8.1 Tipi derivati

Tipi fondamentali:

- **da questi si possono *derivare* altri tipi;**
- **dal tipo *int* si deriva il tipo *puntatore a int*.**
  - **variabile appartenente a questo tipo: può assumere come valori indirizzi di interi.**
- **dal tipo *int* si deriva il tipo *array di 4 int*:**
  - **variabile appartenente a questo tipo: può assumere come valori 4 interi.**

Meccanismi di derivazione:

- **possono essere composti fra di loro, permettendo la definizione di tipi derivati arbitrariamente complessi;**
- **prendendo gli interi come base, si possono definire array di puntatori a interi, puntatori ad array di interi, array di array di interi, eccetera.**

Tipi derivati:

- **riferimenti;**
- **puntatori;**
- **array;**
- **strutture;**
- **unioni;**
- **classi.**

8.2 Riferimenti (I)

Riferimento:

- **identificatore che individua un oggetto;**
- **riferimento default: il nome di un oggetto, quando questo è un identificatore.**
- **oltre a quello default, si possono definire altri riferimenti di un oggetto (sinonimi o alias).**

Tipo riferimento:

- **possibili identificatori di oggetti di un dato tipo (il tipo dell'oggetto determina il tipo del riferimento).**

Dichiarazione di un tipo riferimento e definizione di un riferimento sono contestuali.

Sintassi:

basic-reference-definition
reference-type-indicator identifier
***reference-initializer*|opt**

reference-type-indicator
type-indicator &

reference-initializer
= ***object-name***

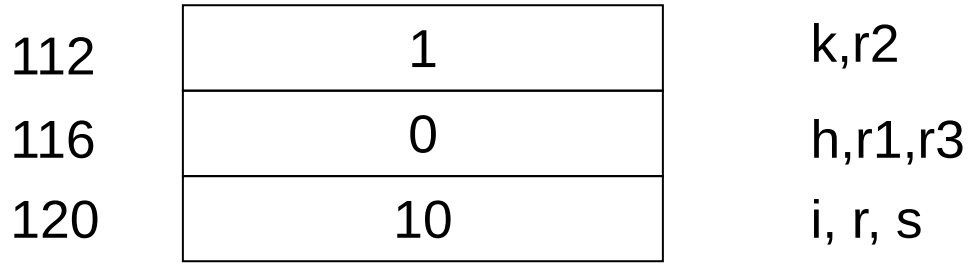
- **indicatore di tipo:**
  - *specifica tipo dell'oggetto riferito;*

Non si possono definire riferimenti di riferimenti.

8.2 Riferimenti (II)

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

int main ()
{
        int i = 10;
        int &r = i;
        int &s = r;
//      int &s;          ERRORE, deve essere sempre iniz.
//      int &s = 10;     ERRORE
        cout << i << '\t' << r << '\t' << s << endl; // 10 10 10
        r++;
        cout << i << '\t' << r << '\t' << s << endl; // 11 11 11

        int h = 0, k = 1;
        int &r1 = h, &r2 = k;   // due riferimenti
        int &r3 = h, r4;        // un riferimento ed un intero

        system("PAUSE");
        return 0;
}
```

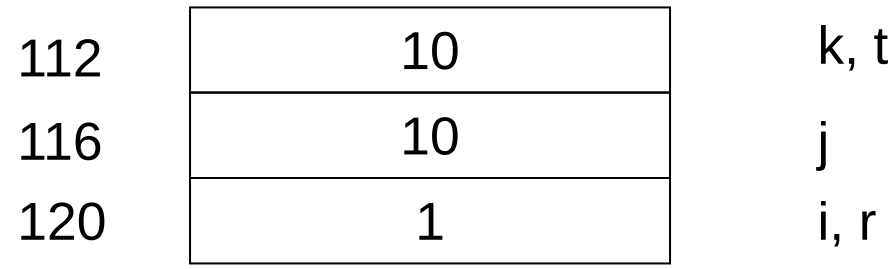
```
10      10      10
11      11      11
Premere un tasto per continuare . . .
```



8.2.1 Riferimenti const (I)

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

int main ()
{
        int i = 1;
        const int &r = i;       // Notare: i non è costante

        int j = 10;
        j = r;                  // OK
        cout << j << endl;      // 1

//      r = 1;                  ERRORE!
        i++;
        const int k = 10;
        const int &t = k;       // OK
        cout << t << endl;      // 10
//      int &tt = k;            ERRORE!
        system("PAUSE");
        return 0;
}
```

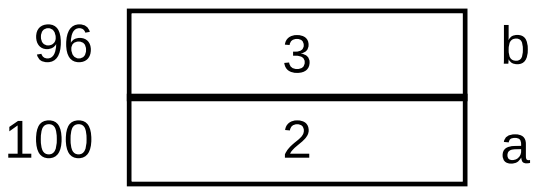
```
1
10
Premere un tasto per continuare . . .
```



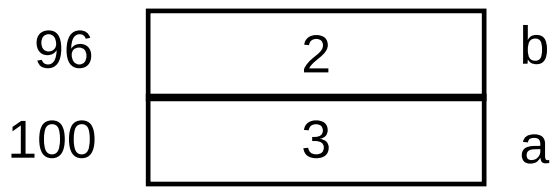
8.2.2 Riferimenti come argomenti (I)

```
// Scambia interi (ERRATO)

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

void scambia (int a, int b)
{    // scambia gli argomenti attuali
     int c = a;
     a = b;
     b = c;
}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                          // Esempio: 2     3
     scambia (i, j);
     cout << i << '\t' << j << endl;         // Esempio: 2     3
     system("PAUSE");
     return 0;
}
```

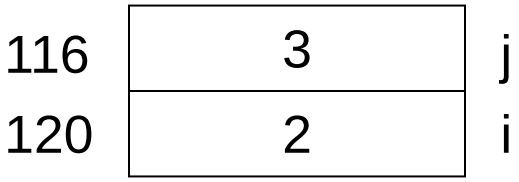
Istanza (scambia)



Istanza (scambia)



Istanza (main)



Istanza (main)

8.2.2 Riferimenti come argomenti (II)

Argomento di una funzione:

- **può essere di un tipo riferimento;**
- **in questo caso:**
  - *l'argomento formale corrisponde a un contenitore senza nome, che ha per valore il riferimento;*
  - *nel corpo della funzione, ogni operazione che coinvolge l'argomento formale agisce sull'entità riferita.*

Chiamata della funzione:

- **il riferimento argomento formale viene inizializzato con un riferimento del corrispondente argomento attuale;**

Argomenti riferimento:

- **devono essere utilizzati quando l'entità attuale può essere modificata.**

In sintesi:

- **la funzione agisce sulle entità riferite dagli argomenti attuali.**

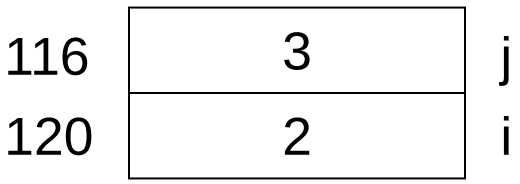
8.2.2 Riferimenti come argomenti (III)

```
// Scambia interi (trasmissione mediante riferimenti)
#include <cstdlib>
#include <iostream>
using namespace std;

void scambia (int &a, int &b)
{    // scambia i valori degli oggetti riferiti
     int c = a;
     a = b;
     b = c;
}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                          // Esempio: 2     3
     scambia (i, j);
     cout << i << '\t' << j << endl;         // Esempio: 3     2
     system("PAUSE");
     return 0;
}
```

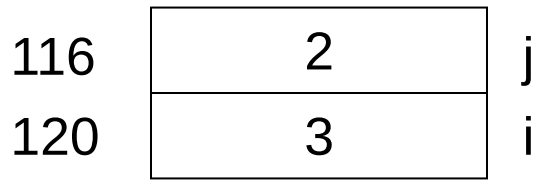
Istanza (scambia)

Istanza (scambia)

Istanza (main)



Istanza (main)



8.2.3 Riferimenti const come argomenti (I)

```
// Calcolo dell'interesse (trasmissione mediante
// riferimenti di oggetti costanti)

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

float interesse(int importo, const float& rateo)
{
     float inter = rateo*importo;   //OK
//   rateo += 0.5;                    ERRORE!
     return inter;
}

int main()
{
   cout << "Interesse : " << interesse(1000,1.2) << endl;
   system("PAUSE");
   return 0;
}
```

```
Interesse : 1200
Premere un tasto per continuare . . .
```

8.2.2 Riferimenti risultato di funzione (I)

```
// Riferimenti come valori restituiti (i)

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

int& massimo(int &a, int &b)
{
    return a > b ? a : b;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;                          // Esempio: 1 3
    cout << "Valore massimo ";
    cout << massimo(i, j) << endl;
    massimo(i, j) = 5;
    cout << i << '\t' << j << endl;         // Esempio: 1 5
    massimo(i, j)++;                        // l-value
    cout << i << '\t' << j << endl;         // Esempio: 1 6
    system("PAUSE");
    return 0;
}
```

```
Inserisci due interi:
1 3
Valore massimo 3
1       5
1       6
Premere un tasto per continuare . . .
```

8.2.2 Riferimenti risultato di funzioni (II)

```
// Riferimenti come valori restituiti (ii)

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

int& massimo(int &a, int &b)
{
    int &p = a > b ? a : b;
    return p;                  // OK. Restituisce un riferimento
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Riferimenti come valori restituiti (iii)

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

int& massimo(int a, int b)
{
    return a > b ? a : b;
    // ERRORE. Riferimento ad un argomento attuale che
    // viene distrutto
}
```

NOTA BENE: l'errore non è segnalato dal compilatore.

8.2.3 Riferimenti risultato di funzioni (III)

```
// Riferimenti const come valori restituiti (iii)

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

const int& massimo(const int& a, const int& b)
{
   return a > b ? a : b;
}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;
     cout << "Valore massimo ";
     cout << massimo(i, j) << endl;

//   massimo(i, j) = 5;                ERRORE
     system("PAUSE");
     return 0;
}
```

```
Inserisci due interi:
1 3
Valore massimo 3
Premere un tasto per continuare . . .
```

8.2.3 Riferimenti (IV)

Argomento formale di una funzione e risultato prodotto da una funzione:

- **possono essere riferimenti con l'attributo *const*.**

Argomento formale con l'attributo *const*:

- **può avere come corrispondente un argomento attuale senza tale attributo, ma non è lecito il contrario.**

Risultato con l'attributo *const*:

- **una istruzione *return* può contenere un'espressione senza tale attributo, ma non è lecito il contrario.**

Operatore *const_cast*:

- **converte un riferimento *const* in un riferimento non *const.***

8.2.3 Riferimenti (V)

```
// Riferimenti come valori restituiti (i)
#include <cstdlib>
#include <iostream>
using namespace std;

int& massimo(const int &a, const int &b)
{
    return const_cast<int&>( a > b ? a : b);
}

//ERRATO
//int& massimoErrato(const int& a, const int& b)
//{
//    return a > b ? a : b;
//}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                              // Esempio: 1 3
     cout << "Valore massimo ";
     cout << massimo(i, j) << endl;
     massimo(i, j) = 5;
     cout << i << '\t' << j << endl;             // Esempio: 1 5
     system("PAUSE");
     return 0;
}
```

```
Inserisci due interi:
1 3
Valore massimo 3
1       5
Premere un tasto per continuare . . .
```

8.2.3 Riferimenti (VI)

Esempio:

- **il risultato della funzione è di tipo *int&*;**
- **nella istruzione *return* compare un riferimento *const*;**
- **si rende opportuna una conversione esplicita di tipo:**

```
int& maxr1(const int& ra, const int& rb)
{
        if (ra >= rb) return const_cast<int&>( ra);
        return const_cast<int&>( rb);
}


const int& maxr1(const int& ra, const int& rb)
{
        if (ra >= rb) return const_cast<int&>( ra);
        return rb;
}
```

8.3 Puntatori (I)

Puntatore:

- **oggetto il cui valore rappresenta l'indirizzo di un altro oggetto o di una funzione.**

Tipo puntatore:

- **insieme di valori: indirizzi di oggetti o di funzioni di un dato tipo (il tipo dell'oggetto o della funzione determina il tipo del puntatore).**

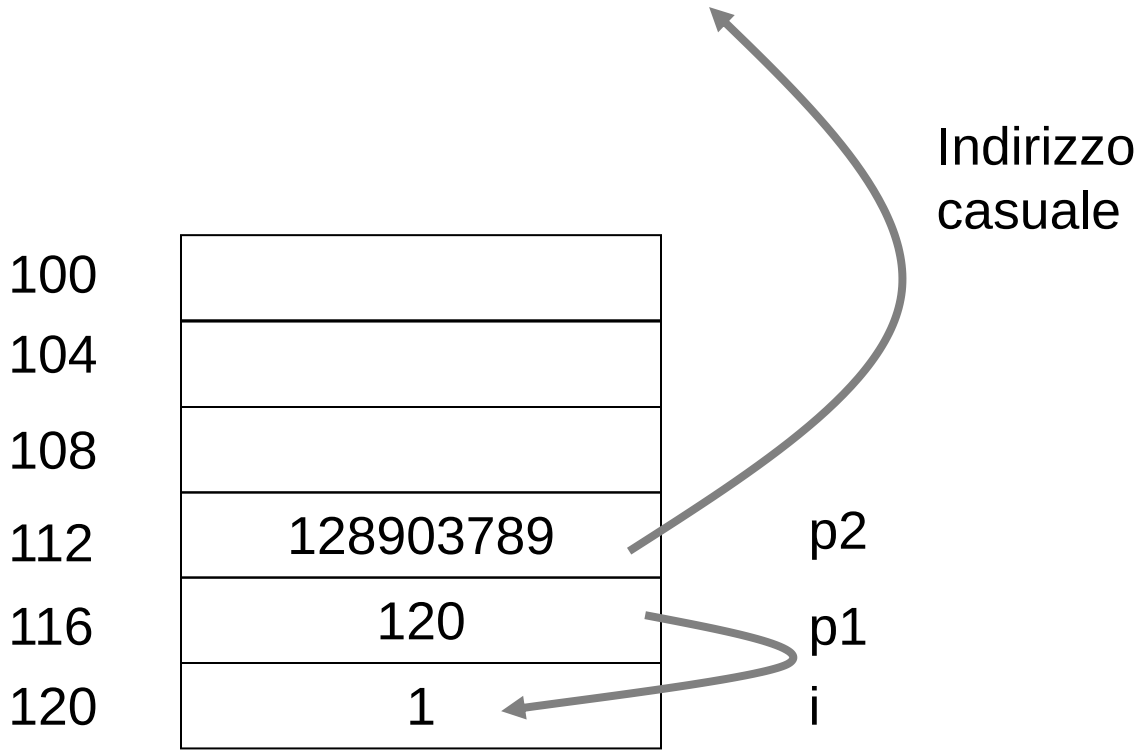
Dichiarazione di un tipo puntatore e definizione di un puntatore sono contestuali.

