

esercizi su funzioni:

- passaggio dei parametri
- restituzione dei risultati
- invocazione

esempio 0:

```
int * f(int * x){ *x=5; return x;}  
main()  
{ int y=1;  
  *f(&y)=25;  
  cout<<y<<endl;  
}
```

è corretto e stampa 25

f riceve e ritorna un puntatore a *y*

y riceve 5 in *f* e poi 25

esempio 1 :

```
#include <iostream>
int * f(int x){ x=5; return &x;}
main()
{ int y=1;
*f(y)=7*y;
cout<<y<<endl;}
```

ERRORE LOGICO, ma no warning, *f* ritorna un puntatore a *x* (var. locale che viene deallocata) quindi **f(y)=7*y* ha effetto su *x* (ORRORE!) e non su *y* che rimane 1.

esempio 2:

```
#include <iostream>
int & f(int & x){ x=5; return x;} main()
{ int y=1;
*f(y)=25;
cout<<y<<endl;}
```

Errore:

In function 'int main' : invalid type argument of 'unary *'

esempio 3:

```
#include <iostream>
int * f(int & x){ x=5; return &x;}
main() { int y=1;
*f(y)=25;
cout<<y<<endl;
}
```

è corretto e stampa 25
x è un alias di *y* ed *f* ritorna un puntatore a *y*
quindi *y* diventa 5 in *f* e poi 25

esempio 4:

```
#include <iostream>
int & f(int & x){ x=5; return &x;}
main() { int y=1;
*f(y)=25;
cout<<y<<endl;
}
```

ERROR: In function 'int main()': invalid type
argument of 'unary *'

esempio 5:

```
#include <iostream>
int * *f(int * x){ *x=5; return &x;}
main() { int y=1;
**f(&y)=25;
cout<<y<<endl;
}
```

ERRORE LOGICO, ma no warning,

esempio 6:

```
#include <iostream>
int * f(int * & x){ *x=5; return x;}
main()
{ int y=1, x;
x=*f(&y);
cout<<y<<" "<<x<<endl;}
```

In function 'int main()': initialization of non-const reference 'int *&' from r-value 'int *' in passing argument 1 of 'f(int * &)' .

esempio 7:

```
#include <iostream>
int & f(int * & x){ *x=5; return *x;}
main() { int y=1, x; int *z=&y;
x= f(z);
cout<<y<<" "<<x<<endl; }
```

CORRETTA: stampa 5 5

f ritorna un alias di *y* ed *x* è un alias di *z*

esempio 8:

```
#include <iostream>
int & f(int * & x){ *x=5; return *x;}
main() { int y=1; int *z=&y;
f(z)=25;
cout<<y<<endl;
}
```

CORRETTA: stampa 25

come nell'esempio precedente, *f* ritorna un alias di *y*, quindi *y* riceve 25

esempio !

```
#include <iostream>
int * f(int * x){ *x=5; return x;}
main() { int y=1;
*f(&y)=25*y;
cout<<y<<endl;
}
```

è corretto, ma può stampare 125 oppure 25 a seconda che venga eseguita prima f(&y) (cosa probabile) oppure 25*y

esercizio:

```
int k=5, *z=&k;
*f(&z)=k+5;
cout<<*z <<endl;
vogliamo una f() t. c. stampi 10
```

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

Domande a risposta multipla - 1

```
int F(int *x){*x = 10; return *x;}  
main(){  
    int x =1, y=1, *p = &x, &q = y;  
    y = F(&q) + 1;  
    cout << x <<y<<endl;}
```

1. La compilazione da un errore di tipo
2. Il programma compila e stampa 1 11
3. Il programma compila e stampa 10 11

Domande a risposta multipla - 2

```
int F(int *x){*x = 10; return *x;}  
main(){  
    int x =1, y=1, *p = &x, * &q = &y;  
    y = F(p);  
    cout << x <<y<<endl;  
}
```

1. La compilazione da un errore di tipo
2. Il programma compila e stampa 10 10

Domande a risposta multipla - 3

```
main(){  
    int y=1, *ptr = &y, *&q = ptr;  
    *ptr = 2; cout << y;  
    *q = 3; cout << y <<endl;  
}
```

