

**LANFRANCO LOPRIORE**

**PROGRAMMAZIONE MEDIANTE ESEMPI**  
**Utilizzando il Linguaggio C++**

*Raccolta di lucidi*

**Febbraio 1999**

I titoli dei lucidi fanno riferimento ai capitoli del libro di A. Domenici e G. Frosini, *Introduzione alla Programmazione ed Elementi di Strutture Dati con il Linguaggio C++*, Seconda Edizione. Milano: Franco Angeli, 1997.

**I PROGRAMMI INCLUSI IN QUESTA DISPENSA HANNO VALORE ESCLUSIVAMENTE DIDATTICO. ESSI SONO STATI VERIFICATI CON CURA, MA NON SONO GARANTITI PER NESSUNO SCOPO SPECIFICO. L'AUTORE NON SI ASSUME NESSUNA RESPONSABILITÀ RIGUARDO AD ESSI.**

## 2.1 Metalinguaggio per la sintassi C++ (I)

```
identifier :          // categorie sintattiche in corsivo
    letter           // alternative su righe separate
    letter identifier-char-list

identifier-char-list : // definizione (ricorsiva) di lista
    identifier-char
    identifier-char identifier-char-list // ripetizione

identifier-char :
    letter
    digit

letter :
    one of          // alternative
                    // simboli terminali a spaziatura uniforme
    a b c d e f g h i j k l m n o p q r s t
    u v w x y z
    A B C D E F G H I J K L M N O P Q R S T
    U V W X Y Z

digit :
    one of
    0 1 2 3 4 5 6 7 8 9

//~~~~~
element-list :        // definizione di lista di elementi
    element
    element element-list

//~~~~~
element-clist :       // definizione di lista di elementi separati da virgola
    element
    element , element-clist
```

## 2.1 Metalinguaggio per la sintassi C++ (II)

```
if-statement :
    if ( expression ) statement
    if ( expression ) statement else statement

//~~~~~
while-statement :
    while ( expression ) statement

//~~~~~
do-statement :
    do statement while ( expression ) ;

//~~~~~
for-statement :        // parti opzionali con pedice opt
    for ( init-statement expressionopt ; expressionopt )
                    statement // proseguimento su riga rientrata
init-statement :
    expression-statement
    declaration-statement
```

## 3.2 Tipo intero

```
#include <iostream.h>

void main() {
    int i;
    int j = 7;
    int i1(5);
    int i2 = 0, i3;
    i2 = i1 * 2;
    i3 = 1 / 2;           // 0

    short int s1 = 1;
    short s2 = 2;

    long int ln1 = 7777;
    long ln2 = 7778L;     // literal long int, suffisso L (l)

    int ott = 011;        // 9
    // literal int ottale, prefisso 0 (zero)

    int esad = 0xF;        // 15
    // literal int esadecimale, prefisso 0x

    cout << ott << '\n' << esad << '\n';
}
```

### 3.2.1 Tipo unsigned (l)

```
#include <iostream.h>

void main() {
    unsigned int u1 = 1U;
    // literal unsigned int, suffisso U (u)
    unsigned u2 = 2;

    unsigned short int u3 = 3;
    unsigned short u4 = 4;

    unsigned long int u5 = 7779;
    unsigned long u6 = 7779UL;
    // literal unsigned long int, suffisso UL (ul)

    cout << u1 << '\n' << u2 << '\n' << u3 << '\n';
    cout << u4 << '\n' << u5 << '\n' << u6 << '\n';
}
```

### 3.2.1 Tipo unsigned (II)

// Vettori di bit

```
#include <iostream.h>

void main() {
    unsigned short a = 0xFFFFA;
    // 1111 1111 1111 1010 (65530)

    unsigned short b = ~a;
    // 0000 0000 0000 0101 (5)

    unsigned short c = 0x0013;
    // 0000 0000 0001 0011 (19)
    unsigned short c1, c2, c3;

    c1 = b | c;          // 0000 0000 0001 0111 (23)
    c2 = b & c;          // 0000 0000 0000 0001 (1)
    c3 = b ^ c;          // 0000 0000 0001 0110 (22)

    unsigned short b1, b2;

    b1 = b << 2;        // 0000 0000 0001 0100 (20)
    b2 = b >> 1;        // 0000 0000 0000 0010 (2)

    cout << b << '\n' << c << '\n';
    cout << c1 << '\n' << c2 << '\n' << c3 << '\n';
    cout << b1 << '\n' << b2 << '\n';
}
```

### 3.3 Tipo reale

```
#include <iostream.h>
```

```
void main() {
    double d = 2.2;
    double d1 = -14.12e-2, d2 = 2.62, d3 = .5, d4 = 6.;
```

```
float f = -1.1F;
float g = f - 12.12F;
// literal float, suffisso F (f)
```

```
long double h = +0.1;
long double k = 1.23e+12L;
// literal long double, suffisso L (l)
```

```
cout << d << '\t' << d1 << '\t' << d2 << '\t';
cout << d3 << '\t' << d4 << '\n';
cout << f << '\t' << g << '\n';
cout << h << '\t' << k << '\n';
}
```

```
//----------------------------------------------------------------
// Divisione tra interi e tra reali
```

```
#include <iostream.h>
```

```
void main() {
    int i = 1 / 2;           // 0
    float f = 1.0 / 2.0;     // 0.5
```

```
cout << i << '\n' << f << '\n';
}
```

### 3.4 Operatori di confronto e logici

```
#include <iostream.h>

void main() {
    int i = 10, j = 0, h, k, n;
    h = i > 3 && j < 5;           // 1
    k = i || j;                 // 1
    n = !i;                     // 0

    cout << h << '\n' << k << '\n' << n << '\n';
}
```

### 3.5 Tipi enumerazione

```
#include <iostream.h>

void main() {
    enum Giorno {LUN,MAR,MER,GIO,VEN,SAB,DOM};
    Giorno oggi = LUN;
    oggi = MAR;

    oggi = 3;                  // ERRORE! enum = const int

    oggi = i;                  // ERRORE! enum = int

    int i = oggi;              // 1, conversione implicita

    cout << int(oggi) << '\n'; // 1
    cout << oggi << '\n';      // 1, conversione implicita

    enum {ROSSO, GIALLO, VERDE} colore;
    colore = GIALLO;
    cout << colore << '\n';   // 1

    enum {STRANO1 = 10, STRANO2 = 12, STRANO3,
          STRANO4 = 1};
    cout << STRANO1 << '\t' << STRANO2 << '\t';
    cout << STRANO3 << '\t' << STRANO4 << '\n';
                                  // 10 12 13 1
}
```

### 3.6 Tipo carattere

```
#include <iostream.h>

void main() {
    char c = 'f', d = '\n';
    char e = '\x40';           // '@' (in esadecimale)
    char f = '\100';          // '@' (in ottale)
    char g = 64;              // '@' (in decimale)

    cout << c << d << e << f << g << d;

    char c1 = c + 1;          // 'g'
    char c2 = c - 2;          // 'd'
    char c3 = 4 * d + 3;      // '+' (!!!)
    int i = c - 'a';          // 5

    cout << c1 << '\n' << c2 << '\n' << c3 << '\n';
    cout << i << '\n';

    int m = 'a' < 'b', n = 'a' > 'c';      // 1 0

    cout << m << '\n' << n << '\n';
}
```

### 3.7.1 Conversioni implicite

```
#include <iostream.h>
```

```
void main() {
    int i = 10, j;
    float f = 2.5, h;
    double d = 1.2e+1;
    char c = 'd';

    h = f + 1;           // 3.5
    cout << h << '\n';

    j = f + 3.1;         // 5
    cout << j << '\n';

    d = i + 1;           // 11
    cout << d << '\n';

    d = f + d;           // 13.5
    cout << d << '\n';

    j = c - 'a';         // 3
    cout << j << '\n';
}
```

### 3.7.2 Conversioni esplicite

```
#include <iostream.h>

void main() {
    int i = (int) 1.1;           // cast, 1
    float f = float(2);         // notazione funzionale

    cout << i << '\n' << f << '\n';
}
```

### 3.8 Dichiarazioni di oggetti costanti

```
#include <iostream.h>

void main() {
    const int i = 0;
    const long double e = 2.718;
    const long double e2 = 2 * e;

    cout << i << '\t' << e << '\t' << e2 << '\n';

    i = 5;                      // ERRORE!
    const int j;                 // ERRORE!
}
```

### 3.9 Operatore sizeof (I)

```
#include <iostream.h>

void main() {
    cout << "char " << sizeof(char) << '\n';      // 1
    cout << "short " << sizeof(short) << '\n';    // 2
    cout << "int " << sizeof(int) << '\n';        // 4
    cout << "long " << sizeof(long) << '\n';      // 4

    cout << "unsigned char ";
    cout << sizeof(unsigned char) << '\n';        // 1
    cout << "unsigned short ";
    cout << sizeof(unsigned short) << '\n';       // 2
    cout << "unsigned int ";
    cout << sizeof(unsigned int) << '\n';         // 4
    cout << "unsigned long ";
    cout << sizeof(unsigned long) << '\n';        // 4
    cout << "signed char ";
    cout << sizeof(signed char) << '\n';          // 1

    cout << "float " << sizeof(float) << '\n';   // 4
    cout << "double ";
    cout << sizeof(double) << '\n';                // 8
    cout << "long double ";
    cout << sizeof(long double) << '\n';           // 8
}
```

### 3.9 Operatore sizeof (II)

```
#include <iostream.h>

void main() {
    cout << "costante carattere ";
    cout << sizeof('c') << '\n';                  // 1
    cout << "costante intera ";
    cout << sizeof 3 << '\n';                     // 4
    cout << "costante reale ";
    cout << sizeof 3.5 << '\n';                   // 8

    int i = 0;
    cout << "variabile intera " << sizeof i << '\n'; // 4

    cout << "char " << sizeof char << '\n'; // ERRORE!
}
```

### 3.10.1 Funzioni aritmetiche

(in `<stdlib.h>`)

- `abs(n)`
- `div(m, n)`
- `rand()`
- `srand(n)`

(in `<math.h>`)

- `fabs(x)`
- `ceil(x)`
- `floor(x)`
- `fmod(x, y)`
- `pow(x, y)`
- `sqrt(x)`
- `ldexp(x, n)`
- `frexp(x, p)`
- `modf(x, p)`

Funzioni trigonometriche (in `<math.h>`)

- `sin(x)`
- `cos(x)`
- `tan(x)`
- `asin(x)`
- `acos(x)`
- `atan(x)`
- `atan2(x, y)`

Funzioni esponenziali e logaritmiche (in `<math.h>`)

- `exp(x)`
- `log(x)`
- `log10(x)`
- `sinh(x)`
- `cosh(x)`
- `tanh(x)`

### 3.10.2 Funzioni sui caratteri

(in `<ctype.h>`)

- `isalnum(c)`
- `isalpha(c)`
- `iscntrl(c)`
- `isdigit(c)`
- `isgraph(c)`
- `islower(c)`
- `ispunct(c)`
- `isspace(c)`
- `isupper(c)`
- `isxdigit(c)`
- `tolower(c)`
- `toupper(c)`
- `isprint(c)`

## 4.2 Espressioni di assegnamento

```
#include <iostream.h>

void main() {
    int i = 1, j = 2, k;

    i = 10;
    cout << i << '\n';                                // 10

    i = j;
    cout << i << '\t' << j << '\n';                // 2 2

    k = j = i = 12;          // associativo a destra
    cout << i << '\t' << j << '\t' << k << '\n'; // 12 12 12

    k = j = 2 * (i = 100);
    cout << i << '\t' << j << '\t' << k << '\n'; // 100 200 200

    k = j +1 = 2 * (i = 100);                         // ERRORE!
}
```

### 4.2.1 Altri operatori di assegnamento

```
#include <iostream.h>
```

```
void main() {
    int i = 1, j = 10;

    i += 1;           // i = i + 1
    cout << i << '\n'; // 2

    i *= j + 1;      // i = i * (j+1);
    cout << i << '\n'; // 22
}
```

## 4.2.2 Incremento e decremento

```
#include <iostream.h>

void main() {
    int i, j;

    i = 0;
    ++i;                                // i += 1;
    cout << i << '\n';                  // 1

    i = 0;
    j = ++i;                            // i += 1; j = i;
    cout << i << '\t' << j << '\n';  // 1 1

    i = 0;
    i++;                                // 1
    cout << i << '\n';

    i = 0;
    j = i++;                           // j = i; i += 1;
    cout << i << '\t' << j << '\n';  // 1 0

    j = ++i++;                          // ERRORE!
}
```

## 4.3 Espressioni aritmetiche e logiche (I)

### // Precedenza

```
#include <iostream.h>

void main() {
    int i = 1, j;
    j = 3 * i + 1;
    cout << j << '\n';                // 4

    j = 3 * (i + 1);
    cout << j << '\n';                // 6
}
```

~~~~~  
// Associatività

```
#include <iostream.h>

void main() {
    double e = 27.0, f = 3.0, g;
    g = e / f / 2;
    cout << g << '\n';                // 4.5

    g = e / f * 2;
    cout << g << '\n';                // 18

    g = e / (f * 2);
    cout << g << '\n';                // 4.5
}
```

## 4.3 Espressioni aritmetiche e logiche (II)

### // Espressioni con operatori di relazione e logici (i)

```
#include <iostream.h>

void main() {
    int i = 1, j = 11, k;
    k = i >= 2 && j <= 1;
    // forma equivalente (i >= 2) && (j <= 1)
    cout << k << '\n';           // 0

    int a = 0, b = 1, c;
    c = a && b || !a;
    // forma equivalente (a && b) || (!a)
    cout << c << '\n';           // 1
}
```

```
//-----
// Espressioni con operatori di relazione e logici (ii)
```

```
#include <iostream.h>

void main() {
    int i = 15, j;
    j = 0 < i < 10;           // ATTENZIONE!
    cout << j << '\n';           // 1

    j = 0 < i && i < 10;
    cout << j << '\n';           // 0
}
```

## 4.3 Espressioni aritmetiche e logiche (III)

### // Cortocircuito

```
#include <iostream.h>
```

```
void main() {
    int i, j;
    cin >> i;

    j = (i < 0) || (i > 100);
    cout << j << '\n';

    j = (i >= 0) && (i <= 100);
    cout << j << '\n';

    j = (i != 0) && (200 / i > 10);      // OK
    cout << j << '\n';

    j = (2 / i > 10) && (i != 0);          // NO!
    cout << j << '\n';
}
```

## 4.4 Operatore condizionale

// Legge due interi e scrive il maggiore

```
#include <iostream.h>
```

```
void main() {
    int a, b, max;
    cout << "? ";
    cin >> a >> b;
    max = (a > b ? a : b);
    cout << max << '\n';
}
```

//~~~~~  
// Legge due interi ed incrementa il maggiore

```
#include <iostream.h>
```

```
void main() {
    int a, b;
    cout << "? ";
    cin >> a >> b;
    a >= b ? a++ : b++;
    cout << a << '\t' << b << '\n';
}
```

### 6.3.1 Istruzione if (I)

// Legge due interi e scrive il maggiore

```
#include <iostream.h>
```

```
void main() {
    int a, b, max;
    cin >> a >> b;
    if (a > b)
        max = a;
    else
        max = b;
    cout << max << '\n';
}
```

//~~~~~  
// Incrementa o decrementa

```
#include <iostream.h>
```

```
void main() {
    int a, b;
    cin >> a >> b;
    if (a >= b) {
        a++;
        b++;
    }
    else {
        a--;
        b--;
    }
    cout << a << '\t' << b << '\n';
}
```

### 6.3.1 Istruzione if (II)

// Valore assoluto (if senza parte else)

```
#include <iostream.h>
```

```
void main() {
    int a;
    cin >> a;
    if (a < 0)
        a = -a;
    cout << a << '\n';
}
```

