Titolo unità didattica: Function e procedure [05]

Titolo modulo : Function [01-T]

Organizzazione di algoritmi come function

Argomenti trattati:

- ✓ organizzazione modulare
- ✓ function
- ✓ parametri e argomenti
- ✓ restituzione del risultato

Prerequisiti richiesti: AP-02-*-T

problema:
algoritmo per il calcolo della circonferenza c di un cerchio, di cui è noto il raggio r

dato di input: il numero r (variabile **raggio**)

dato di output: il numero c
(variabile **circonferenza**)

```
float raggio, circonferenza;
const float pi_greco = 3.1415926;
  read (raggio) ;
  circonferenza = 2.0*pi_greco*raggio ;
  printf (circonferenza) ;
```

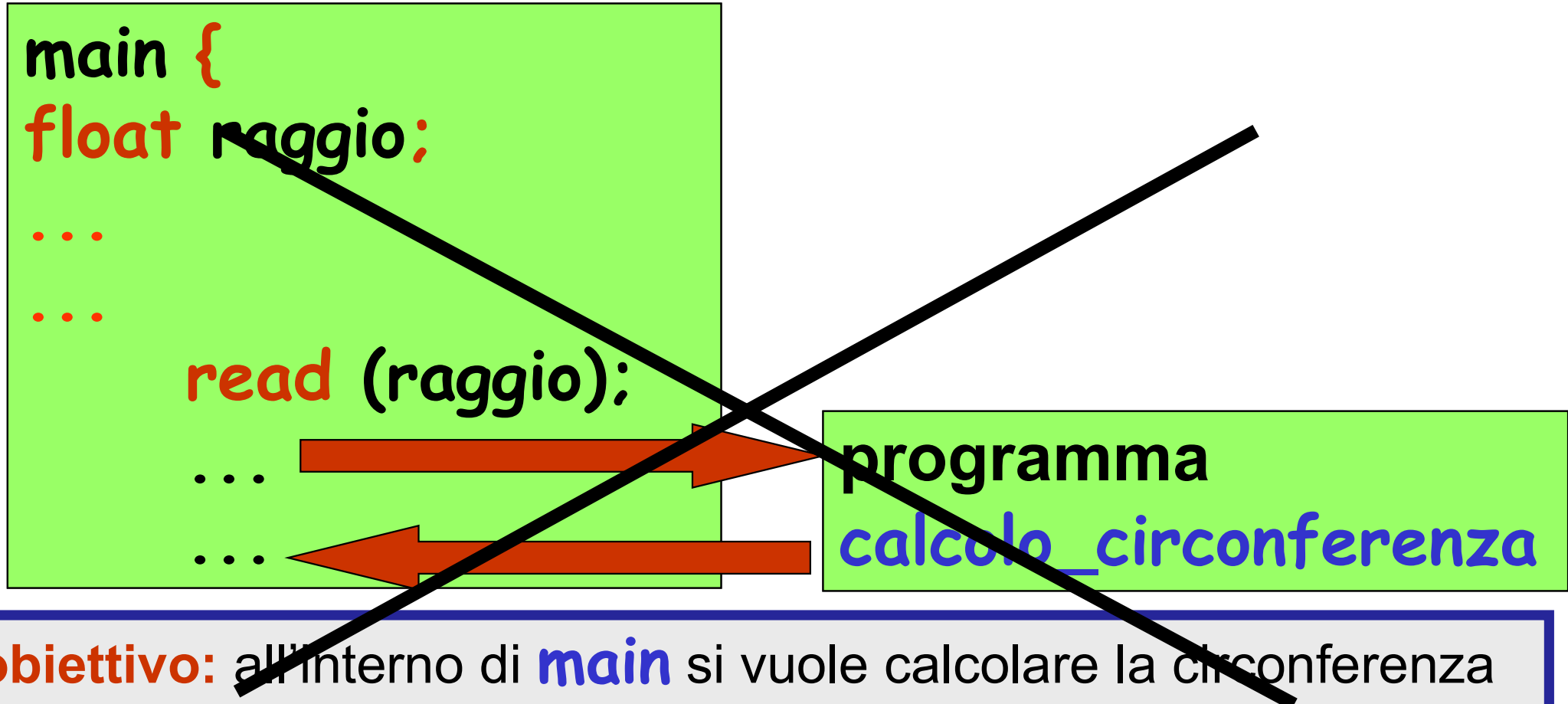
a un algoritmo può essere associato un **nome**

```
programma  calcolo_circonferenza
{
float raggio, circonferenza;
const float pi_greco = 3.1415926;
  read (raggio) ;
  circonferenza = 2.0*pi_greco*raggio ;
  printf (circonferenza) ;
}
```

l'algoritmo è formalmente organizzato come una entità individuata da un **nome**

limitazione!

