

# Informatica - Mod. Programmazione

## Lezione 06

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica  
Università di Bergamo

- Un **Sotto-Programma** è una porzione di programma che svolge un lavoro necessario ad altre parti del programma (una o più)
- Nel C/C++, lo strumento per creare sotto-programmi si chiama **FUNZIONE**

## Funzione

- Il concetto deriva dal concetto matematico di funzione
- Esempio:  
$$y = f(x) = x^2 + 2x + 3$$
- $x$  viene chiamata **argomento** o **parametro** della funzione.  
in base al suo valore, la funzione  $f(x)$  fornisce un diverso valore.
- Come scriverlo nel linguaggio di programmazione?

```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

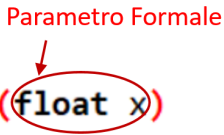
Nome della funzione

```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

Parametro Formale

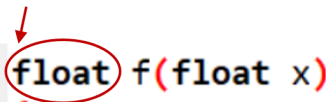


```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

Tipo del Valore Restituito

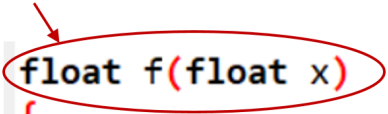


```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

Intestazione della Funzione



```
float f(float x)
```

```
{
```

```
    float r;
```


```
    r = x*x + 2*x + 3;
```

```
    return r;
```

```
}
```



Corpo/Implementazione della Funzione



```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

```
float f(float x)
{
    float r;

    r = x*x + 2*x + 3;

    return r;
}
```

Interrompe la Funzione  
Restituendo il valore  
della variabile r

return r;

## Uso della Funzione

- Una volta definita, la funzione deve essere **CHIAMATA**
- Per esempio, dal programma principale
- Esempio: scriviamo un programma che legge da tastiera un numero (in virgola mobile) e stampa il valore ottenuto applicando la funzione  $f(x)$  a quel numero.

Programma: Funzioni\_01.cpp

```
int main()  
{  
    float v;  
    float ris;  
  
    cout << "Inserire un valore: ";  
    cin >> v;  
  
    ris = f( v );  
    cout << "Risultato: " << ris;  
  
    return 0;  
}
```

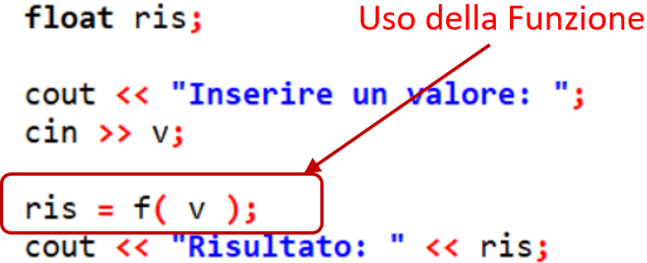
```
int main()
{
    float v;
    float ris;

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

    ris = f( v );
    cout << "Risultato: " << ris;

    return 0;
}
```

Uso della Funzione



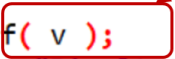
```
int main()
{
    float v;
    float ris;

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

    ris = f( v );
    cout << "Risultato: " << ris;

    return 0;
}
```

Chiamata della Funzione



```
int main()
```

```
{
```

```
    float v;
```

```
    float ris;
```

```
    cout << "Inserire un valore: ";
```

```
    cin >> v;
```

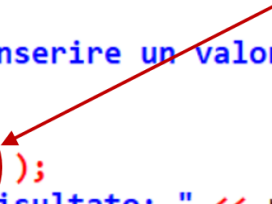
```
    ris = f(v);
```

```
    cout << "Risultato: " << ris;
```

```
    return 0;
```

```
}
```

Parametro Attuale  
(il valore di v)



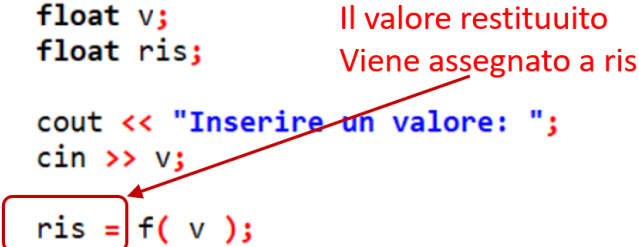
```
int main()
{
    float v;
    float ris;

