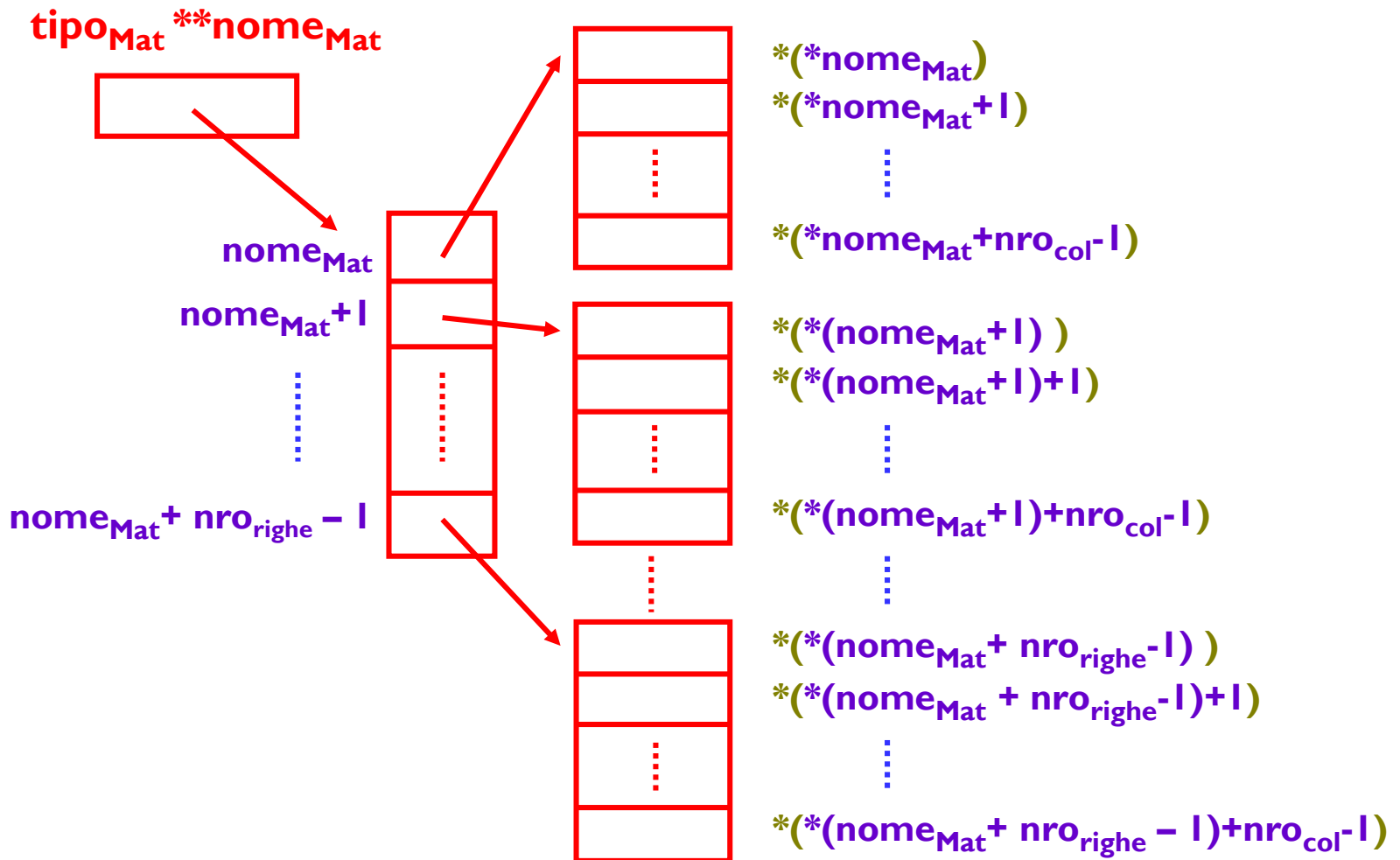
# Accesso agli elementi di una matrice

- Indirizzo:



# Accesso agli elementi di una matrice

- Contenuto:**



# Accesso agli elementi di una matrice

- **Nome:**  $0 \leq \text{espressione a valore intero} \leq \text{nro}_{\text{col}} - 1$

$\text{nome}_{\text{Mat}} [\text{indice}_{\text{riga}}] [\text{indice}_{\text{col}}]$

$0 \leq \text{espressione a valore intero} \leq \text{nro}_{\text{righe}} - 1$

- **Indirizzo:**

$0 \leq \text{espressione a valore intero} \leq \text{nro}_{\text{col}} - 1$

$*(*(\text{nome}_{\text{Mat}} + \text{indice}_{\text{riga}}) + \text{indice}_{\text{col}})$

$0 \leq \text{espressione a valore intero} \leq \text{nro}_{\text{righe}} - 1$

# Allocazione a run-time di una matrice

- **Modifiche allo stato della memoria:**

```
// definisce la  
// variabile di  
// accesso alla  
// matrice
```

```
tipo_Mat **nome_Mat;
```

```
tipo_Mat **nome_Mat
```