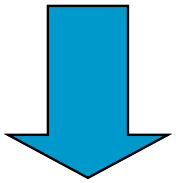
il programma calcolo_circonferenza non può essere utilizzato da altri programmi



obiettivo: all'interno di **main** si vuole calcolare la circonferenza del cerchio di raggio **raggio**, eseguendo il programma il cui nome è **calcolo_circonferenza**

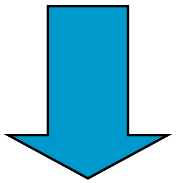
limitazione!

il programma calcolo_circonferenza non può essere utilizzato da altri programmi

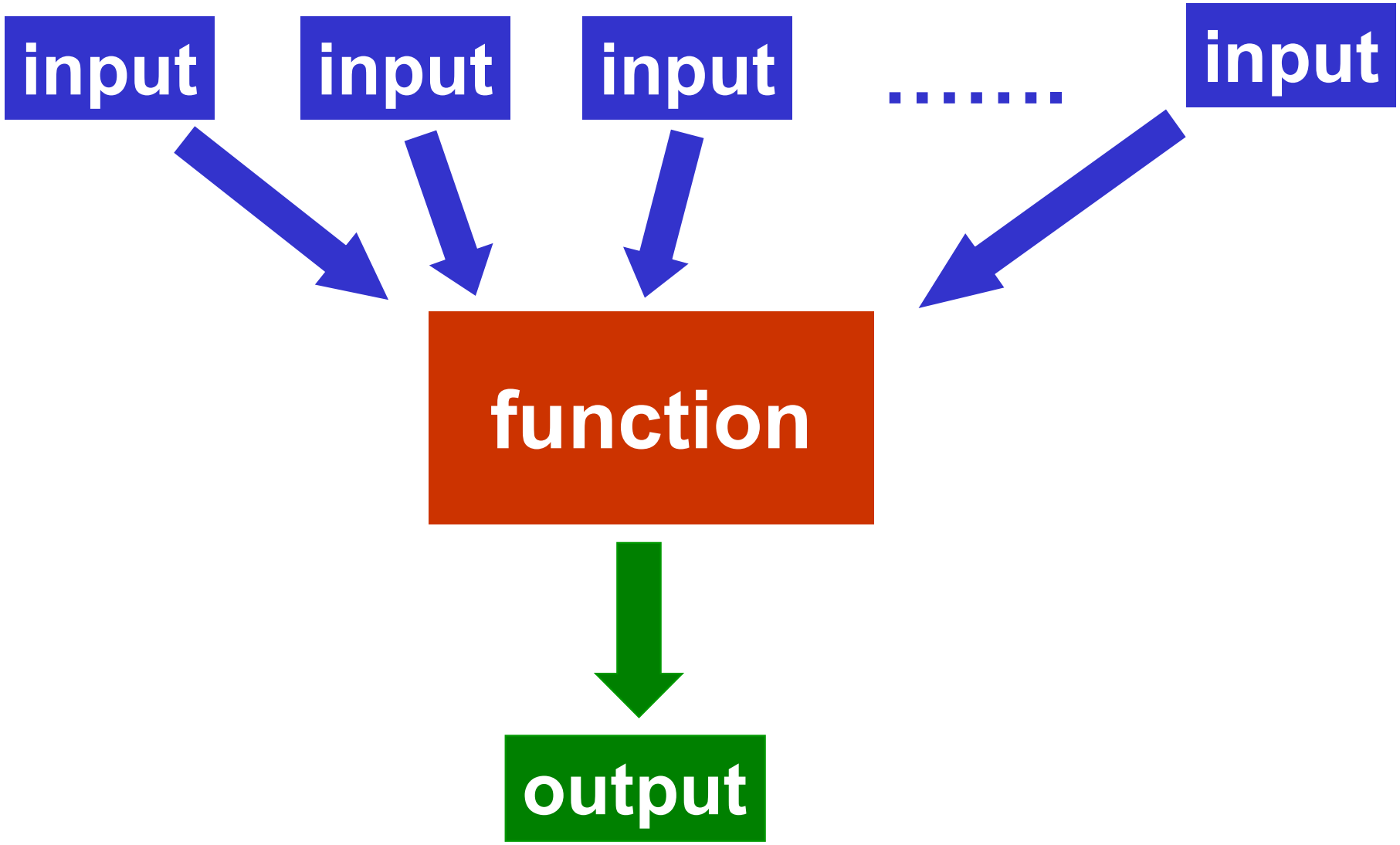


function e **procedura**

l'algoritmo deve essere **organizzato** in modo da poter essere utilizzato da altri algoritmi



l'algoritmo deve poter "**scambiare**" dati con gli altri algoritmi che lo utilizzano


input
input
input
.....
input
function
output

la **intestazione di una function** deve specificare

- ✓ il **nome** della function
- ✓ le variabili per i dati di input (**parametri**)
- ✓ il tipo del valore da restituire (il dato di output)

```
<tipo> <nome>(<tipo><variabile>,…)
```

```
float circon (float raggio) {
const float pi_greco = 3.1415926;
   ...
```

la **function** deve specificare

✓ il **valore che restituisce** (**dato di output**)

```
return <espressione>;
```

```
float circon(float raggio) {
 float risultato;
 const float pi_greco = 3.1415926;
   risultato = 2.0*pi_greco*raggio;
   return risultato;
}
```

utilizzo di una **function**:

```
main {
 float raggio,circonferenza;
   raggio = 1.1;
   circonferenza = circon(raggio);
   printf (circonferenza);
}
```

chiamata (o **attivazione**) della function **circon**

raggio è l'**argomento** della **chiamata**

utilizzo di una **function**:

```
main {
   float mio_raggio, circonferenza;
   mio_raggio = 1.1 ;
   circonferenza = circon(mio_raggio) ;
   printf (circonferenza) ;
}
```

mio_raggio è l'**argomento** della **chiamata**

utilizzo di una **function**:

```
main {
   float mio_raggio, circonferenza;
   circonferenza = circon(1.1) ;
   printf (circonferenza) ;
}
```

1.1 è l'**argomento** della **chiamata**