// Definizione di puntatori

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int *p1;            // puntatore a interi
    int* p2;
    int * p3;

    int *p4, *p5;       // due puntatori
    int *p6, i1;        // un puntatore ed un intero
    int i2, *p7;        // un intero ed un puntatore

    system("PAUSE");
    return 0;
}
```

8.3 Puntatori (II)

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
        int i = 1;
        int *p1 = &i;        // operatore indirizzo
        int *p2;
        …
}
```



8.3 Puntatori (III)

```
// Operatore indirizzo e operatore di indirezione

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int i = 1;
    int *p1 = &i;          // operatore indirizzo
    *p1 = 10;              // operatore di indirezione
                           // restituisce un l-value
    int *p2 = p1;          // Due puntatori allo stesso oggetto

    cout << i << '\t'<< *p1 << '\t' << *p2 << endl;

    *p2 = 20;
    cout << i << '\t'<< *p1 << '\t' << *p2 << endl;

    cout << p1 << '\t' << p2 << endl;

//  p2 = *p1;          // ERRORE: assegna un int ad un punt.
//  *p2 = p1           // ERRORE: assegna un punt. ad un int.

    int *p3;
    *p3 = 2;               // ATTENZIONE: puntatore non iniz.

    system("PAUSE");
    return 0;
}
```

```
10      10      10
20      20      20
0x22ff6c        0x22ff6c
Premere un tasto per continuare . . .
```

8.3 Puntatori (IV)

```
// Puntatori allo stesso oggetto

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    char a, b;
    char *p = &a, *q = &b;
    cout << "Inserisci due caratteri " << endl;
    cin >> a >> b;                          // Esempio: 'a'    'b'
    *p = *q;
    cout << a << '\t' << b << endl;     // Esempio: 'b'    'b'
    cout << *p << '\t' << *q << endl;  // Esempio: 'b'    'b'

    cout << "Inserisci due caratteri " << endl;
    cin >> a >> b;                          // Esempio: 'c'    'f'
    p = q;
    cout << a << '\t' << b << endl;     // Esempio: 'c'    'f'
    cout << *p << '\t' << *q << endl;  // Esempio: 'f'    'f'

    system("PAUSE");
    return 0;
}
```

```
Inserisci due caratteri
ab
b       b
b       b
Inserisci due caratteri
cf
c       f
f       f
Premere un tasto per continuare . . .
```

8.3 Puntatori (V)

```
// Puntatori a costanti

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
     int i = 0;
     const int *p;                  // Nessuna inizializzazione

     p = &i;                        // OK
      // N.B.: i non e` costante

     int j;
     j = *p;                  // OK
//   *p = 1;                  // ERRORE! Il valore di i non puo'
                              // essere modificato attraverso p

     const int k = 10;
     const int *q = &k;       // OK

     int *qq;
//   qq = &k;                 // ERRORE! int* = const int*
//   qq = q;                  // ERRORE! int* = const int*

     system("PAUSE");
     return 0;
}
```

8.3 Puntatori (VI)

```
// Puntatori costanti

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
     char c = 'a';
     char *const p = &c;                // Sempre inizializzato
     cout << *p << endl;                // 'a'

     *p = 'b';
     cout << *p << endl;                // 'b'

     char d = 'c';
//   p = &d;                            // ERRORE!

     char  *p1, *const p2 = &d;
     p1 = p;
     cout << *p1 << endl;               // 'b'
//   p = p1;                            // ERRORE
//   p = p2;                            // ERRORE!

     system("PAUSE");
     return 0;
}
```

```
a
b
b
Premere un tasto per continuare . . .
```

8.3 Puntatori (VII)

```
// Puntatore a puntatore

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int i = 1, j = 10;
    int *pi = &i, *pj = &j;
    int **q1 = &pi;
    cout << **q1 << endl;                    // 1
    *q1 = pj;
    cout << **q1 << endl;                    // 10

    int **q2;
    *q2 = pj;      // ATTENZIONE: nessun oggetto puntato

    system("PAUSE");
    return 0;
}
```

```
1
10
Premere un tasto per continuare . . .
```

8.3 Puntatori (VIII)

```
// Puntatori nulli

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int *p = NULL;          // forma equivalente: int *p = 0;

    *p = 1;                 // ERRORE IN ESECUZIONE!

    if (p == NULL)
        cout << "Puntatore nullo " << endl;
    if (p == 0)
        cout << "Puntatore nullo " << endl;
    if (!p)
        cout << "Puntatore nullo " << endl;

    system("PAUSE");
    return 0;
}
```

8.3 Operazioni sui puntatori (I)

Operazioni possibili:

- **assegnamento di un'espressione che produce un valore indirizzo ad un puntatore;**
- **uso di un puntatore per riferirsi all'oggetto puntato;**
- **confronto fra puntatori mediante gli operatori '==' e '!=';**
- **Stampa su uscita standard utilizzando l'operatore di uscita '<<'. In questo caso, viene stampato il valore in esadecimale del puntatore ossia l'indirizzo dell'oggetto puntato.**

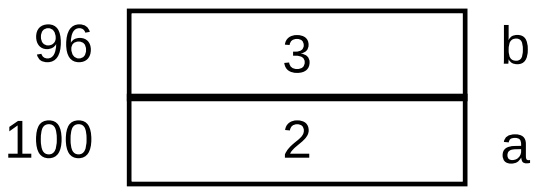
Un puntatore può costituire un argomento di una funzione:

- **nel corpo della funzione, per mezzo di indirezioni, si possono modificare gli oggetti puntati.**

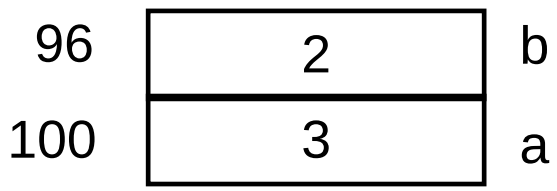
8.3.4 Puntatori come argomenti (I)

```
// Scambia interi (ERRATO)

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

void scambia (int a, int b)
{    // scambia gli argomenti attuali
     int c = a;
     a = b;
     b = c;
}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                          // Esempio: 2     3
     scambia (i, j);
     cout << i << '\t' << j << endl;         // Esempio: 2     3
     system("PAUSE");
     return 0;
}
```

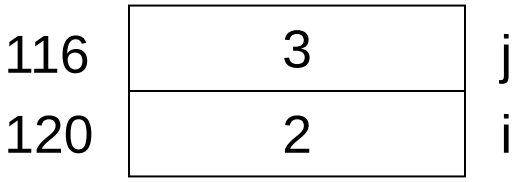
Istanza (scambia)



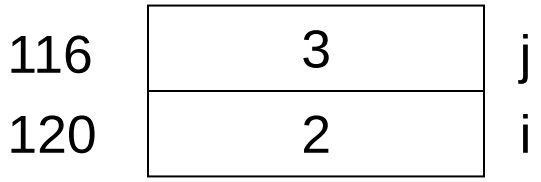
Istanza (scambia)



Istanza (main)



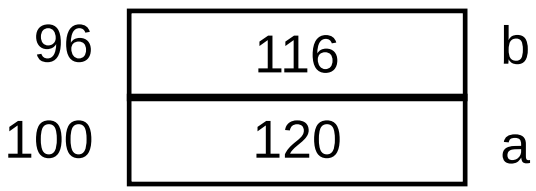
Istanza (main)



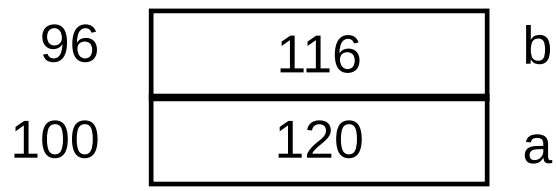
8.3.4 Puntatori come argomenti (II)

```
// Scambia interi (trasmissione mediante puntatori)

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

void scambia (int *a, int *b)
{    // scambia i valori degli oggetti puntati
     int c = *a;
     *a = *b;
     *b = c;
}

int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                          // Esempio: 2     3
     scambia (&i, &j);
     cout << i << '\t' << j << endl;         // Esempio: 3     2
     system("PAUSE");
     return 0;
}
```

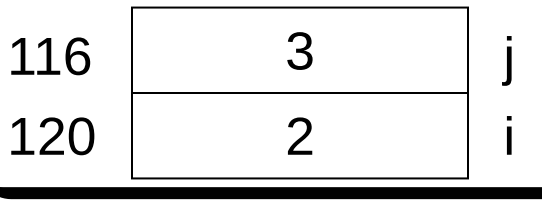
Istanza (scambia)



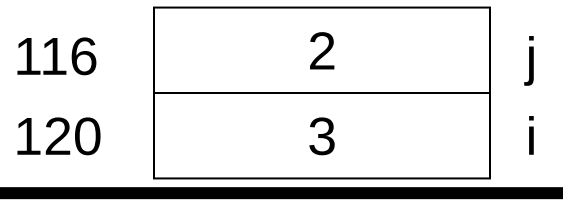
Istanza (scambia)



Istanza (main)

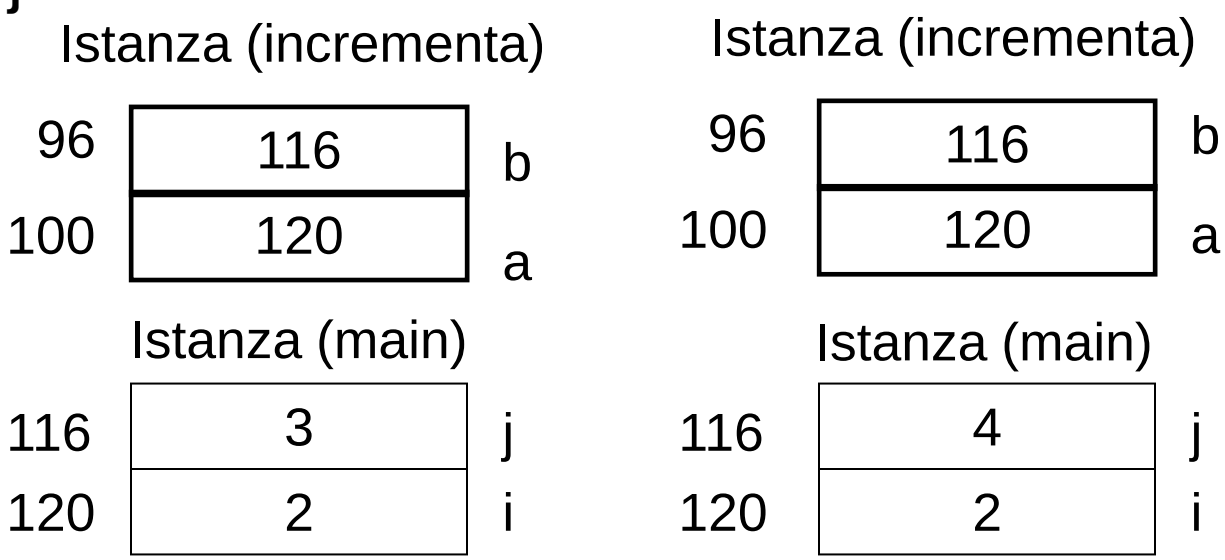


Istanza (main)



8.3.4 Puntatori come argomenti (III)

```
// Incrementa il maggiore tra due interi
#include <cstdlib>
#include <iostream>
using namespace std;
void incrementa(int *a, int *b)
{
     if (*a > *b)
          (*a)++;                              // ATTENZIONE *a++
     else
          (*b)++;
}
int main ()
{
     int i, j;
     cout << "Inserisci due interi: " << endl;
     cin >> i >> j;                            // Esempio: 2 3
     incrementa(&i, &j);
     cout << i << '\t' << j << endl;           // Esempio: 2 4
     system("PAUSE");
     return 0;
}
```



8.3.4 Puntatori come risultato di funzioni (I)

```
// Puntatori come valori restituiti (i)

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

int* massimo(int *a, int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i, j;
    cout << "Inserisci due interi: " << endl;
    cin >> i >> j;                          // Esempio: 2 3
    cout << "Valore massimo ";
    cout << *massimo(&i, &j) << endl;

    *massimo(&i, &j) = 5;
    cout << i << '\t' << j << endl;         // Esempio: 2 5

//  massimo(&i, &j)++;
// ERRORE: il valore restituito non e' un l-value

    (*massimo(&i, &j))++;
    cout << i << '\t' << j << endl;         // Esempio: 2 6

    system("PAUSE");
    return 0;
}
```

```
Inserisci due interi:
2 3
Valore massimo 3
2       5
2       6
Premere un tasto per continuare . . .
```

8.3.4 Puntatori come risultato di funzioni (II)

```
// Puntatori come valori restituiti (ii)

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

int* massimo(int *a, int *b)
{
    int *p = *a > *b ? a : b;
    return p;                    // Restituisce un puntatore
}

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Puntatori come valori restituiti (iii)

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

int* massimo(int *a, int *b)
{
    int i = *a > *b ? *a : *b;
    return &i;
    // ATTENZIONE: restituisce l'indirizzo di una variabile
    // locale
    // [Warning] address of local variable 'i' returned
}
```

8.3.4 Puntatori come argomenti costanti (I)

```
// Trasmissione dei parametri

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

int* massimo(int *a, int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;
    // ERRORE: invalid  conversion from 'const int*' to 'int*'

    system("PAUSE");
    return 0;
}
```

8.3.4 Puntatori come argomenti costanti (II)

```
// Trasmissione di parametri mediante puntatori a
// costanti

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

int* massimo(const int *a, const int *b)
{
    return const_cast<int*> (*a > *b ? a : b);
}

/*
int* massimo(const int *a, const int *b)
{
    return *a > *b ? a : b;
    // ERRORE: invalid  conversion from 'const int*' to 'int*'
}
*/

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;
    system("PAUSE");
    return 0;
}
```

8.3.4 Puntatori come risultato di funzioni

```
// Puntatori a costanti come valori restituiti

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

const int* massimo(const int *a, const int *b)
{
    return *a > *b ? a : b;
}

int main ()
{
    int i = 2;
    const int N = 3;
    cout << "Valore massimo ";
    cout << *massimo(&i, &N) << endl;

    //   int *p1 =  massimo(&i, &N);      ERRORE

    const int *p2 = massimo(&i, &N);

    //   *massimo(&i, &N) = 1;            ERRORE

    system("PAUSE");
    return 0;
}
```

```
Valore massimo 3
Premere un tasto per continuare . . .
```

9.1 Tipi e oggetti array (I)

Array di dimensione *n*:

- **n-upla ordinata di elementi dello stesso tipo, ai quali ci si riferisce mediante un indice, che rappresenta la loro posizione all'interno dell'array.**

Tipo dell'array:

- **dipende dal tipo degli elementi.**

Dichiarazione di un tipo array e definizione di un array sono contestuali.

```
// Somma gli elementi di un dato vettore di interi
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    const int N = 5;
    int v[N];              // dimensione del vettore costante
    cout << "Inserisci 5 numeri interi " << endl;
    for (int i = 0; i < N; i++)
        cin >> v[i];       // operatore di selezione con indice
    int s = v[0];          // restituisce un l-value
    for (int i = 1; i < N; i++)
        s += v[i];
    cout << s << endl;
    system("PAUSE");
    return 0;
}
```

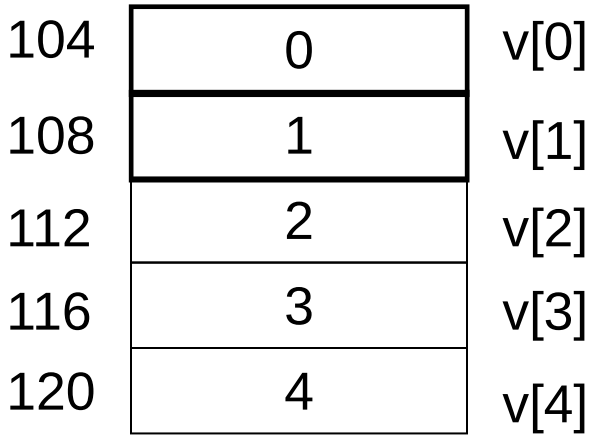
```
Inserisci 5 numeri interi
1 2 3 4 5
15
Premere un tasto per continuare . . .
```

