

Materiale didattico di supporto alle lezioni del corso di

Linguaggio di programmazione C++

Corso di Laurea in Ingegneria Biomedica

Prof. Cinzia Bernardeschi

Dipartimento di Ingegneria dell'Informazione

Anno Accademico 2023-2024

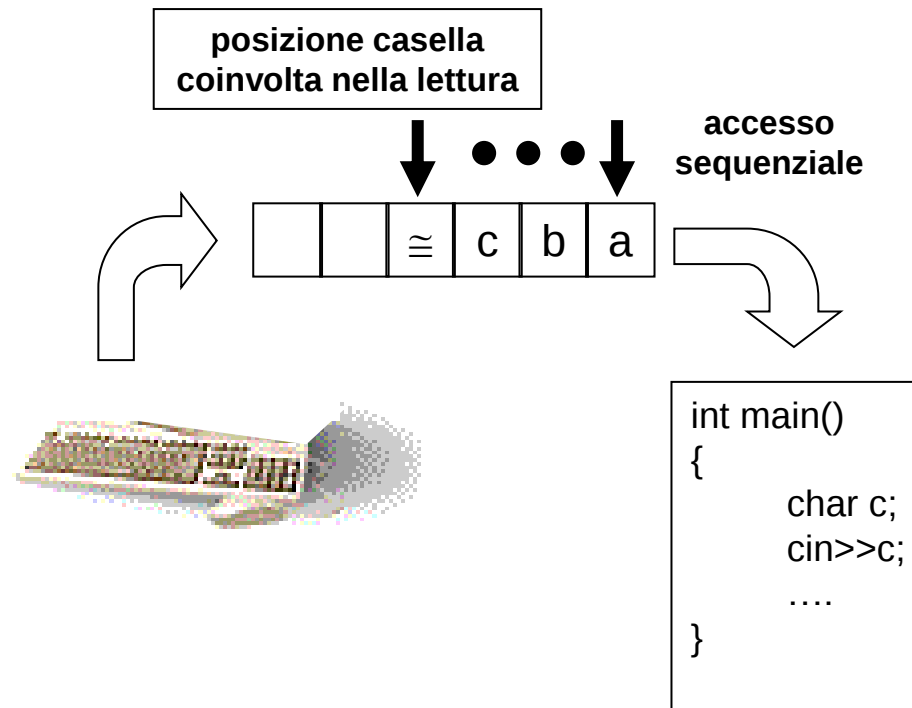
5.1 Concetto di stream (I)

Programma:

- comunica con l'esterno tramite uno o più *flussi* (stream);

Stream:

- struttura logica costituita da una sequenza di caselle (o celle), ciascuna di un byte, che termina con una marca di fine stream (il numero delle caselle è illimitato);



5.2 Concetto di stream (II)

Gli stream predefiniti sono tre:

- *cin*, *cout* e *cerr*;

Funzioni che operano su questi stream:

- si trovano in una libreria di ingresso/uscita, e per il loro uso occorre includere il file di intestazione `<iostream>`.

Osservazione:

- quanto verrà detto per lo stream *cout* vale anche per lo stream *cerr*.

Stream di ingresso standard (*cin*):

- per prelevarvi dati, si usa l'istruzione di lettura (o di ingresso):

basic-input-expression-statement

input-stream >> variable-name ;

5.2 Concetto di stream (II)

Azioni:

- prelievo dallo stream di una sequenza di caratteri, consistente con la sintassi delle costanti associate al tipo della variabile (tipo intero: eventuale segno e sequenza di cifre, e così via);
- la conversione di tale sequenza in un valore che viene assegnato alla variabile.

5.2 Operatore di Lettura

Operatore di lettura definito per:

- singoli caratteri;
- numeri interi;
- numeri reali;
- sequenze di caratteri (costanti stringa).