```
//~~~~~
// Incrementa, e scrive se il risultato e` diverso da 0
// (if senza espressione relazionale)
```

```
#include <iostream.h>
```

```
void main() {
    int i;
    cin >> i;
    if (i++)
        cout << i << '\n';
}
```

### 6.3.1 Istruzione if (III)

// If in cascata

```
// if (a > 0 ) if ( b > 0 ) a++; else b++;
```

```
#include <iostream.h>
```

```
void main() {
    int a, b;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
        else
            b++;
    cout << a << '\t' << b << '\n';
}
```

```
//~~~~~
// Scrittura fuorviante
```

```
#include <iostream.h>
```

```
void main() {
    int a, b;
    cin >> a >> b;
    if (a > 0)
        if (b > 0)
            a++;
        else
            b++;
    cout << a << '\t' << b << '\n';
}
```

### 6.3.1 Istruzioni if (IV)

// Scrive asterischi

```
#include <iostream.h>

void main() {
    int i;
    cin >> i;
    if (i == 1)
        cout << "*";
    else if (i == 2)
        cout << "**";
    else if (i == 3)
        cout << "***";
    else
        cout << "!";
    cout << '\n';
}
```

### 6.3.1 Istruzione if (V)

// Equazione di secondo grado

```
#include <iostream.h>
#include <math.h>

void main() {
    double a, b, c;
    cout << "Coeffienti? ";
    cin >> a >> b >> c;
    if ((a == 0) && (b == 0))
        cout << "Equazione degenere\n";
    else
        if (a == 0) {
            cout << "Equazione di primo grado\n";
            int x = -c / b;
            cout << "x: " << x << '\n';
        }
        else {
            double delta = b * b - 4 * a * c;
            if (delta < 0)
                cout << "Soluzioni immaginarie\n";
            else {
                delta = sqrt(delta);
                double x1 = (-b + delta) / (2 * a);
                double x2 = (-b - delta) / (2 * a);
                cout << "x1: " << x1 << '\n';
                cout << "x2: " << x2 << '\n';
            }
        }
}
```

### 6.3.2 Istruzioni switch e break (I)

// Scrive asterischi

```
#include <iostream.h>
```

```
void main() {
    int i;
    cin >> i;
    switch (i) {
        case 1:
            cout << "*";
            break;
        case 2:
            cout << "***";
            break;
        case 3:
            cout << "****";
            break;
        default:
            cout << '!';
    }
    cout << '\n';
}
```

### 6.3.2 Istruzioni switch e break (II)

// Scrive asterischi (in cascata)

```
#include <iostream.h>
```

```
void main() {
    int i;
    cin >> i;
    switch (i) {
        case 3: // in cascata
            cout << "*";
        case 2: // in cascata
            cout << "***";
        case 1:
            cout << "****";
            break;
        default:
            cout << '!';
    }
    cout << '\n';
}
```

### 6.3.2 Istruzioni switch e break (III)

// Selezione tramite caratteri

```
#include <iostream.h>
```

```
void main() {
    char c;
    cin >> c;
    switch (c) {
        case 'a': case 'e': case 'i':
        case 'o': case 'u':
            cout << "Vocale minuscola\n";
            break;
        case 'A': case 'E': case 'I':
        case 'O': case 'U':
            cout << "Vocale maiuscola\n";
    // Manca il caso di default
    // Se non è una vocale, non scrive niente
    }
}
```

### 6.3.2 Istruzioni switch e break (IV)

// Scrittura di enumerazioni

```
#include <iostream.h>
```

```
void main() {
    enum {ROSSO, GIALLO, VERDE} colore;
    char c;
    cin >> c;
    switch (c) {
        case 'r': case 'R':
            colore = ROSSO;
            break;
        case 'g': case 'G':
            colore = GIALLO;
            break;
        case 'v': case 'V':
            colore = VERDE;
    }
    /* ... */
    switch (colore) {
        case ROSSO:
            cout << "ROSSO";
            break;
        case GIALLO:
            cout << "GIALLO";
            break;
        case VERDE:
            cout << "VERDE";
    }
    cout << '\n';
}
```

#### 6.4.1 Istruzione while (I)

// Scrive n asterischi, con n dato (i)

```
#include <iostream.h>

void main() {
    int n, i = 0;
    cin >> n;
    while (i < n) {
        cout << "*";
        i++;
    }           // al termine, n conserva il valore iniziale
    cout << '\n';
}
```

//~~~~~//

// Scrive n asterischi, con n dato (ii)

```
#include <iostream.h>

void main() {
    int n;
    cin >> n;
    while (n > 0) {
        cout << "*";
        n--;
    }           // al termine, n vale 0
    cout << '\n';
}
```

#### 6.4.1 Istruzione while (II)

// Scrive n asterischi, con n dato (iii)

```
#include <iostream.h>
```

```
void main() {
    int n;
    cin >> n;
    while (n-- > 0)
        cout << "*";           // al termine, n vale -1
    cout << '\n';
}
```

//~~~~~//

// Scrive n asterischi, con n dato (iv)

```
#include <iostream.h>
```

```
void main() {
    int n;
    cin >> n;
    while (n--)
        cout << "*";           // non termina se n <= 0
    cout << '\n';
}
```

### 6.4.1 Istruzione while (III)

// Legge, raddoppia e scrive interi non negativi  
// Termina al primo negativo

```
#include <iostream.h>

void main() {
    int i;
    cout << "? ";
    cin >> i;
    while (i >= 0) {
        cout << 2 * i << '\n';
        cout << "? ";
        cin >> i;
    }
}
```

```
//----------------------------------------------------------------
// Calcola il massimo m tale che la somma dei primi
// m interi positivi è minore o uguale al dato intero
// positivo n
```

```
#include <iostream.h>

void main() {
    int somma = 0, m = 0, n;
    cin >> n;
    while (somma <= n)
        somma += ++m;
    m--;
    cout << m << '\n';
}
```

### 6.4.1 Istruzione while (IV)

// Calcola il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

```
#include <iostream.h>

void main() {
    int i = 0, j = 1;
    int s, n;
    cin >> n;
    while ((s = i + j) <= n) {
        i = j;
        j = s;
    }
    cout << j << '\n';
}
```

## 6.4.2 Istruzione do

```
// Calcola il massimo termine della successione di  
// Fibonacci minore o uguale al dato intero positivo n  
  
#include <iostream.h>  
  
void main() {  
    int i, j = 0, s = 1, n;  
    cin >> n;  
    do {  
        i = j;  
        j = s;  
    } while ((s = i + j) <= n);  
    cout << j << '\n';  
}
```

## 6.4.3 Istruzione for (I)

```
// Scrive n asterischi, con n dato (i)  
  
#include <iostream.h>  
  
void main() {  
    int n;  
    cin >> n;  
    for (int i = 0; i < n; i++)  
        cout << "*";           // al termine, i vale n  
    cout << '\n';  
}  
  
//~~~~~  
// Scrive n asterischi, con n dato (ii)  
  
#include <iostream.h>  
  
void main() {  
    int n;  
    cin >> n;  
    for (; n > 0; n--)  
        cout << "*";           // al termine, n vale 0  
    cout << '\n';  
}
```

### 6.4.3 Istruzione for (II)

// Scrive asterischi e punti esclamativi

```
#include <iostream.h>
```

```
void main() {
    const int N = 5;
    for (int i = 0; i < N; i++)
        cout << "*";
    for (i = 0; i < N; i++)          // i già definita
        cout << "!";
    cout << '\n';
}
```

```
////////////////////////////////////////////////////////////////////////
```

// Calcola la somma dei primi n interi

```
#include <iostream.h>
```

```
void main() {
    int n, somma = 0;
    cin >> n;
    for (int j = 1; j <= n; j++)
        somma += j;
    cout << somma << '\n';
}
```

### 6.4.3 Istruzione for (III)

// Scrive una matrice di asterischi formata da r righe  
// e c colonne, con r e c dati

```
#include <iostream.h>
```

```
void main() {
    int r, c;
    cin >> r >> c;
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            cout << "*";
            cout << '\n';
        }
    }
}
```

### 6.5.1 Istruzione break

```
// Legge e scrive interi non negativi
// Termina al primo negativo

#include <iostream.h>

void main() {
    int j;
    for (;;) { // ciclo infinito; altra forma: while (1)
        cout << "? ";
        cin >> j;
        if (j < 0)
            break;
        cout << j << '\n';
    }
}

//~~~~~
// Legge e scrive al piu` cinque interi non negativi
// Termina al primo negativo

#include <iostream.h>

void main() {
    const int n = 5;
    for (int i = 0, j; i < n; i++) {
        cout << "? ";
        cin >> j;
        if (j < 0)
            break;
        cout << j << '\n';
    }
}
```

### 6.5.2 Istruzione continue

```
// Legge cinque interi e scrive i soli non negativi

#include <iostream.h>

void main() {
    const int n = 5;
    for (int i = 0, j; i < n; i++) {
        cout << "? ";
        cin >> j;
        if (j < 0) continue;
        cout << j << '\n';
    }
}
```

## 7.1 Concetto di funzione (I)

// Problema

```
#include <iostream.h>

int n;

void main() {
    int f;
    /* ... */
    cin >> n;
    f = 1;
    for (int i = 2; i <= n; i++)
        f *= i;
    cout << f << '\n';
    /* ... */
    cin >> n;
    f = 1;
    for (i = 2; i <= n; i++)
        f *= i;
    cout << f << '\n';
    /* ... */
}
```

## 7.1 Concetto di funzione (II)

// Soluzione

```
#include <iostream.h>

int n;

int fatt() {
    int ris = 1;
    for (int i = 2; i <= n; i++)
        ris *= i;
    return ris;
}

void main() {
    int f;
    /* ... */
    cin >> n;
    f = fatt();
    cout << f << '\n';
    /* ... */
    cin >> n;
    f = fatt();
    cout << f << '\n';
    /* ... */
}
```

### 7.3.1 Istruzione return (I)

```
// Calcola il massimo termine della successione di  
// Fibonacci minore o uguale al dato intero positivo n
```

```
#include <iostream.h>
```

```
int fibo(int n) {  
    int i = 0, j = 1, s;  
    for (;;) {  
        if ((s = i + j) > n) return j;  
        i = j;  
        j = s;  
    }  
}  
  
void main() {  
    int h, k;  
    cin >> h;  
    k = fibo(h);  
    cout << k << '\n';  
}
```

### 7.3.1 Istruzione return (II)

```
// Ritorna -1, 0 oppure 1 in base al valore negativo,  
// nullo o positivo di un dato intero n
```

```
#include <iostream.h>
```

```
int segno(int n) {  
    if (n > 0)  
        return +1;  
    if (n == 0)  
        return 0;  
    return -1;  
}  
  
void main () {  
    int i;  
    cin >> i;  
    switch (segno(i)) {  
        case -1:  
            cout << "Valore negativo\n";  
            break;  
        case 0:  
            cout << "Valore nullo\n";  
            break;  
        case 1:  
            cout << "Valore positivo\n";  
    }  
}
```

## 7.4 Dichiarazioni di funzioni (I)

// Valore assoluto (ERRATO)

```
#include <iostream.h>

void main() {
    int i;
    cout << "? ";
    cin >> i;
    int j = abs(i);      // ERRORE!
                        // Identificatore non dichiarato
    cout << j << '\n';
}

int abs(int n) {
    return n > 0 ? n : -n;
}
```

## 7.4 Dichiarazioni di funzioni (II)

// Valore assoluto (dichiarazione e definizione)

```
#include <iostream.h>

int abs(int n);          // anche: int abs(int)

void main() {
    int i;
    cout << "? ";
    cin >> i;
    int j = abs(i);
    cout << j << '\n';
}

int abs(int n) {
    return n > 0 ? n : -n;
}
```

## 7.6 Argomenti e variabili locali (I)

// Problema

```
#include <iostream.h>

int h, k;

void main() {
    int f;
    /* ... */
    cin >> h >> k;
    /* ... */
    f = 1;
    for (int i = 2; i <= h; i++)
        f *= i;
    cout << f << '\n';
    /* ... */
    f = 1;
    for (i = 2; i <= k; i++)
        f *= i;
    cout << f << '\n';
    /* ... */
}
```

## 7.6 Argomenti e variabili locali (II)

// Soluzione

```
#include <iostream.h>

int h, k;

int fatt(int n) {
    int ris = 1;
    for (int i = 2; i <= n; i++)
        ris *= i;
    return ris;
}

void main() {
    int f;
    /* ... */
    cin >> h >> k;
    f = fatt(h);
    cout << f << '\n';
    /* ... */
    f = fatt(k);
    cout << f << '\n';
    /* ... */
}
```

## 7.6 Argomenti e variabili locali (III)

```
// Somma gli interi compresi tra i dati interi n ed m,  
// estremi inclusi, con n <= m  
  
#include <iostream.h>  
  
int sommalInteri(int n, int m) {  
    int s = n;  
    for (int i = n+1; i <= m; i++)  
        s += i;  
    return s;  
}  
  
void main () {  
    int a, b;  
    while (cout << "? ", cin >> a >> b)  
        // termina se legge un valore illegale per a, b  
        cout << sommalInteri(a, b) << '\n';  
}
```

## 7.6 Argomenti e variabili locali (IV)

// Massimo fra tre

```
#include <iostream.h>
```

```
int massimo(int a, int b, int c) {  
    return ((a > b) ? ((a > c) ? a : c) : (b > c) ? b : c);  
}  
  
void main() {  
    int i, j, k;  
    cout << "? ";  
    cin >> i >> j >> k;  
    int m = massimo (i, j, k);  
    cout << m << '\n';  
    /*...*/  
    double x, y, z;  
    cout << "? ";  
    cin >> x >> y >> z;  
    double w = massimo (x, y, z);  
    // Si applicano le regole standard per la conversione di  
    // tipo. Esempio: ingresso 3.3 4.4 5.5, uscita 5  
    cout << w << '\n';  
}
```

## 7.7 Funzioni void

```
#include <iostream.h>

void scriviAsterischi(int n) {
    for (int i = 0; i < n; i++)
        cout << "*";
}

void main() {
    int i;
    cout << "? ";
    cin >> i;
    scriviAsterischi(i);
    cout << '\n';
}
```

## 7.8 Funzioni senza argomenti

```
#include <iostream.h>

const int N = 5;

void scriviAsterischi() {
    for (int i = 0; i < N; i++)
        cout << "*";
}

void main() {
    /* ... */
    scriviAsterischi();
    cout << '\n';
    /* ... */
}
```

## 7.9 Ricorsione (I)

```
// Fattoriale

#include <iostream.h>

int fatt(int n) {
    return (n < 2) ? 1 : n * fatt(n-1);
}

void main() {
    for (int i = 0; i <= 10; i++)
        cout << i << '\t' << fatt(i) << '\n';
}
```

## 7.9 Ricorsione (II)

```
// Calcola l'n-esimo termine della successione di
// Fibonacci

#include <iostream.h>

int fibo(int n) {
    if (n == 1)
        return 0;
    if (n == 2)
        return 1;
    return fibo(n-1) + fibo(n-2);
}

void main() {
    for (int i = 1; i <= 10; i++)
        cout << i << '\t' << fibo(i) << '\n';
}
```

## 7.9 Ricorsione (III)

```
// Legge una parola terminata da un punto, e la scrive
// in senso inverso. Esempio: legge "qwerty.", scrive
// "ytrewq"

#include <iostream.h>

void leggilnverti() {
    char c;
    cin >> c;
    if (c != '.') {
        leggilnverti();
        cout << c;
    }
}

void main() {
    leggilnverti();
    cout << '\n';
}
```

## 8.2 Puntatori (I)

// Definizione di puntatori

```
void main () {
    int *p1;           // puntatore a interi
    int* p2;
    int * p3;

    int *q1, *q2;     // due puntatori
    int* q3, q4;     // un puntatore ed un intero