9.1 Tipi e oggetti array (II)

ATTENZIONE: l'identificatore dell'array identifica l'indirizzo del primo elemento dell'array

v = &v[0]

Nell'esempio, v = 104;



9.1 Tipi e oggetti array (III)

```
// Inizializzazione degli array

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    const int N = 6;
    int a[] = {0, 1, 2, 3};              // array di 4 elementi
    int b[N] = {0, 1, 2, 3};             // array di N elementi
    cout << "Dimensioni array: ";
    cout << sizeof a << '\t' << sizeof b << endl;    // 16  24
    cout << "Numero di elementi: ";
    cout << sizeof a / sizeof(int) << '\t';          // 4
    cout << sizeof b / sizeof(int) << endl;          // 6

    // ERRORE! NON SEGNALATO IN COMPILAZIONE
    // Nessun controllo sul valore degli indici
    for (int i = 0; i < N; i++)
        cout << a[i] << '\t';
     cout << endl;

     for (int i = 0; i < N; i++)
        cout << b[i] << '\t';
    cout << endl;
    system("PAUSE");
    return 0;
}
```

```
Dimensioni array: 16    24
Numero di elementi: 4   6
0       1       2       3       37879712        2009179755
0       1       2       3       0       0
Premere un tasto per continuare . . .
```

9.1 Tipi e oggetti array (IV)

```
// ATTENZIONE: il DEV-C++ NON SEGUE LO STANDARD
// Per imporre il rispetto dello standard si deve usare
// l'opzione -pedantic-errors.
// L'opzione si puo' inserire selezionando dal menu'
// Strumenti, la voce "Opzioni di Compilazione"


#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
      int n = 2;
//    int a[n];
//    ERRORE: numero degli elementi non costante

      system("PAUSE");
      return 0;
}
```

Il compilatore usato senza l'opzione -pedantic-errors non segnala l'errore.
Il codice scritto non è comunque portabile (un altro compilatore potrebbe non compilarlo).

9.1 Tipi e oggetti array (V)

```
// Operazioni sugli array. NON SONO PERMESSE
// OPERAZIONI ARITMETICHE, DI CONFRONTO, DI
// ASSEGNAMENTO

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

int main ()
{
    const int N = 5;
    int u[N] = {0, 1, 2, 3, 4}, v[N] = {5, 6, 7, 8, 9};
//  v = u;        ERRORE: assegnamento non permesso

    cout << "Ind. v:" << v << "\t Ind. u: " << u << endl;

    if (v == u)            // Attenzione confronta gli indirizzi
        cout << "Array uguali " << endl;
    else
        cout << "Array diversi " << endl;
    if (v > u) // operatori di confronto agiscono sugli indirizzi
        cout << "Indirizzo v > u " << endl;
    else
        cout << "Indirizzo v <= u " << endl;

    // v + u;     operatori aritmetici non definiti

    system("PAUSE");
    return 0;
}
```

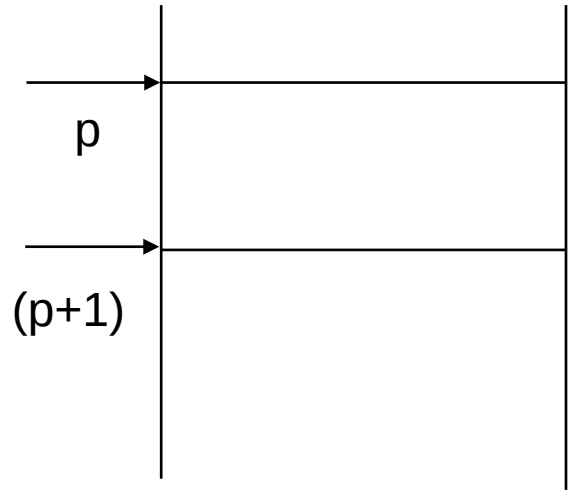
```
Ind. v:0x22ff18  Ind. u: 0x22ff38
Array diversi
Indirizzo v <= u
Premere un tasto per continuare . . .
```

8.3.3 Array e puntatori (I)

// Aritmetica dei puntatori

Permette di calcolare indirizzi con la regola seguente:

- **se l'espressione *p* rappresenta un valore indirizzo di un oggetto di tipo *T*, allora l'espressione (*p*+1) rappresenta l'indirizzo di un oggetto, sempre di tipo T, che si trova consecutivamente in memoria.**



In generale:

- **se *i* è un intero, allora l'espressione (*p*+*i*) rappresenta l'indirizzo di un oggetto, sempre di tipo T, che si trova in memoria, dopo *i* posizioni.**

Nota:

- **Se l'espressione *p* ha come valore *addr* e se *T* occupa *n* locazioni di memoria, l'espressione *p*+*i* ha come valore *addr*+*n***i*.**

Aritmetica dei puntatori:

- **si utilizza quando si hanno degli oggetti dello stesso tipo in posizioni adiacenti in memoria (array).**

8.3.3 Array e puntatori (II)

```
// Aritmetica dei puntatori

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
     int v[4];
     int *p = v;                                   // v <=> &v[0]

     *p = 1;
     *(p + 1) = 10;
     p += 3;
     *(p - 1) = 100;
     *p = 1000;
     p = v;
     cout << "v[" << 4 << "] = [" << *p;
     for (int i = 1; i < 4; i++)
          cout <<  '\t' << *(p + i);       // v[4] = [1 10 100 1000]
     cout << ']' << endl;
     cout << p + 1 - p << endl;      // 1 aritmetica dei puntatori
     cout << int(p + 1) - int(p) << endl;          // 4 (byte)

     char c[5];
     char* q = c;
     cout << int(q + 1) - int(q) << endl;          // 1 (byte)

     int* p1 = &v[1];
     int* p2 = &v[2];
     cout << p2 - p1 << endl;                      // 1 (elementi)
     cout << int(p2) - int(p1) << endl;            // 4 (byte)
     system("PAUSE");
     return 0;
}
```

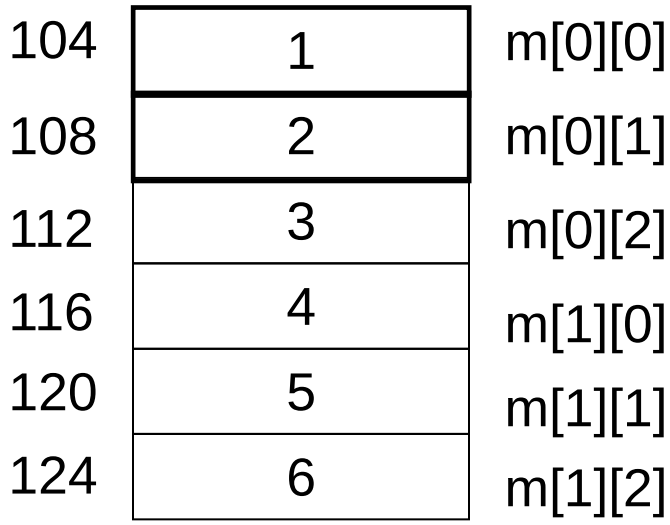
8.3.3 Array e puntatori (III)

```
// Inizializza a 1

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
     const int N = 5;
     int v[N];

     int *p = v;
     while (p <= &v[N-1])
          *p++ = 1;                              // *(p++)

     p = v;
     cout << "v[" << N << "] = [" << *p;
     for (int i = 1; i < N; i++)
          cout <<  '\t' << *(p + i);             // v[5] = [1 1 1 1 1]
     cout << ']' << endl;

     system("PAUSE");
     return 0;
}
```

```
v[5] = [1       1       1       1       1]
Premere un tasto per continuare . . .
```

9.2 Array multidimensionali (I)

```
// Legge e scrive gli elementi di una matrice di R righe
// e C colonne

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

int main ()
{
    const int R = 2;
    const int C = 3;
    int m[R][C];

    cout << "Inserisci gli elementi della matrice" << endl;
    for (int i = 0; i < R; i++)
        for (int j = 0; j < C; j++)
            cin >> m[i][j];                    // 1 2 3 4 5 6

    int* p = &m[0][0];                         // anche: m[0]

    for (int i = 0; i < R; i++)
    {                                          // memorizzazione per righe
        for (int j = 0; j < C; j++)
            cout << *(p + i*C + j) << '\t';
        cout <<  endl;
    }
    system("PAUSE");
    return 0;
}
```




```
Inserisci gli elementi della matrice
1 2 3 4 5 6
1       2       3
4       5       6
Premere un tasto per continuare . . .
```

9.2 Array multidimensionali (II)

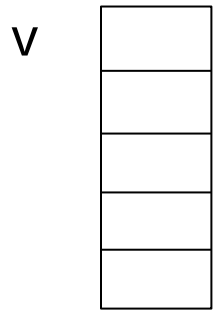
```
// Inizializzazione di vettori multidimensionali

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    int m1[2][3] = {1, 2, 3, 4, 5, 6};    // anche: int m1[][3]
    cout << "Matrice m1 " << endl;
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << m1[i][j] << '\t';
        cout << endl;
    }
    int m2[3][3] = {{0, 1, 2}, {10, 11}, {100, 101, 102}};
    // anche: int m2[][3]. N.B.:  m2[1][2] inizializzato a 0
    cout << "Matrice m2 " << endl;
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << m2[i][j] << '\t';
        cout << endl;
    }
    system("PAUSE");
    return 0;
}
```

```
Matrice m1
1       2       3
4       5       6
Matrice m2
0       1       2
10      11      0
100     101     102
Premere un tasto per continuare . . .
```

Vettore


```
const int N = 5;
int v[N];
```

allocazione in memoria



v è di tipo: int[N]

Accesso all'elemento in posizione i

```
v[i] = 2;
```

con indice

```
int* q = v;
*(q+i) = 2;
```

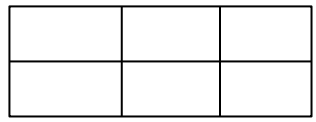
con aritmetica dei puntatori

Conversione implicita del nome del vettore v a puntatore al tipo degli elementi:
v conversione implicita a int*

La notazione v[i] viene convertita dal compilatore in *(v+i)

Matrice

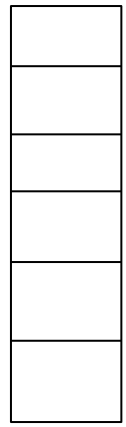
m



```
const int R = 2;
const int C = 3;
int m[R][C];
```

allocazione in memoria

m



vettore di R elementi il cui tipo è:
vettore di C elementi int [C]

Accesso all'elemento in posizione (i,j)

```
m[i][j] = 3;
```

con indice

```
int* p = &m[0][0];
*(p + i*C +j) = 3;
```

con aritmetica dei puntatori

Conversione implicita del nome della matrice a puntatore a vettore di C elementi

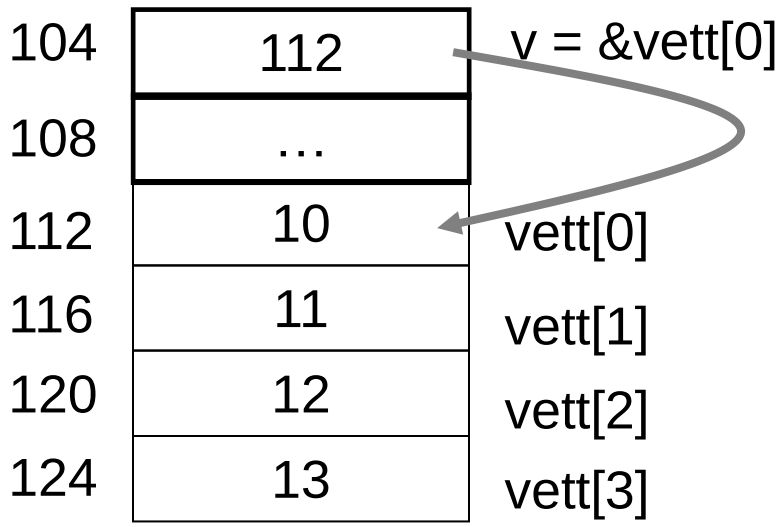
m conversione implicita a int*[C]

```
int* p = m;
```

errore di tipo. Il tipo di m è int*[C]

9.4 Array come argomenti di funzioni (I)

```
// Somma gli elementi di un dato vettore di interi (i)

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

int somma(int v[4])        // La dimensione non ha significato
{          // anche: (int v[])
           // X v[] <=> X *v
     int s = 0;
     for (int i = 0; i < 4; i++)
          s += v[i];
     return s;
}

int main ()
{
     int vett[] = {10, 11, 12, 13};
     cout << "La somma degli elementi e': ";
     cout  << somma(vett) << endl;                    // 46
     system("PAUSE");
     return 0;
}
```



9.4 Array come argomenti di funzioni (II)

```
// Somma gli elementi di un dato vettore di interi (ii)
// (ERRATO)

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

int somma(int v[])
{
    int s = 0;
    int n = sizeof v / sizeof(int);                    // 1
    for (int i = 0; i < n; i++)
        s += v[i];
    return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ";
    cout << somma(vett) << endl;                       // 10
    system("PAUSE");
    return 0;
}
```

```
La somma degli elementi e':10
Premere un tasto per continuare . . .
```

9.4 Array come argomenti di funzioni (III)

```
// Somma gli elementi di un dato vettore di interi (iii)

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

int somma(int v[], int n)      // Dimensione come argomento
{
        int s = 0;
        for (int i = 0; i < n; i++)
            s += v[i];
        return s;
}

int main ()
{
     int vett[] = {10, 11, 12, 13};
     cout << "La somma degli elementi e': ",
     cout  << somma(vett, sizeof vett / sizeof(int)) << endl;
     system("PAUSE");
     return 0;
}
```

```
La somma degli elementi e': 46
Premere un tasto per continuare . . .
```

9.4 Array come argomenti di funzioni (IV)

```
// Incrementa gli elementi di un dato vettore di interi (i)

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

void stampa(int v[], int n)
{
     if (n > 0)
     {
          cout << '[' << v[0];
          for (int i = 1; i < n; i++)
               cout << ' ' << v[i];
          cout << ']' << endl;
     }
}

void incrementa(int v[], int n)
{
    for (int i = 0; i < n; i++)
          v[i]++;
}

int main ()
{
     int vett[] = {10, 11, 12, 13};
     stampa(vett, 4);
     incrementa(vett, 4);
     stampa(vett, 4);
     system("PAUSE");
     return 0;
}
```

```
[10 11 12 13]
[11 12 13 14]
Premere un tasto per continuare . . .
```

9.4 Argomenti array costanti (I)

```
// Somma gli elementi di un dato vettore di interi (iv)

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

int somma(const int v[], int n)      // gli elementi dell'array
{                                    // non possono essere
        int s = 0;                   // modificati
        for (int i = 0; i < n; i++)
            s += v[i];
        return s;
}

int main ()
{
    int vett[] = {10, 11, 12, 13};
    cout << "La somma degli elementi e': ",
    cout  << somma(vett, sizeof vett / sizeof(int)) << endl;
    system("PAUSE");
    return 0;
}
```

