Funzioni - Parte I
Le funzioni

Corso AMAT C++

Liceo Scientifico Statale “A. Volta”

A.S. 2018/19

Il main è una funzione...

```
1 #include <iostream>
2 int main(){
3    std::cout << "A ";
4    std::cout << "B ";
5    return 0;
6 }
```

...ma non per forza l'unica!

```
1  #include <iostream>
2  int piero(){
3     std::cout << "Ciao ";
4     std::cout << "sono Piero ";
5     return 0;
6  }
7  int main(){
8     std::cout << "A ";
9     std::cout << "B ";
10    return 0;
11 }
```

...ma non per forza l'unica!

```
1  #include <iostream>
2  int piero(){
3     std::cout << "Ciao ";
4     std::cout << "sono Piero ";
5     return 0;
6  }
7  int main(){
8     std::cout << "A ";
9     std::cout << "B ";
10    return 0;
11 }
```

Se eseguiamo questo programma, il computer esegue esclusivamente la funzione `main`: a terminale viene scritto:

```
A B
```

La chiamata alla funzione

```
1  #include <iostream>
2  int piero(){
3     std::cout << "Ciao ";
4     std::cout << "son Piero ";
5     return 0;
6  }
7  int main(){
8     std::cout << "A ";
9     piero();
10    std::cout << "B ";
11    piero();
12    return 0;
13 }
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
```

```
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
```

```
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero B
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero B
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero B
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero B Ciao
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero B Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

piero:

```
std::cout<<"Ciao ";
std::cout<<"son Piero ";
return 0;
```

Terminale:

```
A Ciao son Piero B Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero B Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
```

```
retrurn 0;
```

Terminale:

```
A Ciao son Piero B Ciao son Piero
```

La chiamata alla funzione

main:

```
std::cout << "A ";
piero();
std::cout << "B ";
piero();
retrurn 0;
```

Terminale:

```
A Ciao son Piero B Ciao son Piero
```

Funzioni e loro esecuzione

Nel programma che abbiamo visto ci sono:

- due **funzioni**: `piero` e `main`
- tre **esecuzioni** di funzioni: l'esecuzione del `main`, la prima esecuzione di `piero`, la seconda esecuzione di `piero`.

Ciascuna esecuzione di `piero` ha inizio con una **chiamata** o **invocazione** da parte del `main`, che ha qui il ruolo di **chiamante**. L'esecuzione del chiamante rimane **sospesa** fintanto che quella della funzione chiamata non si sia conclusa.

Dichiarazione della funzione e sua chiamata

La dichiarazione della funzione deve sempre precedere, nel codice, le sue chiamate. Così, se avessimo scritto così il codice:

```
1  #include <iostream>
2  int main(){
3     std::cout << "A ";
4     piero();
5     std::cout << "B ";
6     piero();
7     return 0;
8  }
9  int piero(){
10    std::cout << "Ciao ";
11    std::cout << "son Piero ";
12    return 0;
13 }
```

NON avrebbe funzionato!

L'isolamento

Ogni esecuzione di funzione avviene all'interno di uno spazio di memoria **completamente isolato**, che viene creato al momento della chiamata della funzione e viene distrutto al termine della sua esecuzione.

```
1  #include <iostream>
2  int piero(){
3     int b, c;
4     b = 3; c = 4;
5     std::cout << b+c;
6     return 0;
7  }
8  int main(){
9     int a, b;
10    a = 1; b = 2;
11    piero();
12    std::cout << a+b;
13    return 0;
14 }
```

L'isolamento

main:

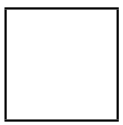
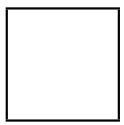
```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

Terminale:

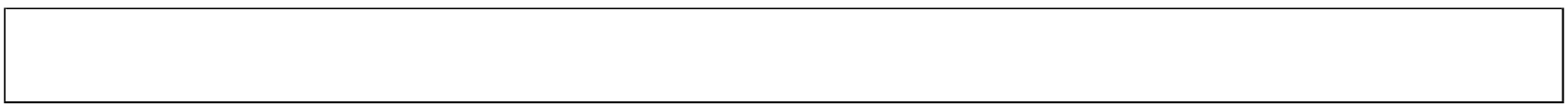
L'isolamento

main:

```
int a; int b;
---
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

a:  b: 

Terminale:



L'isolamento

main:

