



# UNIVERSITÀ DEGLI STUDI DI PADOVA

## Funzioni ed efficienza

Stefano Ghidoni



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE



# Agenda

- Funzioni
- Passaggio di argomenti per copia e per riferimento
- Questioni di efficienza



# Funzioni – recap

- Suddividono la computazione in blocchi logici e funzionali
- Input: argomenti (0+)
- Output: valore di ritorno (0 o 1)
- Passare un argomento significa inizializzare un argomento formale con il valore dell'argomento

```
double fct(int a, double d);  
double fct(int a, double d) { return a*d; }
```

Parametri / argomenti  
formali (formal arguments)

// ...  
double d2 = fct(i, d1);

Argomenti



# Passaggio per valore

- Il passaggio di argomenti in C++ è **per valore**
- Un argomento è una variabile locale della funzione
  - Inizializzata a ogni chiamata
  - Valore copiato

è uguale in Java, quando si parla di tipi built-in, mentre è differente per gli UDT.

In C++ il comportamento è lo stesso tra tipi built-in e tipi UDT (es: se si passasse un'istanza di una classe BankAccount, non verrebbe modificata quella originale)



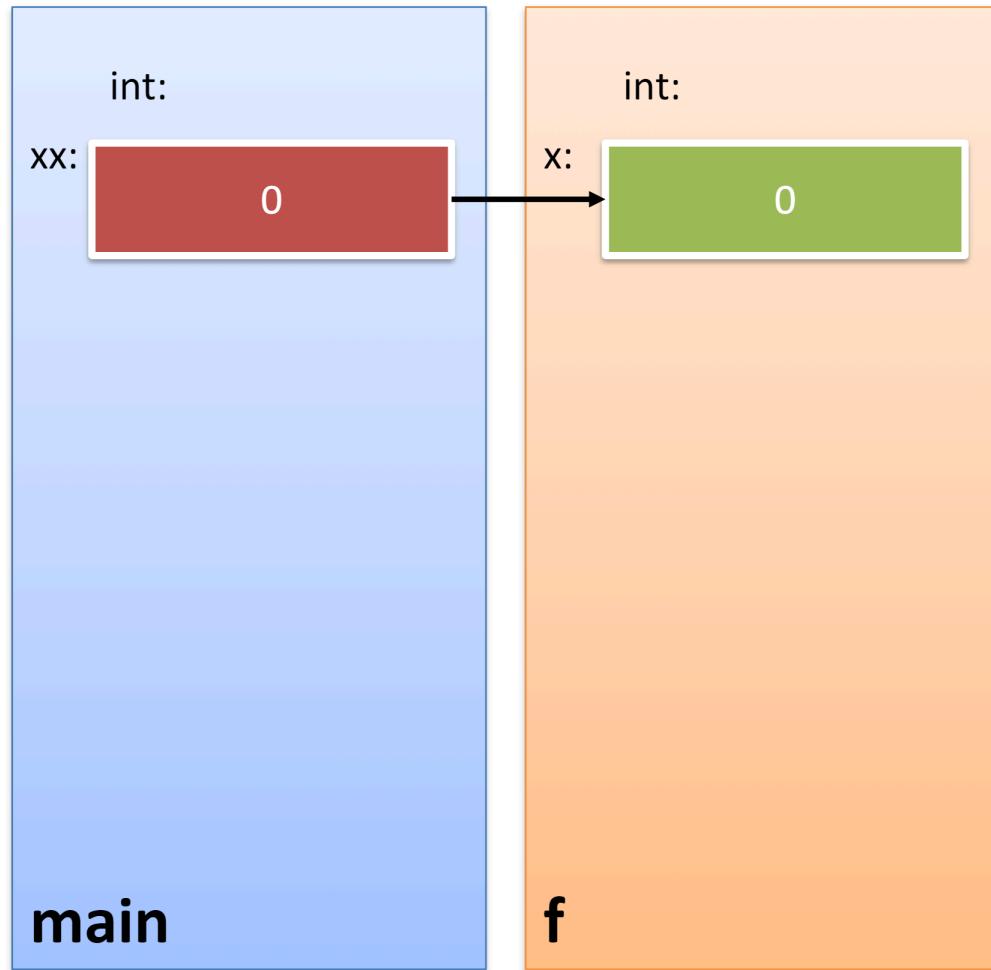
# Passaggio per valore

```
int f(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';
    cout << xx << '\n';

    int yy = 7;
    cout << f(yy) << '\n';
    cout << yy << '\n';

    return 0;
}
```