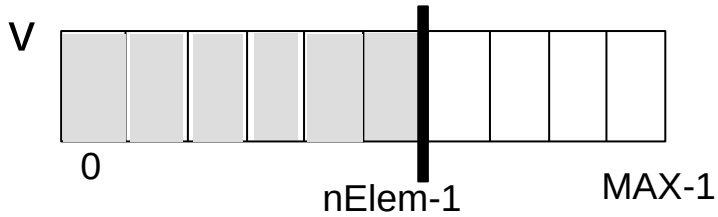
```
La somma degli elementi e': 46
Premere un tasto per continuare . . .
```

9.4 Argomenti array costanti (II)

```
// Incrementa gli elementi di un dato vettore di interi (ii)

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

void stampa(const int v[], int n)            // OK!!!
{
     if (n > 0)
     {
          cout << "[" << v[0];
          for (int i = 1; i < n; i++)
               cout << ' ' << v[i];
          cout << ']' << endl;
     }
}

void incrementa(const int v[], int n)       // ERRORE
{
    for (int i = 0; i < n; i++)
          v[i]++;
}

int main ()
{
     int vett[] = {10, 11, 12, 13};
     stampa(vett, 4);
     incrementa(vett, 4);
     stampa(vett, 4);
     system("PAUSE");
     return 0;
}
```

9.5.3 Argomenti array costanti (III)

```
// Trova il massimo valore in un dato vettore di interi

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

void leggi(int v[], int n)
{
    for (int i = 0; i < n; i++)
    {   cout << '[' << i << "] = ";     cin >> v[i];    }
}

int massimo(const int v[], int n)
{
     int m = v[0];
     for (int i = 1; i < n; i++)
          m = m >= v[i] ? m : v[i];
     return m;
}

int main ()
{
     const int MAX = 10; int v[MAX], nElem;
     cout << "Quanti elementi? ";
     cin >> nElem;
     leggi(v, nElem);
     cout << "Massimo: " << massimo(v, nElem) << endl;
     system("PAUSE");
     return 0;
}
```




```
Quanti elementi? 2
[0] = 13
[1] = 45
Massimo: 45
Premere un tasto per continuare . . .
```

9.5.3 Array multidimensionali (I)

```
// La dichiarazione di un vettore a piu' dimensioni come
// argomento formale deve specificare la grandezza di
// tutte le dimensioni tranne la prima.
// Se M e' il numero delle dimensioni, l'argomento
// formale e' un puntatore ad array M-1 dimensionali

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

const int C = 3;

void inizializza(int m[][C], int r)
{
    for (int i = 0; i < r; i++)
        for (int j = 0; j < C; j++)
            m[i][j] = i + j;
}

void dim(const int m[][C])
{
    cout << "Dimensione (ERRATA) ";
    cout <<  sizeof m / sizeof(int) << endl;
}

// void riempiErrata(int m[][]);
```

ERRORE!

9.5.3 Array multidimensionali (II)

```
void stampa(const int m[][C], int r)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < C; j++)
            cout << m[i][j] << '\t';
        cout << endl;
    }
}

int main ()
{
    int mat1[2][C], mat2[2][5];
    inizializza(mat1, 2);
    dim(mat1);
    stampa(mat1, 2);
    // inizializza(mat2, 2);          ERRORE: tipo arg. diverso
    system("PAUSE");
    return 0;
}
```

```
Dimensione (ERRATA) 1
0       1       2
1       2       3
Premere un tasto per continuare . . .
```

9.5.3 Array multidimensionali (III)

```
// Trasmissione mediante puntatori

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

void inizializza(int* m, int r, int c)
{
    for (int i = 0; i < r; i++)
        for (int j = 0; j < c; j++)
            *(m + i * c + j) = i + j;
}

void stampa(const int* m, int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < c; j++)
            cout << *(m + i * c + j) << '\t';
        cout << endl;
    }
}

int main ()
{
    int mat1[2][3], mat2[2][6];
//  inizializza(mat1, 2, 3);    ERRORE  passing 'int (*)[3]' as
//                  argument 1 of 'inizializza(int *, int, int)'
    inizializza(&mat1[0][0], 2, 3);
    stampa((int*) mat1, 2, 3);
    inizializza((int*) mat2, 2, 6);
    stampa(&mat2[0][0], 2, 6);
    system("PAUSE");
    return 0;
}
```

9.3 Stringhe (I)

Stringa:

- **Sequenza di caratteri.**

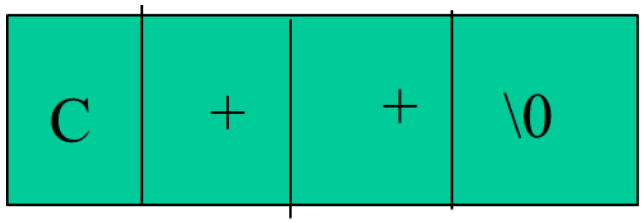
In C++ non esiste il tipo stringa.
Variabili stringa:

- **array di caratteri, che memorizzano stringhe (un carattere per elemento) e il carattere nullo ('\0') finale.**

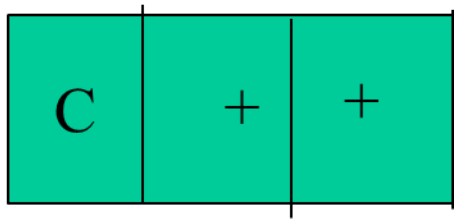
```
// Lunghezza ed inizializzazione di stringhe

#include <cstdlib>
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    char c1[] = "C++";                  // inizializzazione
    cout << sizeof c1 << endl;          // 4
    cout << strlen(c1) << endl;         // 3

    char c2[] = {'C', '+', '+'};        // Manca il '\0'!
    cout << sizeof c2 << endl;          // 3

    char c3[] = {'C', '+', '+', '\0'};  // OK
    cout << sizeof c3 << endl;          // 4

    char c4[4];
//  c4 = "C++";     ERRORE! Assegnamento tra array
    system("PAUSE");
    return 0;
}
```

```
4       3       3       4
Premere un tasto per continuare . . .
```

c1



c2



9.3 Stringhe (II)

Operatori di ingresso e di uscita:

- **accettano una variabile stringa come argomento.**

Operatore di ingresso:

- **legge caratteri dallo stream di ingresso (saltando eventuali caratteri bianchi di testa) e li memorizza in sequenza, fino a che incontra un carattere spazio: un tale carattere (che non viene letto) causa il termine dell'operazione e la memorizzazione nella variabile stringa del carattere nullo dopo l'ultimo carattere letto;**
- **l'array che riceve i caratteri deve essere dimensionato adeguatamente.**

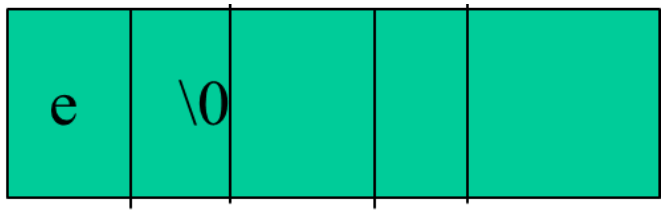
Operatore di uscita:

- **scrive i caratteri della stringa (escluso il carattere nullo finale) sullo stream di uscita.**

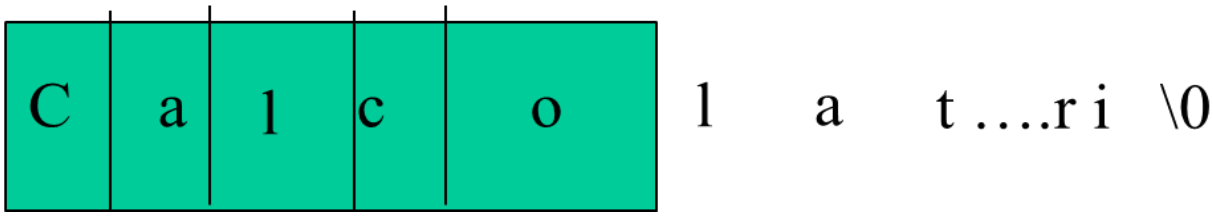
```
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    char stringa[12];     // al piu` 11 caratteri oltre a '\0'
    cout << "? ";
    cin >> stringa;       // Esempio: Informatica e Calcolatori
                  // Attenzione nessun controllo sulla dimensione
    cout << stringa << endl;       // Esempio: Informatica
    system("PAUSE");
    return 0;
}
```

```
? Informatica e Calcolatori
Informatica
Premere un tasto per continuare . . .
```

```
char  s[5];
cin >> s;     // e
```




```
cin >> s;     // Calcolatori
```



ERRORE
vettore s ha solo 5 posizioni

non segnalato dal compilatore
continuo a scrivere in memoria

9.3 Stringhe (III)

```
// Stringhe e puntatori

#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    char s1[] = "Universita' ";
    char s2[] = {'d','i',' ','\0'};
    char *s3 = "Pisa";
    char *s4 = "Toscana";

    cout << s1 << s2 << s3 << s4 << endl;
            // puntatori a caratteri interpretati come stringhe
    s4 = s3;

    cout << s3 << endl;                 // Pisa
    cout << s4 << endl;                 // Pisa

    char *const s5 = "oggi";
    char *const s6 = "domani";

//  s6 = s5;                            ERRORE!

    cout << (void *)s3 << endl;   // Per stampare il puntatore
    system("PAUSE");
    return 0;
}
```

```
Universita' di Pisa Toscana
Pisa
Pisa
0x40121d
Premere un tasto per continuare . . .
```

9.3 Stringhe (IV)

```
// Conta le occorrenze di ciascuna lettera in una stringa
#include <cstdlib>
#include <iostream>
using namespace std;
int main ()
{
    const int LETTERE = 26;
    char str[100];
    int conta[LETTERE];
    for (int i = 0; i < LETTERE; i++)
        conta[i] = 0;
    cout << "Inserisci una stringa: ";
    cin >> str;

    for (int i = 0; str[i] != '\0'; i++)
        if (str[i] >= 'a' && str[i] <= 'z')
            ++conta[str[i] - 'a'];
        else if (str[i] >= 'A' && str[i] <= 'Z')
            ++conta[str[i] - 'A'];

    for (int i = 0; i < LETTERE; i++)
        cout << char('a' + i) << ": " << conta[i] << '\t';
    cout << endl;
    system("PAUSE");
    return 0;
}
```

```
Inserisci una stringa: prova
a: 1    b: 0    c: 0    d: 0    e: 0    f: 0    g: 0    h: 0    i: 0    j: 0
k: 0    l: 0    m: 0    n: 0    o: 1    p: 1    q: 0    r: 1    s: 0    t: 0
u: 0    v: 1    w: 0    x: 0    y: 0    z: 0
Premere un tasto per continuare . . .
```

9.6 Funzioni di libreria sulle stringhe (I)

Dichiarazioni contenute nel file <cstring>

char *strcpy(char *dest, const char *sorg);

Copia *sorg* in *dest*, incluso il carattere nullo (terminatore di stringa), e restituisce *dest*;

ATTENZIONE: non viene effettuato nessun controllo per verificare se la dimensione di *dest* è sufficiente per contenere *sorg*.

char *strcat(char **dest*, const char **sorg*);

Concatena *sorg* al termine di *dest* e restituisce *dest* (il carattere nullo compare solo alla fine della stringa risultante);

ATTENZIONE: non viene effettuato nessun controllo per verificare se la dimensione di *dest* è sufficiente per contenere la concatenazione di *sorg e dest*.

ATTENZIONE: sia *sorg* che *dest* devono essere delle stringhe.

9.6 Funzioni di libreria sulle stringhe (II)

int strlen(const char *_string_);

Restituisce la lunghezza di *string*; il valore restituito è inferiore di 1 al numero di caratteri effettivi, perché il carattere nullo che termina *string* non viene contato.

int strcmp(const char *_s1_, const char *_s2_);

Confronta *s1* con *s2*:

- restituisce un valore negativo se *s1* è alfabeticamente minore di *s2*;
- un valore nullo se le due stringhe sono uguali,
- un valore positivo se *s1* è alfabeticamente maggiore di *s2; (la funzione distingue tra maiuscole e minuscole).*

char *strchr(const char *_string_, char _c_);

Restituisce il puntatore alla prima occorrenza di *c* in *string* oppure 0 se *c* non si trova in *string*.

9.6 Funzioni di libreria sulle stringhe (III)

```
// ESEMPIO

#include <cstdlib>
#include <iostream>
#include <cstring>
using namespace std;
int main ()
{
    const int N = 30;
    char s1[] = "Corso ";
    char s2[] = "di ";
    char s3[] = "Informatica\n";
    char s4[N] = "Corso ";

    cout << "Dimensione degli array s1 e s4 " << endl;
    cout << sizeof s1 << " " << sizeof s4 << endl;

    cout << "Dimensione delle stringhe s1 e s4 " << endl;
    cout << strlen(s1) << " " << strlen(s4) << endl;
    if (!strcmp(s1,s4)) cout << "Stringhe uguali " << endl;
    else cout << "Stringhe diverse " << endl;

    if (!strcmp(s1,s2)) cout << "Stringhe uguali " << endl;
    else cout << "Stringhe diverse " << endl << endl;

    char s5[N];
    strcpy(s5,s1);
    strcat(s5,s2);
    strcat(s5,s3);
```

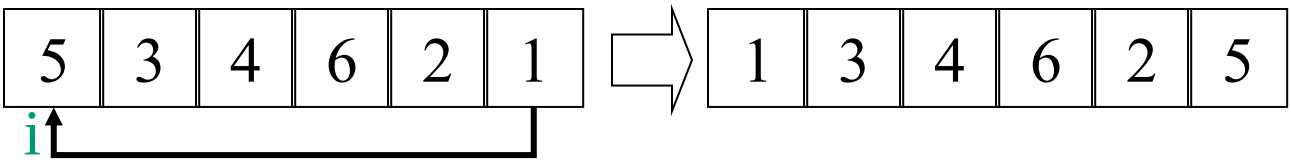
9.6 Funzioni di libreria sulle stringhe (IV)

```
    cout << "Concatenazione di s1, s2 e s3 " << endl;
    cout << s5 << endl;
    char *s=strchr(s5,'I');
    cout << "Stringa dalla prima istanza di I " << endl;
    cout << s << endl;
    system("PAUSE");
    return 0;
}
```

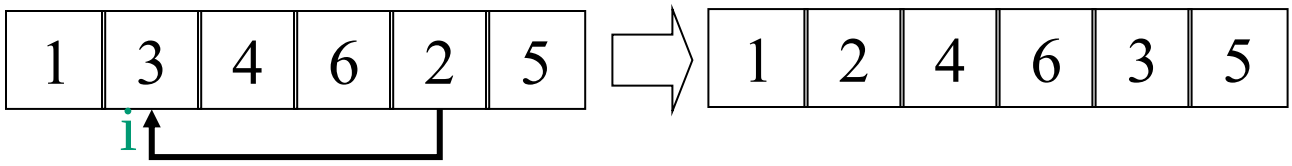
```
Dimensione degli array s1 e s4
7 30
Dimensione delle stringhe s1 e s4
6 6
Stringhe uguali
Stringhe diverse

Concatenazione di s1, s2 e s3
Corso di Informatica

Stringa dalla prima istanza di I
Informatica

Premere un tasto per continuare . . .
```