Esecuzione del programma:

- quando viene incontrata un'istruzione di lettura, il programma si arresta in attesa di dati;
- i caratteri che vengono battuti sulla tastiera vanno a riempire lo stream *cin*;
- per consentire eventuali correzioni, i caratteri battuti compaiono anche in eco sul video;
- tali caratteri vanno effettivamente a far parte di *cin* solo quando viene battuto il tasto di ritorno carrello.

5.2 Operatore di Lettura

Ridirezione:

- col comando di esecuzione del programma, lo stream *cin* può essere ridiretto su un file *file.in* residente su memoria di massa;
- comando DOS/Windows (*leggi.exe* è un file eseguibile):
leggi.exe < file.in

5.2 Lettura di Caratteri (I)

```
char c;  
cin >> c;
```

Azione:

- se il carattere contenuto nella casella selezionata dal puntatore non è uno spazio bianco:
 - viene prelevato;
 - viene assegnato alla variabile *c*;
 - il puntatore si sposta sulla casella successiva;
- se il carattere contenuto nella casella selezionata dal puntatore è uno spazio bianco:
 - viene ignorato;
 - il puntatore si sposta sulla casella successiva, e così via, finché non viene letto un carattere che non sia uno spazio bianco.

Lettura di un carattere qualsivoglia (anche di uno spazio bianco):

```
char c;  
cin.get(c);
```

5.2 Lettura di Caratteri (II)

**// Legge caratteri e li stampa su video
// Termina al primo carattere 'q'.**

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    char c;
    while (true)    // while(1)
    {
        cout << "Inserisci un carattere " << endl;
        cin >> c;
        if (c != 'q')
            cout << c << endl;
        else
            break;
    }
    return 0;
}
```

Inserisci un carattere
a wq
a
Inserisci un carattere
w
Inserisci un carattere

5.2 Lettura di Caratteri (III)

// Legge caratteri e li stampa su video

// Termina al primo carattere 'q'.

```
int main()
{
    char c;
    while (true)
    {
        cout << "Inserisci un carattere " << endl;
        cin.get(c);
        if (c != 'q')
            cout << c << endl;
        else
            break;
    }

    return 0;
}
```

Inserisci un carattere

a wq

a

Inserisci un carattere

Inserisci un carattere

Inserisci un carattere

w

Inserisci un carattere

5.2 Lettura di Interi

```
int i;  
cin >> i;
```

Azione:

- il puntatore si sposta da una casella alla successiva fintanto che trova caratteri consistenti con la sintassi delle costanti intere, saltando eventuali spazi bianchi in testa, e si ferma sul primo carattere non previsto dalla sintassi stessa;
- il numero intero corrispondente alla sequenza di caratteri letti viene assegnato alla variabile *i*.

5.2 Lettura di Interi

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int i, j;
    cout << "Inserisci due numeri interi " << endl;
    cin >> i;
    cin >> j;
    cout << i << endl << j << endl;

    return 0;
}
```

Inserisci due numeri interi

-10 3

-10

3

5.2 Lettura di Reali

```
float f;  
cin >> f;
```

Azione:

- il puntatore si sposta da una casella alla successiva fintanto che trova caratteri consistenti con la sintassi delle costanti reali, saltando eventuali spazi bianchi in testa, e si ferma sul primo carattere non previsto dalla sintassi stessa;
- il numero reale corrispondente alla sequenza di caratteri letti viene assegnato alla variabile *f*.