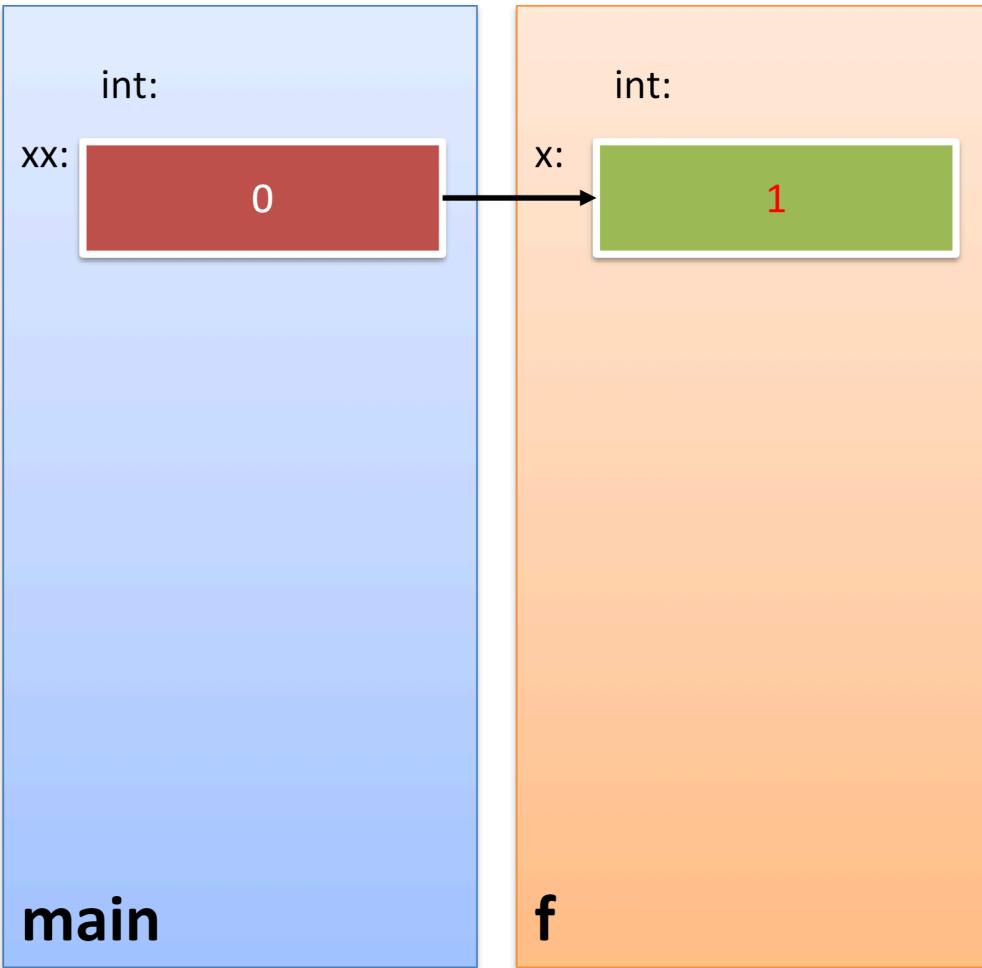
# Passaggio per valore

```
int f(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';
    cout << xx << '\n';

    int yy = 7;
    cout << f(yy) << '\n';
    cout << yy << '\n';

    return 0;
}
```