    /* ... */
}

//~~~~~
// Operatori di indirizzo e indirezione

#include <iostream.h>

void main () {
    int i = 1;
    int* p = &i;
    *p = 10;

    cout << i << '\t' << *p << '\n'; // 10      10
    cout << p << '\n';
// Esempio: 0x0012FF78 (di solito in esadecimale)
}
```

## 8.2 Puntatori (II)

// Due puntatori per lo stesso oggetto

```
#include <iostream.h>
```

```
void main () {
    int i = 0;
    int *h = &i, *k;
    k = h;
    *k = 1;
    cout << i << '\t' << *h << '\n'; // 1 1
}
```

////////////////////////////////////////////////////////////////////////

```
#include <iostream.h>
```

```
void main () {
    int i, j;
    int *p = &i, *q = &j;
    cin >> i >> j; // Esempio: 3 5
    *p = *q;
    cout << i << '\t' << j << '\n'; // Esempio: 5 5
    cout << *p << '\t' << *q << '\n'; // Esempio: 5 5

    cin >> i >> j; // Esempio: 4 6
    p = q;
    cout << i << '\t' << j << '\n'; // Esempio: 4 6
    cout << *p << '\t' << *q << '\n'; // Esempio: 6 6

    p = *q; // ERRORE! int* = int
    *p = q; // ERRORE! int = int*
}
```

## 8.2 Puntatori (III)

// Puntatori a costanti

```
#include <iostream.h>
```

```
void main () {
    int i = 0;
    const int *p = &i; // OK
    // N.B.: i non e` costante
```

```
int j;
j = *p; // OK
*p = 1; // ERRORE!
```

```
const int k = 10;
const int *q = &k; // OK
```

```
int *qq;
qq = &k; // ERRORE! int* = const int*
}
```

## 8.2 Puntatori (IV)

// Puntatori costanti

```
#include <iostream.h>
```

```
void main () {
    int i = 1;
    int *const p = &i;
    cout << *p << '\n';           // 1

    *p = 10;
    cout << *p << '\n';           // 10

    int j = 10;
    p = &j;                      // ERRORE!
}
```

```
//~~~~~
// Puntatore a puntatore
```

```
#include <iostream.h>

void main () {
    int i = 1, j = 10;
    int *pi = &i, *pj = &j;
    int **q = &pi;
    *q = pj;
    cout << **q << '\n';           // 10
}
```

## 8.2 Puntatori (V)

// Puntatori nulli

```
#include <iostream.h>
```

```
void main () {
    int *p = NULL;             // forma equivalente: int *p = 0

    *p = 1;                   // ERRORE!

    if (p == NULL) { /* ... */ }
    if (p == 0) { /* ... */ }
    if (!p) { /* ... */ }
}
```

```
//~~~~~
// Puntatore non inizializzato
```

```
#include <iostream.h>
```

```
void main () {
    int *p;
    cout << *p << '\n';         // ATTENZIONE!
}
```

## 8.2.2 Operazioni sui puntatori (I)

// Scambia interi (ERRATO)

```
#include <iostream.h>

void scambia (int a, int b) {          // non scambia
    int c = a;
    a = b;
    b = c;
}

void main() {
    int i, j; cin >> i >> j;           // Esempio: 2  3
    scambia (i, j);
    cout << i << '\t' << j << '\n';   // Esempio: 2  3
}

//~~~~~
// Scambia interi (trasmissione mediante puntatori)

#include <iostream.h>

void scambia (int* a, int* b) {
    int c = *a;
    *a = *b;
    *b = c;
}

void main() {
    int i, j; cin >> i >> j;           // Esempio: 2  3
    scambia (&i, &j);
    cout << i << '\t' << j << '\n';
}
```

## 8.2.2 Operazioni sui puntatori (II)

// Incrementa il maggiore tra due interi trasmessi  
// mediante puntatori

```
#include <iostream.h>

void incrementa(int* a, int* b) {
    if (*a > *b)
        (*a)++;
    else
        (*b)++;
}

void main () {
    int i, j;
    cin >> i >> j;
    incrementa(&i, &j);
    cout << i << '\t' << j << '\n';
}
```

## 8.2.2 Operazioni sui puntatori (III)

// Puntatore come valori di ritorno (i)

```
#include <iostream.h>

int* sel(int s, int* a, int* b, int* c) {
    // Seleziona a, b oppure c in base al valore negativo,
    // nullo o positivo di s
    if (s < 0)
        return a;
    if (s == 0)
        return b;
    return c;
    // OK: ritorna l'indirizzo di una variabile trasmessa dal
    // chiamante
}

void main () {
    int sl, h = 1, k = 10, r = 100;
    cin >> sl;                                // Esempio: -1
    *sel(sl, &h, &k, &r) = 0;
    cout << h << '\t' << k << '\t' << r << '\n';
                                            // Esempio: 0 10 100

    cin >> sl;                                // Esempio: +1
    int i = (*sel(sl, &h, &k, &r))++;
    cout << h << '\t' << k << '\t' << r << '\t' << i << '\n';
                                            // Esempio: 0 10 101 100
}
```

## 8.2.2 Operazioni sui puntatori (IV)

// Puntatori come valori di ritorno (ii)

```
#include <iostream.h>
```

```
int* sel(int s, int* a, int* b, int* c) {
    int* i = c;
    if (s < 0)
        i = a;
    else if (s == 0)
        i = b;
    return i;
}
```

~~~~~  
// Puntatori come valori di ritorno (iii)

```
#include <iostream.h>
```

```
int* sel(int s, int* a, int* b, int* c) { // ERRATO!
    int i = *c;
    if (s < 0)
        i = *a;
    else if (s == 0)
        i = *b;
    return &i;
    // ATTENZIONE!
    // Ritorna l'indirizzo di una variabile locale
}
```

## 8.2.2 Operazioni sui puntatori (V)

// Puntatori come valori di ritorno (iv)

```
#include <iostream.h>

int* sel(int s, int* a, int* b, int* c) {
    if (s < 0)
        return a;
    if (s == 0)
        return b;
    return c;
}

void main () {
    const int N = 100;
    int sl, h = 1, k = 10;
    cin >> sl;
    int i = sel(sl, &h, &k, &N);      // ERRORE!
    cout << i << '\n';
}
```

## 8.2.2 Operazioni sui puntatori (VI)

// Trasmissione di parametri mediante puntatori a  
// costanti

```
#include <iostream.h>
```

```
int* selc(int s, const int* a, const int* b, const int* c) {
    if (s < 0)
        return (int*)a;           // conversione esplicita
    if (s == 0)
        return (int*)b;
    return (int*)c;
}
```

```
void main () {
```

```
    int sl, h = 1, k = 10, r = 100;
    cin >> sl;                  // Esempio: -1
    *selc(sl, &h, &k, &r) = 0;
    cout << h << '\t' << k << '\t' << r << '\n';
                                // Esempio: 0 10 100
```

```
    cin >> sl;                  // Esempio: +1
```

```
    int i = (*selc(sl, &h, &k, &r))++;
    cout << h << '\t' << k << '\t' << r << '\t' << i << '\n';
                                // Esempio: 0 10 101 100
}
```

## 8.2.2 Operazioni sui puntatori (VII)

// Puntatori a costanti come valori di ritorno

```
#include <iostream.h>

const int* cselc(int s, const int* a, const int* b,
                 const int* c) {
    if (s < 0)
        return a;
    if (s == 0)
        return b;
    return c;
}

void main () {
    int sl, h = 1, k = 10;
    const int N = 100;

    cin >> sl;
    int i = *cselc(sl, &h, &k, &N);           // OK
    cout << i << '\n';

    const int* p = cselc(sl, &h, &k, &N);
    cout << *p << '\n';

    *cselc(sl, &h, &k, &N) = 0;             // ERRORE!
}
```

## 8.3 Riferimenti (I)

#include <iostream.h>

```
void main() {
    int i = 1;
    int &r = i;
    cout << i << '\t' << r << '\n';          // 1 1
    r++;
    cout << i << '\t' << r << '\n';          // 2 2

    int h = 0, k = 1;
    int& r1 = h, &r2 = k; // due riferimenti
    int& r3 = h, r4;    // un riferimento ed un intero
}
```

//-----//  
// Scambia interi

#include <iostream.h>

```
void scambia (int& a, int& b) {
    int c = a;
    a = b;
    b = c;
}

void main() {
    int i, j; cin >> i >> j;
    scambia (i, j);
    cout << i << '\t' << j << '\n';
}
```

### 8.3 Riferimenti (II)

// Riferimenti a costanti

```
#include <iostream.h>

void main () {
    int i = 1;
    const int &r = i;          // Notare: i non è costante

    int j = 10;
    j = r;                   // OK
    cout << j << '\n';       // 1

    r = 1;                   // ERRORE!

    const int k = 10;
    const int &t = k;
    cout << t << '\n';       // OK
                            // 10

    int &tt = k;              // ERRORE!
}
```

### 8.3 Riferimenti (III)

// Riferimenti a costanti come argomenti formali

```
#include <iostream.h>
```

```
int f(const int& n) {
    // const previene modifiche della variabile trasmessa
    // per riferimento
    int i = n + 10;           // OK
    n += 10;                 // ERRORE!
    return n;                // OK
}
```

### 8.3 Riferimenti (IV)

// Riferimento di ritorno (i)

```
#include <iostream.h>

int& sel(int s, int& a, int& b, int& c) {
    // Seleziona a, b oppure c in base al valore negativo,
    // nullo o positivo di s
    if (s < 0)
        return a;           // riferimento
    if (s == 0)
        return b;
    return c;
}

void main () {
    int sl, h = 1, k = 10, r = 100;
    cin >> sl;           // Esempio: -1
    sel(sl, h, k, r) = 0;
    cout << h << '\t' << k << '\t' << r << '\n';
                                // Esempio: 0 10 100

    cin >> sl;           // Esempio: +1
    int i;
    i = sel(sl, h, k, r)++;
    cout << h << '\t' << k << '\t' << r << '\t' << i << '\n';
                                // Esempio: 0 10 101 100
}
```

### 8.3 Riferimenti (V)

// Riferimento di ritorno (ii)

```
#include <iostream.h>

int& sel(int s, int* a, int* b, int* c) {
    // Seleziona a, b oppure c in base al valore negativo,
    // nullo o positivo di s
    if (s < 0)
        return *a;
    // espressione che produce un valore sinistro
    if (s == 0)
        return *b;
    return *c;
}

void main () {
    int sl, h = 1, k = 10, r = 100;
    cin >> sl;           // Esempio: -1
    sel(sl, &h, &k, &r) = 0;
    cout << h << '\t' << k << '\t' << r << '\n';
                                // Esempio: 0 10 100

    cin >> sl;           // Esempio: +1
    int i;
    i = sel(sl, &h, &k, &r)++;
    cout << h << '\t' << k << '\t' << r << '\t' << i << '\n';
                                // Esempio: 0 10 101 100
}
```

### 8.3 Riferimenti (VI)

// Riferimento di ritorno (iii)  
// (ERRATO)

```
#include <iostream.h>

int& sel(int s, int& a, int& b, int& c) { // ERRATO!
    int i = c;
    if (s < 0)
        i = a;
    else if (s == 0)
        i = b;
    return i;
// ATTENZIONE!
// Ritorna un riferimento ad una variabile locale
}
```

### 8.3 Riferimenti (VII)

// Trasmissione di parametri mediante riferimenti a  
// costanti

```
#include <iostream.h>

int& selc(int s, const int& a, const int& b, const int& c) {
    if (s < 0)
        return (int&)a;           // conversione esplicita
    if (s == 0)
        return (int&)b;
    return (int&)c;
}

void main () {
    int sl, h = 1, k = 10;
    const int N = 100;
    cin >> sl;
    int& p = selc(sl, h, k, N); // OK
    cout << p << '\n';
}
```

### 8.3 Riferimenti (VIII)

// Riferimenti a costanti come valori di ritorno

```
#include <iostream.h>

const int& cselc(int s, const int& a, const int& b,
                  const int& c) {
    if (s < 0)
        return a;
    if (s == 0)
        return b;
    return c;
}

void main () {
    int sl, h = 1, k = 10;
    const int N = 100;
    cin >> sl;
    const int& p = cselc(sl, h, k, N);      // OK
    cout << p << '\n';

    cselc(sl, h, k, N) = 0;                  // ERRORE!
}
```

### 9.1 Tipi e oggetti array (I)

// Somma gli elementi di un dato vettore di interi

```
#include <iostream.h>
```

```
void main() {
    const int N = 5;
    int v[N];
    // dimensione del vettore: espressione costante
    // (calcolabile a tempo di compilazione)
```

```
    for (int i = 0; i < N; i++)
        cin >> v[i];
```

```
    int s = v[0];
    for (i = 1; i < N; i++)
        s += v[i];
```

```
    cout << s << '\n';
}
```

```
//~~~~~
// Nessun controllo sul valore degli indici
```

```
#include <iostream.h>
```

```
void main() {
    int a[] = {0, 1, 2, 3, 4};
    for (int i = 0; i <= 5; i++)          // ERRORE!
        cout << a[i] << '\n';
}
```

## 9.1 Tipi e oggetti array (II)

// Copia tra vettori

```
#include <iostream.h>

void main() {
    const int N = 5;
    int u[N], v[N];

    for (int i = 0; i < N; i++)
        cin >> u[i];

    for (i = 0; i < N; i++)
        v[i] = u[i];

    cout << '[' << N << "] <" << u[0];
    for (i = 1; i < N; i++)
        cout << ", " << u[i];
    cout << ">\n";

    cout << '[' << N << "] <" << v[0];
    for (i = 1; i < N; i++)
        cout << ", " << v[i];
    cout << ">\n";
}
```

## 9.1 Tipi e oggetti array (III)

// Inverte l'ordine degli elementi di un vettore dato

```
#include <iostream.h>
```

```
void scambia(int& a, int& b) {
    int c = a;
    a = b;
    b = c;
}
```

```
void main() {
    const int MAX = 10;
    int v[MAX];
    int nElem;
    cout << "Quanti elementi? ";
    cin >> nElem;
    cout << "Elementi? ";
    for (int i = 0; i < nElem; i++)
        cin >> v[i];
```

```
for (int j = 0, k = nElem - 1; j < k; j++, k--)
    scambia(v[j], v[k]);
```

```
cout << '[' << nElem << "] <" << v[0];
for (i = 1; i < nElem; i++)
    cout << ", " << v[i];
cout << ">\n";
```

## 9.1 Tipi e oggetti array (IV)

// Inizializzazione di vettori

```
#include <iostream.h>

void main() {
    int giorniDelMese[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}; // anche: [12]
    cout << sizeof(int) << '\n'; // 4 (byte)
    cout << sizeof(giorniDelMese) << '\n'; // 48 (byte)
    cout << sizeof(giorniDelMese) / sizeof(int) << '\n';
    // 12 (elementi)

    int a[] = {1, 2};
    cout << sizeof(a) / sizeof(int) << '\n'; // 2 (elementi)

    int b[5] = {1, 2};
    cout << sizeof(b) / sizeof(int) << '\n'; // 5 (elementi)
}
```

## 9.2 Array e puntatori (I)

// Aritmetica dei puntatori

```
#include <iostream.h>
```

```
void main () {
    int v[4];
    int* p = v; // v <=> &v[0]

    *p = 1;
    *(p + 1) = 10;
    p += 3;
    *(p - 1) = 100;
    *p = 1000;

    p = v;
    cout << '[' << 4 << "] <" << *p;
    for (int i = 1; i < 4; i++) // [4] <1, 10, 100, 1000>
        cout << ", " << *(p + i);
    cout << ">\n";

    cout << long(p + 1) - long(p) << '\n'; // 4 (byte)

    char c[5];
    char* q = c;
    cout << long(q + 1) - long(q) << '\n'; // 1 (byte)

    int* p1 = &v[1];
    int* p2 = &v[2];
    cout << p2 - p1 << '\n';
    cout << long(p2) - long(p1) << '\n'; // 8 (byte)
}
```