5.2 Lettura di Reali

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    float f, g;
    cout << "Inserisci due numeri reali " << endl;
    cin >> f;
    cin >> g;
    cout << f << endl << g << endl;

    return 0;
}
```

```
Inserisci due numeri reali
-1.25e-3      .1e4
-0.00125
1000
```

Istruzione di ingresso:

- rientra nella categoria delle istruzione espressione;
- l'espressione produce come risultato lo stream coinvolto;
- consente di effettuare più letture in sequenza.

`cin >> x >> y;`

equivalente a:

`cin >> x;      cin >> y;`

Infatti, essendo l'operatore `>>` associativo a sinistra, prima viene calcolata la subespressione `cin >> x`, che produce come risultato *cin*, quindi la subespressione `cin >> y`.

5.3 Errori sullo stream di ingresso

Istruzione di lettura:

- il puntatore non individua una sequenza di caratteri consistente con la sintassi delle costanti associate al tipo della variabile:
 - l'operazione di prelievo non ha luogo e lo stream si porta in uno *stato di errore*;

Caso tipico:

- si vuole leggere un numero intero, e il puntatore individua un carattere che non sia il segno o una cifra;
- caso particolare:
 - si tenta di leggere la marca di fine stream.

Stream di ingresso in stato di errore:

- occorre un ripristino, che si ottiene con la funzione *cin.clear()*.

Stream di ingresso:

- può costituire una *condizione* (nelle istruzioni condizionali o ripetitive);
- la condizione è vera se lo stream non è in uno stato di errore, falsa altrimenti.

Tastiera del terminale:

- se un programma legge dati da terminale fino ad incontrare la marca di fine stream, l'utente deve poter introdurre tale marca;
- questo si ottiene premendo Control e Z nei sistemi DOS/Windows (Ctrl-Z).

5.3 Errori sullo stream di ingresso

// Legge e stampa numeri interi.
// Termina quando viene inserito un carattere non
// consistente con la sintassi delle costanti intere

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int i;
    while (cin)
    {
        cout << "Inserisci un numero intero " << endl;
        cin >> i;
        if (cin)
            cout << "Numero intero: " << i << endl;
    }

    return 0;
}
```

```
Inserisci un numero intero
3
Numero intero: 3
Inserisci un numero intero
x
```

5.3 Errori sullo stream di ingresso

```
// Legge e stampa numeri interi.  
// Termina quando viene inserito un carattere non  
// consistente con la sintassi delle costanti intere
```

```
#include <cstdlib>  
#include <iostream>  
using namespace std;  
int main()  
{  
    int i;  
    cout << "Inserisci un numero intero " << endl;  
    while (cin >> i)  
    {  
        cout << "Numero intero: " << i << endl;  
        cout << "Inserisci un numero intero " << endl;  
    }  
    return 0;  
}
```

```
Inserisci un numero intero  
3  
Numero intero: 3  
Inserisci un numero intero  
x
```

5.3 Errori sullo stream di ingresso

```
// Legge e stampa caratteri.  
// Termina quando viene inserito il fine stream  
  
#include <cstdlib>  
#include <iostream>  
using namespace std;  
int main()  
{  
    char c;  
    while (cout << "Inserisci un carattere " << endl && cin>>c)  
        cout << "Carattere: " << c << endl;  
  
    return 0;  
}
```

```
Inserisci un carattere  
s e  
Carattere: s  
Inserisci un carattere  
Carattere: e  
Inserisci un carattere  
a  
Carattere: a  
Inserisci un carattere  
^Z
```

5.4 Stream di uscita

Stream di uscita standard (cout):

- per scrivere su cout si usa l'istruzione di scrittura (o di uscita), che ha la forma:

basic-output-expression-statement

output-stream << expression ;

Azione:

- calcolo dell'espressione e sua conversione in una sequenza di caratteri;
- trasferimento di questi nelle varie caselle dello stream, a partire dalla prima;
- il puntatore e la marca di fine stream si spostano in avanti, e in ogni momento il puntatore individua la marca di fine stream.

Possono essere scritti:

- numeri interi;
- numeri reali;
- singoli caratteri;
- sequenze di caratteri (costanti stringa).

Nota:

- un valore di tipo booleano o di un tipo enumerato viene implicitamente convertito in intero (codifica del valore) .

5.4 Istruzione di scrittura

Istruzione di uscita:

- è un'istruzione espressione;
- il risultato è lo stream;
- analogamente all'istruzione di ingresso, consente di effettuare più scritture in sequenza.