utilizzo di una function:

```
main {
printf (circon(1.1));
}
```

```
main {
float r ;
read (r);
 printf (circon(r));
}
```

~~
```
main {
 const float pi_greco = 3.1415926;
 circon(1.1) = 2.0*pi_greco*1.1 ;
  printf (circon(1.1)) ;
}
```

le espressioni

```
circonferenza = circon(mio_raggio)
circonferenza = circon(raggio)
```

contengono un **riferimento** alla

```
function circon
```

la valutazione di questa espressione richiede l'esecuzione (**attivazione**) della **function circon**

l'esecuzione dell'istruzione di **chiamata** provoca l'esecuzione della function (**attivazione** della function)

lo scambio di informazioni tra il **programma chiamante** e la **function chiamata** avviene attraverso la **corrispondenza** tra **argomento** di chiamata e **parametro** della function e attraverso il **risultato** restituito nell'espressione di chiamata

argomenti

variabili che appaiono nella **chiamata** di una function

parametri

variabili che appaiono nell'**intestazione** di una function

il **valore dell'argomento** viene **assegnato** al **parametro** corrispondente, al momento dell'attivazione della function

argomento e **parametro non sono la stessa variabile**

<tipo> <nome> (<tipo> <variabile> , . .)

la lista delle variabili che appaiono nella **intestazione** è la lista dei **parametri** della function e indica le variabili che identificano i **dati di input** dell'algoritmo

c'è un **unico dato di output** dell'algoritmo, che è associato al nome della function e il cui tipo è precisato in **<tipo>**

tipo dell'output

```
float circon (float raggio)
```

parametro di input

- la **function circon** ha come unico parametro la variabile **raggio**
- non ci sono istruzioni di **I/O** nella function
- la variabile **raggio** è lasciata indefinita (senza valore associato) nel corpo della function
- la function **deve** contenere una istruzione che restituisca il valore della function (dato di output)
  (**return 2.0*pi_greco*raggio**)