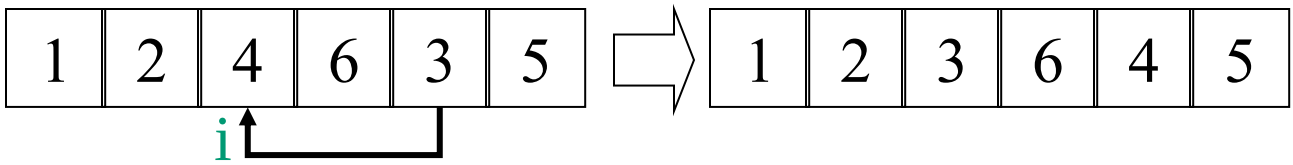
Ordinamento per selezione (selection-sort)

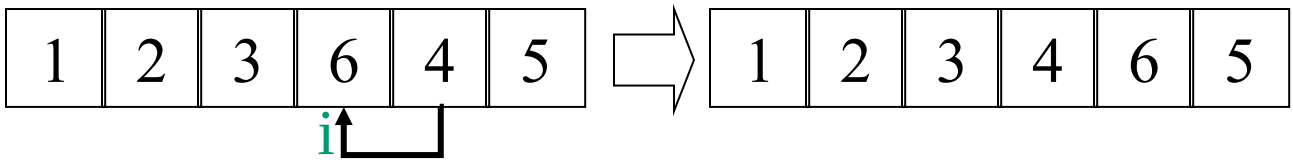
- **Si cerca l'elemento più piccolo e si scambia con l'elemento in posizione i = 0**

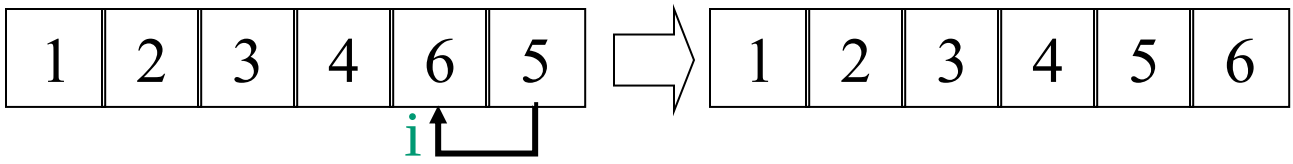


- **Si cerca l'elemento più piccolo tra i rimanenti N-i e si scambia con l'elemento in posizione i, per i = 1..N-1**









Ordinamento per selezione (selection-sort)

```
#include <cstdlib>
#include <iostream>
using namespace std;
typedef int T;      // introduce identificatori per individuare tipi

void stampa(const T v[], int n)
{
    if (n != 0)
    {
        cout << "[" << v[0];
        for (int i = 1; i < n; i++)
            cout << ' ' << v[i];
        cout << ']' << endl;
    }
}

void scambia(T vettore[], int x, int y)
{
    T lavoro = vettore[x];
    vettore[x] = vettore[y];
    vettore[y] = lavoro;
}
```

Ordinamento per selezione (selection-sort)

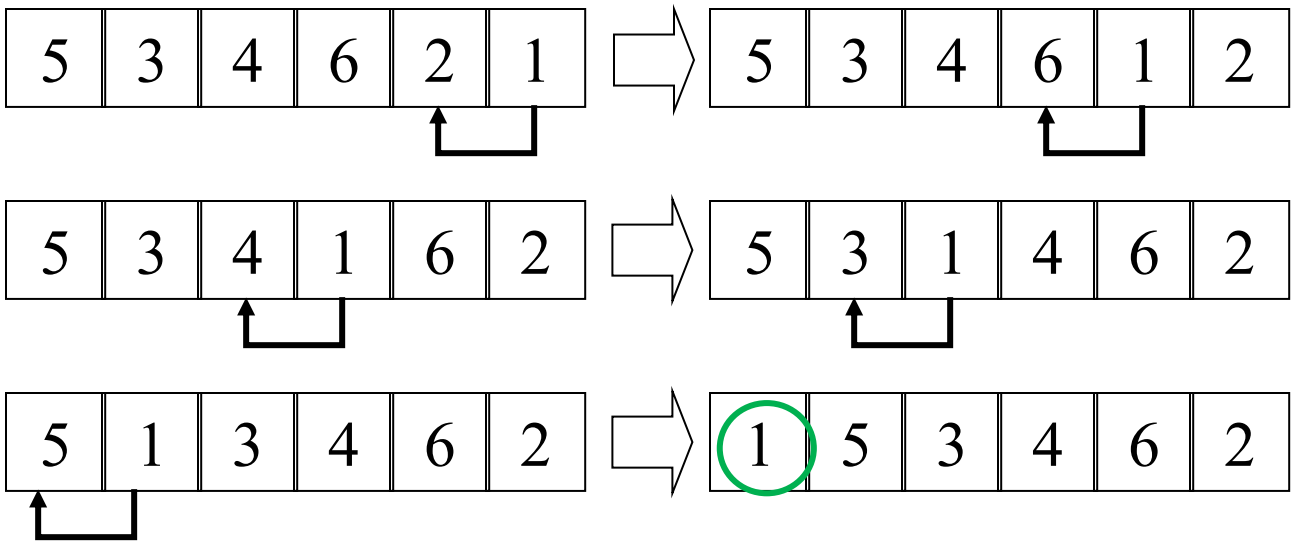
```
void selectionSort(T vettore[], int n)
{
    int min;
    for (int i = 0 ; i < n-1; i++)
    {
        min = i;
        for (int j = i + 1; j < n; j++)
            if (vettore[j] < vettore[min]) min = j;
        scambia(vettore,i,min);
    }
}

int main()
{
    T v[] =  {2, 26, 8, 2, 23};
    selectionSort(v, 5);
    stampa(v, 5);

    return 0;
}
```

```
[2 2 8 23 26]
```

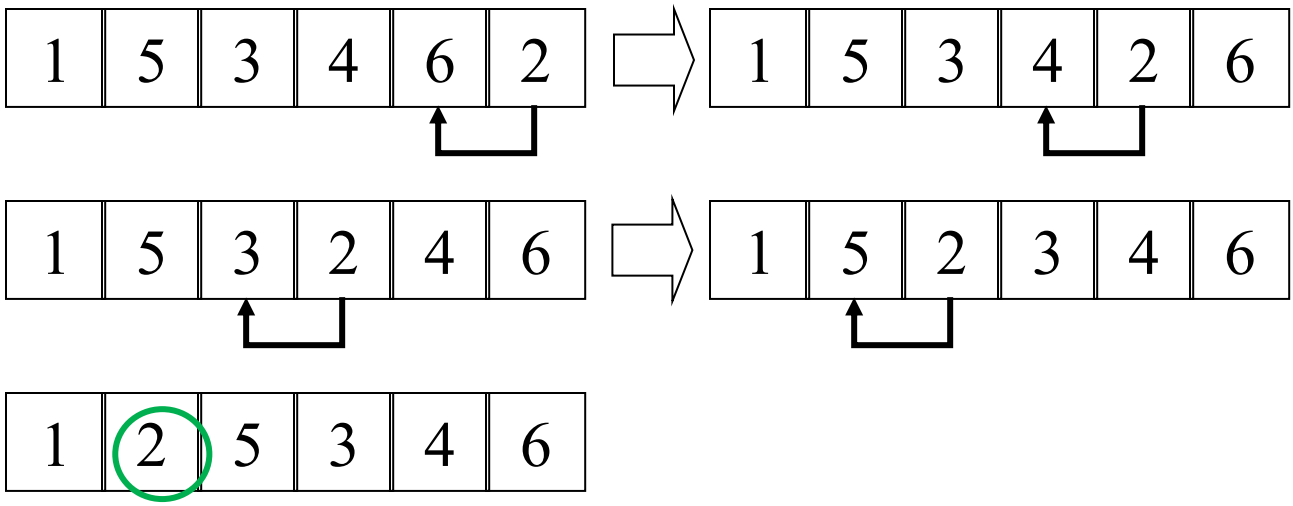
- Complessità dell'algoritmo dell'ordine di n^2, dove n è il numero di elementi nel vettore.

Ordinamento bubble-sort

- **Si scorre l'array n-1 volte, dove n è il numero di elementi nell'array, da destra a sinistra, scambiando due elementi contigui se non sono nell'ordine giusto.**
  - **Prima volta**



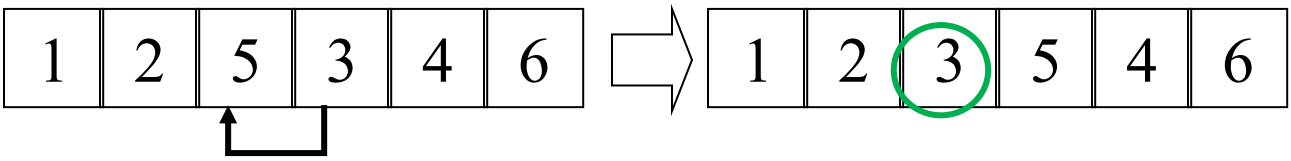
  - **Seconda volta**



N.B.: i primi due elementi risultano ordinati

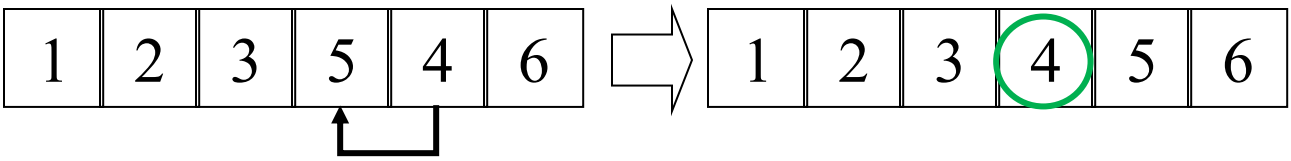
Ordinamento bubble-sort

- **Terza volta**



N.B.: i primi tre elementi risultano ordinati

- **Quarta volta**



N.B.: i primi quattro elementi risultano ordinati

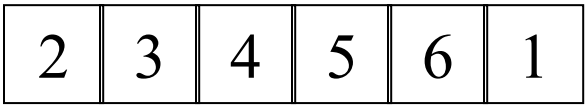
- **Quinta volta**
  - **Nessun cambiamento**

```
void bubble(T vettore[], int n)
{
    for (int i = 0 ; i < n-1; i++)
        for (int j = n-1; j > i; j--)
            if (vettore[j] < vettore[j-1])
                scambia(vettore, j, j-1);
}
```

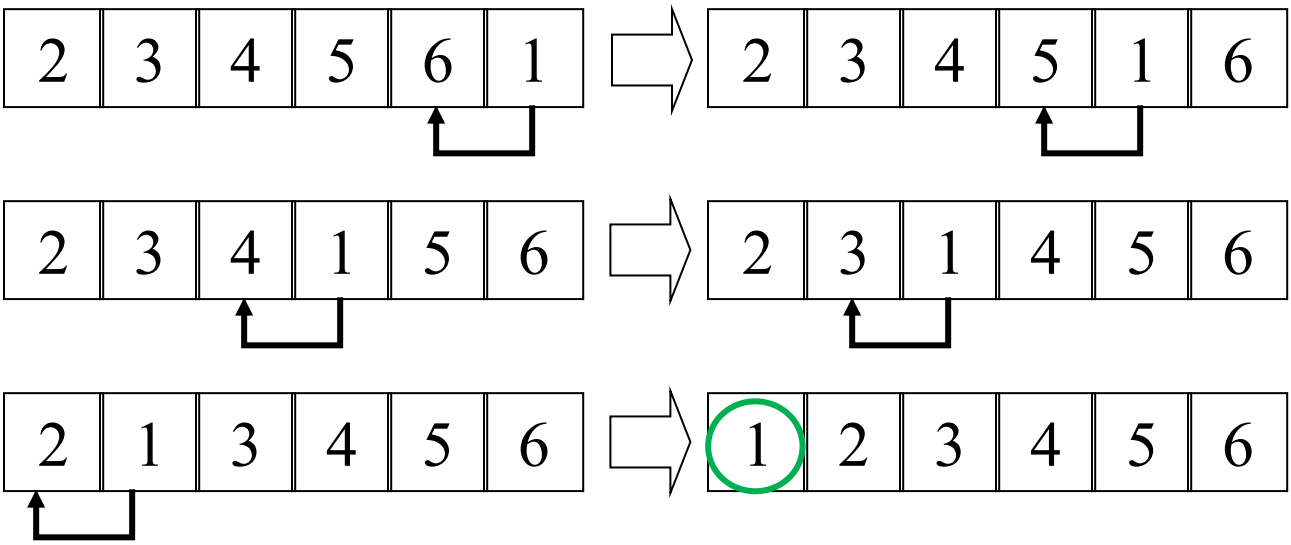
- **Complessità dell'algoritmo dell'ordine di n^2, dove n è il numero di elementi nel vettore.**

Ordinamento bubble-sort ottimizzato

Supponiamo che il vettore sia



- **Prima volta**



- **Il vettore dopo il primo passo risulta ordinato.**
- **Inutile eseguire tutti i passi.**

Ordinamento bubble-sort ottimizzato

```
void bubble(T vettore[], int n)
{
    bool ordinato = false;
    for (int i = 0 ; i < n-1 && !ordinato; i++)
    {
        ordinato = true;
        for (int j = n-1; j >= i+1; j--)
        if(vettore[j] < vettore[j-1])
        {
            scambia(vettore, j, j-1);
            ordinato = false;
        }
    }
}

int main()
{
    T v[] =  {2, 1, 3, 4, 5};
    bubble(v, 5);
    stampa(v, 5);
    system("PAUSE");
    return 0;
}
```

```
[1 2 3 4 5]
Premere un tasto per continuare . . .
```

- L'algoritmo ottimizzato esegue due sole iterazioni invece delle quattro dell'algoritmo non ottimizzato

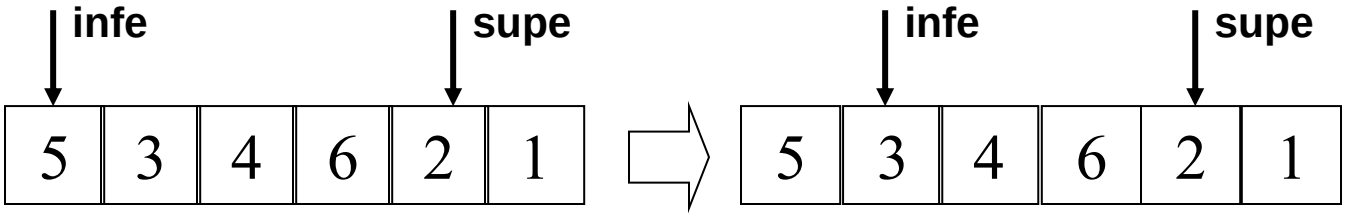
9.7 Ricerca lineare (I)

Problema

- **cercare un elemento in un array tra l'elemento in posizione *infe* e quello in posizione *supe*.**

Possibile soluzione

- **Scorrere il vettore in sequenza a partire dall'elemento in posizione *infe* fino all'elemento cercato oppure all'elemento in posizione *supe*.**




```
#include <cstdlib>
#include <iostream>
using namespace std;
typedef int T;

bool ricerca(T vett[], int infe, int supe, T k, int &pos)
{
    bool trovato = false;
    while ((!trovato) && (infe <= supe))
    {
        if (vett[infe] == k)
        {
            pos = infe;
            trovato = true;
        }
        infe++;
    }
    return trovato;
}
```

9.7 Ricerca lineare (II)

```
int main()
{
    T v[] =  {1, 2, 3, 4, 5};
    int i;                                  // individua la posizione
    if (!ricerca(v, 0, 4, 5, i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato " << i << endl;
    if (!ricerca(v, 0, 4, 10, i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato " << i << endl;
    system("PAUSE");
    return 0;
}
```

```
Posizione elemento cercato 4
Elemento cercato non presente
Premere un tasto per continuare . . .
```

N.B: Per la ricerca dell'elemento 5 è necessario esaminare 5 elementi.

