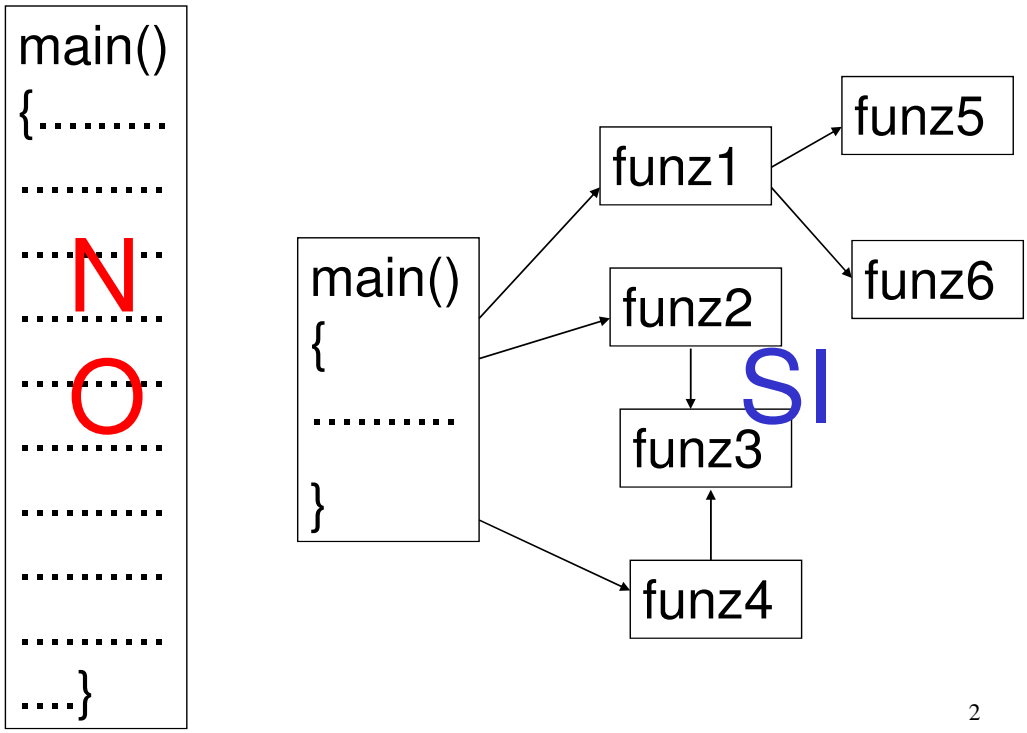
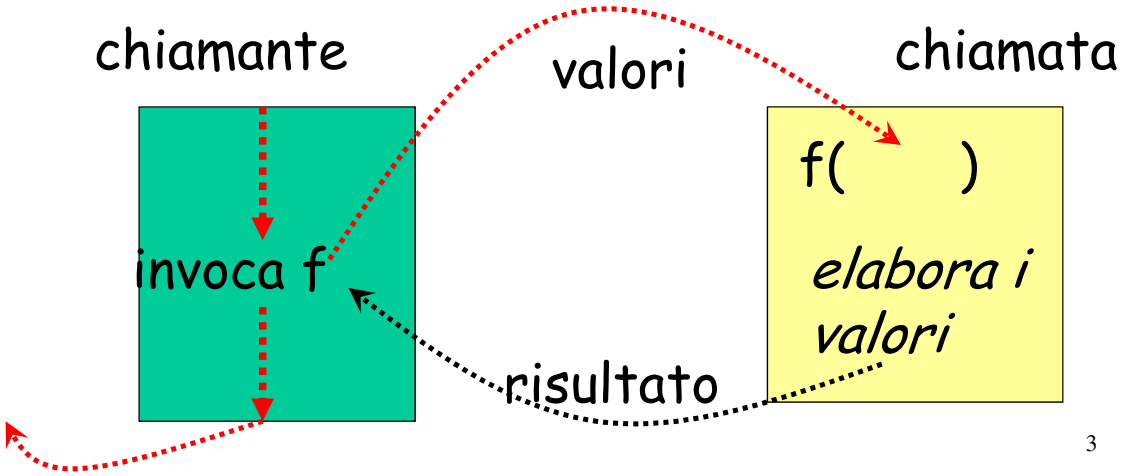
FUNZIONI

