Informatica - Mod. Programmazione
Lezione 07

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Ricorsione

- Il meccanismo di esecuzione delle funzioni è stato progettato così per consentire le chiamate **ricorsive**
- cioè una funzione che, in modo diretto o indiretto, chiama sé stessa
- Questo è consentito dalla creazione del record di attivazione nello stack delle chiamate
- Ma come progettare una soluzione ricorsiva ad un problema?

Definizioni Matematiche Induttive

- Consideriamo un problema che in matematica viene definito in modo **induttivo**
- Calcolo del Fattoriale di un numero $n \geq 0$

  $n! = n \times (n-1)!$ se $n \geq 1$, **Passo Induttivo**

  $0! = 1$ **Base dell'Induzione**

Definizione Funzionale Ricorsiva

- Se usiamo le funzioni, la definizione diventa **Ricorsiva**
- $fatt(n) = n \times fatt(n-1)$ se $n \geq 1$,

  Passo Ricorsivo

  $fatt(0) = 1$ **Base della Ricorsione**

Funzione del Programma: Fattoriale.cpp

```
int fatt(int n)
{
    int r;

    if( n == 0 )
        r = 1;
    else
        r = n * fatt( n-1);

    return r;
}
```

Resto del Programma: Fattoriale.cpp

```
int fattoriale(int n)
{
    int r;

    if(n < 0)
        r = -1;
    else
        r = fatt( n );
    return r;
}
```

Resto del Programma: Fattoriale.cpp

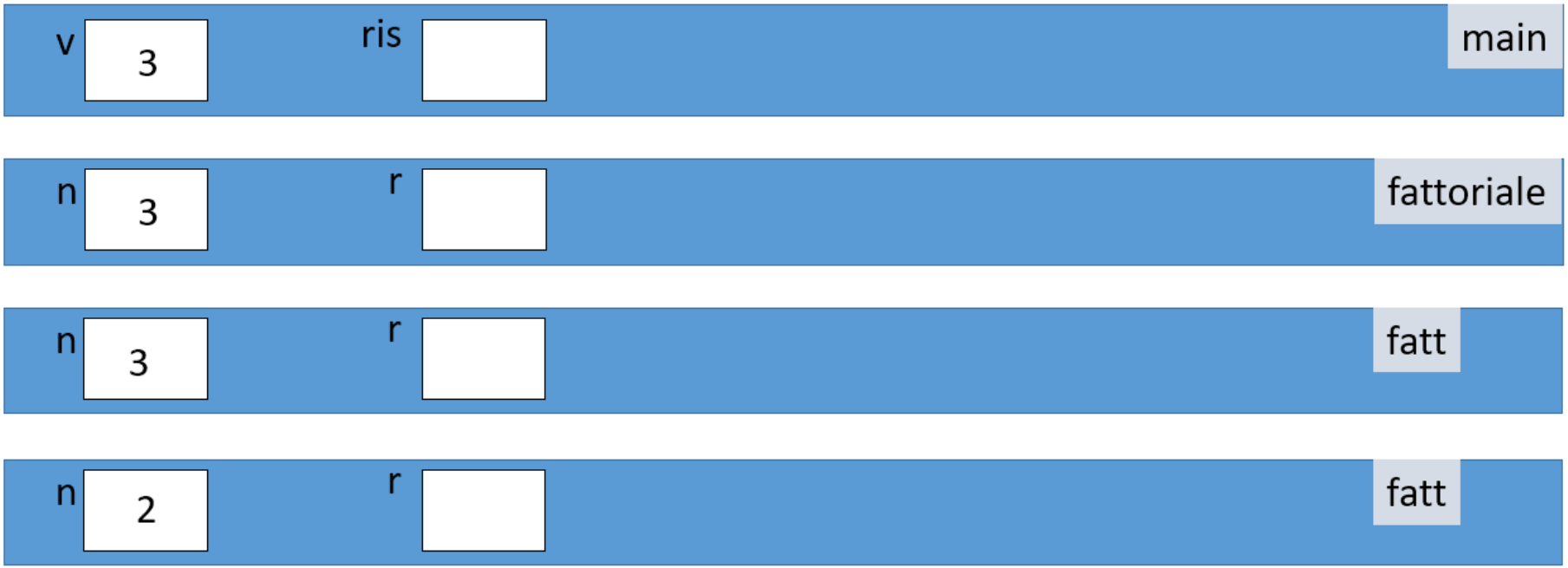
```
int main()
{
    int v;
    int ris;

    cout << "Inserire un intero: ";
    cin >> v;

    ris = fattoriale( v );
    if( ris < 0 )
        cout << "Fattoriale non definito";
    else
        cout << "Risultato: " << ris;

   return 0;
}
```