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

    ris = f( v );
    cout << "Risultato: " << ris;

    return 0;
}
```

Il valore restituito  
Viene assegnato a ris





## Meccanismo della Chiamata

- Il frammento di codice che effettua la chiamata si sospende
- L'esecuzione salta al codice della funzione
- Quando la funzione chiamata termina e restituisce il valore,
- l'esecuzione ritorna nel frammento chiamante, riprendendo con l'eventuale uso del valore restituito

## La **Funzione** `main`

- Ora sappiamo finalmente che cosa è il programma principale
- È una **FUNZIONE** che viene chiamata all'avvio del programma dal Sistema Operativo
- Quindi l'istruzione `return` restituisce il valore al Sistema Operativo

## Variabili Locali

- Le variabili definite in una funzione sono dette **Variabili LOCALI**
- Queste variabili sono **Visibili SOLAMENTE all'interno della funzione** nella quale sono definite

## Variabili Globali

- Le variabili possono essere definite anche **FUORI** dalle funzioni
- Queste variabili sono dette **GLOBALI** e sono viste da **TUTTE** le funzioni
- **ATTENZIONE:** le variabili globali possono creare seri problemi, quindi  
**MENO SI USANO, MEGLIO È**

## Ambiente di Visibilità

- L'insieme di variabili visibili all'interno di una funzione viene detto

### **Ambiente di Visibilità** della funzione

- L'ambiente di una funzione contiene:
  - Tutte le variabili locali (inclusi i parametri formali)
  - Tutte le variabili globali che non sono mascherate da omonime variabili locali

## Parametri Formali

- I **Parametri Formali**, di fatto, sono delle **Variabili Locali PRE-INIZIALIZZATE**
- Il loro valore arriva dal **Chiamante**
- E viene letteralmente **COPIATO** nel parametro formale
- Si parla, infatti, di **Passaggio dei Parametri per COPIA**

# Funzioni

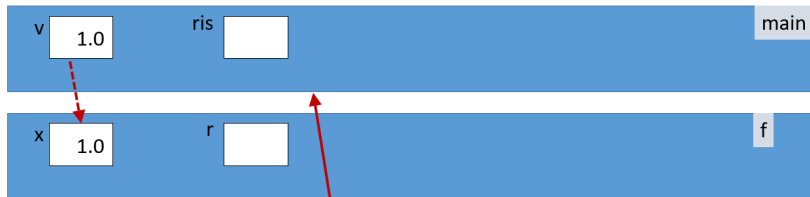


# Funzioni





# Funzioni



Record di Attivazione  
delle Funzioni

## Record di Attivazione e Stack delle Chiamate

- Quando una funzione viene attivata, il suo record di attivazione viene creato
- in una specifica area di memoria detta

### **Stack delle Chiamate**

- stack (pila) perchè il record di attivazione viene impilato sotto (o sopra, dipende da come lo disegniamo) il record di attivazione della funzione chiamante

## Prima della Chiamata



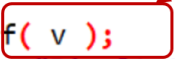
```
int main()
{
    float v;
    float ris;

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

    ris = f( v );
    cout << "Risultato: " << ris;

    return 0;
}
```

Chiamata della Funzione



## Creazione del Record di Attivazione

v	<input type="text" value="1.0"/>	ris	<input type="text"/>	main
x	<input type="text"/>	r	<input type="text"/>	f

## Copia del Valore dei Parametri Attuali nei Parametri Formali



## Record di Attivazione e Stack delle Chiamate

- La funzione chiamata lavora, usando il valore del parametro formale
- L'istruzione `return` memorizza il valore tra i due record di attivazione
- Poi il record di attivazione della funzione chiamata viene rimosso dallo stack delle chiamate

## Calcola il Valore della Variabile $r$





## Copia il Valore da Restituire tra i due Record di Attivazione



return r;

## Rimuove il Record di Attivazione della Funzione che Ha Terminato



## Record di Attivazione e Stack delle Chiamate

- La chiamata è terminata
- Riprende l'esecuzione del chiamante
- Il valore restituito dalla funzione chiamata è ora disponibile e viene usato

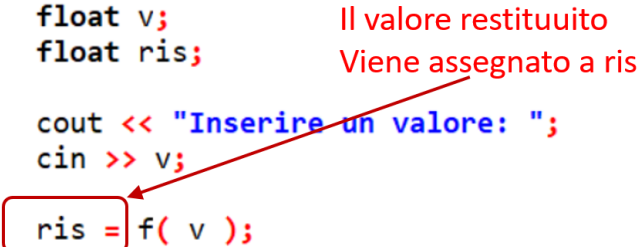
```
int main()
{
    float v;
    float ris;