```
int a; int b;
a=1; b=2;
---
piero();
std::cout << a+b;
retrurn 0;
```

a: 1 b: 2

Terminale:

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

piero:

```
int b, c;
b=3; c=4;
std::cout << b+c;
return 0;
```

a: 1 b: 2

Terminale:

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

piero:

```
int b, c;
b=3; c=4;
std::cout << b+c;
return 0;
```

a: 1 b: 2

b: c:

Terminale:

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

piero:

```
int b, c;
b=3; c=4;
std::cout << b+c;
return 0;
```

a: 1 b: 2

b: 3 c: 4

Terminale:

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

piero:

```
int b, c;
b=3; c=4;
std::cout << b+c;
return 0;
```

a: 1 b: 2

b: 3 c: 4

Terminale:

7

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

piero:

```
int b, c;
b=3; c=4;
std::cout << b+c;
return 0;
```

a: 1 b: 2

b: 3 c: 4

Terminale:

7

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

a: 1 b: 2

Terminale:

```
7
```

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
```

```
std::cout << a+b;
retrurn 0;
```

a: 1 b: 2

Terminale:

```
7
```

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
```

```
retrurn 0;
```

a: 1 b: 2

Terminale:

```
7 3
```

L'isolamento

main:

```
int a; int b;
a=1; b=2;
piero();
std::cout << a+b;
retrurn 0;
```

a: 1 b: 2

Terminale:

```
7 3
```

La comunicazione

Se ogni esecuzione ha luogo entro uno spazio di memoria isolato, come possono il chiamante e la funzione invocata comunicare? Vorremmo ottenere che:

- **INPUT**: Al momento della chiamata, il *chiamante* possa fornire alla funzione dei dati su cui lavorare (quelli che si chiamano **parametri**)
- **OUTPUT** La funzione chiamata, concludendo la propria esecuzione, possa **restituire** un risultato al chiamante.

L'input

Perché una funzione possa ricevere un INPUT, occorre che:

L'input

Perché una funzione possa ricevere un INPUT, occorre che:

sia dichiarata in modo da esigere dei parametri in ingresso:

```
int piero(int pera, int mela){
   comando1;
   comando2;
   comando3;
   comando4;
   ...
}
```

L'input

Perché una funzione possa ricevere un INPUT, occorre che:

sia dichiarata in modo da esigere dei parametri in ingresso:

```
int piero(int pera, int mela){
   comando1;
   comando2;
   comando3;
   comando4;
   ...
}
```

il chiamante la invochi passando un valore per ogni parametro

```
int main(){
   ...
   piero(2,3);
   ...
   piero(3,4);
   ...
}
```

L'input

Perché una funzione possa ricevere un INPUT, occorre che:

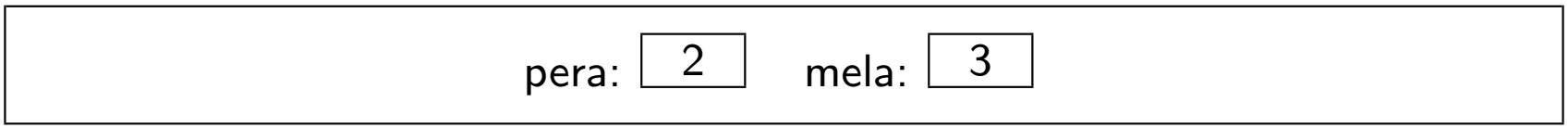
sia dichiarata in modo da esigere dei parametri in ingresso:

```
int piero(int pera, int mela){
   comando1;
   comando2;
   comando3;
   comando4;
   ...
}
```

il chiamante la invochi passando un valore per ogni parametro

```
int main(){
   ...
   piero(2,3);
   ...
   piero(3,4);
   ...
}
```

L'invocazione `piero(2,3)` avvia l'esecuzione della funzione `piero` all'interno un compartimento isolato della memoria, che viene creato già popolato con due variabili: una di nome `pera`, avente valore 2, e una di nome `mela`, avente valore 3.



L'output

Perché una funzione possa restituire un OUTPUT, occorre che:

L'output

Perché una funzione possa restituire un OUTPUT, occorre che:

termini mediante un comando della forma `return espressione;` **con** `espressione` **corrispondente al tipo dichiarato**