Formato di scrittura (parametri):

- *ampiezza del campo* (numero totale di caratteri impiegati, compresi eventuali spazi per l'allineamento);
- *lunghezza della parte frazionaria* (solo per i numeri reali);

Parametri:

- valori default fissati dall'implementazione;
- possono essere cambiati dal programmatore.

Ridirezione:

- col comando di esecuzione del programma, lo stream *cout* può essere ridiretto su un file *file.out* residente su memoria di massa;
- comando DOS/Windows (*scrivi.exe* è un file eseguibile):
scrivi.exe > file.out

5.4 Istruzione di scrittura

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    double d = 1.564e-2, f=1.2345, i;
    cout << d << endl;

    cout << setprecision(2) << d << '\t' << f << endl;
    cout << d << endl;

    cout << setw(10) << d << ' ' << f << endl;
    cout << d << endl;

    cout << hex << 10 << '\t' << 11 << endl;
    cout << oct << 10 << '\t' << 11 << endl;
    cout << dec << 10 << '\t' << 11 << endl;
    return 0;
}
```

```
0.01564
0.016  1.2
0.016
      0.016 1.2
0.016
a      b
12     13
10     11
```

5.5 Manipolazione dei file

Stream associati ai file visti dal sistema operativo:

- gestiti da una apposita libreria;
- occorre includere il file di intestazione `<fstream>`.

Dichiarazione:

basic-file-stream-definition

`fstream identifier-list ;`

Esempio: `fstream ingr, usc;`

Funzione *open()*:

- associa uno stream ad un file;
- apre lo stream secondo opportune modalità;
- le modalità di apertura sono rappresentate da opportune costanti
 - lettura => costante `ios::in`;
 - scrittura => costante `ios::out`;
 - scrittura alla fine del file (append)
=> costante `ios::out | ios::app`;

5.5 Manipolazione dei file

Esempio:

```
ingr.open("file1.in", ios::in);  
usc.open("file2.out", ios::out);
```

il nome del file viene specificato come stringa (in particolare, come costante stringa).

Stream aperto in lettura:

- il file associato deve essere già presente;
- il puntatore si sposta sulla prima casella.

Stream aperto in scrittura:

- il file associato, se non presente, viene creato;
- il puntatore si posiziona all'inizio dello stream , sul quale compare solo la marca di fine stream (eventuali dati presenti nel file vengono perduti).

5.5 Manipolazione dei file

Stream aperto in append:

- il file associato, se non presente, viene creato;
- il puntatore si sposta alla fine dello stream, in corrispondenza della marca di fine stream (eventuali dati presenti nel file non vengono perduti).

Stream aperto:

- utilizzato con le stesse modalità e gli stessi operatori visti per gli stream standard;
- le operazioni effettuate sugli stream coinvolgono i file a cui sono stati associati.

–Scrittura:

```
usc << 10;
```

–Lettura:

```
ingr >> x
```

Funzione *close()*:

- chiude uno stream aperto, una volta che è stato utilizzato.

```
ingr.close();
```

```
usc.close();
```

5.5 Manipolazione dei file

Stream chiuso:

- può essere riaperto, associandolo ad un qualunque file e con una modalità arbitraria.

Fine del programma:

- tutti gli stream aperti vengono automaticamente chiusi.

Errori:

- quanto detto per lo stream *cin* vale anche per qualunque altro stream aperto in lettura.

5.5 Manipolazione dei file

```
// Scrive 4 numeri interi nel file "esempio".
// Apre il file in lettura e stampa su video il suo contenuto
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;

int main()
{
    fstream ff;
    int num;

    ff.open("esempio", ios::out);

    if (!ff)
    {
        cerr << "Il file non puo' essere aperto" << endl;
        exit(1);                // funzione exit
    }
}
```

5.5 Manipolazione dei file