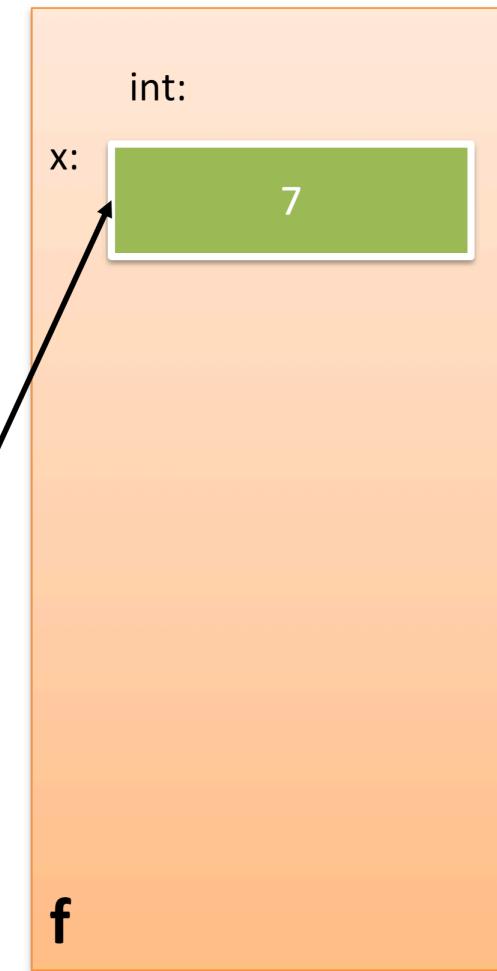
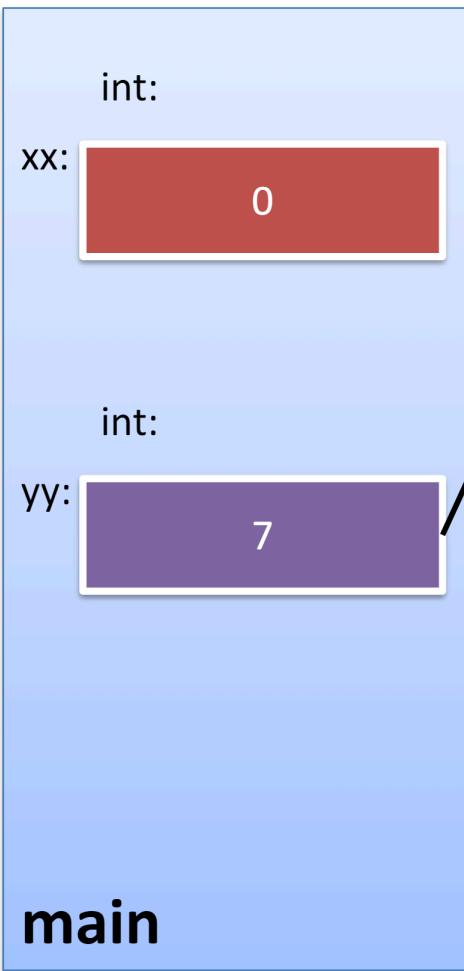
# Passaggio per valore

```
int f(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';
    cout << xx << '\n';

    int yy = 7;
    cout << f(yy) << '\n';
    cout << yy << '\n';

    return 0;
}
```





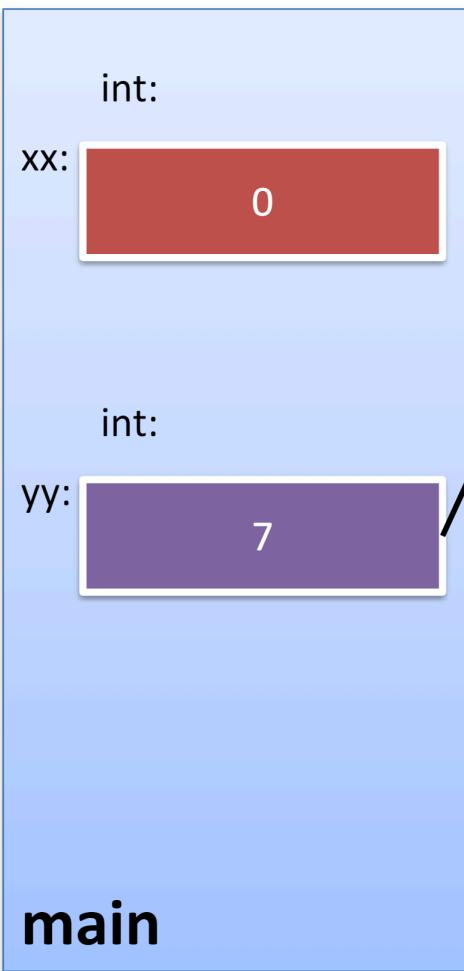
# Passaggio per valore

```
int f(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';
    cout << xx << '\n';

    int yy = 7;
    cout << f(yy) << '\n';
    cout << yy << '\n';

    return 0;
}
```





# Passaggio per valore

```
int f(int x)
{
    x = x + 1;
    return x;
}

int main()
{
    int xx = 0;
    cout << f(xx) << '\n';
    cout << xx << '\n';

    int yy = 7;
    cout << f(yy) << '\n';
    cout << yy << '\n';

    return 0;
}
```

Qual è l'output?



# Passaggio per valore

- Molto comodo, efficace ed efficiente
  - Pochi dati (es., tipi standard)
  - Dati che non devono essere modificati
- Potenziali problemi di **efficienza**
  - Molti dati (la copia è dispendiosa)
    - Vettori
    - Immagini, db, testi, ...

Il fatto che una funzione abbia un comportamento in base a un tipo o un altro crea incoerenze (come succede in Java).  
In C++ il comportamento è coerente.