Esecuzione

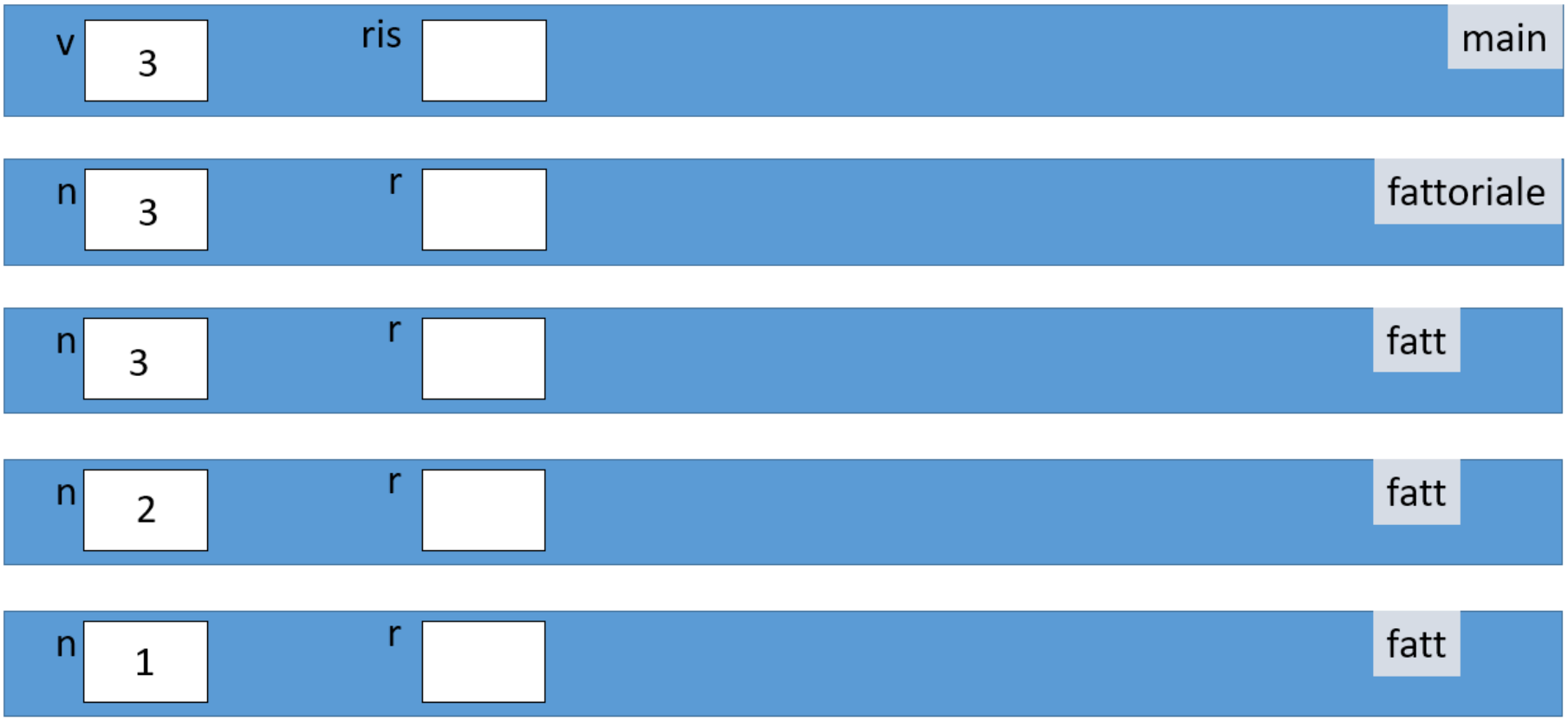
Esecuzione

Esecuzione

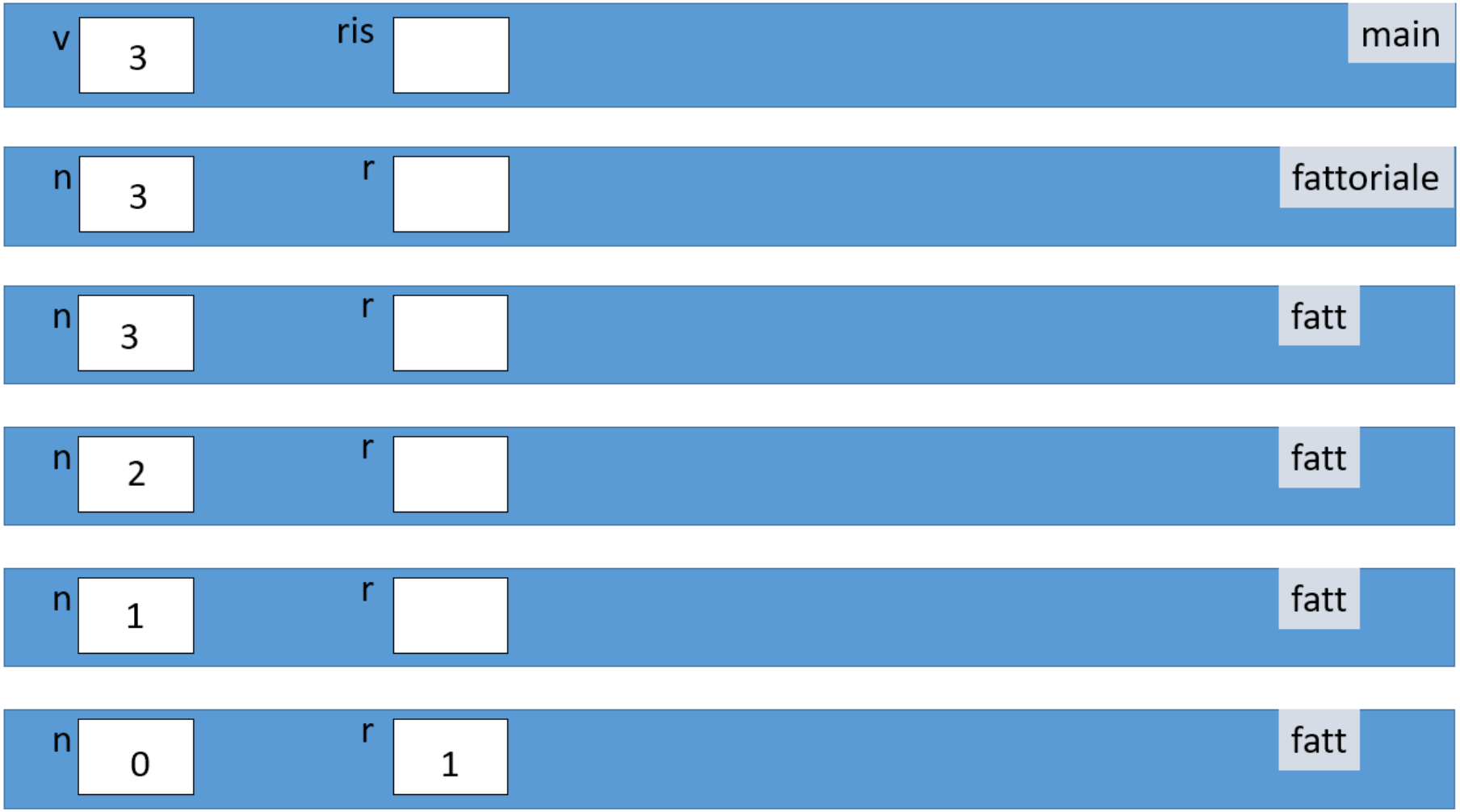
Esecuzione



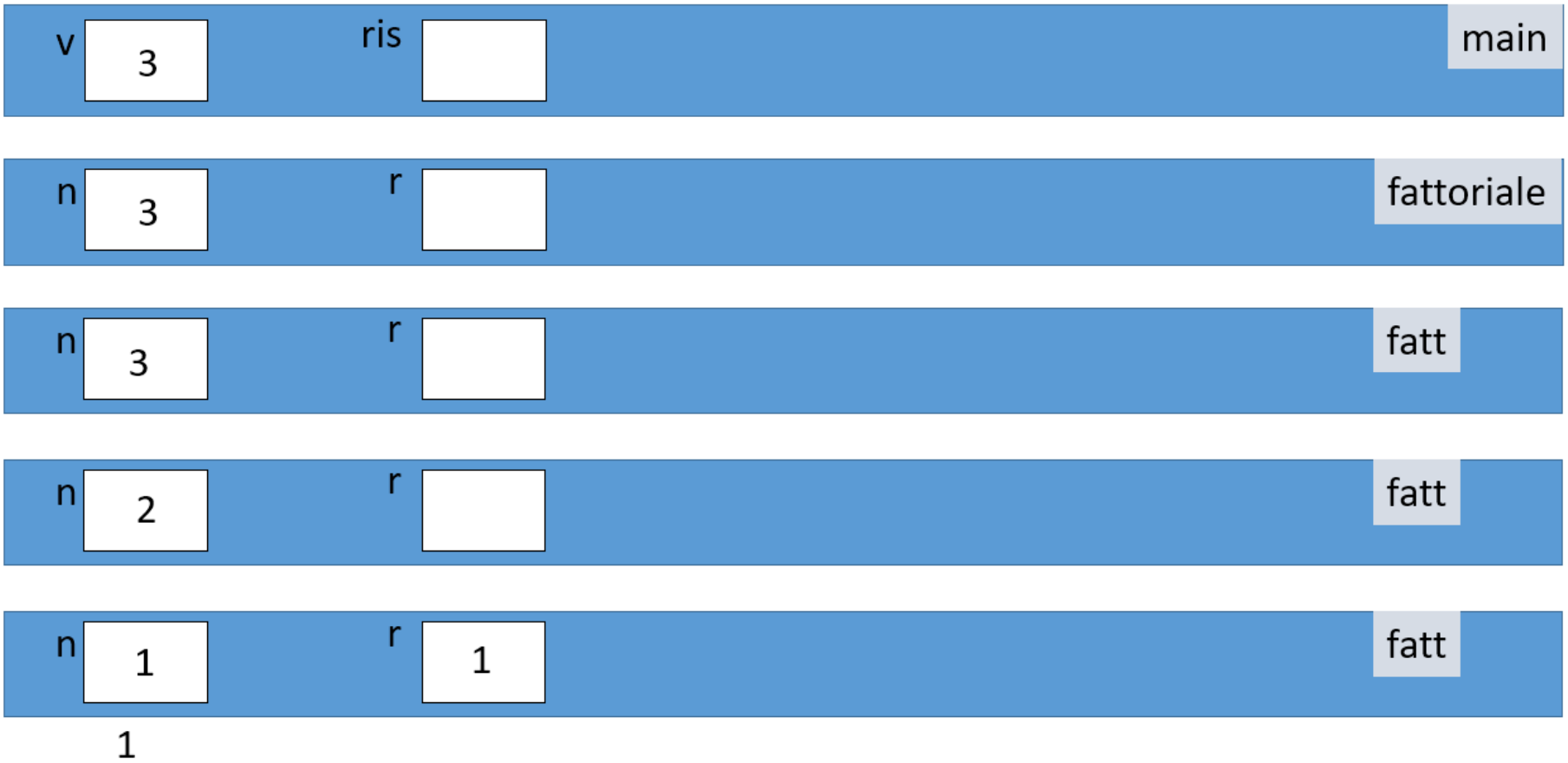
Esecuzione



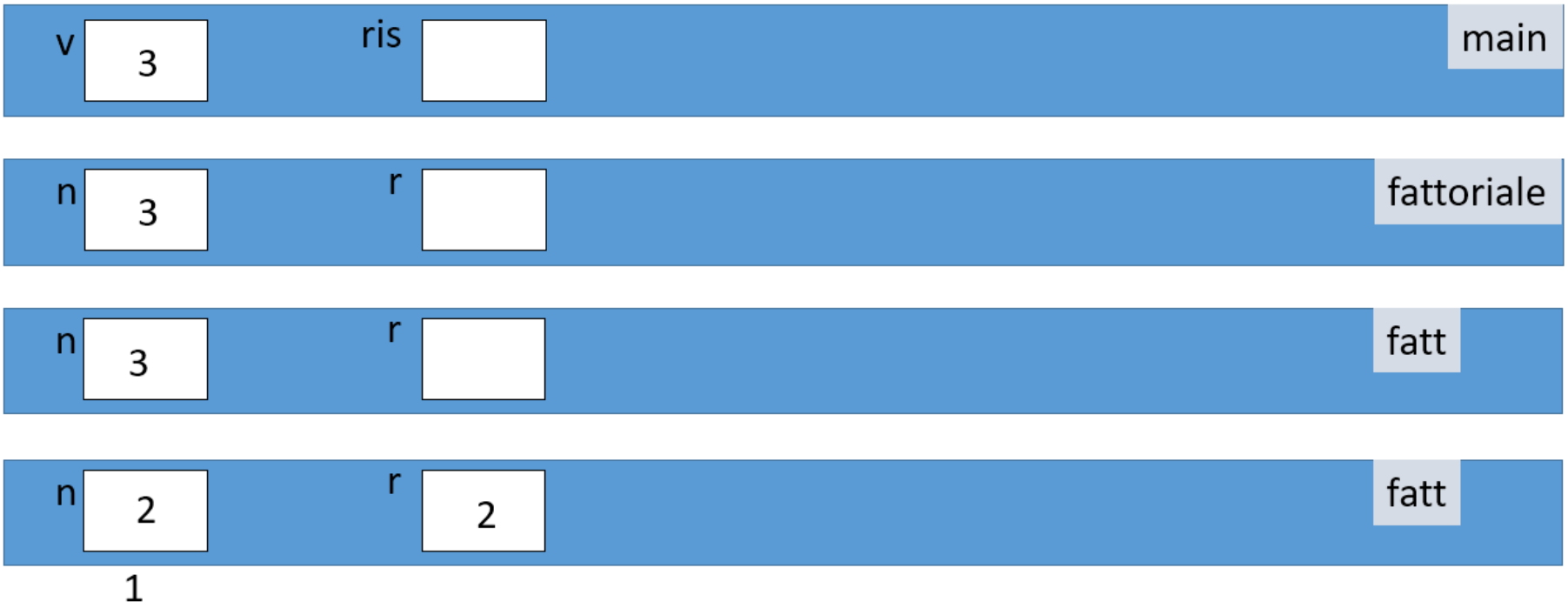
Esecuzione


v
3
ris
main
n
3
r
fattoriale
n
3
r
fatt
n
2
r
fatt
n
1
r
fatt
n
0
r
1
fatt

Esecuzione



Esecuzione



Esecuzione

Esecuzione

Esecuzione

Ricerca Ricorsiva

- Un esempio per imparare a sfruttare i puntatori, anche se così è inutilmente complicato
  Ma è molto utile dal punto di vista didattico
- Scriviamo una funzione
  `int cerca(int vett[], int len, int v)`
- Scopo: restituire la posizione nella quale il valore viene trovato (restituisce -1 se non è presente)

Comportamento della funzione
`int cerca(int vett[], int len, int v)`

- Se `len` vale 0, restituisce -1 (valore non trovato)
- **Base della Ricorsione:**
  se il valore in posizione 0 coincide, restituisce 0