1. Il programma compila e stampa 1 1
2. La compilazione da un errore di tipo
3. Il programma compila e stampa 2 3

Domande a risposta multipla - 4

```
char * C(char x, char &y) {x='b'; y=x; return &x;}  
main(){  
    char A[] = {'a','b','c'}, *p;  
    p=C(A[2],A[0]);  
    cout << A[0] <<endl; }
```

1. Il programma è sbagliato
2. Il programma compila e stampa b
3. Il programma compila e stampa a

Domande a risposta multipla - 5

```
char & C(char &x, char &y) {y=x; return x;}  
main(){  
    char A[] = {'a','b','c'};  
    C(A[0],A[1]) = A[2];  
    cout << A[0] << A[1] << A[2] <<endl; }
```

1. Il programma è sbagliato
2. Il programma compila e stampa c a c
3. Il programma compila e stampa b b c

tipo degli array e aritmetica dei
puntatori

int A[3][3][2][5][10][20]

A : int(*) [3][2][5][10][20]

A+3 : tipo? valore ?

A[3] : int (*) [2][5][10][20]

*(A+3): tipo? valore ?

Domande a risposta multipla - 6

```
char C(char x, char &y) {x=y; return x;}  
main(){  
    char A[] = {'a','b','c'};  
    A[2] = C(A[0],A[1]) ;  
    cout << A[0] << A[1] << A[2] <<endl; }
```

1. Il programma è sbagliato
2. Il programma compila e stampa a b b
3. Il programma compila e stampa b b b

esercizio

Vogliamo gestire un array di interi ordinato e nel quale si possono fare operazioni di cancellazione

- dobbiamo leggere da cin interi fino a che non si legga -11(sentinella)
- cercare l'intero letto nell'array e se c'è cancellarlo
 - tenendo conto del fatto che l'array è ordinato e
 - lasciando l'array ancora ordinato

la cancellazione si può dividere in 2 parti:

1) funzione di ricerca

2) funzione di shift

il numero di elementi buoni dell'array diminuisce

10, 12, 14, 20 25,30,49,87

eliminare 20

↑
top

cerca

Pre=(A[0..top-1] è definita, y è definita)

Post=(ci da la risposta giusta)

=(se trovato → pos in [0..top-1] e A[pos]==y
altrimenti, y non presente in A[0..top-1])

restituisce 2 valori: bool e pos = posizione di y
in A

1) ricerca che tiene conto dell'ordine:

```
bool cerca(int A[], int top, int y, int &pos){  
    bool trovato=false;    invariante ?  
    for(int i=0; i<top && ! trovato && !(A[i]>y); i++)  
        if(A[i]==y) {trovato=true; pos=i;}  
    return trovato;  
}
```

```
for(int i=0; i<top && ! trovato && !(A[i]>y);    ;i++)  
    if(A[i]==y)  
        {trovato=true; pos=i;}
```

ATTENZIONE all'ORDINE delle CONDIZIONI

sarebbe sbagliato

! trovato && !(A[i]>y) && i<top

il C++ valuta le espressioni booleane in un modo speciale detto **short-cut**:

valuta da sinistra a destra e non appena il valore finale è noto, la valutazione del resto non viene più fatta

$A \ \&\& \ B \ \&\& \ C$ se A è falso, allora B e C non sono valutate

$A \ || \ B \ || \ C$ se A è vero, allora B e C non sono valutate

INVARIANTE

```
for(int i=0; i<top && ! trovato && !A[i]>y ;i++)
```

```
    if(A[i]==y)
```

```
        {trovato=true; pos=i;}
```

POST=(trovato \Leftrightarrow esiste k in $[0, \text{top}-1]$, tale che $A[k]==y$ e $\text{pos}==k$)



$R=(\text{trovato} \Leftrightarrow \text{esiste } k \text{ in } [0, i-1] \text{ tale che } A[k]==y \text{ e } \text{pos}==k) \ \&\& \ (0 \leq i \leq \text{top})$

funziona ?

dimostrare che:

1) **inizio**: R vale all'inizio

2) **ciclo**: R è invariante del ciclo
 $R \ \&\& \text{condizione} \langle \text{corpo del ciclo} \rangle R$

