valori restituiti dalle funzioni

una funzione può restituire il risultato col return in 2 modi :

1. per valore si restituisce solo un R-valore (del tipo dichiarato)

2. per riferimento: ritorna una variabile completa, cioè sia il suo L- che il suo R-valore (nella dichiarazione, nel tipo restituito appare T &)

```
int   f(….)
{
 …….
 return espr_intera
}
```

1

```
int & f(….)
{
……..
return a; // a variabile intera
}
```

2

1.risultato restituito *per valore:* *T F(…)*

L'invocazione di F può apparire dovunque serva **solo un R-valore**

```
…=…F()…  // Ok:  alla destra dell'=
         //serve solo R-valore

se bool F(..);

if( F(..) ) ……  // Ok

F(..) =…… // NO, servirebbe un L-valore
```

e SE restituiamo un puntatore per valore ?

```
int * g(...);
x= .....*g(..)  ... // OK, ma anche
*g(...)= ..... // OK
```

se g() restituisce l'indirizzo di una variabile intera allora *g(...) è la variabile

con L- ed R-valore

esempio di return char *:

```
char * max(char X[], int dim)
{int pos=0;
for(int i=1; i<dim ; i++)
   if(X[i] > X[pos])
       pos=i;
return &X[pos];
}
```

come invocare max:

```
char A[10] = "stringati"; //10 char
*(max(A,9))= 'B';
```

che valore ha A ?

stringhe alla C:

"stringati" ➔ 's' 't' 'r' 'i' 'n' 'g' 'a' 't' 'i' '\0'

\0 sentinella

PERICOLO:

bisogna fare attenzione a non restituire (per valore) puntatori a variabili locali della funzione

infatti queste variabili vengono deallocate dopo il return

dangling pointer

puntatore "penzolante"

ERRORE DIFFICILE da TROVARE

esempio di ERRORE di dangling pointer:

```
int * F(int y){return &y;}
```

è invece ok:

```
int * F(int *p){return p;}
```

mentre

```
int * F(int *p){int x; p=&x; return p;}
```

produce un dangling pointer

ATTENZIONE: ERRORE NON SEGNALATO dal COMPILATORE !!

2. Risultato restituito *per riferimento:*

T & F(...)

in questo caso l'invocazione della funzione restituisce una variabile completa (dotata di R- e di L-valore)

Quindi l'invocazione può sempre apparire sia alla destra che alla sinistra delle assegnazioni !!

```
int & F(....);

F(...)=..... // OK si usa L-valore

x=....F(...).... // OK si usa R-valore
```

esempio di return char &:

```
char &  max(char X[], int dim)
{int pos=0;
for(int i=0; i<dim ; i++)

   if(X[i] > X[pos])
       pos=i;

return X[pos];

}
```

max(C,10)='B';

nell'esempio precedente viene restituito un riferimento ad un elemento dell'array C che è un array dichiarato nel chiamante

---

MA attenzione !!!

a non restituire per riferimento una variabile locale della funzione infatti queste variabili spariscono quando si esegue il return

anche questo errore è chiamato di dangling pointer (e non rilevato)

dangling pointer ?

```
char & F(char c)
{ return c;  }
```

e ...

```
char & F(char c)
{ char w='a'; c=w; return c;  }
```

i parametri passati per riferimento possono servire anche a restituire risultati

```
void F(int &x)
{ x=x+2;}
main()
{
int y=0;
F(y);
cout<< y;
}
```

stampa 2

calcola posizione del max e min in X

```
void minmax(char X[], int dim, int & posx,
int & posn)
{posx=0; posn=0;
for(int i=1; i<dim ; i++)

   if(X[i] > X[posx])
       posx=i;

    else
       if(X[i]<X[posn])
        posn=i;
}
```

uso di minmax:

```
char C[]="stringata";

int min, max;

minmax(C, 9, max, min);

cout<< C[min]<<' '<<C[max] << '\n';
```

cosa stampa ?

ATTENZIONE

& ha 2 significati:

1) nelle dichiarazioni e parametri formali delle funzioni: dichiara una variabile riferimento

2) nelle espressioni: &x indica l'l-valore di x

ricorda che una simile situazione vale per *

esercizio-riflessione:

ricorda che con

```
char & F(char *X) {return X[0];}
```

possiamo restituire X[0] come variabile

e con

```
void F(char *X, char & e){e=X[0]  }
```

possiamo restituire X[0] come variabile usando e?