9.7 Ricerca binaria (I)

PREREQUISITO: Vettori ordinati!!!!

Ricerca binaria (vettori ordinati in ordine crescente)
Idea:

- **Si confronta l'elemento cercato con l'elemento in posizione centrale; se sono uguali la ricerca termina;**
- **altrimenti:**
  - **se l'elemento cercato è minore dell'elemento in posizione centrale la ricerca prosegue nella prima metà del vettore; altrimenti prosegue nella seconda metà del vettore.**

```
bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2; // calcola l'indice centrale
        if (k > ordVett[medio])
            infe = medio + 1;    // ricerca nella meta' superiore
        else
            if (k < ordVett[medio])
                supe = medio - 1;
                                   // ricerca nella meta' superiore
            else
            {
                pos = medio;                // trovato
                return true;
            }
    }
   return false;
}
```

N.B: Per la ricerca dell'elemento 5, esempio precedente, è necessario esaminare 3 elementi solamente.

9.7 Ricerca binaria (II)

Esempio: ricerca in un vettore di caratteri

```
#include <cstdlib>
#include <iostream>
using namespace std;
typedef char T;

bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2; //calcola l'indice centrale
        if (k > ordVett[medio])
            infe = medio + 1;    // ricerca nella meta' superiore
        else
            if (k < ordVett[medio])
                supe = medio - 1;
                    // ricerca nella meta' superiore
            else
            {
                pos = medio;       // trovato
                return true;
            }
    }
   return false;
}

int main()
{
    T v[] =  {'a', 'b', 'c', 'r', 's', 't'};
    int i;                                    // individua la posizione
    if (!ricbin(v, 0, 5, 'c', i))
        cerr << "Elemento cercato non presente " << endl;
    else
        cout << "Posizione elemento cercato " << i << endl;
    system("PAUSE");                                    // 2
    return 0;
}
```

9.7 Esempio (I)

Esempio: ordinamento e ricerca in un vettore di stringhe

```
#include <cstdlib>
#include <iostream>
using namespace std;
const int C=20;
typedef char T[C];

void scambia(T vettore[], int x, int y)
{
    T lavoro;
    strcpy(lavoro,vettore[x]);
    strcpy(vettore[x],vettore[y]);
    strcpy(vettore[y],lavoro);
}


void stampa(const T vettore[], int n)
{
    if (n != 0)
    {
        cout << '[' << vettore[0];
        for (int i = 1; i < n; i++)
            cout << ' ' << vettore[i];
        cout << ']' << endl;
    }
}
```

9.7 Esempio (II)

```
void bubble(T vettore[], int n)
{
    bool ordinato = false;
    for (int i = 0 ; i < n-1 && !ordinato; i++)
    {
        ordinato = true;
        for (int j = n-1; j >= i+1; j--)
        if(strcmp(vettore[j],vettore[j-1])<0)
        {
            scambia(vettore, j, j-1);
            ordinato = false;
        }
    }
}

bool ricbin(T ordVett[], int infe, int supe, T k, int &pos)
{
    while (infe <= supe)
    {
        int medio = (infe + supe) / 2;
        if (strcmp(k,ordVett[medio])>0)
            infe = medio + 1;
        else
            if (strcmp(k,ordVett[medio])<0)
                supe = medio - 1;
            else
            {
                pos = medio;
                return true;
            }
    }
   return false;
}
```

9.7 Esempio (III)

```
int main ()
{   T s[4];
    for (int i=0; i<4; i++)
    {   cout << '?' << endl;  cin >> s[i];    }
    stampa(s,4);
    bubble(s,4);
    stampa(s,4);
    int i; T m;
    cout << "Ricerca ?" << endl;
    cin >> m;
    if (ricbin(s,0,4,m,i))
          cout << "Trovato in posizione " << i << endl;
    else cout << "Non trovato" << endl;
    cout << "Ricerca ?" << endl;
    cin >> m;
    if (ricbin(s,0,4,m,i))
          cout << "Trovato in posizione " << i << endl;
      else cout << "Non trovato" << endl;
    system("PAUSE");
    return 0;
}
```

```
?
mucca
?
anatra
?
zebra
?
cavallo
[mucca anatra zebra cavallo]
[anatra cavallo mucca zebra]
Ricerca ?
cavallo
Trovato in posizione 1
Ricerca ?
bue
Non trovato
Press any key to continue . . ..
```

10.1 Strutture (I)

Struttura:

- **n-upla ordinata di elementi, detti membri (o campi), ciascuno dei quali ha uno specifico tipo ed uno specifico nome, e contiene una data informazione;**
- **rappresenta una collezione di informazioni su un dato oggetto.**

basic-structure-type-declaration
structure-typespecifier ;
structure-typespecifier
struct ***identifier*****|opt**
{ ***structure-member-section-seq*** }

Sezione:

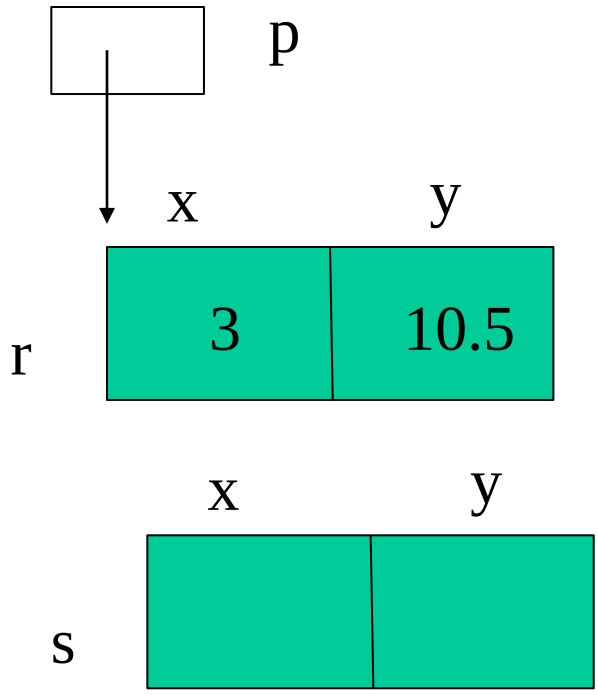
- **membri di un certo tipo, ciascuno destinato a contenere un dato (campi dati);**
- **forma sintatticamente equivalente alla definizione di oggetti non costanti e non inizializzati.**

Esempio:

```
struct persona
{
    char nome[20];
    char cognome[20];
    int g_nascita, m_nascita, a_nascita;
};
```

10.1 Strutture (II)

```
// Punto
#include <cstdlib>
#include <iostream>
using namespace std;

struct punto
{
    double x;
    double y;
};

int main ()
{
    punto r, s;

    r.x = 3;                                    // selettore di membro
    r.y = 10.5;

    s.x = r.x;
    s.y = r.y + 10.0;

    cout << '<' << r.x << ", " << r.y << ">\n";
    cout << '<' << r.x << ", " << s.y << ">\n";

    punto *p = &r;
    cout << '<' << p->x << ", ";               // (*p).x
    cout << p->y << ">\n";                     // (*p).y

    punto t = {1.0, 2.0};                      // inizializzazione
    cout << '<' << t.x << ", " << t.y << ">\n";
    system("PAUSE");
    return 0;
}
```



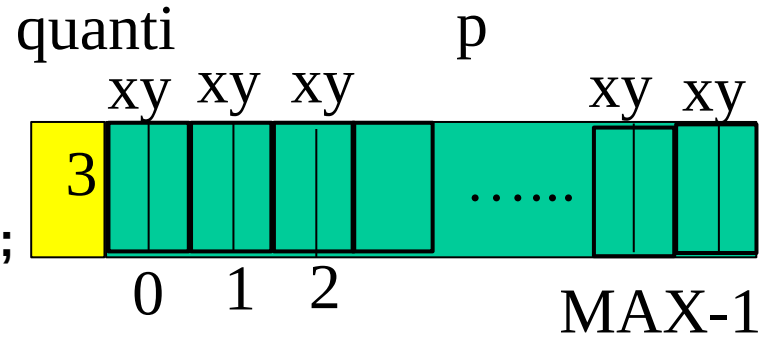

```
<3, 10.5>
<3, 20.5>
<3, 10.5>
<1, 2>
Premere un tasto per continuare . . .
```

10.1 Strutture (III)

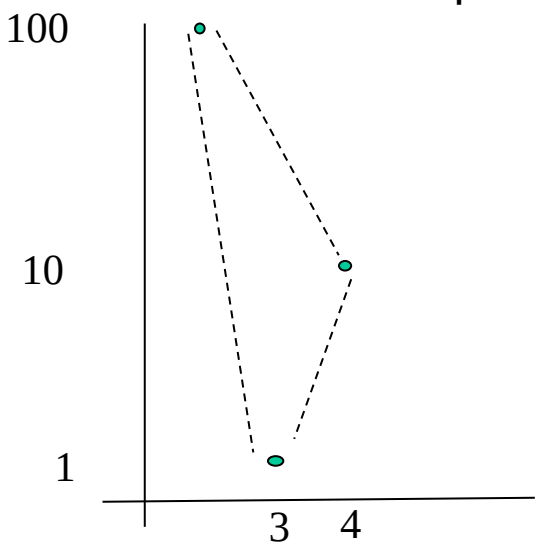
```
struct punto
{
    /* ... */
    punto s;                                // ERRORE!
};

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Struttura contenente un riferimento a se stessa

struct punto {
    /* ... */
    punto* p;                               // OK
};

//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~//
// Strutture con riferimenti intrecciati.
// Dichiarazioni incomplete

struct Parte;

struct Componente {
    /* … */
    Parte* p;
};`

struct Parte {
    /* … */
    Componente* c;
};
```

10.1 Strutture (IV)

```
// Array di strutture

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

struct punto
{    double x;  double y;};

const int MAX = 30;

int main ()
{
     struct poligono
     {
          int quanti;
          punto p[MAX];
     } p;
 // poligono p;

     p.quanti = 3;

     p.p[0].x = 3.0;
     p.p[0].y = 1.0;

     p.p[1].x = 4.0;
     p.p[1].y = 10.0;

     p.p[2].x = 3.0;
     p.p[2].y = 100.0;

     cout << "Stampa del poligono " << endl;
     for (int i = 0; i < p.quanti; i++)
          cout << '<' << p.p[i].x << ", " << p.p[i].y << ">\n";

     return 0;
}
```



// numero effettivo di punti




```
Stampa del poligono
<3, 1>
<4, 10>
<3, 100>
```

10.1.1 Operazioni sulle strutture (I)

```
// Assegnamento tra strutture
#include <cstdlib>
#include <iostream>
using namespace std;

struct punto
{
    double x;
    double y;
};

int main ()
{
    punto r1 = {3.0, 10.0};     // inizializzazione
    punto r2;

    r2 = r1;                    // copia membro a membro

    cout << "r1 = <" << r1.x << ", " << r1.y << ">\n";
                                        // <3, 10>
    cout << "r2 = <" << r2.x << ", " << r2.y << ">\n";
                                        // <3, 10>
//  if (r2 != r1)       ERRORE

    system("PAUSE");
    return 0;
}
```

```
r1 = <3, 10>
r2 = <3, 10>
Premere un tasto per continuare . . .
```

N.B.: Non sono definite operazioni di confronto sulle strutture.

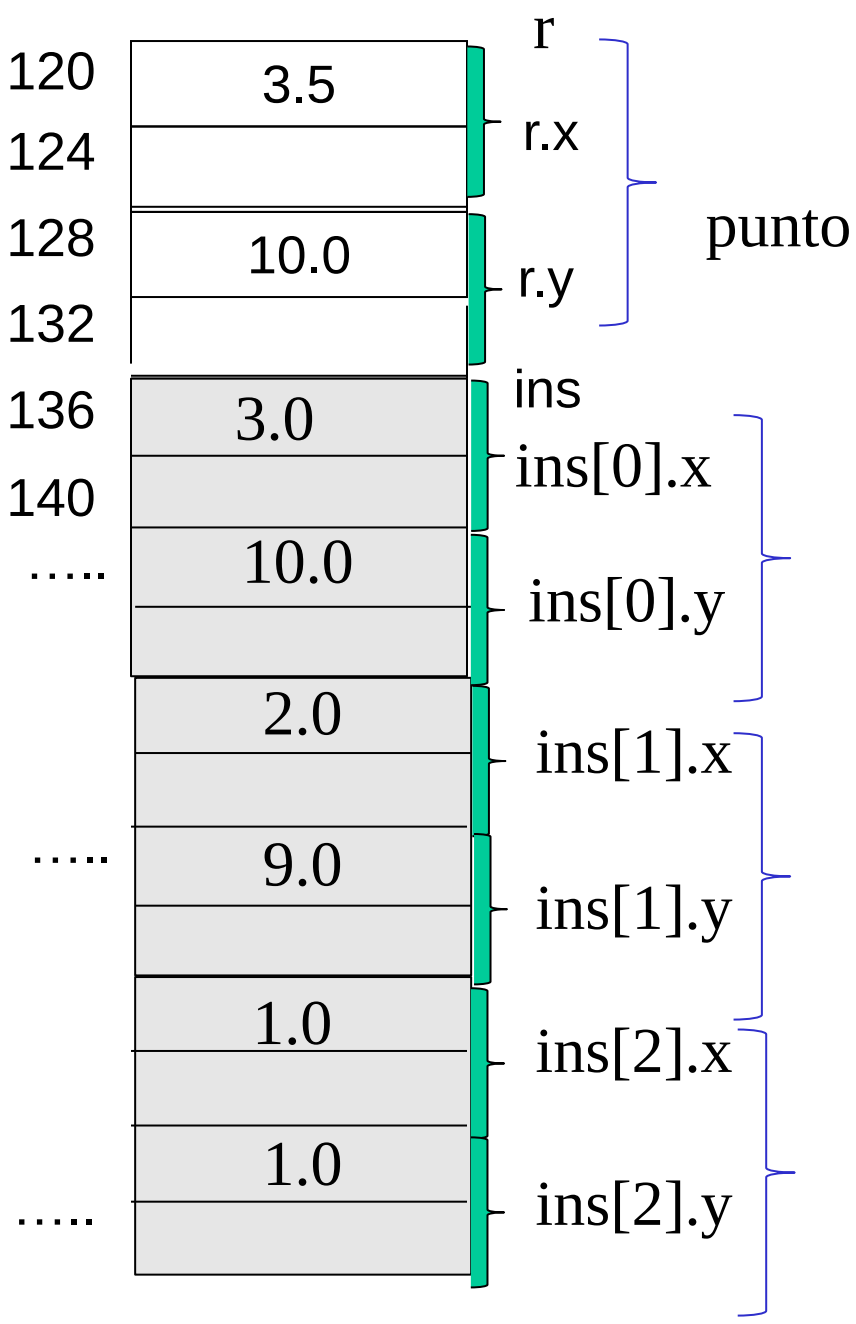
10.1.1 Operazioni sulle strutture (II)

```
// Strutture come argomenti di funzioni e restituite da
// funzioni

#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;

struct punto
{    double x;  double y; };

void test(punto p)
{
     cout << "Dimensione argomento " << sizeof p << endl;
}

void test(const punto* p)                  // Overloading
{
     cout << "Dimensione argomento " << sizeof p << endl;
}

double dist(const punto* p1, const punto* p2)
{
     return sqrt((p1->x - p2->x) * (p1->x - p2->x) +
                 (p1->y - p2->y) * (p1->y - p2->y));
}

punto vicino(punto ins[], int n, const punto* p)
{
     double min = dist(&ins[0], p), t;
     int index = 0;                        // Memorizza l'indice
     for (int i = 1; i < n; i++)
     {
          t = dist(&ins[i], p);
          if (min > t)
               { index = i; min = t; };
      }
      return ins[index];
}
```