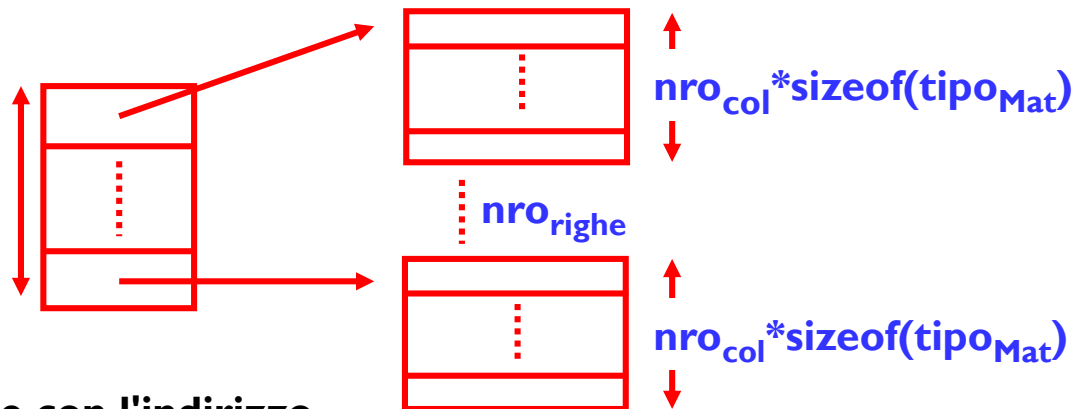


$nro\_righe * sizeof(tipo\_Mat *)$

```
// inizializza ogni elemento del vettore con  
// l'indirizzo di accesso ad un vettore di variabili  
// di tipo tipo_Mat di dimensione pari al numero delle  
// colonne
```

```
for(riga=0; riga < nro_righe; riga++)
```

```
    nome_Mat[riga] = (tipo_Mat *) malloc(nro_col * sizeof(tipo_Mat));
```



```
// inizializza tale variabile con l'indirizzo  
// di un vettore di indirizzi di variabili di tipo tipo_Mat  
// di dimensione pari al numero delle righe
```

```
nome_Mat = (tipo_Mat **) malloc(nro_righe * sizeof(tipo_Mat *))
```

# Rilascio della memoria

- **Modifiche allo stato della memoria:**

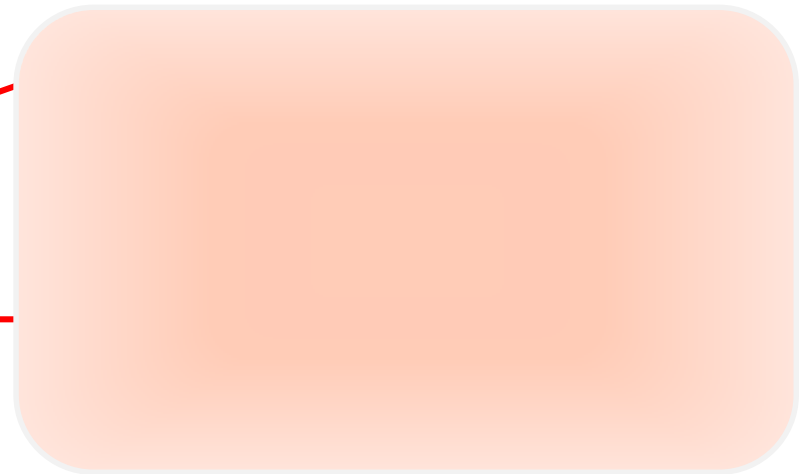
```
// rilascia la memoria allocata per le righe della  
// matrice
```

```
for(riga=0; riga < nro_righe; riga++)  
    free(nome_Mat[riga]);
```

```
tipo_Mat **nome_Mat
```



```
nro_righe * sizeof(*tipo_Mat)
```



```
// rilascia la memoria allocata per  
// le variabili di accesso alle righe della matrice  
free(nome_Mat);
```

# Le Matrici e le funzioni

- **Le matrici come parametri formali:**

```
tipo_fun nome_fun (  
    ...,  
    tipo_Mat ** nome_Mat,  
    int nro_righe,  
    int nro_colonne,  
    ...  
)  
{ ... };
```

- **Le matrici come parametri attuali:**

```
nome_fun (... , nome_Mat, nro_righe, nro_colonne, ...)
```

# I/O di matrici definite a run-time

```
// sorgente: DinMatIO.c
// illustra le modalità di allocazione a run-time, di acquisizione, di
// restituzione, e di rilascio della memoria per una matrice di interi
// inclusione del file di intestazione della libreria standard
// che contiene definizioni di macro, costanti e dichiarazioni
// di funzioni e tipi funzionali alle varie operazioni di I/O
#include <stdio.h>
// inclusione del file di intestazione della libreria standard
// che contiene definizioni di macro, costanti e dichiarazioni
// di funzioni di interesse generale
#include <stdlib.h>
Continua ...
```