- **Passo Ricorsivo:**
  altrimenti chiama sé stessa con `vett+1` e `len-1`; se ottiene un valore diverso da -1, lo incrementa e lo restituisce, altrimenti restituisce il -1

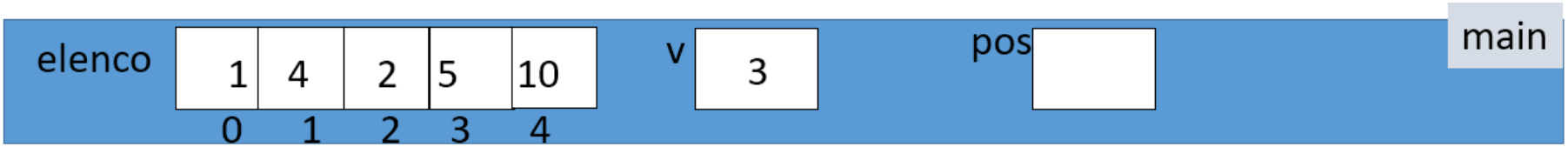
Programma: Ricerca.cpp

```
int cerca(int vett[], int len, int v)
{
    int pos=-1;

    if( len> 0 )
        if( vett[0] == v)
            pos = 0;
        else
        {
            pos = cerca(vett+1, len-1, v);
            if(pos >= 0)
                pos++;
        }

    return pos;
}
```

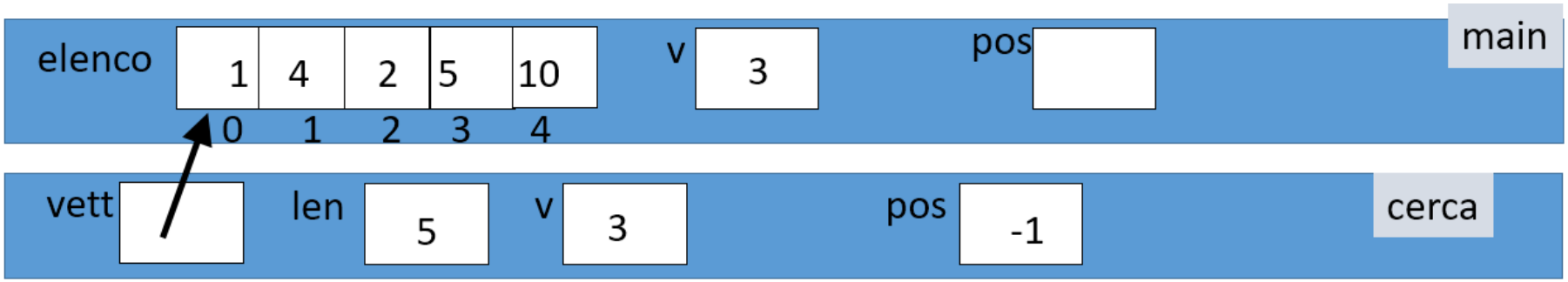
Ricorsione

```
int main()
{
    const int SIZE=5;
    int v, pos;
    int elenco[SIZE] = {1, 4, 2, 5, 10};

    cout << "Inserire un intero: ";
    cin >> v;

    pos = cerca(elenco, SIZE, v);
    if( pos < 0 )
        cout << "Valore non trovato";
    else
        cout << "Posizione: " << pos;

   return 0;
}
```

