29. Passaggio di parametri
In C++ esistono tre modalità di passaggio di parametri a una funzione:

- per valore
- per riferimento
- per puntatore

(Spesso le ultime due sono confuse in letteratura, perché hanno finalità simili).

29.1 Passaggio di parametri per valore
Definizione di parametri formali analoga alla definizione di variabili.
Sintassi lista dei parametri: *(tipo identificatore, ...)*
Esempio: *int fact(int n) {...}*

Simile a definire una nuova variabile locale e assegnarle il valore dell'espressione del parametro attuale.
Esempio: *fact(3*x);* //simile a: *int n = 3*x;*

Il parametro formale acquisisce il **valore** del parametro attuale, il quale può essere un'espressione senza indirizzo.
Può essere di tipo diverso compatibile (viene effettuata una conversione implicita).

L'informazione viene temporaneamente duplicata, causando uno spreco di memoria e di CPU.

Importante: Se modifico il parametro formale, il parametro attuale non viene modificato (passaggio di informazione solo dalla funzione chiamante alla funzione chiamata).

Anche le costanti possono essere passate per valore. Se il parametro formale corrispondente è del tipo *const tipo costante* esso non potrà essere modificato nella funzione. Altrimenti sarà possibile definire il parametro formale senza *const* e sarà modificabile.

29.2 Passaggio di parametri per riferimento
Definizione di parametri formali simile a definizione di riferimenti.
Sintassi lista dei parametri: *(tipo & identificatore,...)*
Esempio: *int swap(int & n, int & m) {...}*

Simile a definire un riferimento "locale" ad un'espressione dotata di indirizzo.
Esempio: *swap(x,y);* //simile a: *int & n=x; int & m=y*

Il parametro formale è un riferimento al parametro attuale il quale deve essere un'espressione dotata di **indirizzo** e deve essere dello **stesso tipo**.

In questo caso l'informazione non viene duplicata evitando uno spreco di CPU e memoria.

Importante: Se modifico il parametro formale, modifico il parametro attuale (il passaggio di informazioni avviene anche dalla chiamata alla chiamante).
! Quindi nelle procedure posso passare dei parametri per riferimento e siccome non ho l'istruzione *return* non restituisco alcun valore, ma modificando i parametri formali, modifico anche i parametri attuali. Stampando quindi a video nel *main* il parametro attuale a cui corrisponde un parametro formale riferimento, stamperò il nuovo valore modificato nella funzione.

29.2.1 Passaggio di parametri per riferimento costante
È possibile definire passaggi per riferimento in sola lettura.
Sintassi lista di parametri: *(const tipo & identificatore, ...)*

Esempio: *int fact(const int & n,...) {...}*

Anche in questo caso l'informazione non viene duplicata evitando uno spreco di memoria e di CPU.

Importante: Non permette di modificare *n*! Quindi scrivere *n = 5;* è un errore. Il passaggio di informazione avviene solo dalla chiamante alla chiamata (solo un input alla funzione).

Questo metodo è usato in input per passare alla funzione oggetti "grossi":

- molto efficiente: (no spreco di CPU e memoria)
- evita errori
- permette di individuare facilmente gli input della funzione0

! Il parametro attuale è ovviamente modificabile, solo il corrispondente parametro formale non è modificabile (si possono però chiaramente fare le ovvie operazioni di confronto come %, ecc.)!

Esempio

```
void prova (const int & z){
cout << z;      //10
z++; //errore
cout << z%2; //0
}

int main (){
int z=10;
cout << z;      //10
prova(z);
z++;
cout << z;      //11
}
```

Grazie al passaggio per parametri per riferimenti è possibile ritornare due valori. Infatti passando alla funzione due parametri attuali e inizializzando i parametri formali della funzione come due riferimenti, quando andrò a modificarli li modificherò anche per la funzione chiamante.

29.3 Passaggio di parametri per puntatore
Definizione di parametri formali: puntatori passati per valore del parametro attuale a cui si riferiscono.
Sintassi lista dei parametri: *(tipo * identificatore,...)*
Esempio: *int swap(int * pn, int * pm) {...}*
N.B.!: nella chiamata, si passa l'indirizzo dell'oggetto passato.

Simile a definire un puntatore "locale" ad un'espressione dotata di indirizzo
Esempio di chiamata: *swap(&x,&y);* //simile a definire un puntatore: *int *pn=&x; int*pm=&y*

Il parametro è un puntatore al(l'oggetto il cui indirizzo è dato dal) parametro attuale che deve essere un'espressione **dotata di indirizzo** e dello **stesso tipo**.