necessità di strutturare i programmi



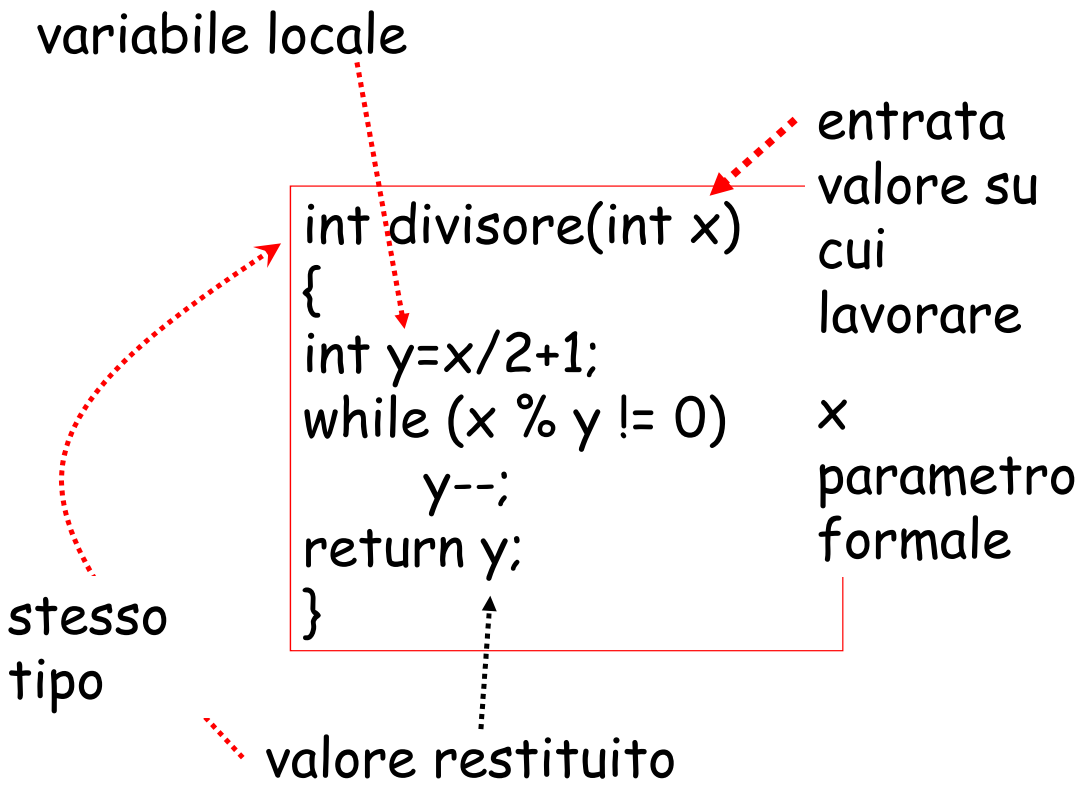
Funzioni

Una funzione è un pezzo di programma con un nome. Essa viene eseguita tramite l'invocazione del suo nome.



il più grande divisore di un valore dato:

```
int divisore(int x)
{
int y=x/2;
while (x % y != 0)
        y--;
return y;
}
```

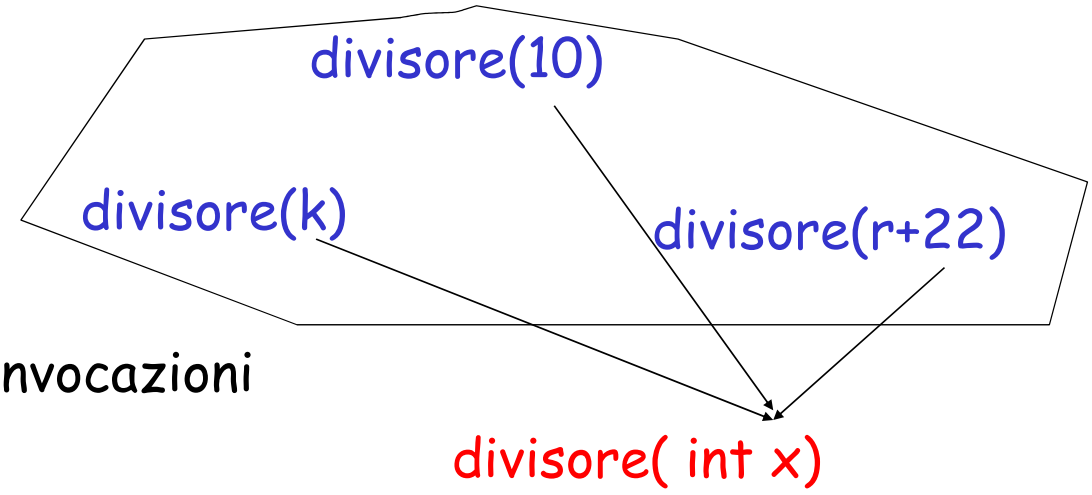


trova il massimo primo più piccolo o uguale di z dato

```
int primo(int z)
{
int k=z;
while(k>1 && ! (divisore(k) == 1))
      k--;
return k;
}
```