## 9.2 Array e puntatori (II)

```
// Inizializza a 1

#include <iostream.h>

void main () {
    const int N = 5;
    int v[N];

    int* p = v;
    while (p <= &v[N-1])
        *p++ = 1;                                // *(p++)

    p = v;
    cout << '[' << N << "] <" << *p;
    for (int i = 1; i < N; i++)
        cout << ", " << *(p + i);
    cout << ">\n";
}
```

## 9.2 Array e puntatori (III)

```
// Somma gli elementi di due vettori dati e scrive il
// vettore a somma maggiore (se le somme sono uguali
// scrive il primo vettore)

#include <iostream.h>

void main() {
    const int N = 5;
    int u[N], v[N];

    cout << "Elementi del primo vettore? ";
    int *p = u;
    for (int i = 0; i < N; i++)
        cin >> *(p + i);
    cout << "Elementi del secondo vettore? ";
    int *q = v;
    for (i = 0; i < N; i++)
        cin >> *(q + i);

    int su = u[0], sv = v[0];
    for (i = 1; i < N; i++) {
        su += *(p + i);
        sv += *(q + i);           // altra forma, che modifica q
    }

    int *r = su >= sv ? u : v;

    cout << '[' << N << "] <" << *r;
    for (i = 1; i < N; i++)
        cout << ", " << *(r + i);
    cout << ">\n";
}
```

## 9.2 Array e puntatori (IV)

// L'operatore [] è commutativo

```
#include <iostream.h>
```

```
void main () {
    int v[] = {1, 2, 3, 4, 5};

    const int N = sizeof v / sizeof (int);

    for (int i = 0; i < N; i++) {
        cout << v[i] << '\t';
        cout << *(v + i) << '\t';
        cout << *(i + v) << '\t';
        cout << i[v] << '\n';
    }
}
```

## 9.3 Array multidimensionali (I)

// Legge e scrive gli elementi di una matrice di R righe  
// e C colonne

```
#include <iostream.h>
```

```
void main () {
    const int R = 2;
    const int C = 3;
    int m[R][C];

    for (int i = 0; i < R; i++)
        for (int j = 0; j < C; j++)
            cin >> m[i][j];

    int* p = &m[0][0];           // anche: m[0]

    for (i = 0; i < R; i++) {    // memorizzazione per righe
        for (int j = 0; j < C; j++)
            cout << *(p + i*C + j) << '\t';
        cout << '\n';
    }
}
```

### 9.3 Array multidimensionali (II)

// Inizializzazione di vettori multidimensionali

```
#include <iostream.h>

void main () {
    int m1[2][3] = {
        1, 2, 3,
        4, 5, 6
    };

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++)
            cout << m1[i][j] << '\t';
        cout << '\n';
    }
    cout << '\n';

    int m2[3][3] = {           // anche: int m[][3]
        {0, 1, 2},
        {10, 11, 13},
        {100, 101, 102}
    };
    /* ... */
}
```

### 9.4 Stringhe (I)

// Lunghezza ed inizializzazione di stringhe

```
#include <iostream.h>
#include <string.h>

void main() {
    char c1[] = "C++";
    cout << sizeof c1 << '\n';      // 4
    cout << strlen(c1) << '\n';     // 3

    char c2[] = {'C', '+', '+'};    // Manca il '\0'!
    cout << sizeof c2 << '\n';     // 3

    char c3[] = {'C', '+', '+', '\0'}; // OK
    cout << sizeof c3 << '\n';     // 4

    char c3[4];
    c3 = "C++";                   // ERRORE!
}

//~~~~~
#include <iostream.h>

void main() {
    char stringa[10];    // al piu` 9 caratteri oltre a '\0'
    cout << "? ";
    cin >> stringa;       // Esempio: qwerty asd
    cout << stringa << '\n'; // Esempio: qwerty
}
```

## 9.4 Stringhe (II)

```
// Stringhe e puntatori

#include <iostream.h>

void main() {
    char s1[] = "Pippo";
    char s2[] = {'p','o','i','\0'};
    char* s3 = "ieri";
    char* s4 = "domani";

    cout << s1 << '\n';           // Pippo
    cout << s2 << '\n';           // poi
    cout << s3 << '\n';           // ieri
    cout << s4 << '\n';           // domani

    s2 = s3;
    // ERRORE! Conversione da char* a char[4]

    s4 = s3;

    cout << s3 << '\n';           // ieri
    cout << s4 << '\n';           // ieri

    char *const s5 = "oggi";
    char *const s6 = "domani";

    s6 = s5;
    // ERRORE! Oggetto costante come valore sinistro

    cout << (void*)s3 << '\n';     // Es.: 0x00419730
}
```

## 9.4 Stringhe (III)

```
// Conta le occorrenze di ciascuna lettera in una stringa
// data

#include <iostream.h>

void main() {
    const int LETTERE = 26;
    char str[100];
    int conta[LETTERE];
    for (int i = 0; i < LETTERE; ++i)
        conta[i] = 0;
    cout << "? ";
    cin >> str;

    for (i = 0; str[i] != '\0'; ++i)
        if (str[i] >= 'a' && str[i] <= 'z')
            ++conta[str[i] - 'a'];
        else if (str[i] >= 'A' && str[i] <= 'Z')
            ++conta[str[i] - 'A'];

    for (i = 0; i < LETTERE; ++i)
        cout << char('a' + i) << ":" << conta[i] << '\n';
}
```

## 9.5 Array come argomenti di funzioni (I)

```
// Scrive gli elementi di un dato vettore di interi (i)  
  
#include <iostream.h>  
  
void scrivi(int v[5]) {          // anche: (int v[])  
    // X v[] <=> X *const v  
    cout << '[' << v[0];  
    for (int i = 1; i < 5; i++)  
        cout << ", " << v[i];  
    cout << ']';  
}  
  
void main () {  
    int vett[] = {10, 11, 12, 11, 10};  
    scrivi(vett);                // [5] < 10, 11, 12, 11, 10 >  
    cout << '\n';  
}
```

## 9.5 Array come argomenti di funzioni (II)

```
// Scrive gli elementi di un dato vettore di interi (ii)  
// (ERRATO)  
  
#include <iostream.h>  
  
void scrivi(int v[]) {  
    const int n = sizeof v;           // ERRORE! 4  
    cout << '[' << v[0];  
    for (int i = 1; i < n; i++)  
        cout << ", " << v[i];  
    cout << ']';  
}  
  
void main () {  
    int vett[] = {10, 11, 12, 11, 10};  
    scrivi(vett);                  // [4] < 10, 11, 12, 11>  
    cout << '\n';  
}
```

## 9.5 Array come argomenti di funzioni (III)

```
#include <iostream.h>

void incrementa(int v[], int n) {
    for (int i = 0; i < n; i++)
        v[i]++;
}

void decrementa(int v[], int n) {
    for (int i = 0; i < n; i++)
        (*(v + i))--;
}

void leggi(int v[], int n) {
    for (int i = 0; i < n; i++)
        cin >> v[i];
}

void scrivi(int v[], int n) {
    cout << '[' << n << "] <" << *v;
    for (int i = 1; i < n; i++)
        cout << ", " << *(v + i);
    cout << '>';
}
```

(continua ...)

## 9.5 Array come argomenti di funzioni (IV)

```
void main () {
    const int MAX = 10;
    int vett[MAX] = {0};
    int nElem;

    cin >> nElem;
    if (nElem > MAX)
        nElem = MAX;
    leggi(vett, nElem);
    incrementa(vett, nElem);
    scrivi(vett, MAX);
    cout << '\n';

    cin >> nElem;
    if (nElem > MAX)
        nElem = MAX;
    leggi(vett, nElem);
    decrementa(vett, nElem);
    scrivi(vett, MAX);
    cout << '\n';
}
```

(... fine)

### 9.5.1 Argomenti array costanti (I)

// Somma gli elementi di un dato vettore di interi (i)  
// (soluzione ricorsiva)

```
#include <iostream.h>

int somma(const int v[], int n) {
    if (n == 0)
        return v[0];
    return v[n] + somma(v, n-1);
}

void leggi(int v[], int n) {
    for (int i = 0; i < n; i++)
        cin >> v[i];
}

void main() {
    const int MAX = 10;
    int v[MAX], nElem;

    cout << "Quanti elementi? ";
    cin >> nElem;
    cout << "Elementi? ";
    leggi(v, nElem);

    for (int i = 1; i < nElem; i++) {
        cout << "Somma dei primi " << i + 1;
        cout << " elementi: " << somma(v, i) << '\n';
    }
}
```

### 9.5.1 Argomenti array costanti (II)

// Somma gli elementi di un dato vettore di interi (ii)  
// (soluzione ricorsiva)

```
int somma(const int *const v, int n) {
    if (n == 0)
        return *v;
    return *(v + n) + somma(v, n-1);
}

//~~~~~
// ERRATA!

void errata(const int v[]) {
    v[3] = 0; // ERRORE!
}
```

### 9.5.1 Argomenti array costanti (III)

```
// Trova il massimo valore in un dato vettore di interi

#include <iostream.h>

int massimo(const int v[], int n) {
    int m = v[0];
    for (int i = 1; i < n; i++)
        m = m >= v[i] ? m : v[i];
    return m;
}

void leggi(int v[], int n) {
    for (int i = 0; i < n; i++)
        cin >> v[i];
}

void main() {
    const MAX = 10; int v[MAX];
    int nElem;
    cout << "Quanti elementi? ";
    cin >> nElem;
    cout << "Elementi? ";
    leggi(v, nElem);
    cout << "Massimo: " << massimo(v, nElem) << '\n';
}
```

### 9.5.1 Argomenti array costanti (IV)

```
// Ordina un dato vettore di interi in ordine crescente

#include <iostream.h>

void scambia(int& a, int& b) {
    int c = a;
    a = b;
    b = c;
}

void leggi(int v[], int n) {
    for (int i = 0; i < n; i++)
        cin >> v[i];
}

void scrivi(const int v[], int n) {
    cout << '[' << n << "] <" << v[0];
    for (int i = 1; i < n; i++)
        cout << ", " << v[i];
    cout << '>';
}

void bubble(int v[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = i + 1; j < n; j++)
            if (v[i] > v[j])
                scambia(v[i], v[j]);
}
```

(continua ...)

### 9.5.1 Argomenti array costanti (V)

```
void main() {  
    const int MAX = 10;  
    int v[MAX], nElem;  
    cout << "Quanti elementi? ";  
    cin >> nElem;  
    cout << "Elementi? ";  
    leggi(v, nElem);  
    bubble(v, nElem);  
    scrivi(v, nElem);  
    cout << '\n';  
}
```

(... fine)

### 9.5.2 Array multidimensionali (I)

```
// La dichiarazione di un vettore a piu` dimensioni come  
// argomento formale deve specificare la grandezza di  
// tutte le dimensioni tranne la prima  
  
#include <iostream.h>  
  
const int C = 3;  
  
void riempi(int m[][C], int r) {  
    for (int i = 0; i < r; ++i)  
        for (int j = 0; j < 3; ++j)  
            m[i][j] = i + j;  
}  
  
void riempiErrata(int m[][]); // ERRORE!  
  
void scrivi(int m[][C], int r) {  
    for (int i = 0; i < r; ++i) {  
        for (int j = 0; j < 3; ++j)  
            cout << m[i][j] << '\t';  
        cout << '\n';  
    }  
}  
  
void main () {  
    int mat[6][3];  
    riempi(mat, 6);  
    scrivi(mat, 6);  
}
```

## 9.5.2 Array multidimensionali (II)

// Trasmissione mediante puntatori

```
#include <iostream.h>

void riempi(int* m, int r, int c) {      // r righe, c colonne
    for (int i = 0; i < r; ++i)
        for (int j = 0; j < c; ++j)
            *(m + i * c + j) = i + j;
}

void scrivi(int* m, int r, int c) {
    for (int i = 0; i < r; ++i) {
        for (int j = 0; j < c; ++j)
            cout << *(m + i * c + j) << '\t';
        cout << '\n';
    }
}

void main () {
    int mat[6][3];
    riempi(&mat[0][0], 6, 3);      // anche: mat[0]
    scrivi((int*)mat, 6, 3);      // conversione esplicita
}
```

## 9.6 Funzioni di libreria per le stringhe

(in `<string.h>`)

- `strcpy(s, t)`
- `strcat(s, t)`
- `strcmp(s, t)`
- `strchr(s, c)`
- `strrchr(s, c)`
- `strlen(s)`

## 10.1 Strutture (I)

```
// Poligono regolare

#include <iostream.h>

struct PolReg {
    int nLati;
    double lunghLato;
};

void main () {
    PolReg r, s;

    r.nLati = 3;           // un triangolo
    r.lunghLato = 10.5;

    s.nLati = r.nLati;
    s.lunghLato = r.lunghLato + 10.0;

    cout << '<' << r.nLati << ", " << r.lunghLato << ">\n";
    cout << '<' << s.nLati << ", " << s.lunghLato << ">\n";

    PolReg *p = &r;
    cout << '<' << p->nLati << ", ";      // (*p).nLati
    cout << p->lunghLato << ">\n";

    PolReg t = {3, 10};           // inizializzazione
    cout << '<' << t.nLati << ", " << t.lunghLato << ">\n";
}
```

## 10.1 Strutture (II)

### // ERRATA!

```
struct Elemento {
    /* ... */
    Elemento elem;           // ERRORE!
};

//-----//
// Struttura contenente un riferimento a se stessa

struct Elemento {
    /* ... */
    Elemento* pElem;         // OK
};

//-----//
// Dichiarazioni incomplete

struct Parte;

struct Componente {
    /* ... */
    Parte* p;
};

struct Parte {
    /* ... */
    Componente* c;
};
```

## 10.1 Strutture (III)

```
struct PolReg {  
    int nLati;  
    double lunghLato;  
};  
  
const int MAX = 30;  
  
void main () {  
    struct Figura {  
        int quanti;  
        PolReg pp[MAX];  
    } f;  
  
    f.quanti = 3;  
  
    f.pp[0].nLati = 3;           // un triangolo  
    f.pp[0].lunghLato = 1.0;  
  
    f.pp[1].nLati = 4;           // un quadrato  
    f.pp[1].lunghLato = 10.0;  
  
    f.pp[2].nLati = 3;           // un triangolo  
    f.pp[2].lunghLato = 100.0;  
  
    /* ... */  
}
```

## 10.1.1 Operazioni sulle strutture (I)

```
// Assegnamento tra strutture  
  
#include <iostream.h>  
  
struct PolReg {  
    int nLati;  
    double lunghLato;  
};  
  
void main () {  
    PolReg r1 = {3, 10};      // un triangolo  
    PolReg r2;  
  
    r2 = r1;                  // copia membro a membro  
  
    cout << '<' << r1.nLati << ", " << r1.lunghLato << ">\n";  
                                // <3, 10>  
    cout << '<' << r2.nLati << ", " << r2.lunghLato << ">\n";  
                                // <3, 10>  
}
```

### 10.1.1 Operazioni sulle strutture (II)

```
// Strutture come argomenti di funzioni e restituite da
// funzioni

#include <iostream.h>

struct PolReg {
    int nLati;
    double lunghLato;
};

double perimetro(PolReg pr) {
    return pr.nLati * pr.lunghLato;
}

PolReg simile(PolReg pr, double scala) {
    PolReg ris = {pr.nLati, pr.lunghLato * scala};
    return ris;
}

void main () {
    PolReg r1 = {3, 10.0};
    PolReg r2 = simile(r1, 5.0);

    cout << '<' << r1.nLati << ", ";
    cout << r1.lunghLato << ">\n";           // <3, 10>

    cout << '<' << r2.nLati << ", ";
    cout << r2.lunghLato << ">\n";           // <3, 50>

    cout << perimetro(r1) << '\n';          // 30
    cout << perimetro(r2) << '\n';
}
```