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

    ris = f( v );
    cout << "Risultato: " << ris;

    return 0;
}
```

Il valore restituito  
Viene assegnato a ris



## Il Valore Restituito Viene Usato



## Rimane Solo il Record di Attivazione del Chiamante



## Procedures e Parametri Formali Multipli

- Vogliamo scrivere una funzione che, dati due numeri  $x$  e  $y$ , fornisca qual è il minimo e qual è il massimo, attraverso due ulteriori parametri `min` e `max`
- Iniziamo dalla versione che non funziona:  
Programma: **Funzioni\_02.cpp**

## Funzione confronto

```
void confronto(float x, float y,  
              float min, float max)  
{  
    if(x > y)  
    {  
        min = y;  
        max = x;  
    }  
    else  
    {  
        min = x;  
        max = y;  
    }  
    return;  
}
```



## Procedures e Parametri Formali Multipli

- Osservate il tipo restituito dalla funzione: **void**
- È il tipo **Vuoto**  
Cioè **Nessun Valore**
- Abbiamo così definito una **PROCEDURA**, cioè una **Funzione che NON Restituisce Alcun Valore**
- Notate l'istruzione `return` senza espressione

## Funzione main

```
int main()
{
    float v1, v2;
    float min=0, max=0;

    cout << "Inserire valore 1: ";
    cin >> v1;
    cout << "Inserire valore 1: ";
    cin >> v2;

    confronto(v1, v2, min, max);

    cout << "min: " << min
    << " max: " << max;

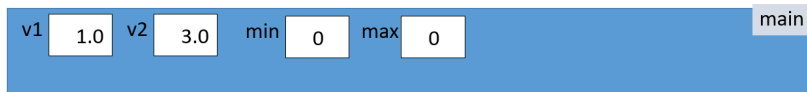
    return 0;
}
```

## Esecuzione

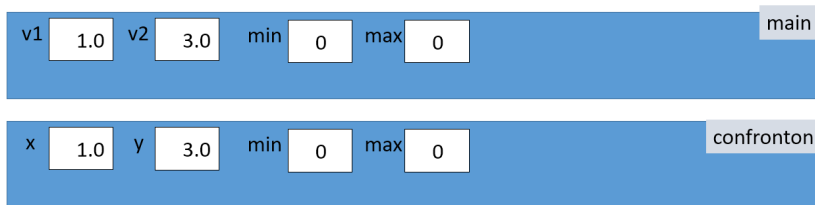
```
Inserire valore 1: 1  
Inserire valore 1: 3  
min: 0 max: 0
```

- Perché non funziona?
- Perché gli assegnamenti che modificano `min` e `max` nella funzione confronto modificano i parametri formali (passaggio dei parametri per copia).

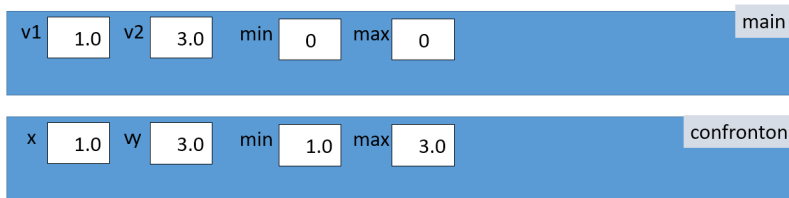
## Variabili nella Funzione `main`



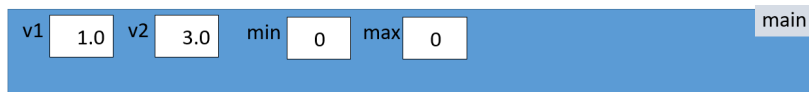
## Chiamata (con Copia dei Parametri Attuali)



## Modifica (delle Copie)



## Ritorno a main





- Come si può risolvere il problema?
- Senza introdurre niente di nuovo
- Usando i **Puntatori**

Programma: **Funzioni\_03.cpp**

## Funzione confronto

```
void confronto(float x, float y,  
               float *min, float *max)  
{  
    if(x > y)  
    {  
        *min = y;  
        *max = x;  
    }  
    else  
    {  
        *min = x;  
        *max = y;  
    }  
    return;  
}
```

## Funzione main