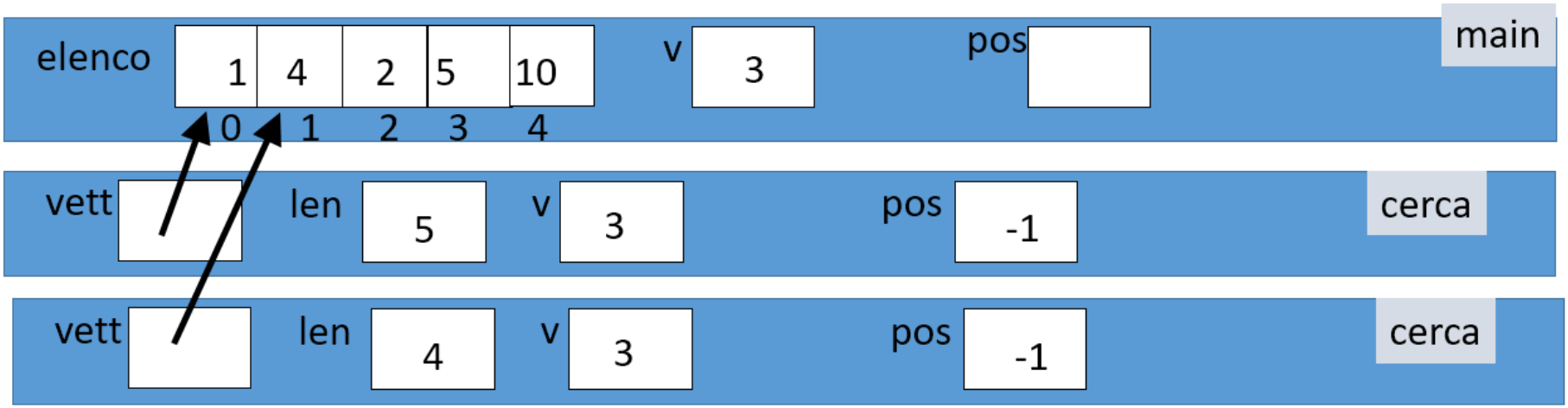
Ricerca Valore: 3



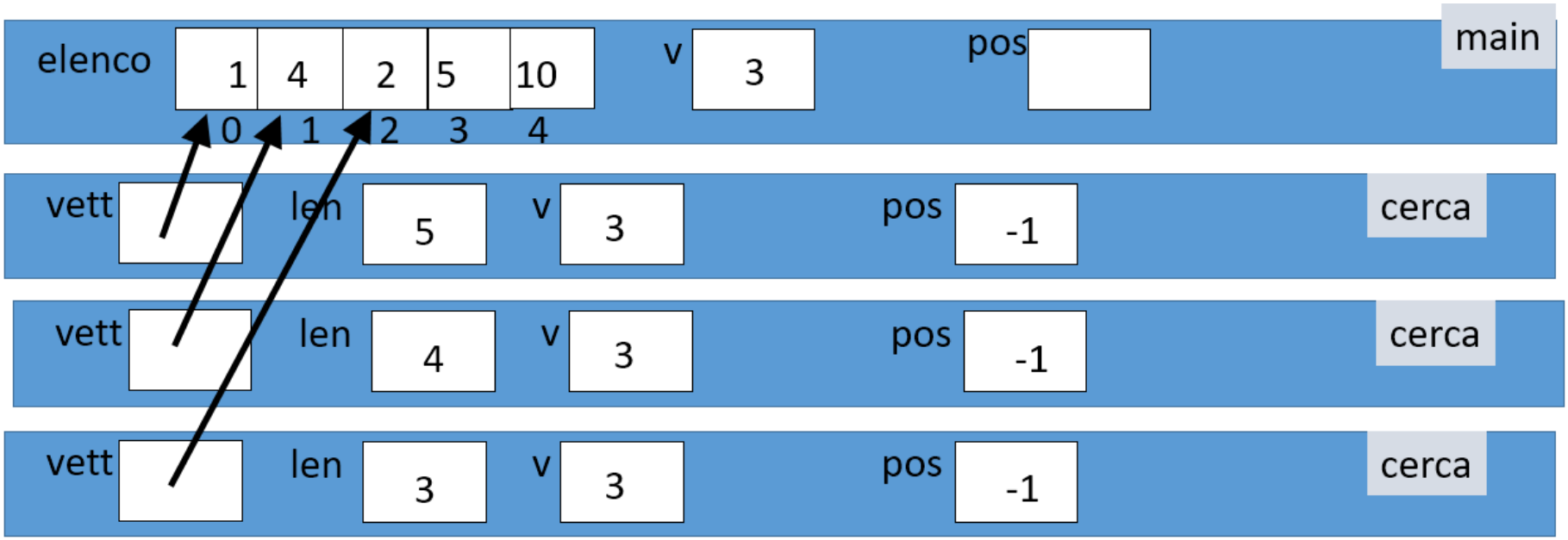
Ricerca Valore: 3



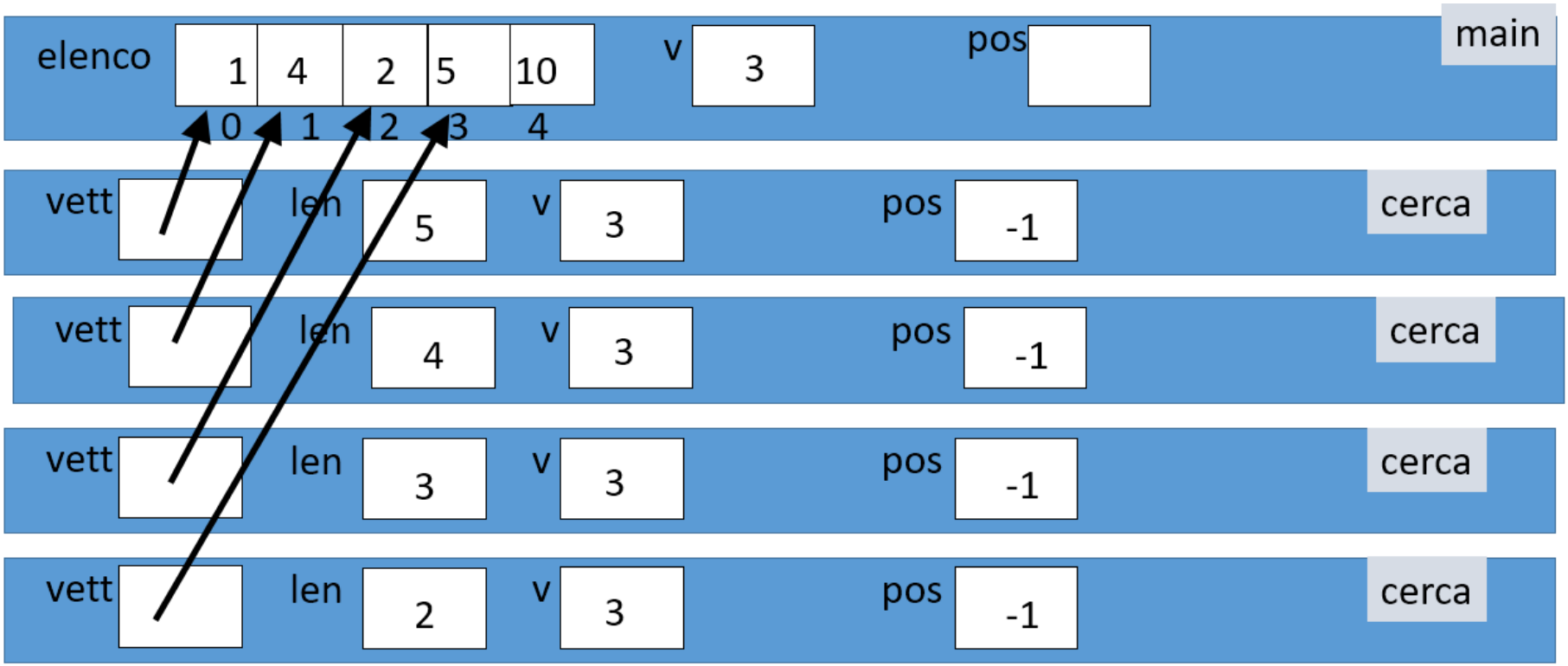
Ricerca Valore: 3



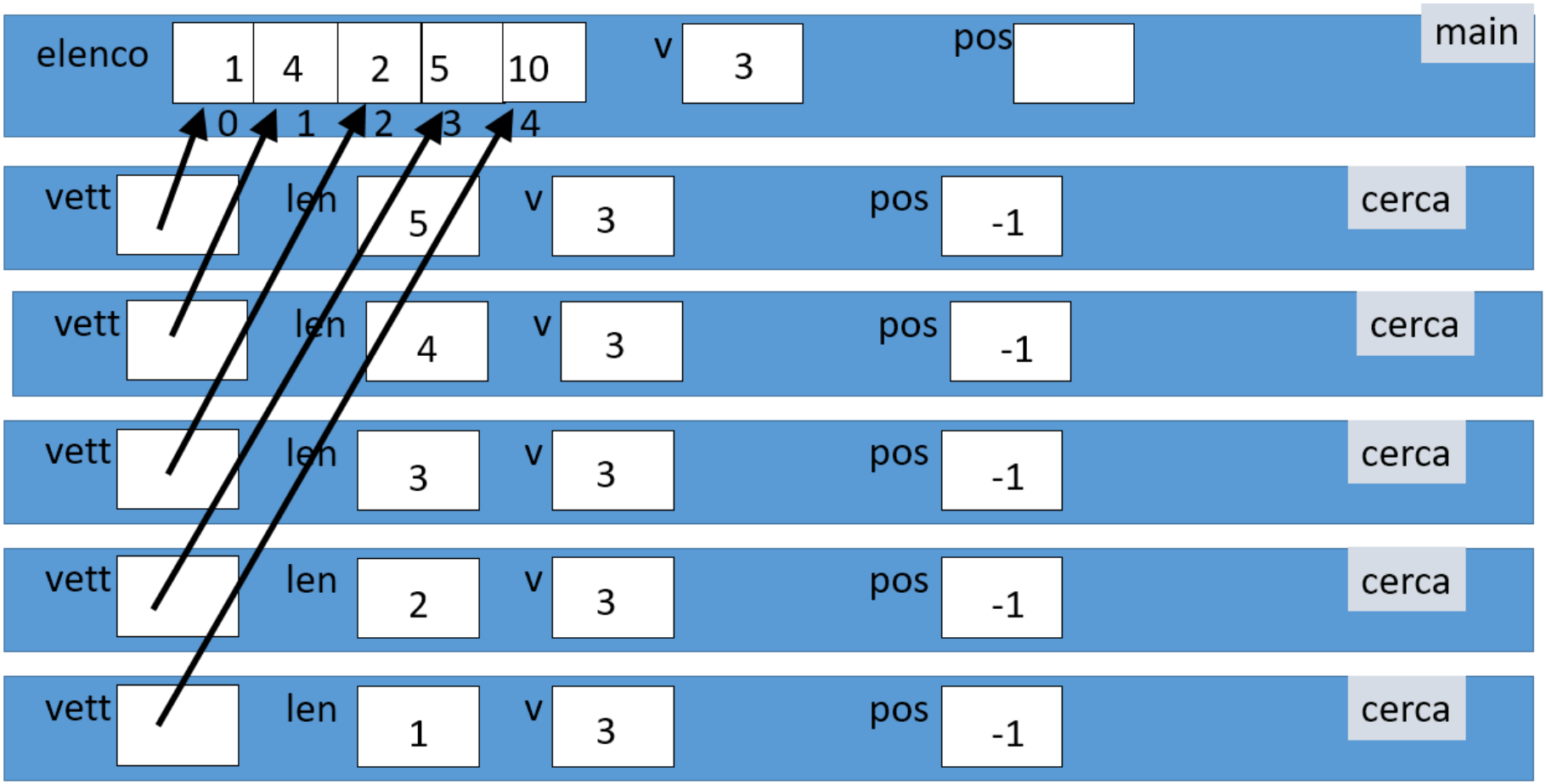
Ricerca Valore: 3



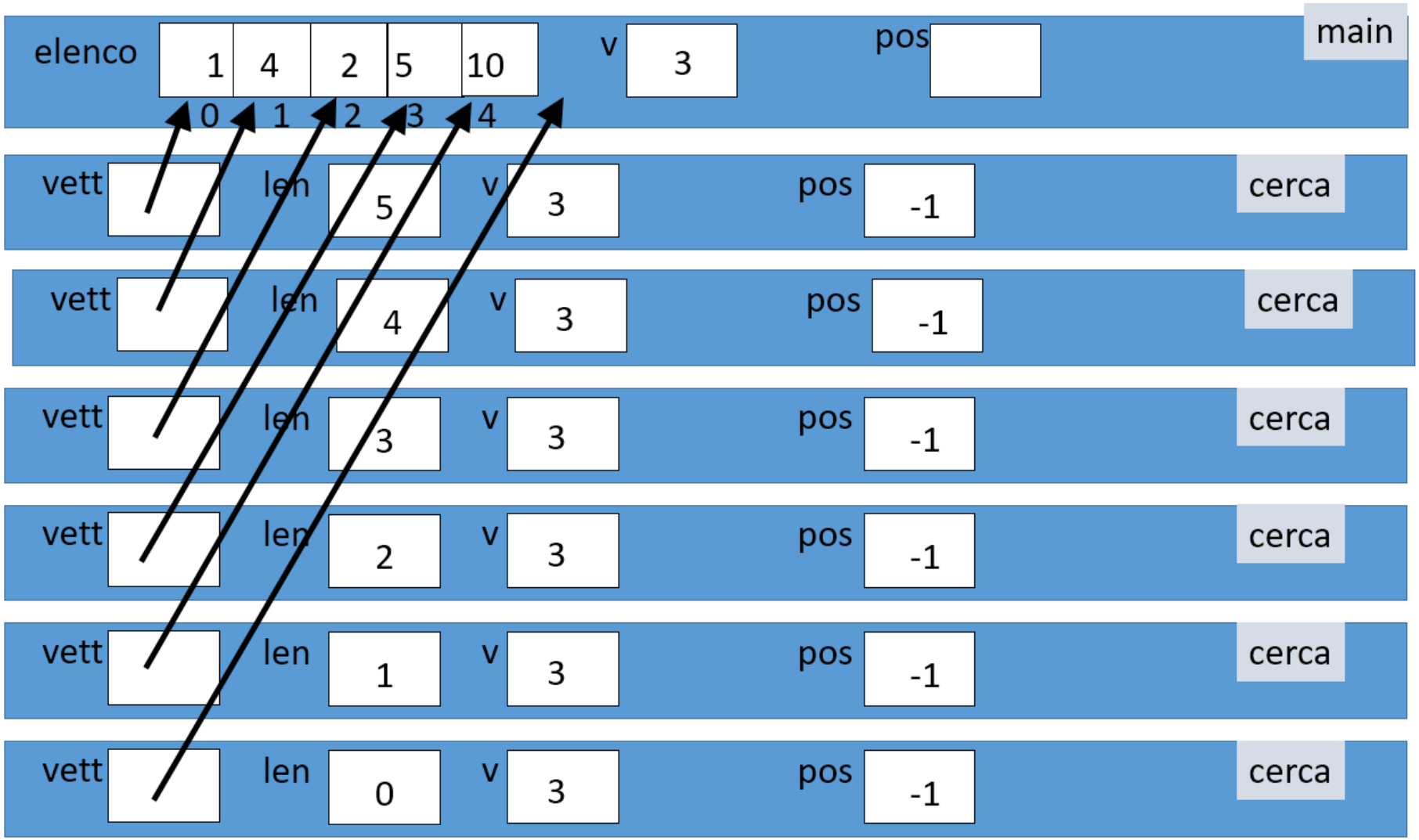
Ricerca Valore: 3



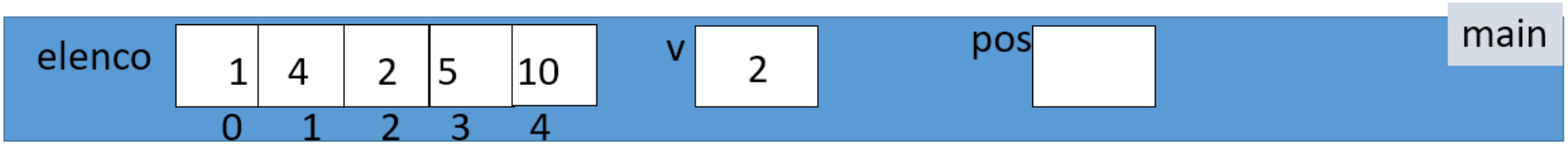
Ricerca Valore: 3