3) **fine**: $R \ \&\& \text{!condizione} \Rightarrow \text{POST}$

$R = (\text{trovato} \Leftrightarrow \text{esiste } k \text{ in } [0, i-1] \text{ tale che } A[k] == y \text{ e } \text{pos} == k) \ \&\& (0 \leq i \leq \text{top})$

1) **inizio**: $\text{trovato} = \text{false}$ e $i = 0$ **banale**

2) **ciclo**:

$R \ \&\& (i < \text{top} \ \&\& \text{!trovato} \ \&\& \text{!}A[i] > y)$

$\langle \text{if}(A[i] == y) \{ \text{trovato} = \text{true}; \text{pos} = i; \}, i++ \rangle$

R

non avevo ancora niente e $i < \text{top}$ dice che $A[i]$ va considerato

se $A[i] == y$ metto le cose a posto **OK**

$R = (\text{trovato} \Leftrightarrow \text{esiste } k \text{ in } [0, i-1] \text{ tale che } A[k] == y \text{ e } \text{pos} == k) \ \&\& (0 \leq i \leq \text{top})$

3) **fine**: $R \ \&\& \ ! (i < \text{top} \ \&\& \ ! \text{trovato} \ \&\& \ ! A[i] > y) \Rightarrow \text{POST}$
 $i == \text{top} \ || \ \text{trovato} \ || \ A[i] > y$

- a) trovato è vero: $R \Rightarrow \text{POST}$
- b) $i == \text{top} \ \&\& \ ! \text{trovato}$: $R \Rightarrow \text{POST}$
- c) $i < \text{top} \ \&\& \ ! \text{trovato} \ \&\& \ A[i] > y$:

$! \text{trovato} \Rightarrow$ non esiste match in $A[0..i-1]$

e $A[i] > y \Rightarrow A[i.. \text{top}-1] > y \Rightarrow A[0.. \text{top}-1]$ non contiene y
 $\Rightarrow \text{POST}$

abbiamo considerato veramente tutti i casi ?

non abbiamo considerato il caso in cui:

trovato $\&\& \ A[i] > y$

è grave ?

NO perchè da R , trovato dice che abbiamo trovato k in $[0..i-1]$ tale che $y == A[k]$ e $\text{pos} = k$

quindi vale **POST**

shift

sposta gli elementi a sinistra, poi decrementa il valore di top di uno e azzerava $A[top]$

```
Pre=(top >=0,  
A[0..top-1] è definita e pos in [0..top-1]  
chiamo vA l'array A iniziale e  
vtop il valore iniziale di top  
)
```

```
Post=(A[0..pos-1]==vA[0..pos-1] &&  
A[pos..vtop-2]==vA[pos+1..vtop-1] &&  
top==vtop-1, A[top]==0  
)
```

```
Post=(A[0..pos-1]==vA[0..pos-1] &&  
A[pos..vtop-2]==vA[pos+1..vtop-1]  
  
&& top==vtop-1, A[top]==0  
)
```

```
R= (A[0..pos-1]==vA[0..pos-1] &&  
A[pos..i-1]==vA[pos+1..i] ) && (pos<=i<=vtop-1)
```



```
R= (A[0..pos-1]==vA[0..pos-1] &&  
A[pos..i-1]==vA[pos+1..i] ) && (pos<=i<=top-1)
```

```
void shift(int A[], int & top, int pos){  
    for(int i = pos; i < top-1; i++){  
        A[i] = A[i+1];  
    }
```

```
(A[pos..top-2]==vA[pos+1..top-1] )
```

```
A[--top] = 0;
```

```
}
```

```
(A[0..pos-1]==vA[0..pos-1] && A[pos..vtop-2] ==  
vA[pos+1..vtop-1] && top==vtop-1, A[top]==0 )
```

Main: lettura dell'array

```
cout << "Inserire un array ordinato" << endl;  
cout << "Inserisci un intero: " << endl;  
cin >> input;
```

```
while(input != -11){  
    A[top] = input;  
    top++;  
    cout << "Inserisci un intero: " << endl;  
    cin >> input;  
}
```

Main: operazione di cancellazione

```
if (top > 0){  
    cout << "Num. da cancellare: "<<endl;  
    cin >> cancella;  
    int pos = -1;  
    if (cerca(A, top, cancella, pos)) {  
        shift(A, top, pos);  
    }  
    stampa(A,top);  
}
```