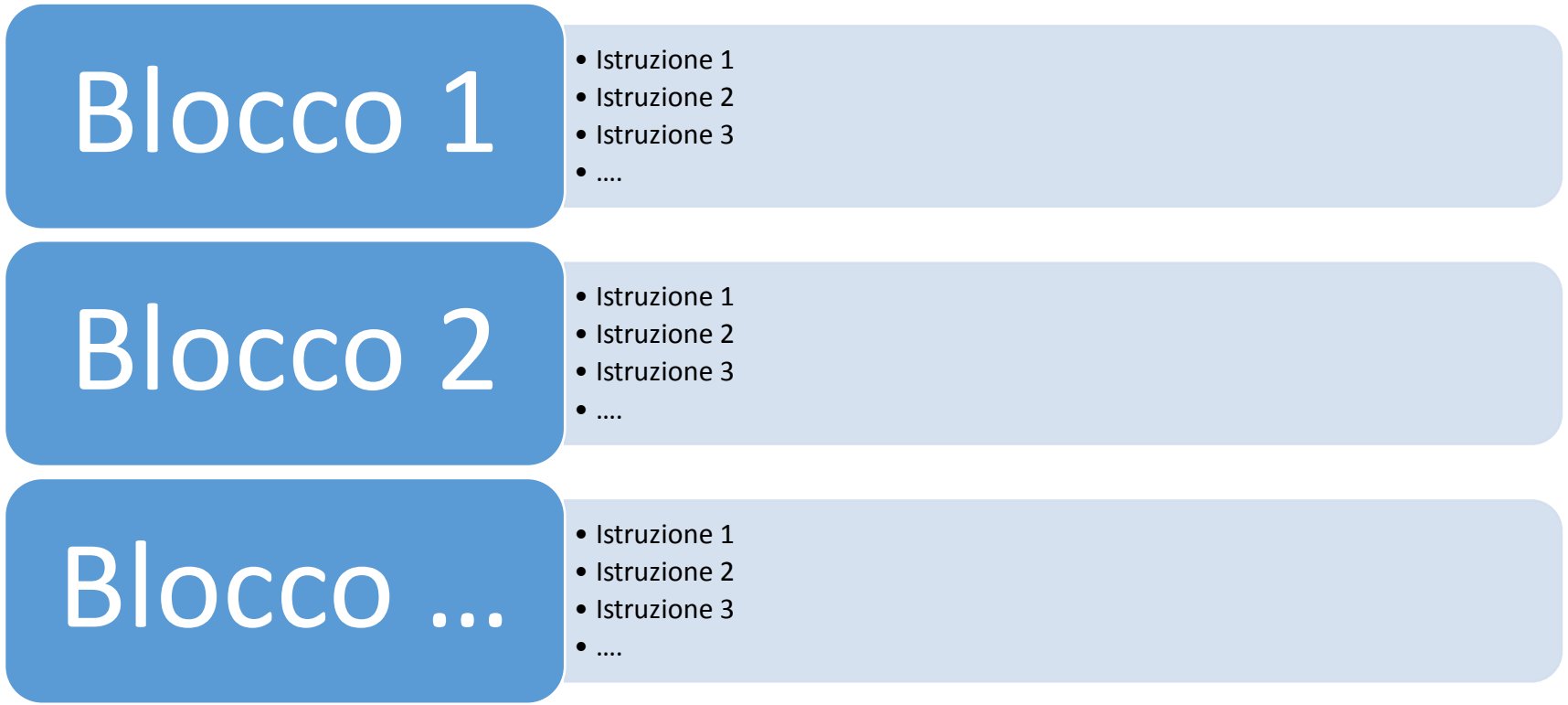
L’organizzazione dei programmi

Lo sviluppo top-down

Lo sviluppo top-down

I problemi complessi



Lo sviluppo top-down

- ogni *blocco* è un modulo che funziona in maniera indipendente;
- più facile costruire programmi sempre più complessi;
- più facile la manutenzione del software;
- i *blocchi* possono essere riutilizzati all'interno di altri programmi.