Esempio:

algoritmo per il calcolo dell'area A di un cerchio di raggio r, utilizzando il valore della circonferenza c

($c=2\pi r$, $A=\pi r^2$ e quindi $A=c^2/(4\pi)$)

```
main {
 float area, raggio;
 const float pi_greco = 3.1415926;
  read (raggio) ;
  area = circon(raggio)^2/(4.0*pi_greco) ;
  printf (area) ;
}
```

$$A = \frac{c^2}{4\pi}$$

```
float area_cerchio (float raggio) {
  const float pi_greco = 3.1415926;
  return circon(raggio)^2/(4.0*pi_greco) ;
}
```

versione 1

$$A = \pi r^2$$

```
float area_cerchio (float raggio) {
 const float pi_greco = 3.1415926;
 return pi_greco*raggio^2 ;
}
```

versione 2

programma che **chiama** (o **attiva**)
la function **area_cerchio**

```
main {
float area, raggio;
read (raggio) ;
   area = area_cerchio(raggio) ;
   printf (area) ;
}
```

per il programma **usa_area_cerchio**

- **non** è necessario conoscere quale algoritmo è implementato nella function **area_cerchio**,
- è necessario sapere solo **come** richiamare (le *specifiche*) la function **area_cerchio**

nascondere i dettagli

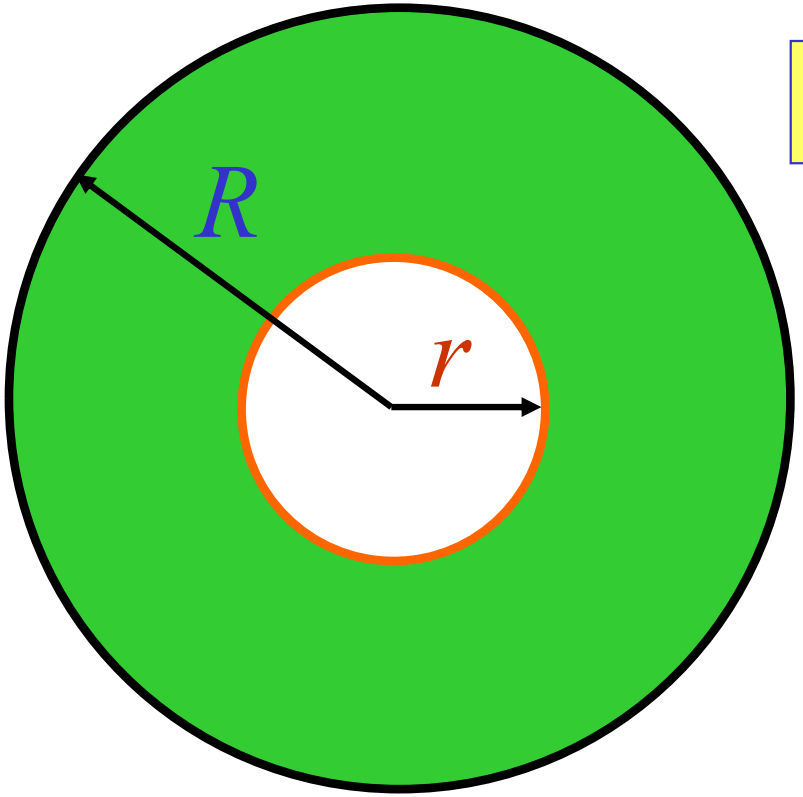
=

astrazione

Esempio:
algoritmo per il calcolo dell'area di una corona circolare

dati di input: il raggio maggiore R (variabile **r_mag**), il raggio minore r (variabile **r_min**)

dato di output: l'area A (variabile **area_corona**)



Area cerchio esterno

$$A = A_R - A_r$$

Area cerchio interno

Esempio:
algoritmo per il calcolo dell'area di una corona circolare

dati di input: il raggio maggiore R (variabile **r_mag**),
il raggio minore r (variabile **r_min**)

dato di output: l'area A (variabile **area_corona**)

```
main {
 float r_mag, r_min, area_corona;
  read (r_mag, r_min) ;
  area_corona = area_cerchio(r_mag)-
                area_cerchio(r_min) ;
  printf (area_corona) ;
}
```

prima chiamata

area_cerchio(r_mag)

attivazione della
function area_cerchio

- al **parametro raggio** è associato il valore dell'**argomento r_mag**
- al termine dell'esecuzione della function si ottiene il valore del primo dei due addendi