```
for (int i = 0; i < 4; i++)  
    ff << i << endl;           // ATT. separare numeri
```

```
ff.close();
```

```
ff.open("esempio", ios::in);
```

```
if (!ff)  
{  
    cerr << "Il file non puo' essere aperto" << endl;  
    exit(1);  
}
```

```
while (ff >> num)               // fino alla fine del file  
    cout << num << '\t';
```

```
cout << endl;  
return 0;
```

```
}
```

0	1	2	3
---	---	---	---

5.5 Manipolazione dei file

```
// Apre in lettura il file "esempio" e legge N numeri interi.  
// Controlla che le istruzioni di lettura non portino  
// lo stream in stato di errore  
#include <cstdlib>  
#include <fstream>  
#include <iostream>  
using namespace std;  
  
int main()  
{  
    fstream ff;  
    int i, num;  
    const int N = 6;  
  
    ff.open("esempio", ios::in);  
    if (!ff)  
    {  
        cerr << "Il file non puo' essere aperto" << endl;  
        exit(1);  
    }  
}
```

5.5 Manipolazione dei file

```
for (i = 0; i < N && ff >> num; i++)  
    cout << num << '\t';
```

```
cout << endl;
```

```
if (i != N)  
    cerr << "Errore nella lettura del file " << endl;
```

```
return 0;
```

```
}
```

0	1	2	3
Errore nella lettura del file			

5.5 Manipolazione dei file

```
// Esempio apertura in append.
#include <cstdlib>
#include <fstream>
#include <iostream>
using namespace std;
int main()
{
    fstream ff;
    int i; char c;

    ff.open("esempio", ios::out);

    if (!ff)
    {
        cerr << "Il file non puo' essere aperto" << endl;
        exit(1);
    }

    for (int i = 0; i < 4; i++)
        ff << i << '\t';

    ff.close();
}
```

5.5 Manipolazione dei file

```
ff.open("esempio", ios::out | ios :: app);

ff << 5 << '\t' << 6 << endl;
ff.close();

ff.open("esempio", ios::in);
if (!ff)
{   cerr << "Il file non puo' essere aperto" << endl;
    exit(1);
}

while (ff.get(c))
    cout << c;

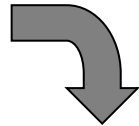
return 0;
}
```

0	1	2	3	5	6
---	---	---	---	---	---

5.5 Manipolazione dei file

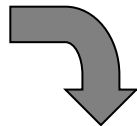
Versioni alternative

```
fstream ff;  
ff.open("file", ios::in);
```



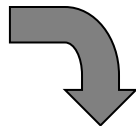
```
ifstream ff("file");
```

```
fstream ff;  
ff.open("file", ios::out);
```



```
ofstream ff("file");
```

```
fstream ff;  
ff.open("file", ios::out|ios::app);
```



```
ofstream ff("file",ios::app);
```

7.1 Concetto di funzione (I)

// Problema: stampare il fattoriale di 4

```
#include <cstdlib>
#include <iostream>
using namespace std;
int main()
{
    int f;
    f = 1;
    for (int i = 2; i <= 4; i++)
        f *= i;
    cout << "Il fattoriale di 4 e': " << f
    << endl;
    /* Altre elaborazioni */

    return 0;
}
```

Il fattoriale di 4 e': 24

7.1 Concetto di funzione (II)

// Soluzione

```
int fatt4()
{
    int ris = 1;
    for (int i = 2; i <= 4; i++)
        ris *= i;
    return ris;
}

int main()
{
    int f;
    f = fatt4();
    cout << "Il fattoriale di 4 e': " << f << endl;

    return 0;
}
```

Il fattoriale di 4 e': 24

7.1 Concetto di funzione (III)

Stampare il fattoriale di un numero letto da tastiera

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

int main()
{
    int i, f;
    cout << "Inserisci un numero intero: ";
    cin >> i;
    f = fatt(i);
    cout << "Il fattoriale di " << i " e': " << f << endl;
    /* Altre elaborazioni */

    return 0;
}
```