### 10.1.1 Operazioni sulle strutture (III)

```
// Copia di vettori

#include <iostream.h>

const int N = 3;

struct Vettore {
    int vv[N];
};

void main () {
    Vettore vett1 = {1, 10, 100}, vett2;

    vett2 = vett1;

    cout << '[' << N << "] <" << vett1.vv[0];
    for (int i = 1; i < N; i++)           // [3] <1, 10, 100>
        cout << ", " << vett1.vv[i];
    cout << ">\n";

    cout << '[' << N << "] <" << vett2.vv[0];
    for (i = 1; i < N; i++)           // [3] <1, 10, 100>
        cout << ", " << vett2.vv[i];
    cout << ">\n";
}
```

### 10.1.1 Operazioni sulle strutture (IV)

// Trasmissione di vettori per valore

```
#include <iostream.h>

struct Vettore {
    int vv[3];
};

void f(Vettore e) {
    /*...*/
    e.vv[1] = 0;
    /*...*/
}

void main () {
    Vettore vett;
    /* ... */
    vett.vv[1] = 10;
    f(vett);
    cout << vett.vv[1] << '\n';           // 10
    /* ... */
}
```

### 10.2 Unioni

```
#include <iostream.h>
```

```
union Uni {
    char c;
    int i;
};

struct Str {
    char c;
    int i;
};

void main() {
    cout << sizeof(char) << '\t' << sizeof(int) << '\n'; // 1 4
    cout << sizeof(Uni) << '\t' << sizeof(Str) << '\n'; // 4 8

    Uni u = {'a'};
    u.i = 10024;
    cout << u.c << '\t' << u.i << '\n';                  // ( 10024

    Str s = {'a', 10024};
    cout << s.c << '\t' << s.i << '\n';
}
```

## 10.2.2 Strutture con varianti (I)

```
#include <iostream.h>

enum Colore {ROSSO, GIALLO, VERDE};
struct Valore {
    int invariante;
    Colore discriminante;
    union {
        char c;
        int v[3];
        double d;
    } variante;
};

void leggi(Valore& x) {
    char col; cout << "Colore? "; cin >> col;
    cout << "Invariante? "; cin >> x.invariante;
    cout << "Variante? ";
    switch (col) {
        case 'r': case 'R':
            x.discriminante = ROSSO; cin >> x.variante.c;
            break;
        case 'g': case 'G':
            x.discriminante = GIALLO;
            for (int j = 0; j < 3; j++) cin >> x.variante.v[j];
            break;
        case 'v': case 'V':
            x.discriminante = VERDE; cin >> x.variante.d;
    }
}
```

(continua ...)

## 10.2.2 Strutture con varianti (II)

```
void scrivi(Valore x) {
    cout << '[' << x.invariante << ", ";
    switch (x.discriminante) {
        case ROSSO: {
            cout << x.variante.c;
            break;
        }
        case GIALLO: {
            cout << '<' << x.variante.v[0];
            for (int j = 1; j < 3; j++)
                cout << ", " << x.variante.v[j];
            cout << '>';
            break;
        }
        case VERDE: {
            cout << x.variante.d;
        }
    }
    cout << ']';
}

void main() {
    Valore w;
    leggi(w);
    scrivi(w);
    cout << '\n';
}
```

(... fine)

## 10.2.2 Strutture con varianti (III)

// Unioni anonne

```
#include <iostream.h>

enum Colore {ROSSO, GIALLO, VERDE};
struct Valore {
    int invariante;
    Colore discriminante;
    union {                                // parte variante
        char c;
        int v[3];
        double d;
    };
};

void leggi(Valore& x) {
    char col; cout << "Colore? "; cin >> col;
    cout << "Invariante? "; cin >> x.invariante;
    cout << "Variante? ";
    switch (col) {
        case 'r': case 'R':
            x.discriminante = ROSSO; cin >> x.c;
            break;
        case 'g': case 'G':
            x.discriminante = GIALLO;
            for (int j = 0; j < 3; j++) cin >> x.v[j];
            break;
        case 'v': case 'V':
            x.discriminante = VERDE; cin >> x.d;
    }
}
/* ... */
```

## 11.2 Puntatori a funzioni (I)

#include <iostream.h>

```
int f1(int i) {
    int v[] = {1, 10, 100, 1000, 1000, 100, 10, 1, 1, 10, 100};
    if (i < 0)
        return v[0];
    if (i > 10)
        return v[10];
    return v[i];
}

int f2(int i) {
    int v[] = {0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512};
    if (i < 0)
        return v[0];
    if (i > 10)
        return v[10];
    return v[i];
}

int sommatoria(int f(int), int inf, int sup) {
    for (int i = inf, s = 0; i <= sup; i++)
        s += f(i);
    return s;
}
```

(continua ...)

## 11.2 Puntatori a funzioni (II)

```
void main () {  
    int a, b;  
    cin >> a >> b;  
    for (int i = a; i <= b; i++) {  
        cout << sommatoria(f1, a, i) << '\t';  
        cout << sommatoria(f2, a, i) << '\n';  
  
    int n;  
    cin >> n;  
  
    int (*p)(int);  
    p = f1;  
    cout << p(n) << '\n'; // (*p)(n)  
  
    p = f2;  
    cout << p(n) << '\n';  
}  
}
```

(... fine)

## 11.3 Argomenti default (I)

```
#include <iostream.h>  
  
double perimetro(int nLati = 3, double lunghLato = 1.0) {  
    return nLati * lunghLato;  
}  
  
double area(int nLati = 3, double lunghLato = 1.0,  
            double precisione = 0.01) {  
    /* ... */  
}  
  
void main () {  
    double perim1 = perimetro(4, 2.5);  
    // quadrato di lato 2.5  
    double perim2 = perimetro(3);  
    // triangolo di lato 1.0  
    double perim3 = perimetro();  
    // triangolo di lato 1.0;  
    double perim4 = perimetro(4.1);  
    // ATTENZIONE! Quadrato di lato 1.0  
    /* ... */  
}  
  
//~~~~~  
  
double perimetro(int nLati = 3, double lunghLato) {  
    // ERRORE!  
    return nLati * lunghLato;  
}
```

### 11.3 Argomenti default (II)

```
#include <iostream.h>

double perimetro(int nLati = 3, double lunghLato = 1.0) {
    return nLati * lunghLato;
}

double area(int nLati, double lunghLato,
            double precisione) {
    /* ... */
}

void f() {
    double p = perimetro();           // OK
    double a = area();                // ERRORE!
    cout << p << '\t' << a << '\n';
}

double area(int nLati = 3, double lungh = 1.0,
            double precisione = 0.01);

void g() {
    double p = perimetro();
    double a = area();               // OK
    cout << p << '\t' << a << '\n';
}

void main () {
    /* ... */
}
```

### 11.4 Overloading (I)

```
#include <iostream.h>

int max(int a, int b) {
    return a > b ? a : b;
}

double max(double a, double b) {
    return a > b ? a : b;
}

int max(double a, double b)           // ERRORE!
{
    return int(a > b ? a : b);
}

void main () {
    int h, k;
    cin >> h >> k;
    cout << max(h, k) << '\n';

    double d, e;
    cin >> d >> e;
    cout << max(d, e) << '\n';
}
```

## 11.4 Overloading (II)

```
// Sovrapposizione const - non const

#include <iostream.h>

int massimo(const int v[], int n) {
// Trova il valore massimo
    int m = v[0];
    for (int i = 1; i < n; i++)
        m = m >= v[i] ? m : v[i];
    return m;
}

int massimo(int v[], int n) {
// Azzera il primo elemento a valore massimo
    int m, im = 0;
    for (int i = 1; i < n; i++)
        if (v[im] < v[i])
            im = i;
    m = v[im];
    v[im] = 0;
    return m;
}
```

(continua ...)

## 11.4 Overloading (III)

```
void leggi(int v[], int n) {
    for (int i = 0; i < n; i++)
        cin >> v[i];
}

void scrivi(const int v[], int n) {
    cout << '[' << n << "] <" << v[0];
    for (int i = 1; i < n; i++)
        cout << ", " << v[i];
    cout << '>';
}

void main() {
    const int N = 5;
    const int cv[N] = {1, 10, 100, 10, 1};
    cout << massimo(cv, N) << '\n';
    scrivi(cv, N);
    cout << '\n';

    int v[N];
    leggi(v, N);
    cout << massimo(v, N) << '\n';
    scrivi(v, N);
    cout << '\n';
}
```

(... fine)

## 11.5 Funzioni inline

```
// Valore assoluto

#include <iostream.h>

inline int abs(int n) {
    return n >= 0 ? n : -n;
}

void main() {
    int a;
    cout << "? ";
    cin >> a;
    cout << abs(a) << '\n';
}
```

## 12.4 Dichiarazioni typedef (I)

```
#include <iostream.h>

void main () {
    int i = 1;

    typedef int* intP;
    intP p = &i;
    cout << *p << '\n';                                // 1

    typedef int& intR;
    intR r = i;
    r++;
    cout << i << '\n';                                // 2

    typedef int vett[5];
    vett v = {1, 10, 100, 10, 1};
    cout << '<' << v[0];
    for (int j = 1; j < 5; j++)                      // <1, 10, 100, 10, 1>
        cout << ", " << v[j];
    cout << '>\n';

    typedef int intero;
    int a = 1;
    intero b = a;
    // OK, typedef non introduce un nuovo tipo
    cout << a << 't' << b << '\n';                  // 1 1
}
```

## 12.4 Dichiarazioni `typedef` (II)

// Puntatori a funzione

```
#include <iostream.h>

int f1(int i) {
    int v[] = {1, 10, 100, 1000, 1000, 100, 10, 1, 1, 10, 100};
    if (i < 0)
        return v[0];
    if (i > 10)
        return v[10];
    return v[i];
}

void main () {
    typedef int (*pf)(int );
    pf q = f1;
    cout << q(3) << '\n';                                // 1000
}
```

## 13.1 Operatori `new` e `delete` (I)

// Operatore `new`

```
#include <iostream.h>
```

```
void main() {
    int* h = new int;
    *h = 10;
    cout << *h << '\n';                                // 10

    const int N = 5;
    int* p = new int[N];
    for (int i = 0; i < N; i++)
        *(p + i) = i;

    cout << '[' << N << "] <" << *p;
    for (i = 1; i < N; i++)                            // [5] <0, 1, 2, 3, 4, 5>
        cout << ", " << *(p + i);
    cout << ">\n";
}
```

## 13.1 Operatori new e delete (II)

// Operatore delete

```
#include <iostream.h>
```

```
void main () {
    int* p = new int;
    *p = 10;
    /*...*/
    delete p;

    int* q = new int [10];
    /*...*/
    delete [] q;

    int i = 0;
    int* pi = &i;
    delete p;                                // ERRORE!
}
```

## 13.1 Operatori new e delete (III)

// Vettori dinamici

```
#include <iostream.h>
```

```
void main() {
    int nElem;
    cin >> nElem;                                // Esempio: 5

    int* p = new int[nElem];
    for (int i = 0; i < nElem; i++)
        p[i] = i;

    cout << '[' << nElem << "] <" << *p;
    for (i = 1; i < nElem; i++)
        cout << ", " << p[i];
    cout << ">\n";                                // Esempio: [5] <0, 1, 2, 3, 4>
}
```

## 13.1 Operatori new e delete (IV)

// Superamento dei limiti di memoria (i)

```
#include <iostream.h>
#include <stdlib.h>

void main () {
    for (int i = 0; i < 1024; i++) {
        char* p = new char[1024 * 1024];      // 1 Mbyte
        cout << i << endl;
        if (p == NULL) {
            cerr << "Memoria libera esaurita!" << endl;
            exit(1);                      // dichiarata in <stdlib.h>
        }
    }
}
```

## 13.1 Operatori new e delete (V)

// Superamento dei limiti di memoria (ii)

```
#include <iostream.h>
#include <new.h>
#include <stdlib.h>

int memoriaEsaurita(size_t size) {
    cerr << "Memoria libera esaurita! Richiesti ";
    cerr << size << " bytes." << endl;
    exit (1);                  // dichiarata in <stdlib.h>
// nessun tentativo di garbage collection
    return 0;
}

void main () {
    _set_new_handler(memoriaEsaurita);
// dichiarata in <new.h>

    for (int i = 0; i < 1024; i++) {
        char* p = new char[1024 * 1024];      // 1 Mbyte
        cout << i << endl;
    }
}
```

## 13.2 Liste semplici (I)

```
#include <iostream.h>

struct elem {
    int info;
    elem* pun;
};

typedef elem* Lista;

void inizializza(Lista& s) {
    s = NULL;
}

void inserisciInTesta(Lista& s, int val) {
// creo il nuovo elemento
    elem* q = new elem;
    q->info = val;

// inserisco in testa
    q->pun = s;
    s = q;
}
```

(continua ...)

## 13.2 Liste semplici (II)

```
// Inserisce un elemento a valore val nella posizione pos
// della lista s. Se pos e` minore o uguale ad 1, inserisce
// in testa. Se pos e` maggiore del numero degli elementi
// della lista, inserisce in fondo.

void inserisciPerPosizione(Lista& s, int val, int pos) {
// creo il nuovo elemento
    elem* q = new elem;
    q->info = val;

    if (pos <= 1 || s == NULL) {
// inserisco in testa
        q->pun = s;
        s = q;
        return;
    }

// sposto p sull'elemento dopo del quale effettuerò
// l'inserzione
    elem* p = s;
    for (int j = 1; j <= pos - 2 && p->pun != NULL; j++)
        p = p->pun;

// inserisco
    q->pun = p->pun;
    p->pun = q;
}
```

(... continua ...)

## 13.2 Liste semplici (III)

```
int estraiInTesta(Lista& s, int& val) {  
    if (s == NULL) return 0;           // lista vuota  
  
    // inserisco in testa  
    elem* q = s;  
    val = s->info;  
    s = s->pun;  
    delete q;  
    return 1;  
}
```

(... continua ...)

## 13.2 Liste semplici (IV)

```
// Estraе l'elemento nella posizione pos della lista s e ne  
// ritorna il valore val. Se pos e` minore o uguale ad 1,  
// estraе il primo elemento. Se pos e` maggiore del  
// numero degli elementi della lista, estraе l'ultimo.
```

```
int estraiPerPosizione(Lista& s, int& val, int pos) {  
    if (s == NULL) return 0;           // lista vuota
```

```
    if (pos <= 1 || s->pun == NULL) {  
        // estraggo il primo  
        elem* q = s;  
        val = s->info;  
        s = s->pun;  
        delete q;  
        return 1;  
    }
```

```
// sposto q sull'elemento da estrarre, e p sull'elemento  
// precedente
```

```
    elem *p = s, *q = s->pun;  
    for (int j = 1; j <= pos - 2 && q->pun != NULL; j++) {  
        p = p->pun;  
        q = q->pun;  
    }
```

```
// estraggo
```

```
    val = q->info;  
    p->pun = q->pun;  
    delete q;  
    return 1;
```

```
}
```

(... continua ...)

## 13.2 Liste semplici (V)

```
void distruggi(Lista& s) {
    elem *q;
    while (s != NULL) {
        q = s->pun;
        delete s;
        s = q;
    }
}

void visualizza(Lista s) {
    cout << '<';
    // lista vuota
    if (s == NULL) {
        cout << '>';
        return;
    }

    elem* q = s;
    cout << q->info;
    q = q->pun;
    while (q != NULL) {
        cout << ", ";
        cout << q->info;
        q = q->pun;
    }
    cout << '>';
}
```

(... continua ...)