```
int main()
{
    float v1, v2;
    float min=0, max=0;

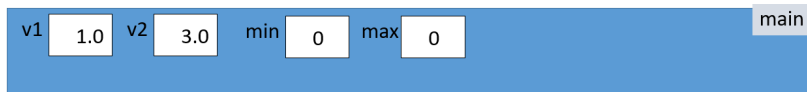
    cout << "Inserire valore 1: ";
    cin >> v1;
    cout << "Inserire valore 1: ";
    cin >> v2;

    confronto(v1, v2, &min, &max);

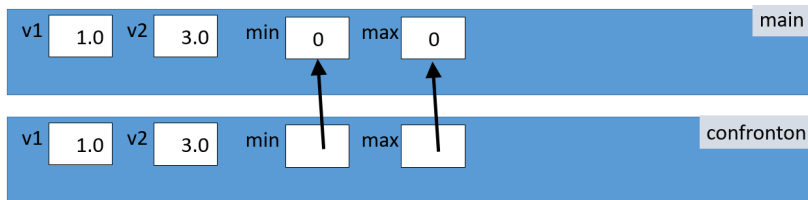
    cout << "min: " << min
         << " max: " << max;

    return 0;
}
```

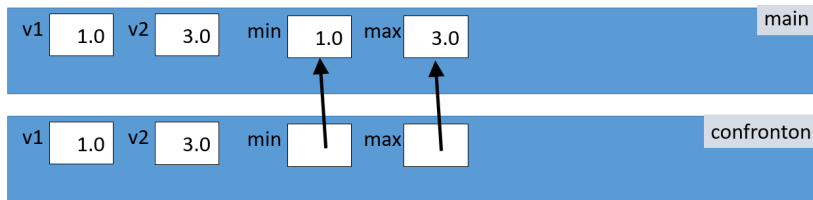
## Variabili nella Funzione `main`



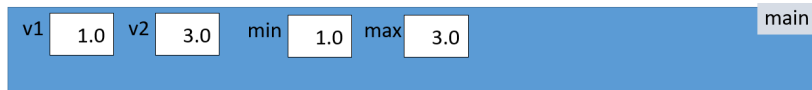
## Attivazione della Funzione confronto



## Modifica



## Situazione Finale



## Passaggio dei Parametri per Reference

- Il C++ ha introdotto una nuova modalità di passaggio dei parametri

### il **Passaggio dei Parametri per Reference** **(Riferimento)**

- In pratica, fa implicitamente quello che abbiamo fatto nell'esempio precedente:

### **Copia l'Indirizzo della Variabile Fornita come Parametro Attuale**

- Ma lo fa in automatico, semplificando l'uso dei parametri formali e la chiamata
- Esempio: Programma **Funzioni\_04.cpp**



## Funzione confronto

```
void confronto(float x, float y,  
               float &min, float &max)  
{  
    if(x > y)  
    {  
        min = y;  
        max = x;  
    }  
    else  
    {  
        min = x;  
        max = y;  
    }  
    return;  
}
```

## Funzione confronto

```
void confronto(float x, float y,  
               float &min, float &max)  
{  
    if(x > y)  
    {  
        min = y;  
        max = x;  
    }  
    else  
    {  
        min = x;  
        max = y;  
    }  
    return;  
}
```

## Funzione main

```
int main()
{
    float v1, v2;
    float min=0, max=0;

    cout << "Inserire valore 1: ";
    cin >> v1;
    cout << "Inserire valore 1: ";
    cin >> v2;

    confronto(v1, v2, min, max);

    cout << "min: " << min
         << " max: " << max;

    return 0;
}
```

## Funzione main

```
int main()
{
    float v1, v2;
    float min=0, max=0;

    cout << "Inserire valore 1: ";
    cin >> v1;
    cout << "Inserire valore 1: ";
    cin >> v2;

    confronto(v1, v2, min, max);

    cout << "min: " << min
         << " max: " << max;

    return 0;
}
```

## Passaggio dei Vettori

- Come possiamo passare dei Vettori alle funzioni?
- Usando i **Puntatori**  
Cioè copiando l'indirizzo del primo elemento del vettore
- Esempio: Programma **Funzioni\_05.cpp**:  
La funzione `Lettura` effettua l'input dei valori nel vettore  
La funzione `Stampa` stampa i valori nel vettore sullo schermo  
La funzione `main` definisce il vettore e usa le due precedenti funzioni

## Funzione Lettura