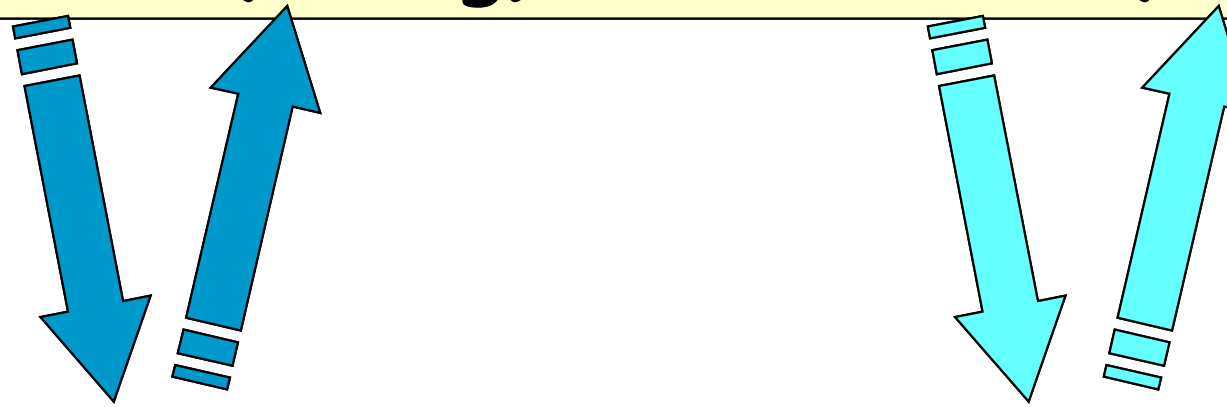
seconda chiamata

area_cerchio(r_min)

attivazione della
function area_cerchio

- al **parametro raggio** è associato il valore dell'**argomento r_min**
- al termine dell'esecuzione è associato un valore anche al secondo addendo
- l'espressione viene valutata e il suo valore viene associato alla variabile **area_corona**

```
area_corona =
area_cerchio(r_mag)-area_cerchio(r_min)
```



```
float area_cerchio(float raggio)
 <corpo della function>
```

scambio delle informazioni (dati) tra
chiamante e chiamato
=
meccanismo di sostituzione,
(passaggio dei parametri)

Esercizio: scrivere una function per il calcolo dell'area di una corona circolare

trasformare in function

```
main {
 float r_mag, r_min, area_corona;
  read (r_mag, r_min) ;
  area_corona = area_cerchio(r_mag)-
                          area_cerchio(r_min) ;
  printf (area_corona) ; }
```

```
float area_cor_circ (float r_mag,float r_min)
{ float area_corona;
  area_corona = area_cerchio(r_mag)-
                        area_cerchio(r_min) ;
  return area_corona;
}
```