In linguaggio informatico i *blocchi* sono chiamati sottoprogrammi.

Nel linguaggio C++ i *sottoprogrammi* vengono detti ***function***

Le function

tipo di dato restituito **nome della function** (*elenco dei parametri*)

{

istruzione 1;

istruzione 2;

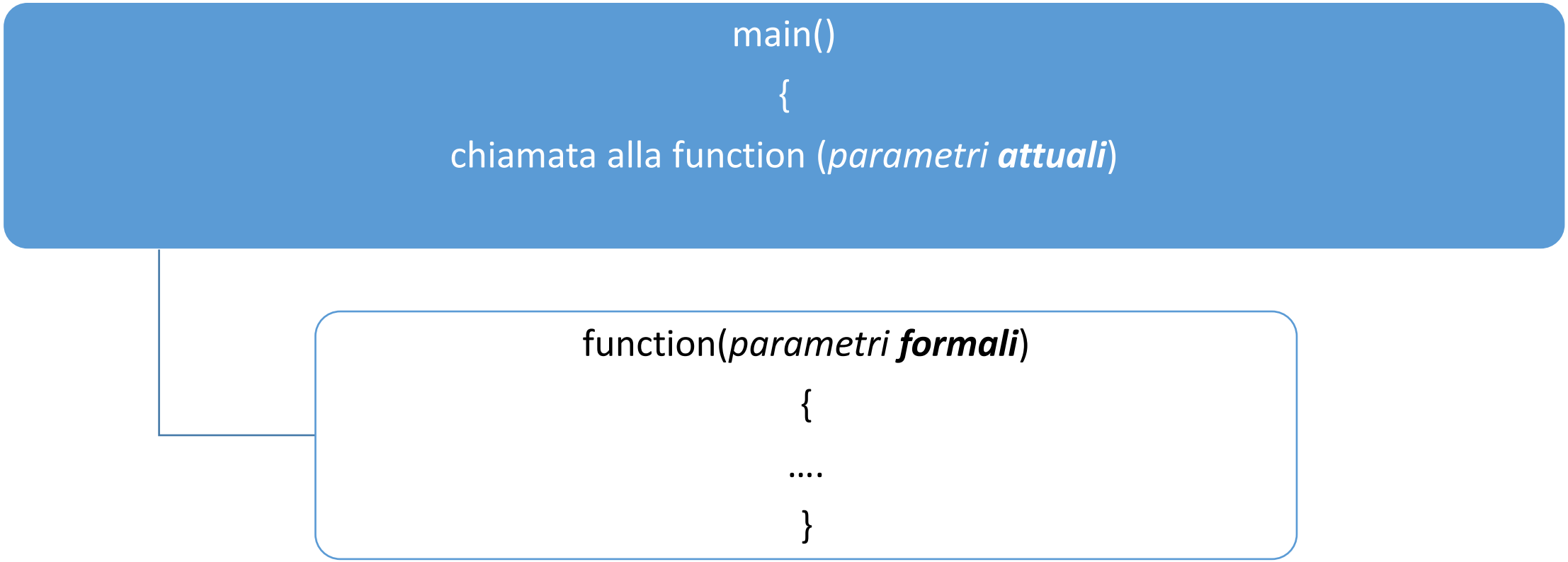
istruzione …;

return *valore restituito*;

}

```
// Prodotto di due numeri interi, con funzione iterativa
#include ….
using namespace std;
int prodotto(int a, int b);  // dichiarazione della function
int main ()
{
   int a, b, p;
   cout << "inserisci i due numeri interi" << endl;
   cin >> a >> b;
   cout << "a * b = " << prodotto(a,b) << endl;
          system ("PAUSE");
       return 0;
}
int prodotto(int a, int b)   // definizione della function
{
   int p = 0;
   do
      {
        p+=a;                        // p = p + a
        b--;                         // b = b - 1
       } while (b != 0);             // !=  «diverso da»
       return p;
}
```