```
void Lettura(int *v, int size)
{
    int i;

    for(i=0; i < size; i++)
    {
        cout << "Inserire valore "
              << (i+1) << ": ";
        cin >> v[i];
    }
}
```

## Funzione Stampa

```
void Stampa(int v[], int size)
{
    int i;

    for(i=0; i < size; i++)
    {
        cout << "Valore "
              << (i+1) << ": "
              << v[i] << endl;
    }
}
```

## Funzione main

```
int main()  
{  
    const int SIZE=5;  
    int elenco[SIZE];  
  
    Lettura( elenco, SIZE);  
    Stampa( elenco, SIZE );  
  
    return 0;  
}
```

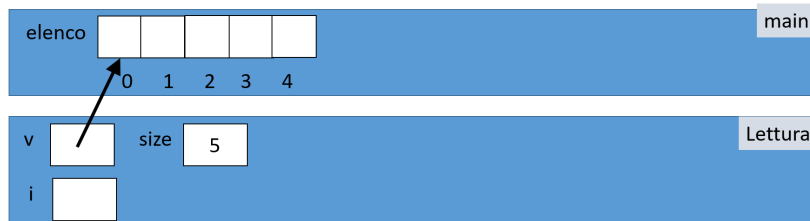


- Notate, nella funzione `Stampa`, come è definito il parametro formale `v`  
`int v[]`
- è equivalente ad  
`int *v`
- ma mette in evidenza che il puntatore da passare serve per **Gestire un Vettore**  
Aumenta la chiarezza, non aumenta il potere espressivo

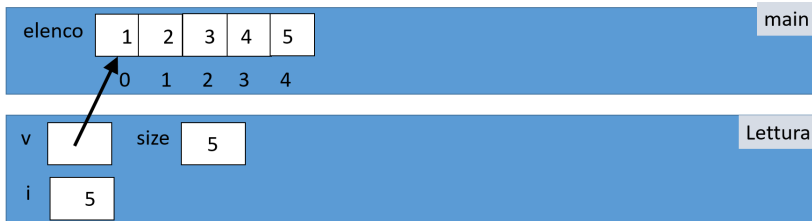
## Solo Funzione main



## Chiamata alla Funzione Lettura



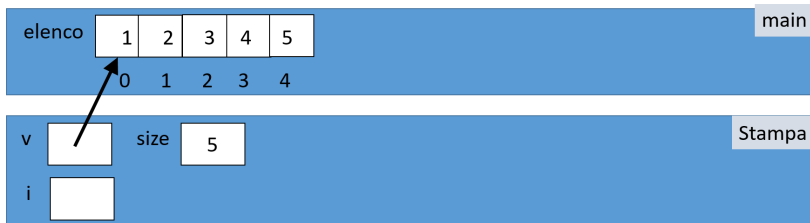
## Prima di Terminare la Funzione Lettura



## Terminata la Funzione Lettura



## Chiamata della Funzione Stampa



## Prototipo di Funzione

- Che succede se la funzione da chiamare è definita più avanti o altrove (nelle librerie)?
- Si definisce il suo **PROTOTIPO**  
il prototipo è **SOLO** l'intestazione, senza corpo.
- Vedi esempio: **Funzioni\_06.cpp**

```
void Lettura(int *v, int size);  
void Stampa(int v[], int size);
```

```
int main()  
{  
    const int SIZE=5;  
    int elenco[SIZE];  
  
    Lettura( elenco, SIZE);  
    Stampa( elenco, SIZE );  
  
    return 0;  
}
```



```
void Lettura(int *v, int size);  
void Stampa(int v[], int size);
```

```
int main()  
{
```

```
    const int SIZE=5;  
    int elenco[SIZE];
```

```
    Lettura( elenco, SIZE);  
    Stampa( elenco, SIZE );
```

```
    return 0;
```

```
}
```

Prototipi  
Delle Funzioni



## Ordinamento (tipo Bubble Sort)

- Ecco un semplice algoritmo di ordinamento
- L'ho etichettato **Tipo Bubble Sort**  
perché è un ordinamento a bolla  
ma normalmente, cercando il Bubble Sort, viene  
proposto un algoritmo diverso
- Programma: **Ordinamento.cpp**  
Legge la solita serie di 5 numeri interi, ma li stampa  
ordinati

## Tipo Bubble Sort