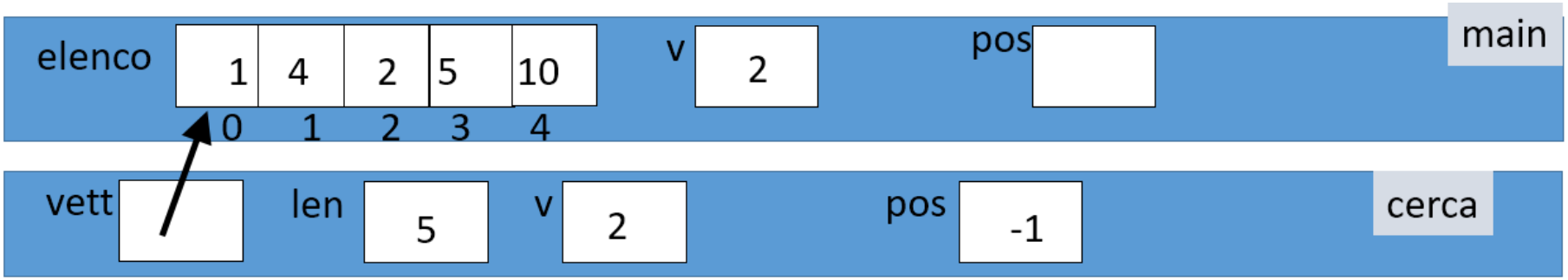
Ricerca Valore: 3



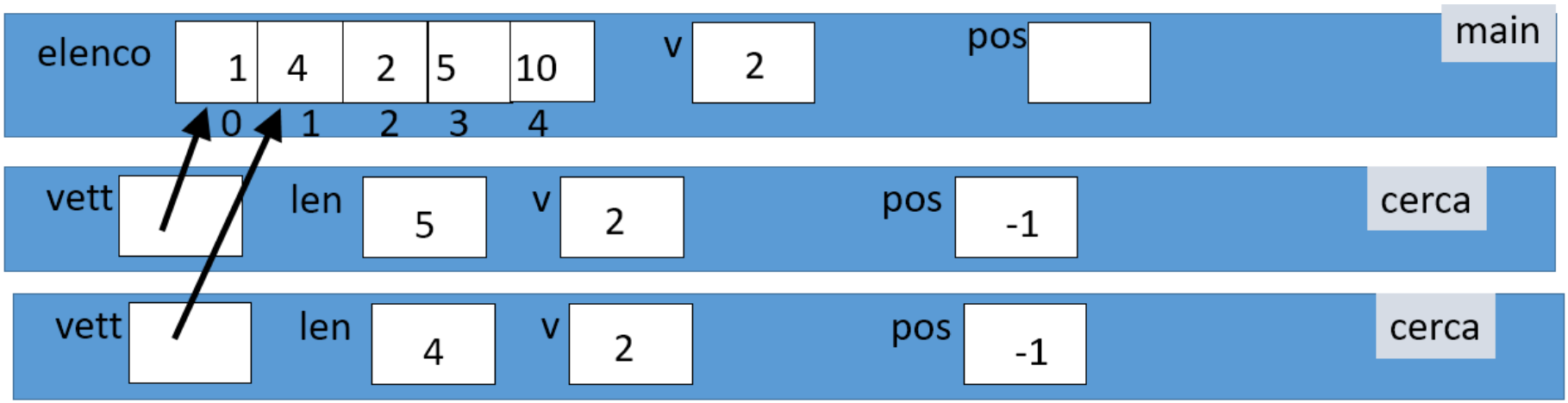
Ricerca Valore: 2



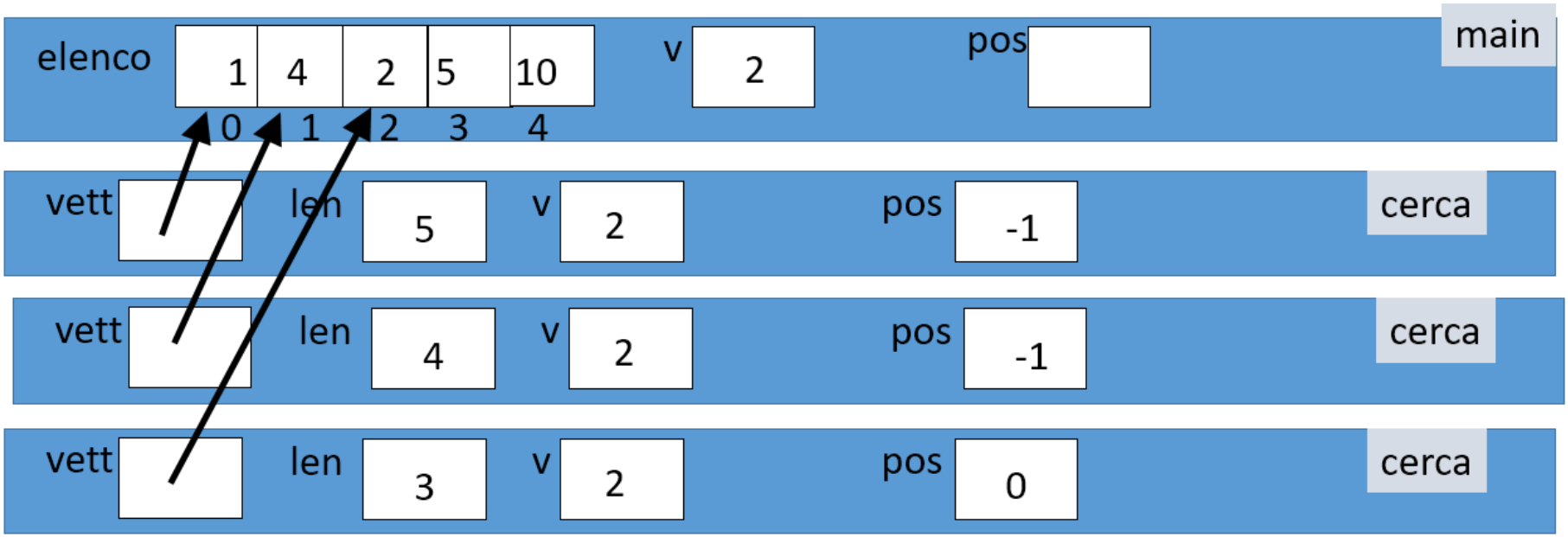
Ricerca Valore: 2



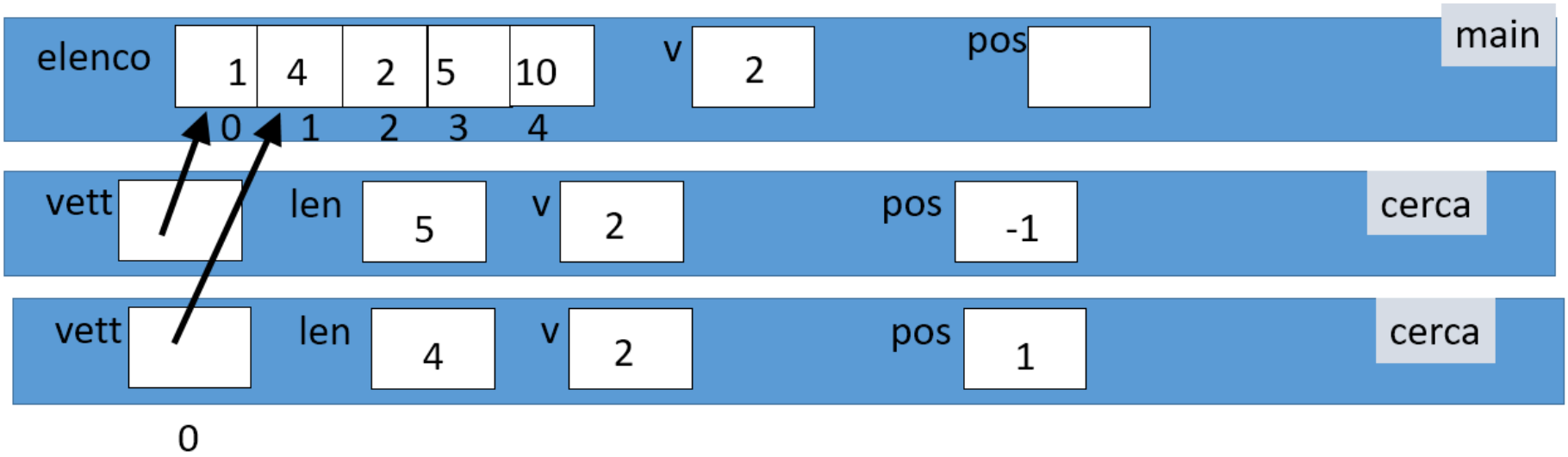
Ricerca Valore: 2



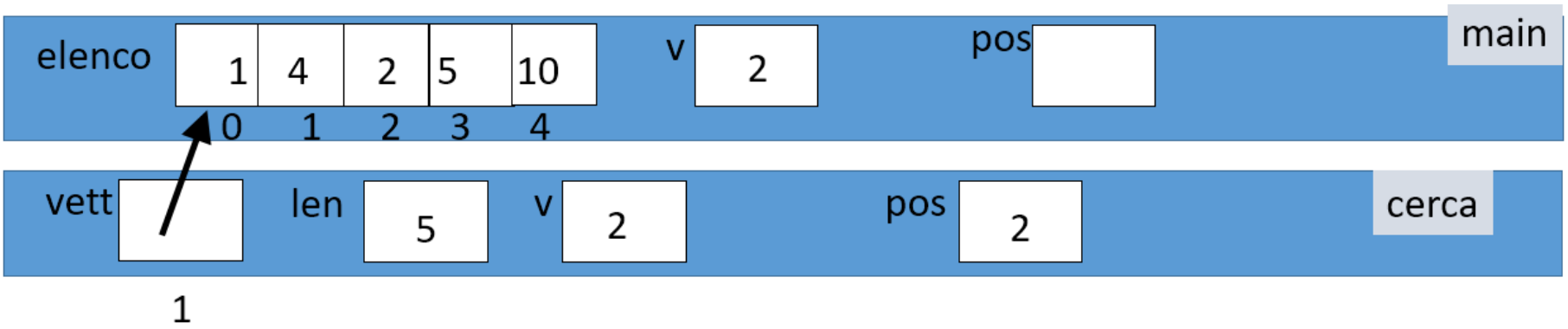
Ricerca Valore: 2



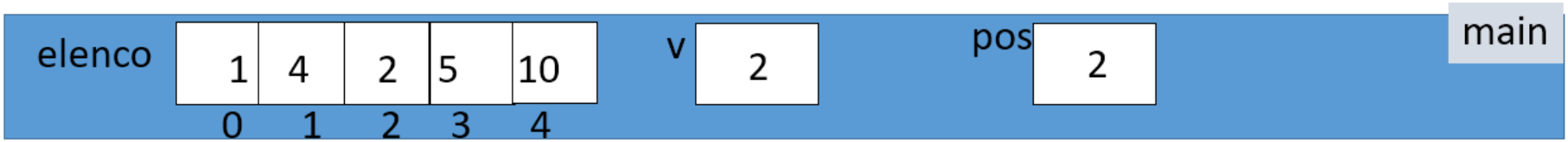