## 13.2 Liste semplici (VI)

```
void main() {
    char comando;
    Lista m; inizializza(m);
    for (;;) {
        cout << "? "; cin >> comando;
        if (comando == '~') break;      // per terminare
        int val, pos, ok;
        switch (comando) {
            case 't':
                cin >> val;
                inserisciInTesta(m, val);
                break;
            case 'p':
                cin >> pos >> val;
                inserisciPerPosizione(m, val, pos);
                break;
            case 'T':
                ok = estraiInTesta(m, val);
                if (ok) cout << "Estratto " << val << '\n';
                else cout << "Estrazione fallita\n";
                break;
            case 'P':
                cin >> pos;
                ok = estraiPerPosizione(m, val, pos);
                if (ok) cout << "Estratto " << val << '\n';
                else cout << "Estrazione fallita\n";
        }
        visualizza(m); cout << '\n';
    }
    distruggi(m);
}
```

(... fine)

## 14.2.1 Blocchi

```
#include <iostream.h>

void f() {
    int i = 2;
    cout << i << '\n';                                // 2
}

void main() {
    cout << i << '\n';
// ERRORE! Identificatore non dichiarato
    int i = 1, j = 5;
    cout << i << '\n';                                // 1
    cout << j << '\n';                                // 5
{
    cout << i << '\n';                                // 1
    cout << j << '\n';                                // 5
    int i = 10;
    cout << i << '\n';                                // 10
}
cout << i << '\n';                                // 1

f();
cout << i << '\n';                                // 1

for (int a = 0; a < 10; a++) {
    int b = 2 * a;
    cout << a << '\t' << b << '\n';    // 0 0 ... 9 18
}
cout << a << '\n';                                // 10
cout << b << '\n';
// ERRORE! Identificatore non dichiarato
}
```

## 14.2.2 Unità di compilazione (I)

```
#include <iostream.h>

int i;
// visibilità a livello di file

void leggi() {
    cin >> i;
}

void scrivi() {
    cout << i;
}

void main() {
    leggi();
    scrivi();
    cout << '\n';
}
```

## 14.2.2 Unità di compilazione (II)

// Operatore :: unario (risolutore globale)

```
#include <iostream.h>

int i = 1; // visibilità a livello di file

void main() {
    cout << i << '\n'; // 1
    {
        int i = 10; // visibilità locale
        cout << ::i << '\n'; // 1
        cout << i << '\n'; // 10
        {
            int i = 100; // visibilità locale
            cout << ::i << '\n'; // 1
            cout << i << '\n'; // 100
        }
        cout << ::i << '\n'; // 1
        cout << i << '\n'; // 10
    }
}
```

## 14.2.3 Collegamento (I)

// ----- file AltroFile.cpp ----- //

```
int a = 1;
// collegamento esterno (visibilità a livello di file)

const int N = 0;
// const, collegamento interno (visibilità a livello di file)

static int b = 10;
// static, collegamento interno (visibilità a livello di file)

void f1(
// collegamento esterno (visibilità a livello di file)
    int a) { // collegamento interno (visibilità locale)
    int k; // collegamento interno (visibilità locale)
    /* ... */
}

static void f2() {
// static, collegamento interno (visibilità a livello di file)
/* ... */
}

inline void f3() {
// inline, collegamento interno (visibilità a livello di file)
/* ... */
}
```

(continua ...)

### 14.2.3 Collegamento (II)

```
// ----- file Main.cpp ----- //

#include <iostream.h>

extern int a;           // solo dichiarazione
void f1(int);          // solo dichiarazione
void f2();              // solo dichiarazione
void f3();              // solo dichiarazione
double f4(double, double); // definizione mancante
                           // OK, non utilizzata

void main() {
    cout << a << '\n';           // OK, 1

    extern int b;               // dichiarazione
    cout << b << '\n';           // ERRORE!

    f1(a);                     // OK
    f2();                      // ERRORE!
    f3();                      // ERRORE!
    /* ... */
}
```

(... fine)

### 14.3 Classi di memorizzazioni

#### // Tempo di vita degli oggetti

```
#include <iostream.h>
```

```
int contaChiamateErrata() { // ERRATA!
    int n = 0;
    return ++n;
}

int contaChiamate() {      // di classe statica
    static int n = 0;
    return ++n;
}

void main() {
    for (int i = 0; i < 5; i++)
        cout << contaChiamateErrata() << '\n'; // 1 1 1 1 1

    for (i = 0; i < 5; i++)
        cout << contaChiamate() << '\n';      // 1 2 3 4 5
}
```

## 14.4 Effetti collaterali (I)

// Variabili globali

```
#include <iostream.h>

int a = 1, b = 10, c = 100;

int azzera(int s) {
    // Azzera a, b oppure c in base al valore negativo,
    // nullo o positivo di s
    int ris = c;
    if (s < 0) {
        ris = a;
        a = 0;
    }
    else if (s == 0) {
        ris = b;
        b = 0;
    }
    else
        c = 0;
    return ris;
}

void main() {
    int s;
    cin >> s;
    cout << azzera(s) << '\n';
    cout << a << '\t' << b << '\t' << c << '\n';
}
```

## 14.4 Effetti collaterali (II)

// Argomenti di tipo puntatore

```
#include <iostream.h>
```

```
int azzera(int s, int* pa, int* pb, int* pc) {
    int ris = *pc;
    if (s < 0) {
        ris = *pa;
        *pa = 0;
    }
    else if (s == 0) {
        ris = *pb;
        *pb = 0;
    }
    else
        *pc = 0;
    return ris;
}

void main() {
    int a = 1, b = 10, c = 100, s;
    cin >> s;
    cout << azzera(s, &a, &b, &c) << '\n';
    cout << a << '\t' << b << '\t' << c << '\n';
}
```

## 14.4 Effetti collaterali (III)

// Argomenti di tipo riferimento

```
#include <iostream.h>
```

```
int azzera(int s, int& ra, int& rb, int& rc) {
    int ris = rc;
    if (s < 0) {
        ris = ra;
        ra = 0;
    }
    else if (s == 0) {
        ris = rb;
        rb = 0;
    }
    else
        rc = 0;
    return ris;
}
```

```
void main() {
    int a = 1, b = 10, c = 100, s;
    cin >> s;
    cout << azzera(s, a, b, c) << '\n';
    cout << a << '\t' << b << '\t' << c << '\n';
}
```

## 14.4 Effetti collaterali (IV)

// Problema: proprieta` associativa

```
#include <iostream.h>
```

```
int azzera(int s, int& ra, int& rb, int& rc) {
    /* ... */
}

void main() {
    int a, b, c, r, s;

    cin >> s;                                // Esempio: 1

    a = 1; b = 10; c = 100;
    r = azzera(s, a, b, c) + azzera(s, a, b, c);
    cout << r << '\n';                      // Esempio: 100

    a = 1; b = 10; c = 100;
    r = 2 * azzera(s, a, b, c);
    cout << r << '\n';                      // Esempio: 200
}
```

## 14.4 Effetti collaterali (V)

// Problema: ordine di valutazione degli operandi

```
#include <iostream.h>

int azzera(int s, int& ra, int& rb, int& rc) {
    /* ... */
}

void main() {
    int a, b, c, r, s;

    cin >> s;                                // Esempio: 1

    a = 1; b = 10; c = 100;
    r = a + b + c + azzera(s, a, b, c);
    cout << r << '\n';                      // Esempio: 211

    a = 1; b = 10; c = 100;
    r = azzera(s, a, b, c) + a + b + c;
    cout << r << '\n';                      // Esempio: 111
}
```

## 14.4 Effetti collaterali (VI)

// Problema: ottimizzazione

```
#include <iostream.h>

int azzera(int s, int& ra, int& rb, int& rc) {
    /* ... */
}

void main() {
    int a, b, c, r1, r2, s;

    cin >> s;                                // Esempio: 1

    a = 1; b = 10; c = 100;
    r1 = a + b + c + azzera(s, a, b, c);
    r2 = a + b + c + azzera(s, a, b, c);
    cout << r1 << '\t' << r2 << '\n'; // Esempio: 211 11

    a = 1; b = 10; c = 100;
    r1 = a + b + c + azzera(s, a, b, c);
    r2 = r1;
    cout << r1 << '\t' << r2 << '\n'; // Esempio: 211 211
}
```

## 16.1 Limitazioni dei tipi derivati

### // Problema

```
#include <iostream.h>

const int N = 10;      // capacita` dello stack

struct Stack {
    int tt;          // indice del primo elemento vuoto
    int ee[N];        // elementi dello stack
};

inline void inizializza(Stack& s) {s.tt = 0;}
inline void push(Stack& s, int v) {s.ee[s.tt++] = v;}
inline int pop(Stack& s) {return s.ee[--s.tt];}
inline int top(Stack s) {return s.ee[s.tt - 1];}
inline int vuoto(Stack s) {return s.tt <= 0;}
inline int pieno(Stack s) {return s.tt >= N;}

void main() {
    Stack st;
    inizializza(st);
    /* ... */
    push(st, 1);
    /* ... */
    cout << top(st) << '\n';
    /* ... */
    st.tt = 10;           // ATTENZIONE!
    /* ... */
}
```

## 16.2 Classi (I)

### // Soluzione

```
#include <iostream.h>

const int N = 100;

class Stack {
    int tt;          // indice del primo elemento vuoto
    int ee[N];        // elementi dello stack
public:
    void inizializza() {tt = 0;}
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= N;}
};
```

(continua ...)

## 16.2 Classi (II)

```
void main() {
    char comando;
    Stack st;
    st.inizializza();
    for (;;) {
        cout << "? "; cin >> comando;
        if (comando == '~') break;      // ~ per terminare
        int val, pos;
        switch (comando) {
            case 'p':
                cin >> val;
                st.push(val);
                break;
            case 'o':
                val = st.pop();
                cout << val << '\n';
                break;
            case 't':
                val = st.top();
                cout << val << '\n';
                break;
            case 'v':
                if (st.vuoto()) cout << "Stack vuoto\n";
                else cout << "Stack non vuoto\n";
                break;
            case 'i':
                if (st.pieno()) cout << "Stack pieno\n";
                else cout << "Stack non pieno\n";
        }
    }
}
```

(... fine)

## 16.2 Classi (III)

### // Numeri complessi

```
#include <iostream.h>

class Complesso {
public:
    void inizializza(double r, double i) {re = r; im = i;}
    double pRe() {return re;}
    double plm() {return im;}
    /* ... */
    void scrivi() {cout << '(' << re << ", " << im << ')';}
private:
    double re, im;
};

void main() {
    Complesso c1, c2;
    c1.inizializza(1.0, -1.0);
    c1.scrivi(); cout << '\n';           // (1, -1)

    c2.inizializza(10.0, -10.0);
    c2.scrivi(); cout << '\n';          // (11, -11)

    Complesso* cp = &c1;
    cp->scrivi(); cout << '\n';          // (1, -1)
}
```

## 16.2 Classi (IV)

```
// Poligoni regolari (i)

#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    void inizializza(int n, double g) {
        nLati = n; lunghLato = g;}
    double perimetro() {return nLati * lunghLato;}
    double area() {/* ... */}
    void rendiSimile(double scala) {lunghLato *= scala;}
    int numeroLati() {return nLati;}
    double lunghezzaLato() {return lunghLato;}
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';
    }
};

void main () {
    PolReg r;
    r.inizializza(3, 1.0);           // un triangolo
    r.scrivi(); cout << '\t';       // <3, 1>
    cout << "Perimetro: " << r.perimetro() << '\n'; // 3

    PolReg* pr = new PolReg;
    pr->inizializza(4, 1.0);         // un quadrato
    pr->rendiSimile(5.0);
    cout << "Lunghezza di ogni lato: " <<      // 5
         pr->lunghezzaLato() << '\n';
}
```

## 16.2 Classi (V)

```
// Poligoni regolari (ii)
// (diversa rappresentazione interna)

#include <iostream.h>

class PolReg {
    double perim;
    int nLati;
public:
    void inizializza(int n, double g) {
        perim = g * n; nLati = n;}
    double perimetro() {return perim;}
    double area() {/* ... */}
    void rendiSimile(double scala) {perim *= scala;}
    int numeroLati() {return nLati;}
    double lunghezzaLato() {return perim / nLati;}
    void scrivi() {
        cout << '<' << nLati << ", " << perim / nLati << '>';
    }
};

void main () {
    PolReg r;
    r.inizializza(3, 1.0);           // un triangolo
    r.scrivi(); cout << '\t';       // <3, 1>
    cout << "Perimetro: " << r.perimetro() << '\n'; // 3

    PolReg* pr = new PolReg;
    pr->inizializza(4, 1.0);         // un quadrato
    pr->rendiSimile(5.0);
    cout << "Lunghezza di ogni lato: " <<      // 5
         pr->lunghezzaLato() << '\n';
}
```

### 16.3 Operazioni predefinite

```
#include <iostream.h>

class PolReg {
    /* ... */
};

void main () {
    PolReg r1, r2;
    r1.inizializza(3, 1.0);

    r2 = r1;                      // copia membro a membro

    r1.scrivi(); cout << '\n'; // <3, 1>
    r2.scrivi(); cout << '\n'; // <3, 1>
}
```

### 16.4 Puntatore this (I)

```
#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    void inizializza(int n, double g) {
        nLati = n; lunghLato = g;}
    PolReg& rendiSimile(double scala) {
        lunghLato *= scala; return *this;}
    /* ... */
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>'}
};

void main () {
    PolReg q1, q2;
    q1.inizializza(4, 10.0);
    q1.scrivi(); cout << '\n';                                // <4, 10>

    q1.rendiSimile(10.0);
    q1.scrivi(); cout << '\n';                                // <4, 100>

    PolReg t;
    t.inizializza(3, 1.0);
    t.rendiSimile(2.0).rendiSimile(2.0).scrivi(); // <3, 4>
// concatenazione di funzioni membro
    cout << '\n';
    t.scrivi(); cout << '\n';                                // <3, 4>
}
```

## 16.4 Puntatore this (II)

```
#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    void inizializza(int n, double g) {
        nLati = n; lunghLato = g;}
    PolReg rendiSimile(double scala) { // ATTENZIONE!
        lunghLato *= scala; return *this;}
    /* ... */
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';
    }
};

void main () {
    PolReg q1, q2;
    q1.inizializza(4, 10.0);
    q1.scrivi(); cout << '\n';           // <4, 10>

    q1.rendiSimile(10.0);
    q1.scrivi(); cout << '\n';           // <4, 100>

    PolReg t;
    t.inizializza(3, 1.0);
    t.rendiSimile(2.0).rendiSimile(2.0).scrivi(); // <3, 4>
    cout << '\n';
    t.scrivi(); cout << '\n';           // ATTENZIONE! <3, 2>
}
```

## 16.5 Funzioni globali

```
// Rende vuoto lo stack
// (realizzazione come funzione globale)

const int N = 100;

class Stack {
    int tt;
    int ee[N];
public:
    void inizializza() {tt = 0;}
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= N;}
};

void rendiVuoto(Stack& s) {
    while (!s.vuoto())
        s.pop();
}

void main () {
    Stack st;
    st.inizializza();
    /*...*/
    rendiVuoto(st);
    /*...*/
}
```

## 16.8 Ricompilazione (I)

```
// ----- File DStack.h -----//
// File di intestazione (estensione h)

const int MAX = 100;

class DStack {
    int nElem;
    int tt;
    int* ee;
public:
    DStack(int N = MAX) {
        ee = new int[nElem = N]; tt = 0;}
    DStack(const DStack& s);
    DStack& DStack::operator=(const DStack& s);
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= nElem;}
    void inverti();
    ~DStack() {delete[] ee;}
};
```

(continua ...)