10.1.1 Operazioni sulle strutture (II)

```
// Strutture come argomenti di funzioni e restituite da
// funzioni

#include <cstdlib>
#include <iostream>
#include <cmath>
using namespace std;

struct punto
{    double x;  double y; };

void test(punto p)
{
     cout << "Dimensione argomento " << sizeof p << endl;
}

void test(const punto* p)                  // Overloading
{
     cout << "Dimensione argomento " << sizeof p << endl;
}

double dist(const punto* p1, const punto* p2)
{
     return sqrt((p1->x - p2->x) * (p1->x - p2->x) +
                 (p1->y - p2->y) * (p1->y - p2->y));
}

punto vicino(punto ins[], int n, const punto* p)
{
     double min = dist(&ins[0], p), t;
     int index = 0;                        // Memorizza l'indice
     for (int i = 1; i < n; i++)
     {
          t = dist(&ins[i], p);
          if (min > t)
               { index = i; min = t; };
      }
      return ins[index];
}
```

10.1.1 Operazioni sulle strutture (III)

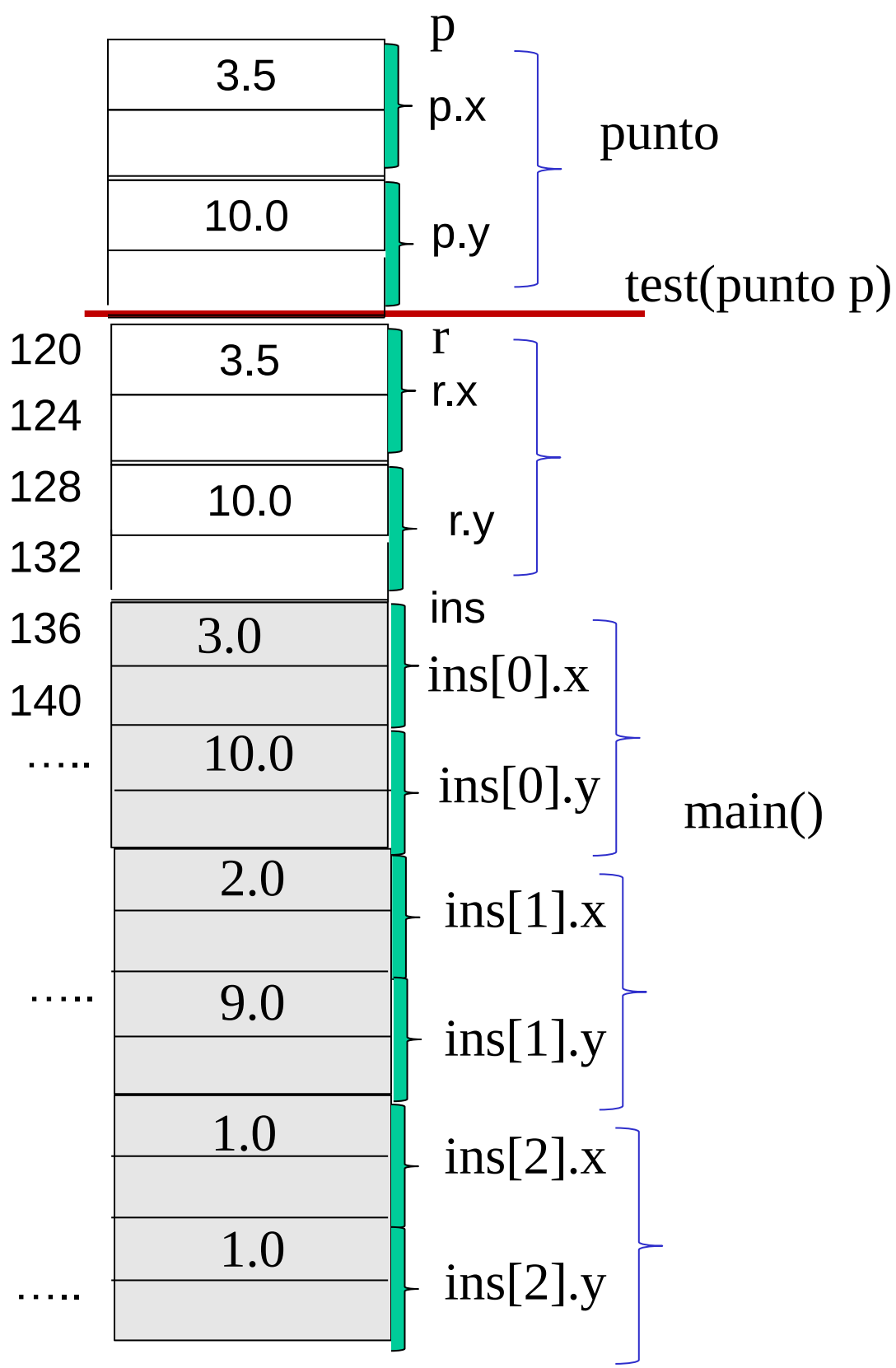


10.1.1 Operazioni sulle strutture (III)

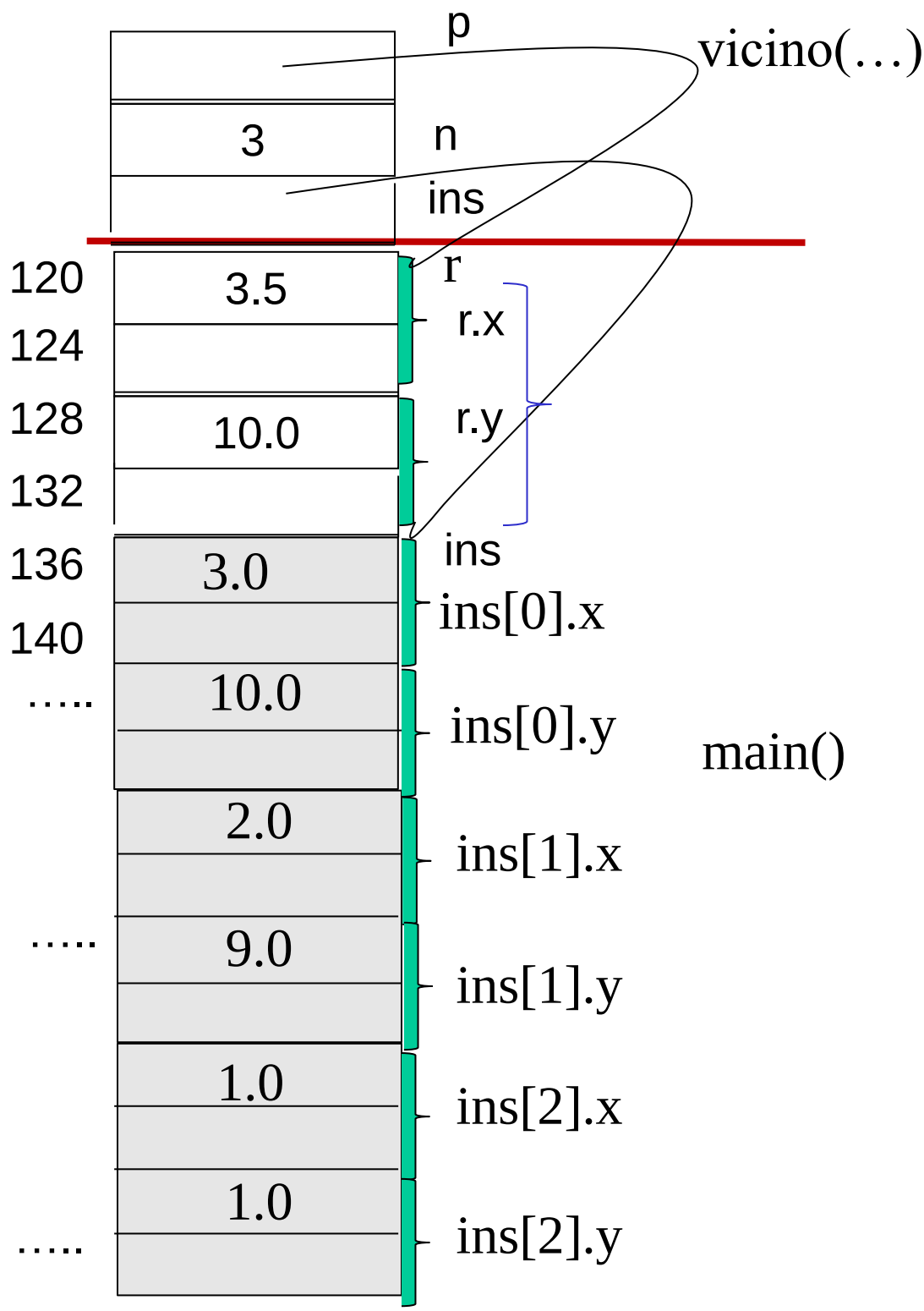
```
// Strutture come argomenti di funzioni e restituite da
// funzioni

int main ()
{
    punto ins[] = {{3.0, 10.0}, {2.0, 9.0}, {1.0, 1.0}};
    punto r = {3.5, 10.0};
    test(r);                                          // 16
    test(&r);                                         // 4
    cout << "Distanza: " << dist(&r, &ins[0]) << endl;
    punto s = vicino(ins, sizeof ins/sizeof(punto), &r);
    cout << "Punto piu' vicino: ";
    cout << '<' << s.x << ", " << s.y << ">\n";
    return 0;
}
```

```
Dimensione argomento 16
Dimensione argomento 4
Distanza: 0.5
Punto piu' vicino: <3, 10>
```

10.1.1 Operazioni sulle strutture



10.1.1 Operazioni sulle strutture

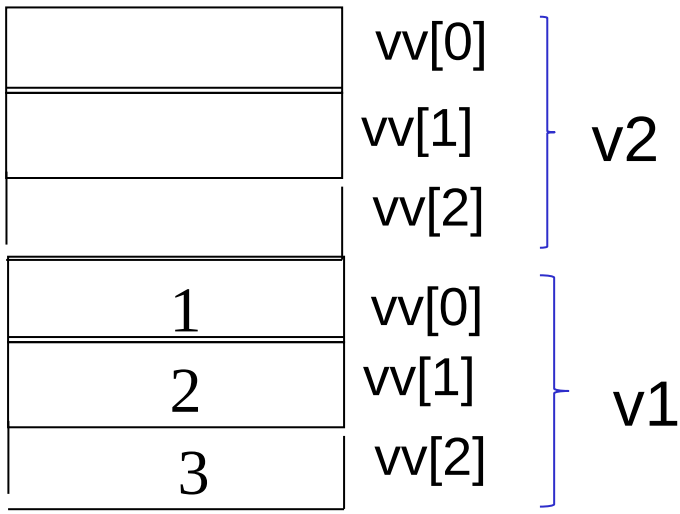


10.1.1 Operazioni sulle strutture (IV)

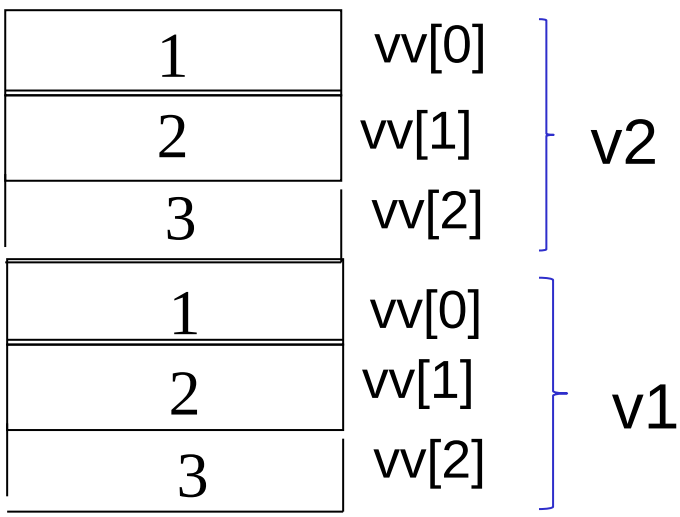
```
// Copia di vettori
#include <cstdlib>
#include <iostream>
using namespace std;
const int N = 3;
struct vettore
{    int vv[N];  };

void stampa(const vettore& v, int n)
{
     cout << '[' << v.vv[0];
     for (int i = 1; i < n; i++)
          cout << ' ' << v.vv[i];
     cout << ']' << endl;
}


int main ()
{
     vettore v1 = {1, 2, 3}, v2;
     v2 = v1;
     cout << "Stampa del vettore v1 " << endl;
     stampa(v1, N);
     cout << "Stampa del vettore v2 " << endl;
     stampa(v2, N);
     system("PAUSE");
     return 0;
}
```

```
Stampa del vettore v1
[1 2 3]
Stampa del vettore v2
[1 2 3]
Premere un tasto per continuare . . .
```