Ricerca Valore: 2



Ricerca Valore: 2



Ricerca Valore: 2



2

Strutture

- Supponiamo di dover gestire dei punti nel piano cartesiano;
- servono due valori: uno è la coordinata sull'asse x, l'altro è la coordinata sull'asse y.
- Soluzione: due variabili.

Strutture

- Ma se dobbiamo gestire degli **Elenchi di Punti**?
- Occorre uno strumento che consenta di gestire il **punto** come un **Tutt'uno**
- Questo concetto prende il nome di **STRUTTURA**

Definizione

```
struct PUNTO
{
    float x;
    float y;
};
```

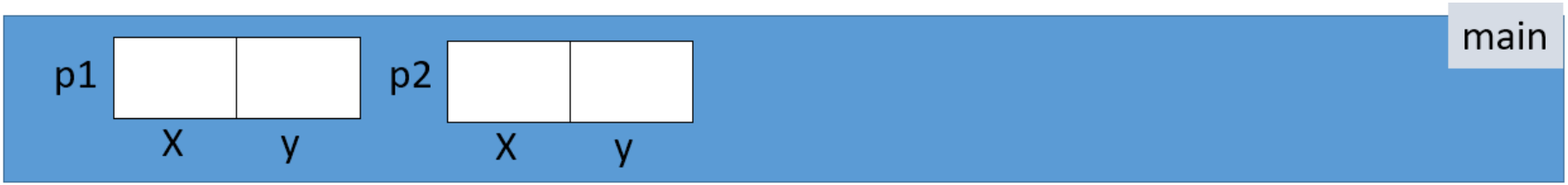
Strutture

- `x` e `y` sono detti **Campi** o **Membri** della struttura.
- In C++, `PUNTO` è un nuovo **Nome di Tipo**, che può essere usato per definire variabili.