# Problemi di efficienza



# Vector (excursus)

- Vector: è la versione moderna dell'array C
  - Gestisce vettori di ogni tipo di dato
    - `vector<int> vi;` `vector<double> vd;` `vector<T> vt;`
    - `<>` contiene un tipo!
  - Conosce quanti elementi ha in memoria
    - `v.size()`
  - Gestisce automaticamente la memoria
- Ulteriori info:  
<http://www.cplusplus.com/reference/vector/vector/>
- Gli oggetti vector possono occupare molta memoria



# Problemi di efficienza

```
void print(vector<double> v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}
```

Potenziale problema  
di efficienza

- È davvero un problema?



# Problemi di efficienza

- È davvero un problema?

```
void int f(int x)
{
    vector<double> vd1 (10);
    vector<double> vd2(1000000);
    vector<double> vd3(x)
    // ... riempimento di vd1, vd2, vd3 ...
    print(vd1);
    print(vd2);
    print(vd3);
}
```

- Non troppa ossessione per l'efficienza, ma attenzione ai dettagli



# Migliorare l'efficienza

- Evitare la copia
  - Accedere direttamente ai dati passati come argomento
- Quale strumento usare?



# Migliorare l'efficienza

- Evitare la copia
  - Accedere direttamente ai dati passati come argomento
- Quale strumento usare?
- Indirizzo del dato
  - Puntatore? Variabile che contiene un indirizzo
  - Riferimento (reference)

# Reference



# Passaggio per riferimento

```
void print(vector<double> v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}

void print(vector<double>& v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}
```



- Trovate le differenze!



# Passaggio per riferimento

```
void print(vector<double> v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}

void print(vector<double>&v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}
```

- Trovate le differenze!



# Passaggio per riferimento

```
void print(vector<double>& v)
{
    cout << "{";
    for (int i = 0; i < v.size(); ++i) {
        cout << v[i];
        if (i != v.size() - 1)
            cout << ", ";
    }
    cout << "}\n";
}
```

- Una piccolissima differenza di sintassi
  - Ora l'argomento di print *si riferisce* (è un riferimento!) all'oggetto definito nel chiamante
- Il main è identico al precedente
  - È un problema?

Dalla chiamata non sappiamo che diritti si prende la funzione chiamata quando si prende l'argomento.



# Passaggio per riferimento

- Riconsideriamo questo esempio supponendo di passare l'argomento a print per riferimento

```
void int f(int x)
{
    vector<double> vd1 (10);
    vector<double> vd2(1000000);
    // ... riempimento di vd1, vd2
    print(vd1);
    print(vd2);
}
```



# Passaggio per riferimento

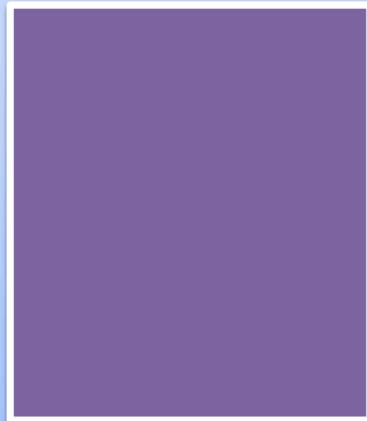
vector<int>:

vd1:



vector<int>:

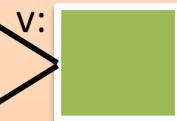
vd2:



**main**

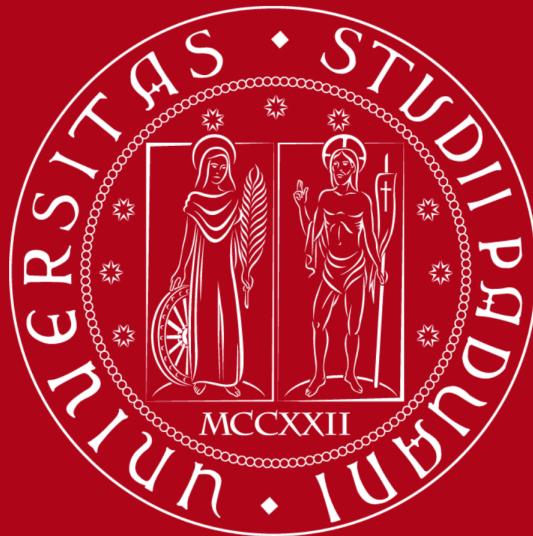
```
void int f(int x)
{
    vector<double> vd1 (10);
    vector<double> vd2(1000000);
    // ... riempimento di vd1, vd2
    print(vd1);
    print(vd2);
}
```

vector<int>&:



v:

**print**



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

**Funzioni ed efficienza**

Stefano Ghidoni