valori restituiti dalle funzioni

testo Sezione 7.5

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

1. per valore: restituisce solo un R-valore

2. per riferimento: ritorna una variabile completa, cioè sia il suo L- che il suo R-valore

funziona come per il passaggio dei parametri

1

```
int   f(….)
{
 ……..
 return espr con valore intero
}
```

2

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

1. per valore: int 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, serve un L-valore
```

e SE restituissimo un puntatore per valore ?

```
int * g(...);

x= .....*g(..)   e anche g(..)+2 // OK

*g(...)= ..... // OK
```

se g() restituisce l'indirizzo di una variabile intera allora *g(…) **E'** la variabile intera con L-valore = g()

esempio di return char* per valore:

```
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”;
*(max(A,9))= ‘B’;

che valore ha A ?

stringati ➔ sBringati

char A[10] = “stringati”;

“stringati” è una stringa alla C e corrisponde ad un array di caratteri

in realtà: “stringati\0”
cout<< A << endl; //cosa stampa?

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 o puntatore “penzolante”

ERRORE DIFFICILE da TROVARE

esempio di dangling pointer (è un ERRORE!!)

```
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: int & 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];
}
```

e possiamo invocare max con:

char C[]=“stringati”;
max(C,9)=‘B’;

e se max(C, 9) mi serve anche per altri usi

char & m=max(C,9);

m= .....

m= ....

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 reference (e non è rilevato dal compilatore)

c'è un dangling reference?

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

e qui

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

risultato restituito per riferimento è diverso da un parametro passato per riferimento

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

char z[10], &w= F(z);
```

l'alias lo decide F e il chiamante lo può usare dopo l'invocazione

mentre con

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

```
char z[10], w;
F(z,w);
```

w è fissato, F può eventualmente cambiarne il valore

restituire array ????

```
int* F()
{
 int A[20];
 return A;
}
```

NON ha SENSO !!!