## 16.8 Ricompilazione (II)

```
// ----- File DStack.cpp -----//
// File di codice (estensione cpp)

#include "DStack.h"

DStack& DStack::operator=(const DStack& s) {
    if (&s != this) {
        nElem = s.nElem;
        tt = s.tt;
        delete[] ee;
        ee = new int[nElem];
        for (int i = 0; i < nElem; i++) ee[i] = s.ee[i];
    }
    return *this;
}

DStack::DStack(const DStack& s) {
    nElem = s.nElem;
    tt = s.tt;
    ee = new int[nElem];
    for (int i = 0; i < nElem; i++)
        ee[i] = s.ee[i];
}

void DStack::inverti() {
    DStack s = *this;
    for (int i = 0; i < nElem; i++)
        ee[i] = s.ee[nElem - 1 - i];
};
```

(... continua ...)

## 16.8 Ricompilazione (III)

```
// ----- File Main.cpp -----//  
  
#include <iostream.h>  
#include "DStack.h"  
  
void main() {  
    int quantiElem;  
    cin >> quantiElem;  
    DStack s1(quantiElem);  
    int i;  
    for (; quantiElem > 0; quantiElem--) {  
        cin >> i;  
        s1.push(i);  
    }  
  
    DStack s2(s1);  
    while (!s2.vuoto())  
        cout << s2.pop() << '\n';  
  
    s2 = s1;  
    s2.inverti();  
    while (!s2.vuoto())  
        cout << s2.pop() << '\n';  
}
```

(... fine)

## 16.9 Costruttori (I)

// Costruttore per poligoni regolari

```
#include <iostream.h>  
  
class PolReg {  
    int nLati;  
    double lunghLato;  
public:  
    PolReg(int n, double g) {  
        nLati = n; lunghLato = g;}  
    double perimetro() {return nLati * lunghLato;}  
    /* ... */  
    void scrivi() {  
        cout << '<' << nLati << ", " << lunghLato << '>';}  
};  
  
void main () {  
    PolReg t(3, 1.0); // un triangolo  
    t.scrivi(); cout << '\t'; // <3, 1>  
    cout << "Perimetro: " << t.perimetro() << '\n'; // 3  
}
```

## 16.9 Costruttori (II)

// Costruttore per complessi

```
#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r, double i) {re = r; im = i;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

void main() {
    Complesso c = Complesso(1.0, 2.0);
    // forma esplicita

    Complesso c1(10.0, 20.0);
    // forma abbreviata

    Complesso c2;
    // ERRORE! (esiste un costruttore)

    /*...*/
}
```

## 16.9 Costruttori (III)

// Costruttore con allocazione di memoria libera

```
#include <iostream.h>
#include <string.h>

class Parola {
    char* nn;
public:
    Parola (const char r[]);
    /*...*/
    void scrivi() {cout << nn;}
};

inline Parola::Parola (const char r[]) {
    nn = new char[strlen(r)+1];
    strcpy(nn, r);
}

void main() {
    Parola c("C++");
    c.scrivi(); cout << '\n';
    /*...*/
}
```

### 16.9.1 Costruttori default (I)

```
#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso() {re = 0; im = 0;};
    // costruttore default mediante sovrapposizione
    Complesso(double r, double i) {re = r; im = i;};
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')'};
};

Complesso cf();
// ATTENZIONE! Dichiarazione di funzione

void main() {
    Complesso c;
    // OK (esiste il costruttore default)
    c.scrivi(); cout << '\n';                                // (0, 0)
}
```

### 16.9.1 Costruttori default (II)

```
#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;};
    // costruttore default mediante argomenti default
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')'};
};

void main() {
    Complesso c1; // OK, esiste il costruttore default
    c1.scrivi(); cout << '\n';                                // (0, 0)

    Complesso c2(1.0);
    c2.scrivi(); cout << '\n';                                // (1, 0)

    Complesso c3 = 10.0;
    // OK, esiste un costruttore con un solo argomento
    c3.scrivi(); cout << '\n';                                // (10, 0)
}

//~~~~~
class Complesso {
    double re, im;
public:
    Complesso() {re = 0; im = 0;};
    Complesso(double r = 0, double i = 0) {re = r; im = i;};
    // ERRORE!
    /*...*/
};
```

## 16.9.2 Costruttori per oggetti dinamici

```
#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    PolReg(int n = 3, double g = 1.0) {
        nLati = n; lunghLato = g;}
    double perimetro() {return nLati * lunghLato;}
/* ... */
};

void main () {
    PolReg* pQ = new PolReg(4, 1.0); // un quadrato
    cout << pQ->perimetro() << '\n'; // 4

    PolReg* pT = new PolReg;          // un triangolo
// OK, esiste il costruttore default
    cout << pT->perimetro() << '\n'; // 3
}
```

## 16.10 Distruttori (I)

```
#include <iostream.h>
#include <string.h>

class Parola {
    char* nn;
public:
    Parola (const char r[] = "") { // costruttore default
        nn = new char[strlen(r)+1]; strcpy(nn, r);}
/*...*/
    void scrivi() {cout << nn;}
    ~Parola() {delete[] nn;} // distruttore
};

void main() {
    Parola c("C++");
    c.scrivi(); cout << '\n';
/*...*/
}
```

## 16.10 Distruttori (II)

```
// Stack dinamico

#include <iostream.h>

class DStack {
    int nElem;          // capacita` dello stack
    int tt;
    int* ee;
public:
    DStack(int N = 100);
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= nElem;}
    ~DStack();
};

DStack::DStack(int N) {
    nElem = N;
    ee = new int[nElem];
    tt = 0;
}

DStack::~DStack() {
    delete[] ee;
}
```

(continua ...)

## 16.10 Distruttori (III)

```
void main() {
    int quanti;
    cin >> quanti;
    DStack s1(quanti);
    int i;
    for (; quanti > 0; quanti--) {
        cin >> i;
        s1.push(i);
    }
    while (!s1.vuoto()) cout << s1.pop() << '\n';
}
```

(... fine)

## 16.10 Distruttori (IV)

```
// Scrive: (1) (C) (2) (C) (3) (C) (4) (D) (5) (D) (6) (D)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {
        re = r; im = i; cout << "(C)\n";
    /*...*/
    ~Complesso() {cout << "(D)\n";}
}

void f() {
    cout << "(3)\n";
    Complesso x;
    cout << "(4)\n";
}

void main() {
    cout << "(1)\n";
    Complesso c(1.0, 2.0);
    {
        cout << "(2)\n";
        Complesso d;
        f();
        cout << "(5)\n";
    }
    cout << "(6)\n";
}
```

## 17.1 Overloading di operatori (I)

```
// Operatore - (meno) UNARIO come funzione GLOBALE
// @h => operator@(h)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    double pRe() {return re;}
    double plm() {return im;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
}

inline Complesso operator-(Complesso a) {
    return Complesso(-a.pRe(), -a.plm());
}

void main() {
    Complesso c1(1.0, 2.0), c2;
    c2 = -c1;
    c2.scrivi(); cout << '\n'; // (-1, -2)
}

//~~~~~
inline Complesso operator-(Complesso a) {
    Complesso x;
    x = Complesso(-a.pRe(), -a.plm());
    return x;
}
```

## 17.1 Overloading di operatori (II)

```
// Operatore - (meno) UNARIO come funzione MEMBRO
// @h => h.operator@()

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    Complesso operator-();
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso Complesso::operator-() {
    return Complesso(-re, -im);
}

void main() {
    Complesso c1(1.0, 2.0), c2;
    c2 = -c1;
    c2.scrivi(); cout << '\n';           // (-1, -2)
}

//~~~~~
inline Complesso Complesso::operator-() {
    Complesso x;
    x.re = -re;
    x.im = im;
    return x;
}
```

## 17.1 Overloading di operatori (III)

```
// Operatore + BINARIO come funzione GLOBALE
// h @ k => operator@(h, k)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    double pRe() {return re;}
    double plm() {return im;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso operator+(Complesso h,
                           Complesso k) {
    return Complesso(h.pRe()+k.pRe(), h.plm()+k.plm());
}

void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0), c3;
    c3 = c1 + c2;
    c3.scrivi(); cout << '\n';           // (11, 22)

    c3 = c1 + 1.0;
    c3.scrivi(); cout << '\n';           // (2, 2)

    c3 = 1.0 + c1;
    c3.scrivi(); cout << '\n';           // (2, 2)
}
```

## 17.1 Overloading di operatori (IV)

```
// Operatore + BINARIO come funzione MEMBRO
// h @ k => h.operator@(k)

// (SOLUZIONE SCONSIGLIATA: gli operatori che
// non cambiano il valore dei propri operandi si
// realizzano meglio come funzioni globali)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    Complesso operator+(Complesso k);
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso Complesso::operator+(Complesso k) {
    return Complesso(re + k.re, im + k.im);
}

void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0), c3;
    c3 = c1 + c2;
    c3.scrivi(); cout << '\n';           // (11, 22)

    c3 = c1 + 1.0;
    c3.scrivi(); cout << '\n';           // (2, 2)

    c3 = 1.0 + c1;                      // ERRORE!
}
```

## 17.1 Overloading di operatori (V)

```
// Operatore += come funzione MEMBRO
// h @ k => h.operator@(k)

// (gli operatori che modificano il valore di uno dei
// propri operandi sono di solito realizzati come funzioni
// membro)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    Complesso& operator+=(Complesso);
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso& Complesso::operator+=(Complesso k) {
    re += k.re;
    im += k.im;
    return *this;
}

void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0);
    c2 += c1;
    c2.scrivi(); cout << '\n';           // (11, 22)
}
```

## 17.1 Overloading di operatori (VI)

```
// Operatore + BINARIO come funzione GLOBALE
// utilizzando +=

// (di solito conviene realizzare operator@= per
// primo, e poi usarlo per realizzare operator@)

#include <iostream.h>

class Complesso {
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    Complesso& operator+=(Complesso k) {
        re += k.re; im += k.im; return *this;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso operator+(Complesso h,
    Complesso k) {
    return h += k;
}
```

(continua ...)

## 17.1 Overloading di operatori (VII)

```
void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0), c3;
    c3 = c1 + c2;
    c3.scrivi(); cout << '\n'; // (11, 22)

    c3 = c1 + 1.0;
    c3.scrivi(); cout << '\n'; // (2, 2)

    c3 = 1.0 + c1;
    c3.scrivi(); cout << '\n'; // (2, 2)
}
```

(... fine)

## 17.2 Funzioni friend (I)

```
// Operatore - (meno) UNARIO come funzione GLOBALE
// AMICA

// (un operatore unario e` piu` opportuno realizzarlo
// come funzione membro)

#include <iostream.h>

class Complesso {
    friend Complesso operator-(Complesso);
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
}

inline Complesso operator-(Complesso a) {
    return Complesso(-a.re, -a.im);
}

void main() {
    Complesso c1 = Complesso(1.0, 2.0);
    Complesso c2 = -c1;
    c2.scrivi(); cout << '\n';           // (-1, -2)
}
```

## 17.2 Funzioni friend (II)

```
// Operatore + BINARIO come funzione GLOBALE
// AMICA

#include <iostream.h>

class Complesso {
    friend Complesso operator+(Complesso,
                                Complesso);
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso operator+(Complesso h,
                           Complesso k) {
    return Complesso(h.re + k.re, h.im + k.im);
}

void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0), c3;
    c3 = c1 + c2;
    c3.scrivi(); cout << '\n';           // (11, 22)

    c3 = c1 + 1.0;
    c3.scrivi(); cout << '\n';           // (2, 2)

    c3 = 1.0 + c1;
    c3.scrivi(); cout << '\n';           // (2, 2)
}
```

## 17.2 Funzioni friend (III)

```
// Molteplici sovrapposizioni dell'operatore + BINARIO

#include <iostream.h>

class Complesso {
    friend Complesso operator+(Complesso,
        Complesso);
    friend Complesso operator+(Complesso, double);
    friend Complesso operator+(double, Complesso);
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso operator+(Complesso h,
    Complesso k) {
    return Complesso(h.re + k.re, h.im + k.im);
}

inline Complesso operator+(Complesso h, double k) {
    return Complesso(h.re + k, h.im);
}

inline Complesso operator+(double h, Complesso k) {
    return Complesso(h + k.re, k.im);
}
```

(continua ...)

## 17.2 Funzioni friend (IV)

```
void main() {
    Complesso c1(1.0, 2.0), c2(10.0, 20.0), c3;
    c3 = c1 + c2;
    c3.scrivi(); cout << '\n'; // (11, 22)

    c3 = c1 + 1.0;
    c3.scrivi(); cout << '\n'; // (2, 2)

    c3 = 1.0 + c1;
    c3.scrivi(); cout << '\n'; // (2, 2)
}
```

(... fine)

## 17.2 Funzioni friend (V)

// Funzione GLOBALE, AMICA di due classi

```
#include <iostream.h>

class Elemento; // pre-dichiarazione

class Valore {
    friend void scrivi(Elemento);
    int v;
public:
    Valore(int i = 0) {v = i;}
    /* ... */
};

class Elemento {
    friend void scrivi(Elemento);
    Valore val;
public:
    Elemento() {}
    /* ... */
};

inline void scrivi(Elemento elem) {
    cout << elem.val.v;
}

void main() {
    Elemento unElemento;
    scrivi(unElemento); cout << '\n'; // 0
}
```

## 17.2 Funzioni friend (VI)

// Funzione MEMBRO di una classe e AMICA di un'altra

```
#include <iostream.h>

class Valore {
    friend class Elemento;
    int v;
public:
    Valore(int i = 0) {v = i;}
    /* ... */
};

class Elemento {
    Valore val;
public:
    Elemento() {}
    /* ... */
    void scrivi();
};

inline void Elemento::scrivi() {
    cout << val.v;
}

void main() {
    Elemento unElemento;
    unElemento.scrivi(); cout << '\n'; // 0
}
```

### 17.3 Assegnamento (I)

```
// Operatore di assegnamento
// X& X::operator=(const X&)

#include <iostream.h>

class DStack {           // stack dinamico
    int nElem;           // capacita` dello stack
    int tt;
    int* ee;
public:
    DStack(int N = 100) {ee = new int[nElem= N]; tt = 0;}
    DStack& operator=(const DStack& s);
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= nElem;}
    ~DStack() {delete[] ee;}
};

DStack& DStack::operator=(const DStack& s) {
    if (&s != this) {
        nElem = s.nElem;
        tt = s.tt;
        delete[] ee;
        ee = new int[nElem];
        for (int i = 0; i < nElem; i++) ee[i] = s.ee[i];
    }
    return *this;
}
```

(continua ...)

### 17.3 Assegnamento (II)

```
void main() {
    int quantiElem;
    cin >> quantiElem;
    DStack s1(quantiElem), s2;
    int i;
    for (; quantiElem > 0; quantiElem--) {
        cin >> i;
        s1.push(i);
    }

    s2 = s1;
    while (!s2.vuoto()) cout << s2.pop() << '\n';
}
```

(... fine)

## 17.4 Copia (I)

```
// Costruttore di copia
// X::X(const X&)

#include <iostream.h>

class DStack {
    int nElem;
    int tt;
    int* ee;
public:
    DStack(int N = 100) {ee = new int[nElem= N]; tt = 0;}
    DStack(const DStack& s);
    DStack& operator=(const DStack& s);
    void push(int v) {ee[tt++] = v;}
    int pop() {return ee[--tt];}
    int top() {return ee[tt - 1];}
    int vuoto() {return tt <= 0;}
    int pieno() {return tt >= nElem;}
    ~DStack() {delete[] ee;}
};

DStack& DStack::operator=(const DStack& s) { /* ... */}
```

```
DStack::DStack(const DStack& s) {
    nElem = s.nElem;
    tt = s.tt;
    ee = new int[nElem];
    for (int i = 0; i < nElem; i++) ee[i] = s.ee[i];
}
```

(continua ...)