```
void Sort( int v[], int size)
{
    int i, j;
    int t;

    for( i=0; i < size-1; i++)
        for(j = i+1; j < size; j++)
        {
            if( v[i] > v[j])
            {
                t = v[i];
                v[i] = v[j];
                v[j] = t;
            }
        }
}
```

## Tecnica di Scambio

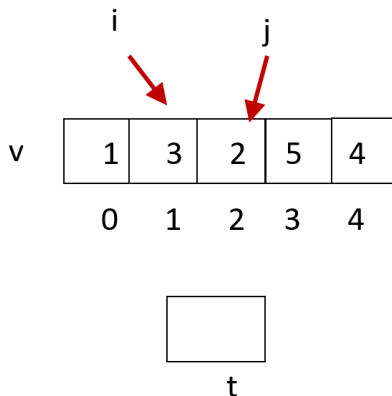
- Per scambiare i valori di due variabili, serve una **Terza Variabile**
- Nella funzione, si chiama *t* (temporanea)
- Ecco la sequenza di scambio:

```
t = v[i];
```

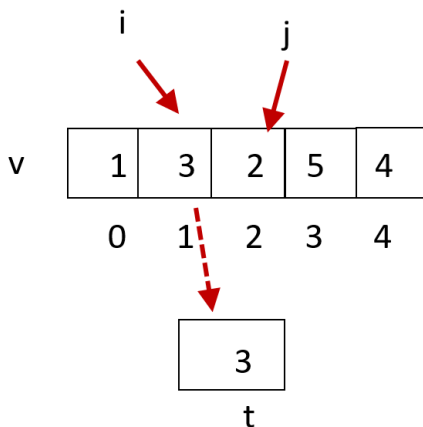
```
v[i] = v[j];
```

```
v[j] = t;
```

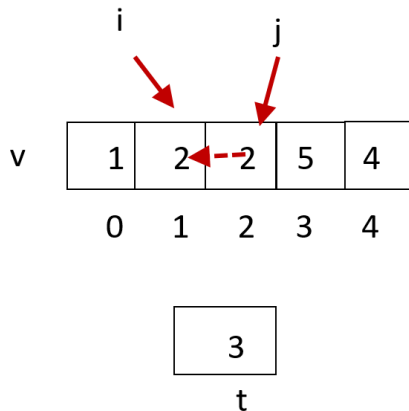
## Sequenza di Scambio



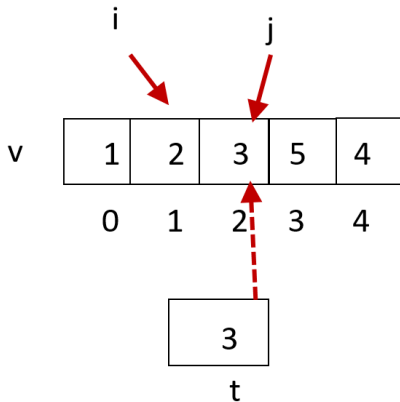
## Sequenza di Scambio



## Sequenza di Scambio



## Sequenza di Scambio





## Ordinamento Bubble Sort Ufficiale

- Ecco l'ordinamento Bubble Sort ufficiale
- Vedrete che è un po' più complicato
- Programma: **Ordinamento\_2.cpp**

## Bubble Sort

```
void BubbleSort( int v[], int size)
{
    int i, continua;
    int t;

    do
    {
        continua = 0;
        for(i = 0; i < size-1; i++)
        {
            if( v[i] > v[i+1])
            {
                t = v[i];
                v[i] = v[i+1];
                v[i+1] = t;
                continua = 1;
            }
        }
    } while( continua );
}
```

## Programma: No\_Duplicati2.cpp

- Partendo dal programma **No\_Duplicati.cpp**
- ristrutturare il programma in modo che la funzione `Verifica_Presenza` effettui la verifica del valore nel vettore: se restituisce 1, il valore è presente, se restituisce 0, il valore non è presente (definite voi i parametri, prendendo spunto da quanto visto fino ad ora).

## Programma: `Trovare_il_Problema.cpp`

- Il programma chiede all'utente quanti valori vuole trattare;
- quindi, richiede all'utente un valore per volta: se questo è maggiore o uguale a 0, stampa la sequenza decrescente (zero incluso).

Trovate l'errore, perché sembra corretto, ma non lo è