Programma: Punti_01.cpp

```
int main()
{
    PUNTO p1, p2;

    cout << "Inserire x: ";
    cin >> p1.x;
    cout << "Inserire y: ";
    cin >> p1.y;
    stampa_punto(p1); cout << endl;

    p2.x = p1.y;
    p2.y = p1.x;
    stampa_punto(p2); cout << endl;


    p1 = p2;
    stampa_punto(p1); cout << endl;

   return 0;
}
```

In Memoria Centrale



Strutture

- Data una variabile strutturata, come `p1`, come accedere ai suoi campi?
- Con la **Notazione Puntata** (in Inglese, **Dot Notation**)
- `p1.x`

Strutture

- È possibile assegnare una struttura ad una variabile strutturata?
- Sì, infatti

```
p1 = p2;
```

Strutture

- È possibile definire un parametro di tipo strutturato?
- Sì, funzione `stampa_punto`

Strutture

```
void stampa_punto(PUNTO pt)
{
    cout << "(" << pt.x << ","
         << pt.y << ")";
}
```

Strutture

- ATTENZIONE: il meccanismo di chiamata **Non Cambia**
- La funzione `stampa_punto` riceve il parametro `pt` per **Copia**

Vettori di Strutture

- Nel programma **Punti_02.cpp**
  viene gestito un **Elenco di Punti**
- Si definisce un vettore
  `PUNTO elenco[SIZE]`
  nella funzione `main`
  che viene ricevuto dalla funzione con il parametro
  `PUNTO v[]`

Vettori di Strutture

- Come accedere ai campi dell'elemento in posizione `i`?
  `v[i].x`
  cioè combinando la notazione vettoriale e la notazione puntata
- In alternativa, dal puntatore si può usare l'operatore freccia `->`
  `v->x` `(v+i)->x`
  che equivalgono a scrivere
  `(*v).x` `(*(v+1)).x`

Strutture e Tipi

- Nel C++, il nome della struttura è automaticamente il nuovo nome di tipo.
- Nel C no.
  È necessario usare sempre la parola `struct`
  Vedi programma **Punti_03.cpp**

Strutture e Tipi

- Tuttavia, ripetere sempre `struct` è scomodo
- Quindi, era diventata prassi usare l'operatore `typedef`
  Vedi Programma **Punti_04.cpp**

```
typedef struct st_PUNTO
{
    float x;
    float y;
} PUNTO;
```

Vettori di Caratteri

- Come rappresentare i testi?
- Il tipo base `char` lo abbiamo già visto
- Ma rappresenta **Un SOLO Carattere**, non un testo.
- Come fa il compilatore a gestire le costanti stringa, per esempio
  `"PIPPO"` ?
- Come un vettore di caratteri

Stringa Costante

- OK, ma questo vuol dire che se lo stampiamo
  `cout << "PIPPO";`
  all'operatore << viene passato dell'altro, oltre al puntatore?
- NO, occorre indicare che la stringa è finita.

Stringa Costante

- Il carattere '\0' è il carattere con il codice ASCII 0
- Viene detto **Marcatore di Fine Stringa**
- Nel caso delle costanti, viene aggiunto automaticamente dal compilatore
- Questa convenzione vale per tutte le funzioni e tutti gli operatori che operano sulle stringhe

Vettori di Caratteri

- Per riservare lo spazio per le stringhe, usiamo i **Vettoi di Caratteri**
- Vogliamo memorizzare un nome di lunghezza massima di 50 caratteri?
  `char nome[51];`
  Serve **SEMPRE** un carattere in più per il `\0`

Input/Output

- Gli operatori di Input/Output che conosciamo funzionano senza problemi (o quasi)
- `char nome[151];`
  `cin >> nome;`
  `cout << nome`
  Programma **Stringhe_01.cpp**

Input/Output

- L'operatore >> interrompe la lettura quando trova uno spazio.
- Come fare a leggere i nomi con gli spazi?
  Usando la funzione `getline` fornita dal dispositivo di ingresso `cin`
  `cin.getline(nome, 51);`
  Programma **Stringe_02.cpp**
- Il secondo parametro è il massimo numero di caratteri da inserire nel vettore, **Incluso il** `\0`

Funzioni di Libreria

- Nella libreria `cstring` (o `string.h`, nel vecchio C) vengono definite molte funzioni per lavorare con le stringhe. Vediamo le principali.
- `char *strcpy(char d[], char s[]);`
  Copia da `s` a `d`
- `char *strcat(char d[], char s[]);`
  concatena da `s` a `d`
- `int strlen( char s[]);`
  lunghezza di `s`

Funzioni di Libreria

- `int strcmp(char a[], char b[]);`
  Confronta i contenuti di `a` e di `b`.
  Restituisce `0` se i contenuti di `a` e di `b` sono uguali.
  Restituisce `< 0` se il contenuto di `a` precede alfabeticamente il contenuto di `b`.
  Restituisce `> 0` se il contenuto di `a` segue alfabeticamente il contenuto di `b`.

Funzioni di Libreria

- Esempio:
  `if(strcmp(nome, cognome)==0)`
  è il confronto corretto per verificare che `nome` e `cognome` siano uguali
- Invece:
  `if(nome ==cognome)`
  è scorretto, ma non dà errore, perché confronta i puntatori (quindi è sempre falso)

Funzioni di Libreria

- Esempio:
  Programma **Stringhe_03.cpp**

Sfida

- Scrivere un programma che legge da tastiera una sequenza di noni;
- l'inserimento termina quando l'utente inserisce il nome `"ESCI"`;
- terminata la lettura, il programma stampa il nome alfabeticamente più grande.
- Facoltativo: dopo aver letto un nome, convertire tutte le sue lettere in maiuscolo, scrivendo e chiamando la funzione

```
void maiuscolo(char s[]);
```

Ricerca Dicotomica

- Dato un vettore ordinato, la ricerca dicotomica sfrutta questo ordinamento.
- Dato un valore da cercare `v`,
  esso viene confrontato con il valore centrale `c` del vettore:
  - se `v < c`, allora `v` si trova nella prima metà del vettore;
  - se `v > c`, allora `v` si trova nella seconda metà del vettore;
  - se `v == c`, allra `v` è il centro del vettore.

Si può fare sia in modo iterativo che ricorsivo.

Sfida

- Scrivere un programma che legge da tastiera un vettore di `SIZE=10` elementi. Il vettore viene ordinato al termine della lettura.
- Richiedere un numero da cercare, quindi effettuare la ricerca chiamando al funzione `Ricerca`, alla quale viene passato l'indirizzo del primo elemento del vettore, il numero di elenti e il valore da cercare; la funzione restituisce `-1` se non trova il valore, altrimenti restituisce la posizione n cui ha trovato il valore.

Sfida (continua)

- Fate due versioni:
  - la versione iterativa realizza la ricerca nella funzione `Ricerca` in modo iterativo;
  - la versione ricorsiva realizza la funzione `Ricerca` in modo ricorsivo.

Approccio 1/2

- `lb`: lower bound, posizione inferiore;
  `uop`: upper bound, posizione superiore;
  all'inizio `lb=0` e `up = SIZE-1`
  `v`: valore cercato
- `dist = ub - lb +1`
  `centro = (dist/2)+lb`
- Se `vett[centro]` contiene il valore cercato
  si restituisce `centro`

Approccio 2/2

Passo Ricorsivo:

- se `v < vett[centro]`
  si cerca a **sinistra** di `centro`, quindi
  `ub = centro-1`
- se `v > vett[centro]`
  si cerca a **destra** di `centro`, quindi
  `lb = centro+1`

Base della Ricorsione:

- Si raggiunge quando `!( lb <= ub)`
  si restituisce `-1` (non trovato)