10.1.1 Operazioni sulle strutture



v2 = v1; assegnamento fra strutture

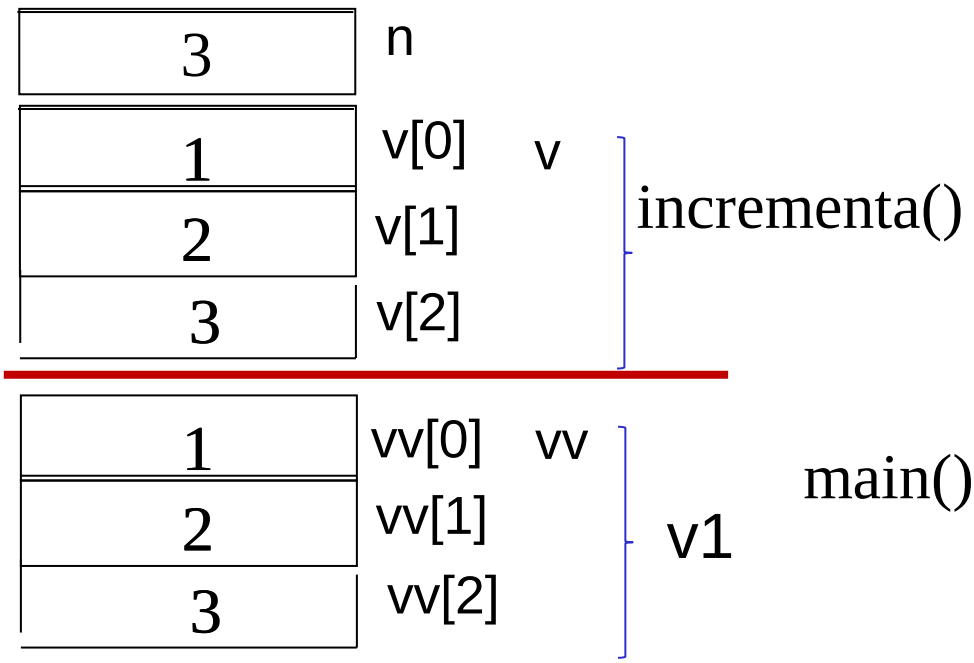


10.1.1 Operazioni sulle strutture (V)

```
// Trasmissione di vettori per valore
#include <cstdlib>
#include <iostream>
using namespace std;
const int N = 3;
struct vettore
{    int vv[N];  };
void stampa(const vettore& v, int n)
{
     cout << '[' << v.vv[0];
     for (int i = 1; i < n; i++)
          cout << ' ' << v.vv[i];
     cout << ']' << endl;
}
void incrementa(vettore v, int n)
{   for (int i = 0; i < n; i++)  v.vv[i]++;}
int main ()
{
     vettore v1 = {1, 2, 3};
     cout << "Stampa del vettore v1 " << endl;
     stampa(v1, N);
     incrementa(v1, N);              // Incrementa la copia
     cout << "Stampa del vettore v1 " << endl;
     stampa(v1, N);                  // [1 2 3]
     return 0;
}
```

```
Stampa del vettore v1
[1 2 3]
Stampa del vettore v2
[1 2 3]
```

10.1.1 Operazioni sulle strutture



10.2 Unioni (I)

Sono dichiarate e usate con la stessa sintassi delle strutture:

- **si utilizza la parola chiave *union* al posto di *struct*.**

Rappresentano un'area di memoria che in tempi diversi può contenere dati di tipo differente:

- **i membri di un'unione corrispondono a diverse "interpretazioni" di un'unica area di memoria.**

Membri di una unione non della stessa dimensione:

- **viene riservato spazio per il più grande.**

Esempio:

```
struct { int i; double d; } x;
union { int i; double d; } y;
```

- **la struttura *x* occupa 96 bit di memoria (32 per *i* e 64 per *d*);**
- **l'unione *y* occupa 64 bit di memoria, che possono essere dedicati ad un valore intero (lasciandone 32 inutilizzati) o ad un valore reale.**

Operazioni:

- **quelle viste per le strutture.**

Valori iniziali delle unioni:

- **solo per il primo membro;**
- **esempio:**

```
union {char c; int i; double f; } a = { 'X' };
```

10.2 Unioni (II)

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

union Uni
{    char c;  int i; };

struct Str
{    char c; int i; };

int main()
{
     cout << sizeof(char) << '\t' << sizeof(int) << endl;
                                                          // 1 4
     cout << sizeof(Uni) << '\t' << sizeof(Str) << endl;
                                                          // 4 8

     Uni u = {'a'};
//   Uni u1 = {'a', 10000};                          ERRATO

     u.i = 0xFF7A;                  // 7A e' la codifica ASCII di z
     cout << u.c << '\t' << u.i << endl;              // z 65402

     Str s = {'a', 0xFF7A};
     cout << s.c << '\t' << s.i << endl;              // a 65402
     system("PAUSE");
     return 0;
}
```

```
1       4
4       8
z       65402
a       65402
Premere un tasto per continuare . . .
```

10.2 Unioni (I)

Sono dichiarate e usate con la stessa sintassi delle strutture:

- **si utilizza la parola chiave *union* al posto di *struct*.**

Rappresentano un'area di memoria che in tempi diversi può contenere dati di tipo differente:

- **i membri di un'unione corrispondono a diverse "interpretazioni" di un'unica area di memoria.**

Membri di una unione non della stessa dimensione:

- **viene riservato spazio per il più grande.**

Esempio:

```
struct { int i; double d; } x;
union { int i; double d; } y;
```

- **la struttura *x* occupa 96 bit di memoria (32 per *i* e 64 per *d*);**
- **l'unione *y* occupa 64 bit di memoria, che possono essere dedicati ad un valore intero (lasciandone 32 inutilizzati) o ad un valore reale.**

Operazioni:

- **quelle viste per le strutture.**

Valori iniziali delle unioni:

- **solo per il primo membro;**
- **esempio:**

```
union {char c; int i; double f; } a = { 'X' };
```

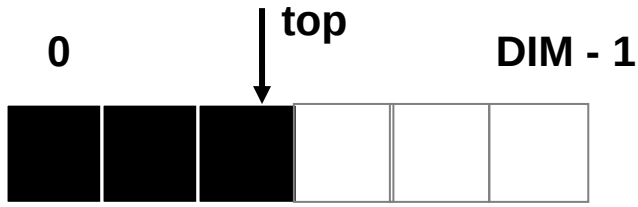
10.2 Unioni (II)

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

union Uni
{    char c;  int i; };

struct Str
{    char c; int i; };

int main()
{
     cout << sizeof(char) << '\t' << sizeof(int) << endl;
                                                          // 1 4
     cout << sizeof(Uni) << '\t' << sizeof(Str) << endl;
                                                          // 4 8

     Uni u = {'a'};
//   Uni u1 = {'a', 10000};                          ERRATO

     u.i = 0xFF7A;                  // 7A e' la codifica ASCII di z
     cout << u.c << '\t' << u.i << endl;              // z 65402

     Str s = {'a', 0xFF7A};
     cout << s.c << '\t' << s.i << endl;              // a 65402
     system("PAUSE");
     return 0;
}
```

```
1       4
4       8
z       65402
a       65402
Premere un tasto per continuare . . .
```

10.3.1 Pila (I)

- **Insieme ordinato di dati di tipo uguale, in cui è possibile effettuare operazioni di inserimento e di estrazione secondo la seguente regola di accesso: l'ultimo dato inserito è il primo ad essere estratto (LIFO: Last In First Out).**



- **Se top == -1, la pila è vuota. Se top == DIM - 1, dove DIM è il numero massimo di elementi nella pila, la pila è piena.**

```
#include <cstdlib>
#include <iostream>
using namespace std;
typedef  int T;
const int DIM = 5;

struct pila
{
   int top;
   T stack[DIM];
};

//inizializzazione della pila
void inip(pila& pp)
{
    pp.top = -1;
}
```

10.3.1 Pila (II)

```
bool empty(const pila& pp)        // pila vuota?
{
    if (pp.top == -1) return true;
    return false;
}

bool full(const pila& pp)         // pila piena?
{
    if (pp.top == DIM - 1) return true;
    return false;
}

bool push(pila& pp, T s)          // inserisce un elemento in pila
{
    if (full(pp)) return false;
    pp.stack[++(pp.top)] = s;
    return true;
}

bool pop(pila& pp, T& s)          // estrae un elemento dalla pila
{
    if (empty(pp)) return false;
    s = pp.stack[(pp.top)--];
    return true;
}

void stampa(const pila& pp)       // stampa gli elementi
{
    cout << "Elementi contenuti nella pila: " << endl;
    for (int i = pp.top; i >= 0; i--)
    cout << '[' << i << "] " << pp.stack[i] << endl;
}
```

10.3.1 Pila (III)

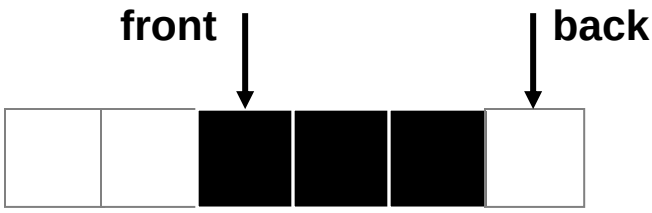
```
int main()
{
    pila st;
    inip(st);
    T num;
    if (empty(st)) cout << "Pila vuota" << endl;
    for (int i = 0; i < DIM; i++)
        if (push(st,DIM - i))
        cout << "Inserito " << DIM - i <<
             ". Valore di top: " << st.top << endl;
        else cerr << "Inserimento di " << i << " fallito" << endl;
    if (full(st)) cout << "Pila piena" << endl;
    stampa(st);
    for (int i = 0; i < DIM - 2; i++)
        if (pop(st, num))
            cout << "Estratto " << num << ". Valore di top: "
                        << st.top << endl;
        else cerr << "Estrazione fallita" <<endl;
     for (int i = 0; i < DIM; i++)
        if (push(st, i))
            cout << "Inserito " << i << ". Valore di top: "
                << st.top << endl;
        else cerr << "Inserimento di " << i << " fallito" << endl;
    stampa(st);
    for (int i = 0; i < 2; i++)
        if (pop(st, num))
                cout << "Estratto " << num << ". Valore di top:
                        " << st.top << endl;
        else cerr << "Estrazione fallita" <<endl;
    stampa(st);
    system("PAUSE");
    return 0;
}
```

10.3.1 Pila (IV)

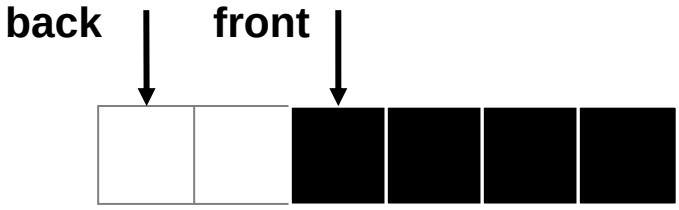
```
Pila vuota
Inserito 5. Valore di top: 0
Inserito 4. Valore di top: 1
Inserito 3. Valore di top: 2
Inserito 2. Valore di top: 3
Inserito 1. Valore di top: 4
Pila piena
Elementi contenuti nella pila:
[4] 1
[3] 2
[2] 3
[1] 4
[0] 5
Estratto 1. Valore di top: 3
Estratto 2. Valore di top: 2
Estratto 3. Valore di top: 1
Inserito 0. Valore di top: 2
Inserito 1. Valore di top: 3
Inserito 2. Valore di top: 4
Inserimento di 3 fallito
Inserimento di 4 fallito
Elementi contenuti nella pila:
[4] 2
[3] 1
[2] 0
[1] 4
[0] 5
Estratto 2. Valore di top: 3
Estratto 1. Valore di top: 2
Elementi contenuti nella pila:
[2] 0
[1] 4
[0] 5
Premere un tasto per continuare . . .
```

10.3.2 Coda (I)

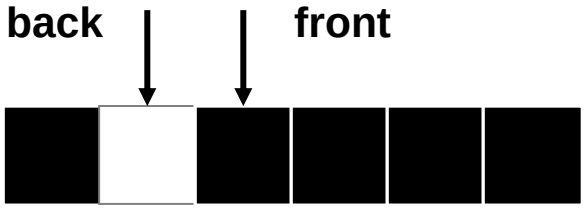
- **Insieme ordinato di dati di tipo uguale, in cui è possibile effettuare operazioni di inserimento e di estrazione secondo la seguente regola di accesso: il primo dato inserito è il primo ad essere estratto (FIFO: First In First Out).**



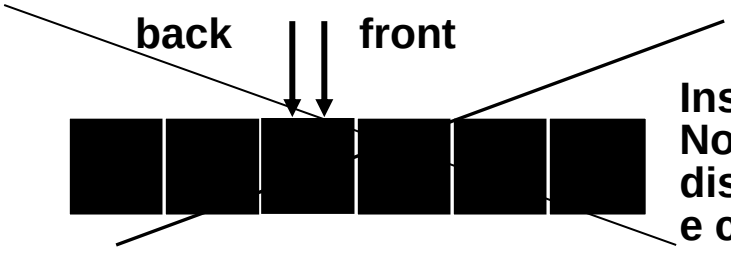
- **Realizzata con un array circolare e due puntatori:**
  - **front – posizione da cui avviene l'estrazione;**
  - **back – posizione in cui avviene l'inserimento.**



Inserimento di un elemento



Inserimento di un elemento



Inserimento di un elemento
Non sarebbe possibile discriminare tra coda vuota e coda piena

- **front == back** **coda vuota**
- **front == (back + 1) % DIM** **coda piena**
  - **(ATTENZIONE al massimo DIM - 1 elementi**

10.3.2 Coda (II)

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

typedef int T;
const int DIM = 5;

struct coda
{
    int front, back;
    T queue[DIM];
};

void inic(coda&  cc)            // inizializzazione della coda
{
     cc.front = cc.back = 0;
}

bool empty(const coda& cc)      // coda vuota?
{
     if (cc.front == cc.back) return true;
     return false;
}

bool full(const coda& cc)              //coda piena?
{
     if (cc.front == (cc.back + 1)%DIM) return true;
     return false;
}
```

10.3.2 Coda (III)

```
bool insqueue(coda& cc, T s) // inserisce un elemento
{
    if (full(cc)) return false;
    cc.queue[cc.back] = s;
    cc.back = (cc.back+1)%DIM;
    return true;
}

bool esqueue(coda& cc, T& s) // estrae un elemento
{
    if (empty(cc)) return false;
    s = cc.queue[cc.front];
    cc.front = (cc.front + 1)%DIM;
    return true;
}

void stampa(const coda& cc)        // stampa gli elementi
{
    for (int i = cc.front; i%DIM != cc.back; i++)
        cout << cc.queue[i%DIM] << endl;
}
```

10.3.2 Coda (IV)

```
int main()
{
 coda qu; T num;
 inic(qu);
 if (empty(qu)) cout << "Coda vuota" << endl;
 for (int i = 0; i < DIM; i++)
      if (insqueue(qu, i))
          cout << "Inserito l'elemento " << i << " in
           posizione " <<  (qu.back + DIM - 1)%DIM << endl;
      else cerr << "Coda piena" << endl;
 if (full(qu)) cout << "Coda piena" << endl;
 stampa(qu);
 for (int i = 0; i < DIM - 2; i++)
      if (esqueue(qu, num))
        cout <<"Estratto l'elemento " << num << " in
          posizione " << (qu.front + DIM - 1)%DIM << endl;
      else cout << "Coda vuota " << endl;
 for (int i = 0; i < DIM; i++)
      if (insqueue(qu, i))
       cout << "Inserito l'elemento " << i << " in posizione "
            << (qu.back + DIM - 1)%DIM << endl;
      else cerr << "Coda piena" << endl;
  stampa(qu);
  for (int i = 0; i < 2; i++)
      if (esqueue(qu, num))
        cout <<"Estratto l'elemento " << num << " in
          posizione "    << (qu.front + DIM - 1)%DIM << endl;
      else cout << "Coda vuota" << endl;
  stampa(qu);
  system("PAUSE");
  return 0;
}
```

10.3.2 Coda (V)

```
Coda vuota
Inserito l'elemento 0 in posizione 0
Inserito l'elemento 1 in posizione 1
Inserito l'elemento 2 in posizione 2
Inserito l'elemento 3 in posizione 3
Coda piena
Coda piena
0
1
2
3
Estratto l'elemento 0 in posizione 0
Estratto l'elemento 1 in posizione 1
Estratto l'elemento 2 in posizione 2
Inserito l'elemento 0 in posizione 4
Inserito l'elemento 1 in posizione 0
Inserito l'elemento 2 in posizione 1
Coda piena
Coda piena
3
0
1
2
Estratto l'elemento 3 in posizione 3
Estratto l'elemento 0 in posizione 4
1
2
Premere un tasto per continuare . . .
```