costanti, conversioni e cast alla C e alla C++

le costanti:

```
const double pigreco=3,14;
```

inizializzazione sempre obbligatoria

anche per arrays:

```
const int A[4]={1,2,3,4};
```

puntatori costanti e puntatori a costanti

```
const char * pc;
```

pc punta ad un carattere costante cioè *pc non può cambiare, ma pc può cambiare!

i puntatori costanti invece sono:

```
char x,  * const PC = &x;  // inizializzazione!
```

PC non può più cambiare, ma *PC e x possono !

principio generale

```
char x;

const char * p=&x;  // ok possiamo
                    // aggiungere const
```

MA.....

```
const char x='a';

char * p=&x;  //  NOOO!! non possiamo
              // togliere const
```

```
int x=1;
const int * p=&x; //OK p puntatore a
                  // costante
cout << *p ;   // stampa 1
*p=*p+1;   // ERRORE
x=x+1;  // OK
cout<< *p; // stampa 2
```

esistono anche puntatori costanti:

```
int x=1, * const p=&x; // OK p è costante
*p=*p+1;  // OK x=2
p=p+1; // ERRORE !!!
```

per garantire che se invochiamo F(…A) A non venga cambiato da F:

- passiamo A per valore
- passiamo A per riferimento costante: const int & p (anche se nel chiamante A non è costante)
- passiamo un puntatore ad A come fosse costante: F(&A) e F(const int * p)

passare array costanti è importante per proteggerne il valore visto che non è possibile passarli per valore

void F(const int X[], int dim)

F può usare gli elementi di X, ma non cambiarne il valore

lo stesso per

void F(const int * X, int dim) e

void F(const int (*X)[10][10], int dim)

conversioni implicite ed esplicite

IMPORTANTE:

x = espressione;

- si valuta l'espressione indipendentemente dal tipo di x
- poi si converte il risultato al tipo di x

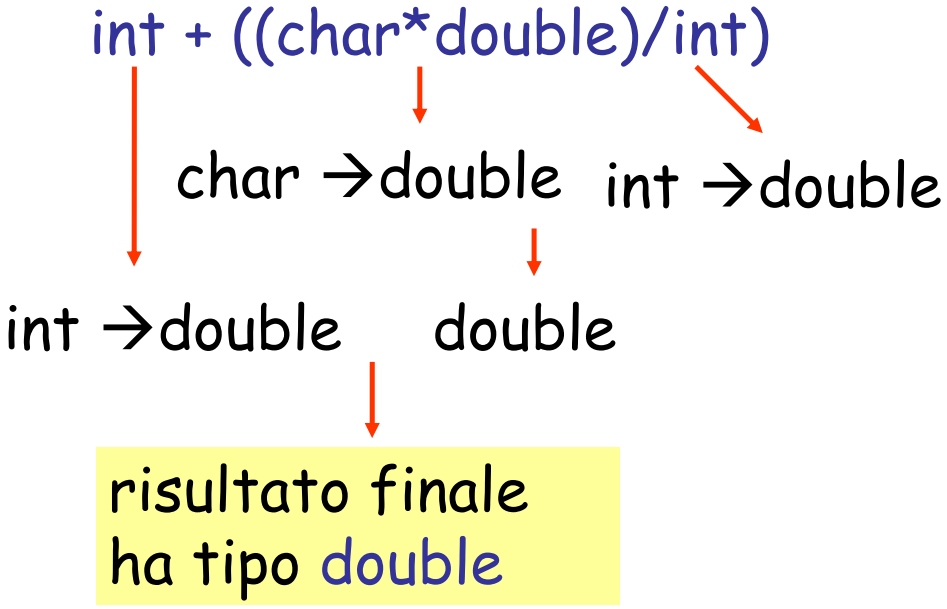
valutare espressioni con tipi diversi

5+ 'a' *3.2/12 → int + ((char*double)/int)

il compilatore esegue automaticamente le seguenti conversioni :

{bool,char} → {int} → {float} → {double}

PROMOZIONI



il valore di un'espressione ha sempre il tipo massimo tra quelli presenti nell'espressione stessa

in un'assegnazione x = espressione

IMPORTANTE: il tipo T di espressione,

non dipende dal tipo di x

se il tipo di x è S diverso da T, è necessaria un'altra conversione

T → S

viene (quasi) "sempre" effettuata, anche se comporta perdita di informazione:

double → char possibile warning

valutare 5 + 'a' * 3.2/12 produce un valore di tipo double

```
double x = 5 + 'a' *3.2/12; // ovvio oK

int y = 5 + 'a' *3.2/12; // warning, ma
                         //compila
e anche:
char z = 5 + 'a' *3.2/12; //warning ma
                          // compila
```

per evitare i warning, conviene chiedere la conversione esplicitamente:

```
int x= (int) (5 + 'a' *3.2/12)
```

tipo da ottenere

espressione il cui risultato va trasformato

i cast sono importanti perché rendono esplicite le conversioni da fare

conversioni esplicite: cast

il C offre una sola operazione di cast

(T) espressione

il valore dell'espressione è convertito in un valore "equivalente" del tipo T

questi cast vengono eseguiti senza discussione

il C++ offre invece 4 tipi di cast: a seconda del caso

1.static_cast

2.const_cast

3.reinterpret_cast

4.dynamic_cast

→ Programmazione a oggetti

per tutti la sintassi è:

xx_cast <tipo>(esp)

tipo da ottenere

valore da convertire

```
double x=3.14;
char y= x; //  ok con warning
char y= static_cast<char>(x); // meglio
```

static_cast

serve per le conversioni inverse delle promozioni:

int → enum

int → char

int → bool

double → int

a volte è utile anche per forzare conversioni sicure:

int x, y; .....

double z=x/y;

se vogliamo la divisione reale

double z= static_cast <double> (x) /y;

forziamo la conversione int→double

const_cast : serve a "togliere" i const:

```
char x='a', y='b', *p=&x;
const char * pp=&y;
p=pp; // errore !!
p=const_cast<char *>(pp); // OK !
```

cosa stampa ?

```
const int x=1, *p=&x;
int *q=const_cast<int*>(p);
(*q)++;
cout<<x<< ' ' << *q<<' '<<*p<<endl;
```

reinterpret_cast: T* --> S*

```
double x=3.14, *p=&x;

char *s= reinterpret_cast<char*>(p);

cout<<*s<<' '<<*(s+1)<<' '<<*(s+2)<<' '<<endl;
```

ATTENZIONE

viene sempre eseguito e non avviene alcuna trasformazione: né del puntatore né dell'oggetto puntato