## 17.4 Copia (II)

```
void main() {
    int quantiElem;
    cin >> quantiElem;
    DStack s1(quantiElem);
    int i;
    for (; quantiElem > 0; quantiElem--) {
        cin >> i;
        s1.push(i);
    }

    DStack s2 = s1;
    while (!s2.vuoto()) cout << s2.pop() << '\n';
}
```

(... fine)

## 18.1 Oggetti costanti nelle classi

```
// Liste di inizializzazione per membri oggetti costanti

#include <iostream.h>

class PolReg {
    const int nLati;
    double lunghLato;
public:
    PolReg(int n = 3, double g = 1.0);
    /* ... */
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';
    }

    inline PolReg::PolReg(int n, double g) : nLati(n) {
        lunghLato = g;
    }

    void main() {
        PolReg q(4, 10.0);      // un quadrato di lato 10.0
        q.scrivi(); cout << '\n'; // <4, 10>
    }
}
```

## 18.2 Classi contenenti oggetti classe (I)

```
// Liste di inizializzazione per membri oggetti classe

#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    PolReg(int n, double g) {nLati = n; lunghLato = g;}
    /* ... */
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';
    }

    enum Qualifica {TRASPIRENTE, OPACO};

    class PolRegQual {
        PolReg pol;
        Qualifica qual;
    public:
        PolRegQual(int n = 3, double g = 1.0,
                   Qualifica q = TRASPIRENTE);
        /* ... */
        void scrivi() {
            cout << (qual ? "OP: " : "TR: "); pol.scrivi();
        }

        inline PolRegQual::PolRegQual(int n, double g,
                                      Qualifica q) : pol(n, g) {qual = q;}
    };
}
```

(continua ...)

## 18.2 Classi contenenti oggetti classe (II)

```
void main() {
    PolRegQual q(4, 10.0, OPACO); // un quadrato opaco
    q.scrivi(); cout << '\n';           // OP: <4, 10>
}
```

(... fine)

## 18.3 Array di oggetti classe (I)

```
#include <iostream.h>

class PolReg {
    int nLati;
    double lunghLato;
public:
    PolReg(int n = 3, double g = 1.0) {
        nLati = n; lunghLato = g;}
    /* ... */
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';
    };
    const int N = 3;
};

void main() {
    PolReg figura[N];
    for (int i = 0; i < N; i++) {
        figura[i].scrivi();           // <3, 1> <3, 1> <3, 1>
        cout << '\t';
    }
    cout << '\n';

    PolReg* pFigura = new PolReg[N];
    for (i = 0; i < N; i++) {
        pFigura[i].scrivi();         // <3, 1> <3, 1> <3, 1>
        cout << '\t';
    }
    cout << '\n';
}
```

(continua ...)

### 18.3 Array di oggetti classe (II)

```
PolReg altraFigura[N] = {  
    PolReg(5, 100.0),  
    PolReg(4, 10.0)  
};  
for (i = 0; i < N; i++) {  
    altraFigura[i].scrivi(); // <5, 100> <4, 10> <3, 1>  
    cout << '\t';  
}  
cout << '\n';  
}
```

(... fine)

### 18.4 Membri statici (I)

```
// Elemento statico privato  
  
#include <iostream.h>  
  
class Prodotto {  
    static int prossimoNdS;  
    int NdS; // numero di serie  
    /* ... */  
public:  
    Prodotto() {NdS = prossimoNdS++;}  
    int numeroDiSerie() {return NdS;}  
    /* ... */  
};  
  
int Prodotto::prossimoNdS = 0;  
  
void main() {  
    Prodotto i;  
    cout << i.numeroDiSerie() << '\n'; // 0  
    Prodotto j;  
    cout << j.numeroDiSerie() << '\n'; // 1  
    /* ... */  
}  
  
//~~~~~  
  
class Valore {  
    int val;  
public:  
    static int quantiValori = 0; // ERRORE!  
    /* ... */  
};
```

## 18.4 Membri statici (II)

// Elemento statico pubblico

```
#include <iostream.h>

class Prodotto {
    int NdS;
    /*...*/
public:
    static int prossimoNdS;
    Prodotto() {NdS = prossimoNdS++;}
    int numeroDiSerie() {return NdS;}
    /* ... */
};

int Prodotto::prossimoNdS = 0;

void main() {
    Prodotto i;
    cout << i.numeroDiSerie() << '\n';           // 0

    Prodotto::prossimoNdS += 10;

    Prodotto j;
    cout << j.numeroDiSerie() << '\n';           // 11

    j.prossimoNdS += 100;

    Prodotto k;
    cout << k.numeroDiSerie() << '\n';           // 112
    /*...*/
}
```

## 18.4 Membri statici (III)

// Funzioni membro statiche

```
#include <iostream.h>

class Prodotto {
    static int prossimoNdS;
    int NdS;
    /*...*/
public:
    Prodotto() {NdS = prossimoNdS++;}
    int numeroDiSerie() {return NdS;}
    static void avanzaNdS(int n) {prossimoNdS += n;}
    /*...*/
};

int Prodotto:: prossimoNdS = 0;

void main() {
    Prodotto i;
    cout << i.numeroDiSerie() << '\n';           // 0

    Prodotto::avanzaNdS(10);
    Prodotto j;
    cout << j.numeroDiSerie() << '\n';           // 11

    j.avanzaNdS(100);
    Prodotto k;
    cout << k.numeroDiSerie() << '\n';           // 112
}
```

## 18.4 Membri statici (IV)

// Funzione membro statica ERRATA

```
#include <iostream.h>

class Prodotto {
    static int prossimoNdS;
    int NdS;
    /*...*/
public:
    Prodotto() {NdS = prossimoNdS++;}
    int numeroDiSerie() {return NdS;}
    static void avanzaNdS(int n);
    /*...*/
};

int Prodotto:: prossimoNdS = 0;

void Prodotto::avanzaNdS(int n) {
    prossimoNdS += n;
    NdS = prossimoNdS++;
// ERRORE! Tentativo di uso del puntatore this
}
```

## 18.5 Funzioni const

// Sovrapposizione const-non const

```
#include <iostream.h>
```

```
class PolReg {
    int nLati;
    double lunghLato;
public:
    PolReg(int n = 3, double g = 1.0) {
        nLati = n; lunghLato = g;}
    /* ... */
    void scrivi() const {
        cout << '[' << nLati << ", " << lunghLato << ']';}
    void scrivi() {
        cout << '<' << nLati << ", " << lunghLato << '>';}

// Sovrapposizione const-non const
};

void main() {
    const PolReg qc(4, 10.0);
    qc.scrivi();                                // [4, 10]
    cout << '\n';

    PolReg q(4, 10.0);
    q.scrivi();                                 // <4, 10>
    cout << '\n';
}
```

## 18.7 Espressioni letterali e conversioni (I)

```
#include <iostream.h>

class Complesso {
    friend Complesso operator+(Complesso,
        Complesso);
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    /*...*/
    void scrivi() {cout << '(' << re << ", " << im << ')';}
};

inline Complesso operator+(Complesso h,
    Complesso k) {
    return Complesso(h.re + k.re, h.im + k.im);
}
```

(continua ...)

## 18.7 Espressioni letterali e conversioni (II)

```
void main() {
    Complesso c1(1.0, 2.0), c2;
    c2 = c1 + Complesso(10.0, 100.0);
    // espressione letterale mediante costruttore
    c2.scrivi(); cout << '\n';           // (11, 102)

    c2 = Complesso(1.0);
    // chiamata del costruttore di conversione
    // (un solo argomento)
    c2.scrivi(); cout << '\n';           // (1, 0)

    c2 = 10.0;
    // chiamata implicita del costruttore di conversione
    c2.scrivi(); cout << '\n';           // (10, 0)

    c2 = c1 + 10.0;
    // chiamata implicita del costruttore di conversione
    c2.scrivi(); cout << '\n';           // (11, 2)
}
```

(... fine)

## 18.7.1 Operatori di conversione

```
// Operatore di conversione X::operator T() da X a T
// (permette di usare un X dovunque occorrerebbe un T)

#include <iostream.h>

class InteroPiccolo { // interi compresi tra 0 e 63
    char n;
public:
    InteroPiccolo (int i) {n = i % 64;}
    operator int(); // converte interi piccoli in interi
    /*...*/
};

InteroPiccolo::operator int() {
    return n;
}

void main() {
    int i;
    InteroPiccolo p = 1;

    i = p; // chiama l'operatore di conversione

    cout << i << '\n'; // 11
}
```

## 20.2 Classi istream e ostream (I)

```
#include <iostream.h>

void main() {
    int i;
    cin >> i;
    cout << i << '\n'; // tipi fondamentali

    int* p = &i;
    cout << p << '\n'; // puntatori

    cout << "C++\n"; // stringhe

    cin >> i; // Esempio: 3
    cout << i << 2 << '\n'; // Esempio: 3 2
    cout << (i << 2) << '\n'; // Esempio: 12
}
```

## 20.2 Classi istream e ostream (II)

```
#include <iostream.h>

void main() {
    char x, y, z;
    cin >> x >> y >> z;           // Esempio: q w e
    cout << x << y << z;         // Esempio: qwe
}

//~~~~~//
```

  

```
#include <iostream.h>

void main() {
    char x, y, z;
    cin.get(x); cin.get(y); cin.get(z);   // Esempio: q w
    cout.put(x); cout.put(y); cout.put(z); // Esempio: q w
}
```

### 20.2.1 Controlli sugli stream

```
#include <iostream.h>
```

  

```
void main() {
    char c; cin >> c;
    while (!cin.eof()) {
        cout << c;           // Esempio: q w e (DOS: ^Z)
        cin >> c;            // Esempio: qwe
    }
}

//~~~~~//
```

  

```
#include <iostream.h>
```

  

```
void main() {
    char c; cin >> c;
    while (cin) {           // Nome del file come condizione
        cout << c;           // Esempio: q w e (^Z)
        cin >> c;            // Esempio: qwe
    }
}

//~~~~~//
```

  

```
#include <iostream.h>
```

  

```
void main() {
    char c;
    while (cin >> c)       // Esempio: q w e (^Z)
        cout << c;          // Esempio: qwe
}
```

### 20.3 Uso dei file (I)

```
// Copia da ingresso.in a cout

#include <fstream.h>
#include <stdlib.h> // per exit()

void main() {
    ifstream ingr("ingresso.in", ios::in | ios::nocreate);
    // ios::in apertura in lettura
    // ios::nocreate non creare il file se questo non esiste

    if (!ngr) {
        cerr << "File ingresso.in: Apertura fallita!\n";
        exit(1);
    }

    char ch;
    while (ngr.get(ch))
        cout.put(ch);
}
```

### 20.3 Uso dei file (II)

```
// Copia da cin a uscita.out

#include <fstream.h>
#include <stdlib.h>

void main() {
    ofstream usc("uscita.out", ios::out | ios::nocreate);
    // ios::out apertura in scrittura

    if (!usc) {
        cerr << "File uscita.out: Apertura fallita!\n";
        exit(1);
    }

    char ch;
    while (cin.get(ch))
        usc.put(ch);
}
```

### 20.3 Uso dei file (III)

// Copia da ingresso.in a uscita.out

```
#include <fstream.h>
#include <stdlib.h>

void main() {
    ifstream ingr("ingresso.in", ios::in | ios::nocreate);
    if (!ngr) {
        cerr << "File ingresso.in: Apertura fallita!\n";
        exit(1);
    }

    ofstream usc("uscita.out", ios::out | ios::nocreate);
    if (!usc) {
        cerr << "File uscita.out: Apertura fallita!\n";
        exit(1);
    }

    char ch;
    while (ngr.get(ch))
        usc.put(ch);
}
```

### 20.3 Uso dei file (IV)

// open() e close()

```
#include <fstream.h>
#include <stdlib.h>

void main() {
    fstream f;
    f.open("F.txt", ios::in | ios::nocreate);
    if (!f) {
        cerr << "File F.txt: Apertura fallita!\n";
        exit(1);
    }
    /*...*/
    f.close();
    f.open("F.txt", ios::out);
    if (!f) {
        cerr << "File F.txt: Apertura fallita!\n";
        exit(1);
    }
    /*...*/
}
```

### 20.3 Uso dei file (V)

```
// Copia alcuni file in cout

#include <fstream.h>
#include <stdlib.h>

const int quanti = 2;
char* tabella[quanti] = {
    "File1.in",
    "File2.in"
};

void main() {
    ifstream ingr;
    for (int i = 0; i < quanti; i++) {
        ingr.open(tabella[i], ios::in | ios::nocreate);
        if (!ingr) {
            cerr << "Apertura fallita!\n";
            continue;
        }

        char ch;
        while (ingr.get(ch))
            cout.put(ch);
        ingr.close();
        cout << '\n';
    }
}
```

### 20.4 Ingresso e uscita per i tipi utente (I)

```
#include <iostream.h>
#include <stdlib.h>

class Complesso {
    friend ostream& operator <<(ostream& os,
        const Complesso& c);
    double re, im;
public:
    Complesso(double r = 0, double i = 0) {re = r; im = i;}
    /* ... */
};

ostream& operator <<(ostream& os,
    const Complesso& c) {
    os << '(' << c.re << ", " << c.im << ')';
    return os;
}
```

(continua ...)

## 20.4 Ingresso e uscita per i tipi utente (II)

```
istream& operator >>(istream& is, Complesso& c) {
    double r, i;
    char ch;
    is >> ch;
    if (!cin || ch != '(') {
        is.clear(ios::failbit); return is;
    }
    is >> r >> ch;
    if (ch != ',') {
        is.clear(ios::failbit); return is;
    }
    is >> i >> ch;
    if (ch != ')') {
        is.clear(ios::failbit); return is;
    }
    c = Complesso(r, i);
    return is;
}

void main() {
    Complesso c1(1.0, 10.0);
    Complesso c2(1.1, 10.1);
    cout << c1 << '\t' << c2 << '\n';
    if (!(cin >> c2)) {
        cout << "Errore nella lettura di un Complesso\n";
        exit(1);
    }
    cout << c1 << '\t' << c2 << '\n';
}
```

(... fine)

## Aliasing (I)

### // Riferimenti

```
#include <iostream.h>
```

```
void main() {
    int i = 1;
    int& ri = i;
    ri++;
    cout << i << '\n'; // 2
/*...*/
}
```

```
////////////////////////////////////////////////////////////////////////
// Puntatori
```

```
#include <iostream.h>
```

```
void main() {
    int i = 1;
    int* pi = &i;
    (*pi)++;
    cout << i << '\n'; // 2
/*...*/
}
```

## Aliasing (II)

// Due argomenti di tipo riferimento

```
#include <iostream.h>

void f(int& a, int& b) {
    a++; b++;
}

void main() {
    int h = 1;
    f(h, h);
    cout << h << '\n';           // 3
    /*...*/
}
```

//-----//  
// Una variabile visibile a livello di file ed un argomento  
// di tipo riferimento

```
#include <iostream.h>

int h = 1;

void f(int& a) {
    a++; h++;
}

void main() {
    f(h);
    cout << h << '\n';           // 3
    /*...*/
}
```

## Aliasing (III)

// Problema: algoritmo

```
#include <iostream.h>

void scambia(int& a, int& b) {
    a = a + b;
    b = a - b;
    a = a - b;
}

void main() {
    int i, j;
    cout << "? ";
    cin >> i >> j;
    scambia (i, j);
    cout << i << '\t' << j << '\n';

    cout << "? ";
    cin >> i;
    scambia (i, i);
    cout << i << '\n';           // 0
}
```

## Aliasing (IV)

// Problema: ottimizzazione

```
#include <iostream.h>

void main() {
    int a, b, c, y;
    int& x = a;
    cout << "? ";
    cin >> a >> b >> c;           // Esempio: 1 10 100
    x = a + b + c;
    y = a + b + c;
    cout << x << '\t' << y << '\n'; // Esempio: 111 221

    cout << "? ";
    cin >> a >> b >> c;           // Esempio: 1 10 100
    x = a + b + c;
    y = x;
    cout << x << '\t' << y << '\n'; // Esempio: 111 111
}
```