# I/O di matrici definite dinamicamente

**Continua ...**

```
// definizione della funzione per il recupero della memoria allocata  
// per una matrice di un qualsiasi numero di colonne
```

```
void FreeMatInt (  
    int **Mat,  
    unsigned int righe  
)
```

```
{  
    // definizione di una variabile per l'indice di riga
```

```
    unsigned int riga;
```

```
    // rilascio della memoria allocata per ognuna delle righe della  
    // matrice
```

```
    for(riga=0; riga< righe; riga++)  
        free(Mat[riga]);
```

```
    // rilascio della memoria allocata per le variabili di accesso alle  
    // righe della matrice
```

```
    free(Mat);  
};
```

**Continua ...**

# I/O di matrici definite dinamicamente

**Continua ...**

```
// definizione della funzione per l'allocazione a run-time della memoria
// per una matrice di interi
int **AllMatInt (
    unsigned int nro_righe,
    unsigned int nro_colonne
)
{
    // definizione di una variabile per l'indirizzo di accesso alla matrice
    // e di una per l'indice di riga
    int** Mat;
    unsigned int riga;
    // inizializzazione della variabile di accesso alla matrice con l'indirizzo
    // di un vettore di riferimenti a variabili intere di dimensione pari al
    // numero delle righe
    Mat = (int **) malloc(nro_righe*sizeof(int *));
    // verifica dell'esito di tale inizializzazione, se negativo termina con
    // NULL
    if (Mat == NULL)
        return(NULL);
}
```

**Continua ...**

# I/O di matrici definite dinamicamente

**Continua ...**

```
// inizializzazione di ogni elemento del vettore con l'indirizzo di un
// vettore di variabili intere di dimensione pari al numero delle
// colonne
for(riga=0; riga< nro_righe; riga++)
{
    Mat[riga] = (int *) malloc(colonne*sizeof(int));
    // verifica dell'esito della inizializzazione, se negativo rilascio di tutta
    // la memoria allocata fino a quel momento e terminazione con
    // NULL
    if (Mat[riga] == NULL)
    {
        FreeMatInt(Mat, riga-1);
        return(NULL);
    };
};
// restituzione dell'indirizzo di accesso alla matrice
return(Mat);
};
```

**Continua ...**

# I/O di matrici definite dinamicamente

**Continua ...**

// funzione per l'acquisizione del contenuto di una matrice di interi

```
void AcqMatInt (
    int **Mat,
    unsigned int dim_righe,
    unsigned int dim_col
)
{
    // definizione delle variabili per l'indice di riga e quello di colonna
    unsigned int riga, col;
    // scansione delle righe della matrice
    for (riga = 0; riga < dim_righe; riga = riga+1)
        // scansione delle colonne della matrice
        for (col = 0; col < dim_col; col = col+1)
            {
                // acquisizione dell'elemento corrente della matrice
                printf("\nMat[%d][%d]: ", riga, col);
                scanf("%d", &(Mat[riga][col]));
            }
};
```

**Continua ...**

# I/O di matrici definite dinamicamente

**Continua ...**

// funzione per la restituzione del contenuto di una matrice di interi

```
void ResMatInt (
    int **Mat,
    unsigned int nro_righe,
    unsigned int nro_col
)
{
    // definizione delle variabili per l'indice di riga e quello di colonna
    unsigned int riga, col;
    // scansione delle righe della matrice
    for (riga = 0; riga < dim_righe; riga = riga+1)
        // scansione delle colonne della matrice
        for (col = 0; col < dim_col; col = col+1)
            // visualizzazione dell'elemento corrente
            printf("\nMat[%d][%d]: %d", riga, col, Mat[riga][col]);
};
```

**Continua ...**

# I/O di matrici definite dinamicamente

**Continua ...**

**// Testing di tutte le funzioni definite in precedenza**

**int main ()**

**{**

**// definizione della variabile per l'indirizzo di accesso alla matrice  
// e per il numero di righe e di colonne**

**int\*\* Matrice;**

**unsigned int nro\_righe, nro\_col;**

**// acquisizione del numero delle righe e delle colonne della matrice**

**printf("Nro righe? "); scanf("%d", &nro\_righe);**

**printf("Nro colonne? "); scanf("%d", &nro\_col);**

**// allocazione della memoria per la matrice e assegnazione**

**// dell'indirizzo di accesso alla variabile preposta**

**Matrice = AllMatInt(nro\_righe, nro\_col);**

**// verifica l'esito dell'allocazione, se negativo terminazione con NULL**

**if (Matrice == NULL)**

**{**

**printf("\nAllocazione di memoria fallita"); return(0);**

**};**

**// acquisizione e restituzione del contenuto della matrice**

**AcqMatInt(Matrice, nro\_righe, nro\_col);**

**ResMatInt(Matrice, nro\_righe, nro\_col);**

**// recupero della memoria allocata per la matrice**

**FreeMatInt(Matrice, nro\_righe);**

**return(1);**

**};**