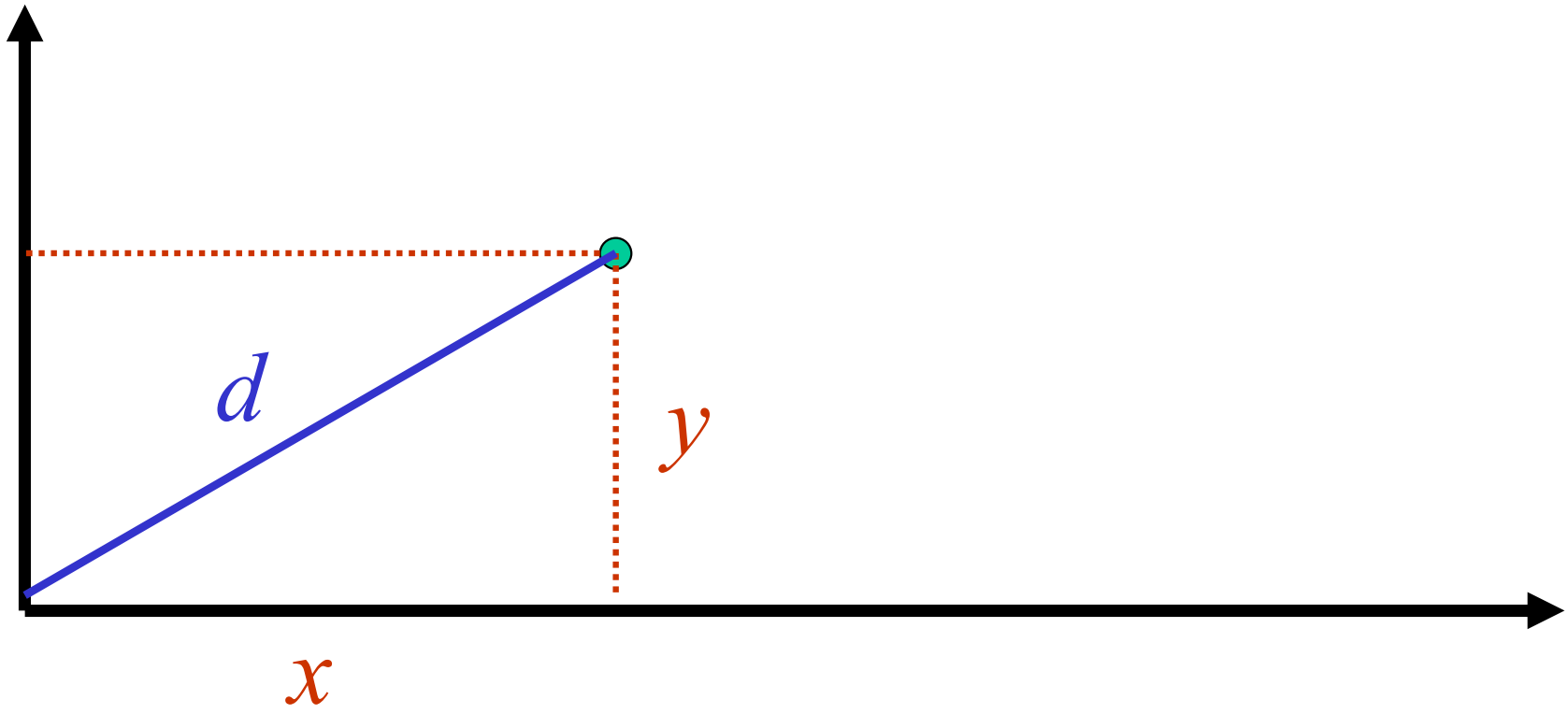
Esercizio: scrivere una function per il calcolo dell'area di una corona circolare

```
main {
   float r_1, r_2;
   read (r_1, r_2) ;
   printf (area_cor_circ(r_1,r_2) ) ;
}
```

```
float area_cor_circ (float r_mag,float r_min)
{ float area_corona;
   area_corona = area_cerchio(r_mag)-
                 area_cerchio(r_min) ;
   return area_corona;
}
```

Esempio:

sviluppare una function per il calcolo della distanza $d = \sqrt{x^2 + y^2}$ di un punto di coordinate (x, y) dall'origine



Esempio:

sviluppare una function per il calcolo della distanza $d = \sqrt{x^2 + y^2}$ di un punto di coordinate (x, y) dall'origine

dati di input: ascissa x (variabile **x**), ordinata y (variabile **y**)

dato di output: la distanza d (da restituire)

Esempio:

sviluppare una function per il calcolo della distanza $d = \sqrt{x^2 + y^2}$ di un punto di coordinate (x, y) dall'origine