Anche qua l'informazione non viene duplicata, evitando uno spreco di memoria e di CPU.

Modificando il parametro formale (ovvero l'area di memoria puntata dal puntatore), si modifica anche il parametro attuale: il passaggio di informazione anche dalla chiamata alla chiamante. C'è un effetto simile al passaggio per riferimento.

! Se voglio lavorare con i valori dei parametri attuali devo usare **pn* (per riferirmi a *x*) e **pm* (per rifermi a *y*) nelle operazioni ! Usando *pn* e *pm* cambierò l'*r-value* del puntatore (ovvero lavorerò sugli indirizzi di memoria di *x* e *y*) !

Esempio
```
void scambia (int * px, int * py)
{ int t;
 t = *px;
 *px = *py;
 *py = t; }

int main (){
int a=2, b=3;
scambia(&a,&b);      //mi scambia il valore delle variabili, ora a=3 e b=2
}
```

Senza usare gli * avremmo scambiato gli indirizzi i *r-value* di *px* e *py* (quindi *px* avrebbe puntato a *b* e *py* avrebbe puntato a *a*).

29.4 Passaggio per valore vs. Passaggio per riferimento/puntatore

Vantaggi del passaggio per *riferimento/puntatore*
- Minore carico di calcolo e di memoria (soprattutto con parametri di grosse dimensioni)
- Permette di restituire informazione da chiamata a chiamante

Svantaggi del passaggio per *riferimento/puntatore*
- Rischio di confusione nel codice (non si sa dove cambiano i valori)
- Aliasing (entità con più di un nome)
- Parametro formale e attuale esattamente dello stesso tipo
- **!** Si possono passare solo espressioni dotate di indirizzo **!**

Alcuni linguaggi (come C) non ammettono il passaggio per riferimento, ma solo per puntatore.

29.5 Funzioni che restituiscono un riferimento
Restituisce un riferimento ad un'espressione (con indirizzo). L'espressione deve riferirsi ad un oggetto del chiamante (es. un parametro formale passato per riferimento, un elemento di un array) e deve essere dello stesso tipo.

Esempio
```
int& max(int& x,int& y)        //restituisce un riferimento
{return (x > y ? x : y);}      //x,y riferimenti a oggetti
```

Si può anche scrivere così:
```
if(x>y)
  int &xx = x;
else
  int &xx=y;

return xx;                     //basta che il valore ritornato sia di tipo int&

…
int m=44, n=22;
max(m,n) = 55; //cambia il valore di m da 44 a 55. Infatti guardando il return della funzione si
                capisce che restituisce il maggiore tra due numeri. Guardando il tipo che restituisce è
                un riferimento ad una variabile quindi restituisce un riferimento a x e/o a m cioè 44.
                Siccome restituisce un'espressione dotata di l-value (cioè m) è possibile associarle un
                valore.
```

29.6 Sovrapposizione di parametri (overloading)
In C++ è possibile dare lo stesso nome a funzioni diverse, purché con liste di parametri diverse, per numero e/o per tipo.
Il compilatore "riconosce" la giusta funzione per ogni chiamata. In caso di ambiguità, il compilatore produce un errore. Le conversioni implicite sono ammissibili, purché non causino ambiguità.

Esempio
```
int max(int, int) {...};
int max(int, int, int) {...};
double max(double, double) {...};    //tre funzioni con lo stesso nome, ma che differiscono per tipo
                                       restituito e lista di parametri formali
cout << max(99,77) << " " << max(55,66,33) << " " << max(3.4,7.2) << endl;
cout << max(3,3.1) << endl; // errore: AMBIGUA
```

29.7 Funzioni con argomenti di default

In C++ è possibile fornire parametri opzionali, con valori di default.
Ciò permette chiamate con liste di parametri attuali ridotte.
I parametri opzionali devono essere gli ultimi della lista e il match viene effettuato da sinistra a destra.

Esempio

```
double p(double, double, double =0, double =0, double =0);     //l'assegnazione dei valori di default va fatta in fase di dichiarazione
cout << p(x, 7) << endl;
cout << p(x, 7, 6) << endl;
cout << p(x, 7, 6, 5) << endl;
cout << p(x, 7, 6, 5, 4) << endl;

double p(double x, double a0, double a1, double a2, double a3)
{ return a0 + (a1 + (a2 + a3*x)*x)*x; }
```

29.8 Firma di una funzione

La firma di una funzione si compone del nome della funzione e del numero, ordine e tipo dei parametri formali.

```
int calcolaSomma (int, int);

int calcolaSomma (int addendo1, int addendo2){
… ;
}
```

Firma della funzione: *calcolaSomma(int, int);*