Il passaggio dei parametri



Il passaggio *per valore*

```
int prodotto(int x, int y);
int main ()
{
   int a, b, p;
…
cout << "a * b = " << prodotto(a,b) << endl;
…}
int prodotto(int x, int y)
{…
   do
      {
        p+=x;
        y--;
       } while (y != 0);
…}
```

Il passaggio (dei parametri) ***per valore***

```
int prodotto(int x, int y);
int main ()
{
   int a, b, p;
…
cout << "a * b = " << prodotto(a,b) << endl;
…}
int prodotto(int x, int y)
{…
   do
      {
        p+=x;
        y--;
       } while (y != 0);
…}
```

I cambiamenti effettuati sui parametri formali durante l'esecuzione della funzione **non** influenzano i valori delle variabili nella funzione chiamante (*main* oppure altra *function*)

Ordinamento (crescente) di tre numeri

```
int main ()
{  int a,b, c;
   int temp;
cout << "inserisci i tre numeri ";
cin >> a >> b >> c;
if (a > b)
                {    temp = a;
                 a = b;
                 b = temp;  }
if (a > c)
                {   temp = a;
                  a = c;
                  c = temp;  }
  if (b > c)
                {  temp = b;
                   b = c;
                  c = temp;  }
     cout  ….
}
```

Il passaggio (dei parametri) ***per referenza***

```
void Ordina(int& x, int& y);   // function
{
    int temp;
    if (x > y)
            {
            temp = x;
            x = y;
            y = temp;
            }
}                       // fine della function
int main()                                  //programma principale
{
   int a, b, c;
   ….
   Ordina(a,b);
   Ordina(a,c);
   Ordina(b,c);
   cout ….
….
}
```

I cambiamenti effettuati sui parametri formali durante l'esecuzione della funzione influenzano i valori delle variabili nella funzione chiamante (*main* oppure altra *function*)

Le funzioni definite per *ricorsione* (*ricorsività*)

Ricorsione: la possibilità che una **funzione** ha di chiamare se stessa

La *chiamata* della funzione è contenuta *all'interno* della funzione stessa.

Hofstadter's Law

It always takes longer than you expect, even when

you take into account the Hofstadter's Law

Calcolo del *fattoriale* (metodo iterativo)

```
int main()
{
int n;
long double fatt;
cout <<"inserisci il numero n: ";
 cin >> n;
  fatt = 1;
for(int i=1; i<=n; i++){
fatt=fatt*i;
  }
 cout ..
•  }
```

Calcolo del *fattoriale* (metodo ricorsivo)

```
long double Fatt(int x);

int main()
{….
   cout <<"il fattoriale di " << n << "  e':  " << Fatt(n) << endl;
….}

long double Fatt(int x)
{
        double long f;
        if (x==0) f = 1;
        else f = x * Fatt(x-1);
        return f;
}
```

I *coefficienti binomiali*

```
long int Disp (int x, int y);
long int Fatt (int x);

int main ()
{
   int n, k;
   long int cnk;
   cout << "inserisci n e k" << endl;
   cin >> n >> k;
   cnk= Disp(n,k)/Fatt(k);
   cout << cnk;
...}
```

```
long int Fatt(int x)
{
        long int f;
        if (x==0) f = 1;
        else f = x * Fatt(x-1);
        return f;
}

long int Disp (int n, int k)
{
    long int t;
    t=n;
for (int i =1; i <=k-1 ; i++)
   t=t*(n-i);
   return t;
}
```

La *successione* di Fibonacci

1, 1, 2, 3, 5, 8, 13, 15, 28, 43, 71…

$f(k) = f(k-1) + f(k-2), \quad k = 3, 4, 5\ldots$

$f(1)=f(2)=1$

Prodotto tra due numeri positivi

$a*b = 0$ se $b = 0$

$a*b = a*(b-1) + a$ se $b > 0$

Somma di tutti i numeri che vanno da 1 a *n*

$$\sum_{i=1}^{1} i = 1$$

$$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i$$