invoca la funzione divisore

passaggio dei parametri: attuali ➔ formali

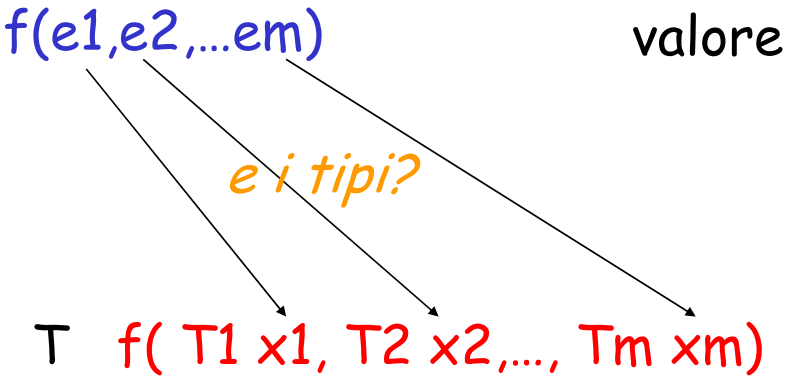


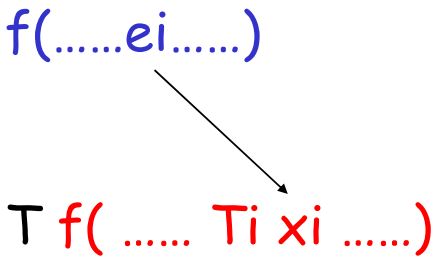
il parametro attuale è un valore che diventa l'R-valore di x

passaggio dei parametri

PER VALORE

il valore di ei diventa l'R-valore di xi

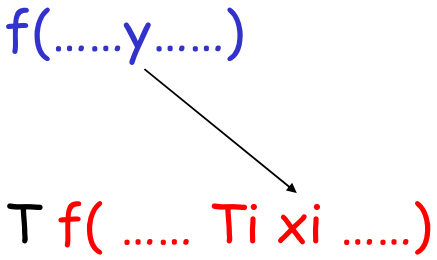




- se il tipo del valore di ei è Ti, facile
- se è diverso ?

conversione obbligata come per le assegnazioni (possibile warning)

IMPORTANTE



l'R-valore di xi == R-valore di y

ma L-valore di xi != L-valore di y

quando una funzione non restituisce alcun risultato, dobbiamo dichiarare che il suo tipo di ritorno è

void

non esistono valori di tipo void

void f(……);

le funzioni viste finora hanno un limite

non è possibile realizzare una funzione void f(int x) tale che :

int A = 10;

f(A);

e qui A ha R-valore 20 (o qualsiasi valore diverso da 10)