```
int piero(){
   ...
   return 5;
   ...
}
double pino(){
   ...
   return 5.5;
   ...
}
```

L'output

Perché una funzione possa restituire un OUTPUT, occorre che:

termini mediante un comando della forma `return espressione;` **con** `espressione` **corrispondente al tipo dichiarato**

```
int piero(){
   ...
   return 5;
   ...
}
double pino(){
   ...
   return 5.5;
   ...
}
```

l'invocazione della funzione sia parte di un'espressione, così che il chiamante possa catturare il valore restituito:

```
int main(){
   ...
   double x;
   x = pino();
   std::cout<< 2*piero()+x;
   ...
}
```

Ad `x` verrà assegnato il valore 5.5; a terminale verrà scritto 15.5.

Esempio 1

Si scriva una funzione `primo` che determini se un numero intero $n \geq 2$ fornito come argomento e' o meno primo; si scriva un `main` che consenta all'utente di interfacciarsi con tale funzione.

Esempio 1

Si scriva una funzione `primo` che determini se un numero intero $n \geq 2$ fornito come argomento e' o meno primo; si scriva un `main` che consenta all'utente di interfacciarsi con tale funzione.

```
1  bool primo(int n){
2     int candidato;
3     candidato = 2;
4     while(candidato<n){
5        if(n%candidato==0){
6           return 0;
7        }
8        candidato=candidato+1;
9     }
10    return 1;
11 }
```

Esempio 1

Si scriva una funzione `primo` che determini se un numero intero $n \geq 2$ fornito come argomento e' o meno primo; si scriva un `main` che consenta all'utente di interfacciarsi con tale funzione.

```
1  bool primo(int n){
2     int candidato;
3     candidato = 2;
4     while(candidato<n){
5        if(n%candidato==0){
6           return 0;
7        }
8        candidato=candidato+1;
9     }
10    return 1;
11 }
```

```
1  int main(){
2     int num;
3     std::cout << "Numero: ";
4     std::cin >> num;
5     if(primo(num)){
6        std::cout << "Primo";
7     }else{
8        std::cout << "Comp.";
9     }
10    return 0;
11 }
```

Esempio 2

Si scriva una funzione `f` che, dato un numero reale x fornito come argomento, restituisca $f(x)$, ove:

$$f(x) = \begin{cases} \sqrt[5]{x} - 1 & \text{se } x \geq 1 \\ x & \text{altrimenti} \end{cases}$$

Si scriva un `main` che mostri a terminale i valori assunti da f per $x \in [0, 5]$ (passo di campionamento 0.05)

Esempio 2

Si scriva una funzione `f` che, dato un numero reale x fornito come argomento, restituisca $f(x)$, ove:

$$f(x) = \begin{cases} \sqrt[5]{x} - 1 & \text{se } x \geq 1 \\ x & \text{altrimenti} \end{cases}$$

Si scriva un `main` che mostri a terminale i valori assunti da f per $x \in [0, 5]$ (passo di campionamento 0.05)

```
1 double f(double x){
2    if(x>=1){
3       return pow(x,0.2)-1;
4    }else{
5       return x;
6    }
7 }
```

Esempio 2

Si scriva una funzione `f` che, dato un numero reale x fornito come argomento, restituisca $f(x)$, ove:

$$f(x) = \begin{cases} \sqrt[5]{x} - 1 & \text{se } x \geq 1 \\ x & \text{altrimenti} \end{cases}$$

Si scriva un `main` che mostri a terminale i valori assunti da f per $x \in [0, 5]$ (passo di campionamento 0.05)

```
1 double f(double x){
2    if(x>=1){
3       return pow(x,0.2)-1;
4    }else{
5       return x;
6    }
7 }
```

```
1 int main(){
2    double x; x = 0;
3    while(x<=5){
4       std::cout<< x << "\t";
5       std::cout<< f(x) << "\n";
6       x = x+0.05;
7    }
8    return 0;
9 }
```