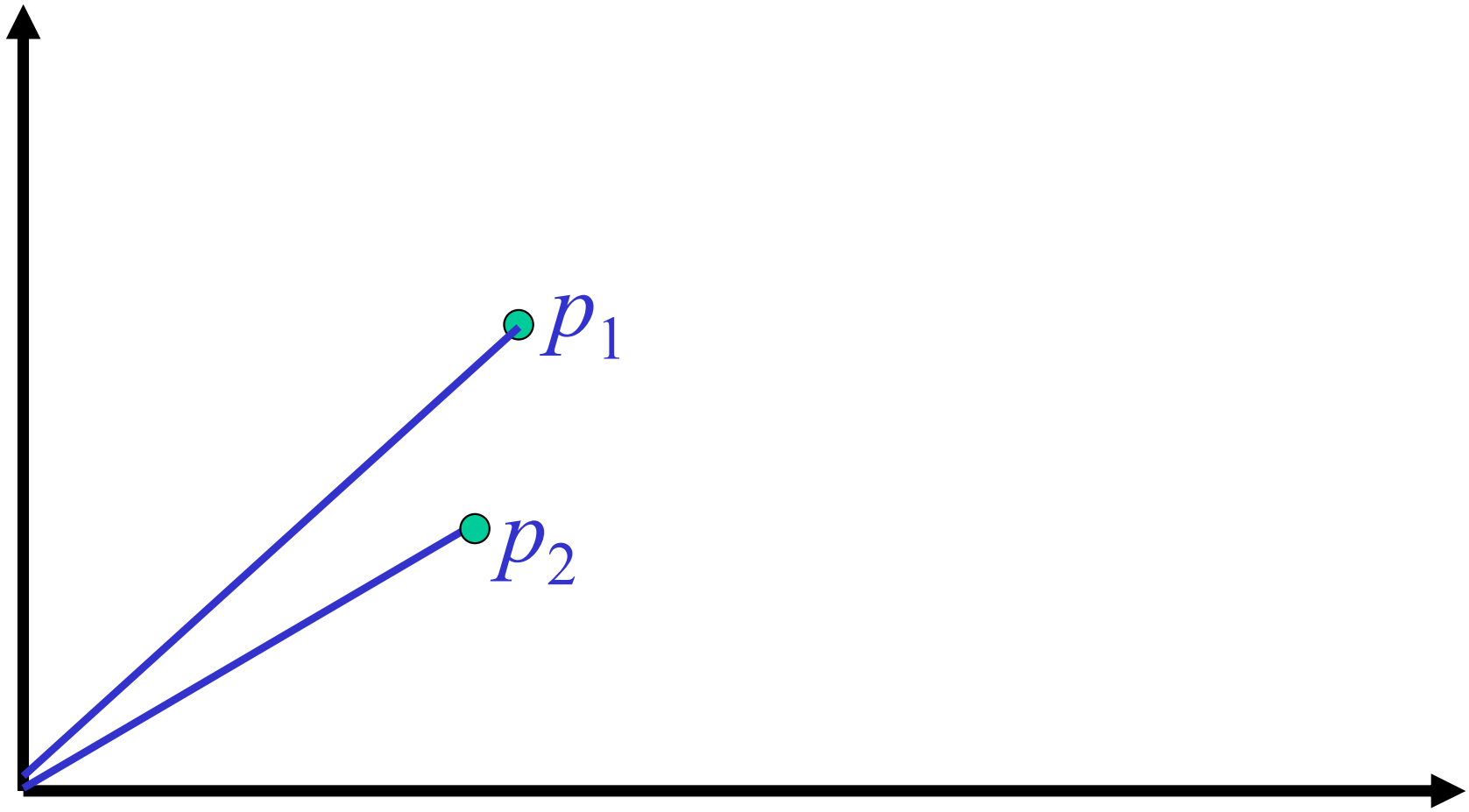
```
float distanza_o(float x,float y) {
   return sqrt(x^2+y^2);
}
```

```
float distanza_o(float x, float y){
 return sqrt(x*x + y*y);
}
```

```
float distanza_o(float x, float y){
 return sqrt(pow(x,2) + pow(y,2));
}
```

Esempio:
calcolare la distanza dall'origine dei due punti

p_1 = (2.1, 4.2) e p_2 = (1.8, 3.2)



Esempio:
calcolare la distanza dall'origine dei due punti

p_1 = (2.1, 4.2) e p_2 = (1.8, 3.2)

```
main {
 float d_primo_punto,d_secondo_punto,...
       primo_x,primo_y,secondo_x,secondo_y;
 read (primo_x,primo_y,secondo_x,secondo_y) ;

  d_primo_punto = distanza_o(primo_x,primo_y) ;
  d_secondo_punto = distanza_o(secondo_x,secondo_y) ;

 printf (d_primo_punto,d_secondo_punto) ;
}
```

la corrispondenza tra

parametri

e

argomenti

è stabilita dalla posizione dei parametri nella lista dei parametri (nell'intestazione) e degli argomenti nella lista degli argomenti (nella chiamata)

argomento e parametro non sono la stessa variabile

Esercizio:
calcolare la distanza tra i due punti
p_1 = (2.1, 4.2) e p_2 = (1.8, 3.2) usando la function
distanza_o