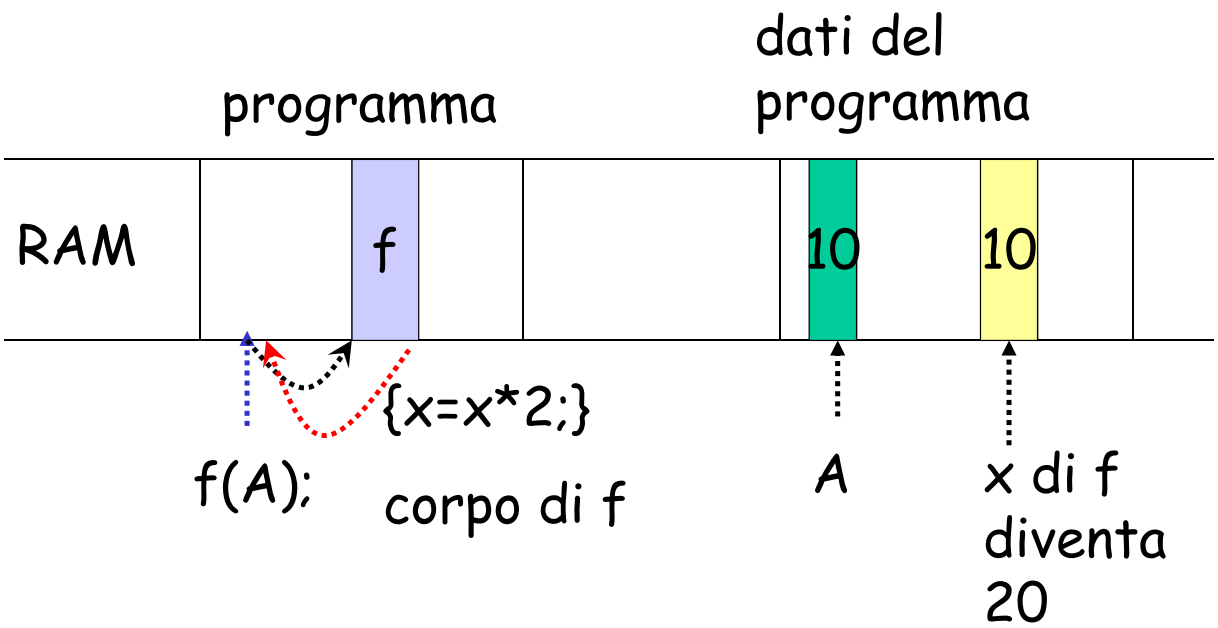
non possiamo realizzare side-effect

A=10

```
void f (int x)
{ x=x*2;}
```

x e A hanno diversi L-valori e quindi x=x*2; non ha alcun effetto su A il cui R-valore resta 10 !!

A non è visibile in f



al ritorno x sparisce e il suo spazio RAM è liberato

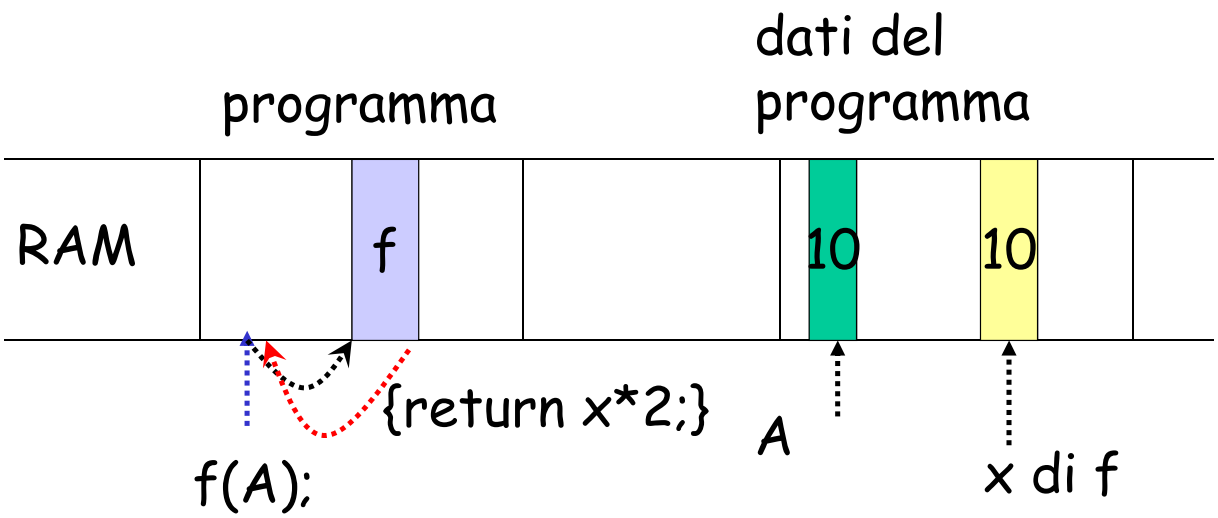
ma x*2 è l'R-valore che vorremmo dare ad A e quindi potremmo fare:

```
int f(int x)
{return x*2;}
```

invocata con:

```
int A=10;
A=f(A);
```

ok per 1 variabile, ma se sono di più?



return x*2 ritorna 20 ad A e poi x sparisce e il suo spazio RAM è liberato

per estendere le funzioni in modo che possano fare side-effect

possiamo usare i puntatori

anziché passare per valore a f l'R-valore della variabile A da modificare, passiamo sempre per valore il puntatore alla variabile A

quindi avremo: void f(int * x);

se un parametro formale è int * x allora il corrispondente parametro attuale deve fornire l'indirizzo di un intero

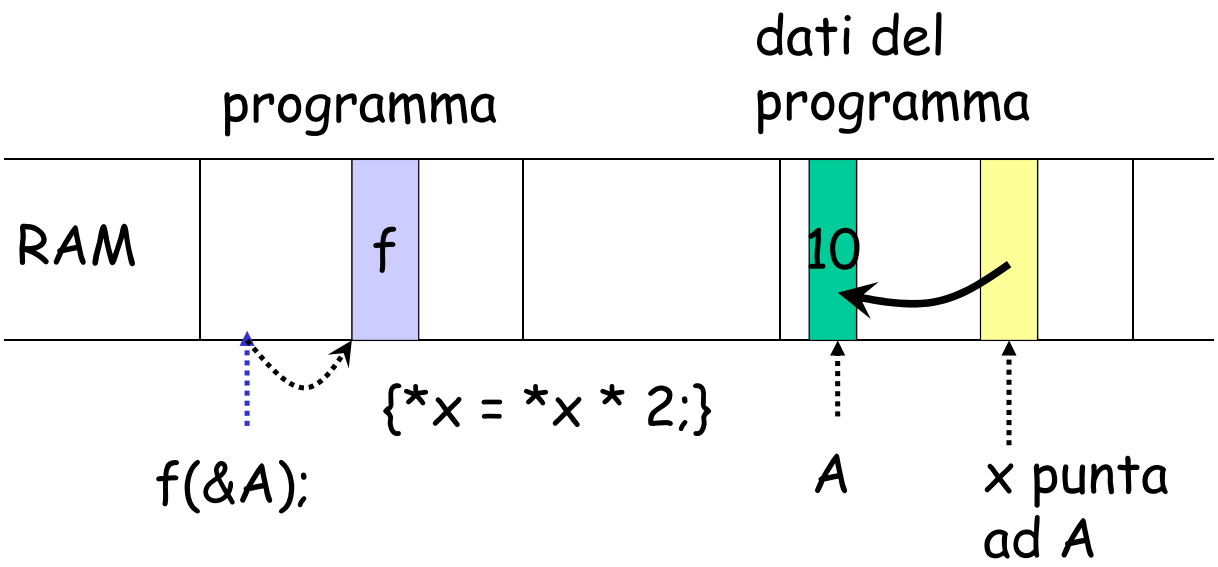
cioè un'espressione che ha un valore di tipo int *

```
void f(int * x)
{
*x=*x+2;
}
```

come invocarla
per cambiare la variabile A ?

f(&A);

passiamo l'L-valore di A



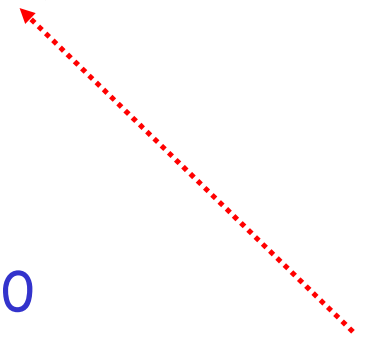
*x = oggetto puntato da x cioè A

posso invocare f anche così:

```
int A=10, *p=&A;
f(p);    // passaggio per valore
qui A==20
```


passaggio dei parametri per riferimento

```
void f(int & x) {x=x*2;}
main()
{
int A=10;
f(A);
} // qui A=20
```



x è passato per riferimento => x è un alias di A

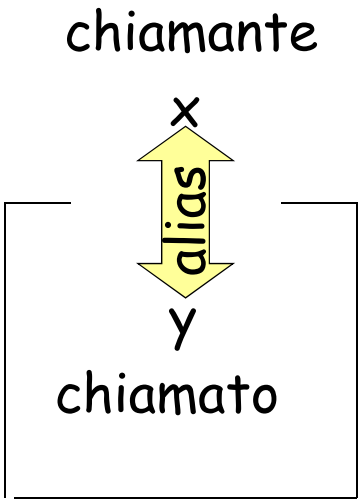
```
void g(int x, int & y)
{ x=y+1; }
main()
{
int A=10;
g(A,A);
} // valore di A ?
```

e con & ?

e se ?

```
void g(int x, int & y)
{ x++; y++;}
```


i parametri passati per riferimento mettono in comune una variabile tra chiamante e chiamato



che può servire in entrambe le direzioni o solo in una

è diverso per i puntatori passati per valore ?

e ha senso passare un puntatore per riferimento?

e…

esercizio: scambiare gli R-valori di 2 variabili:

```
char x='a',y='b';
f(x,y);
// qui x=='b' e y=='a'
```

come deve essere ? f(?, ?)