**Inserisci un numero intero: 4
Il fattoriale di 4 e': 24**

7.1 Concetto di funzione (IV)

Variabili definite in una funzione:

- *locali* alla funzione;

Nomi di variabili locali e di argomenti formali:

- associati ad oggetti che appartengono alla funzione in cui sono definiti;
- se uno stesso nome viene utilizzato in funzioni diverse, si riferisce in ciascuna funzione ad un oggetto diverso;
- in questo caso si dice che il nome è *visibile* solo nella rispettiva funzione.

Chiamata:

- gli argomenti formali e le variabili locali vengono allocati in memoria;
- gli argomenti formali vengono inizializzati con i valori degli argomenti attuali (passaggio per valore);
- gli argomenti formali e le variabili locali vengono utilizzati per le dovute elaborazioni;
- al termine della funzione, essi vengono deallocati, e la memoria da essi occupata viene resa disponibile per altri usi.

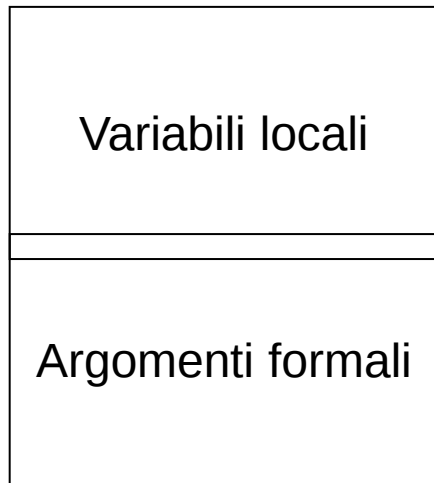
Istanza di funzione:

- nuova copia degli argomenti formali e delle variabili locali (nascono e muoiono con l'inizio e la fine della esecuzione della funzione);
- il valore di una variabile locale ottenuto durante una certa esecuzione della funzione non viene conservato per le successive istanze.

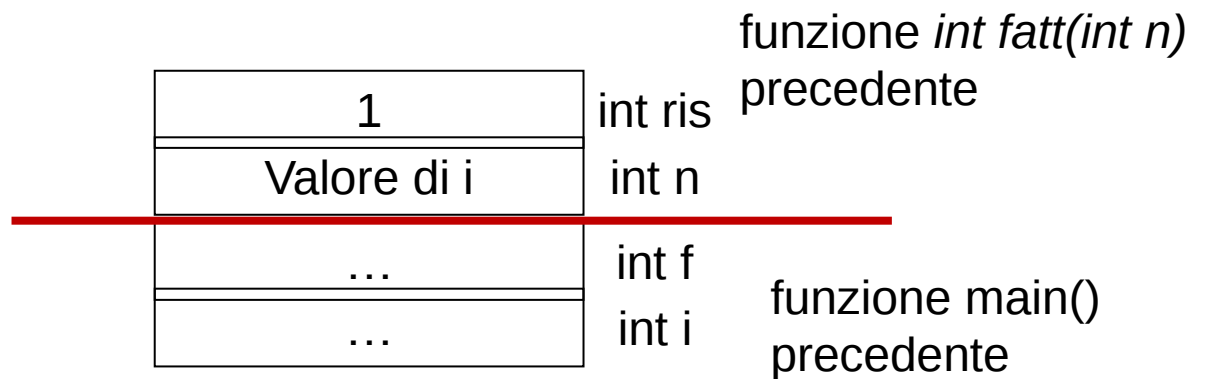
7.1 Concetto di funzione (V)

- Quando una funzione viene invocata, viene creata in memoria un'istanza della funzione;
- L'istanza ha un tempo di vita pari al tempo di esecuzione della funzione

// Istanza di funzione



Esempio:



7.3.1 Istruzione return (I)

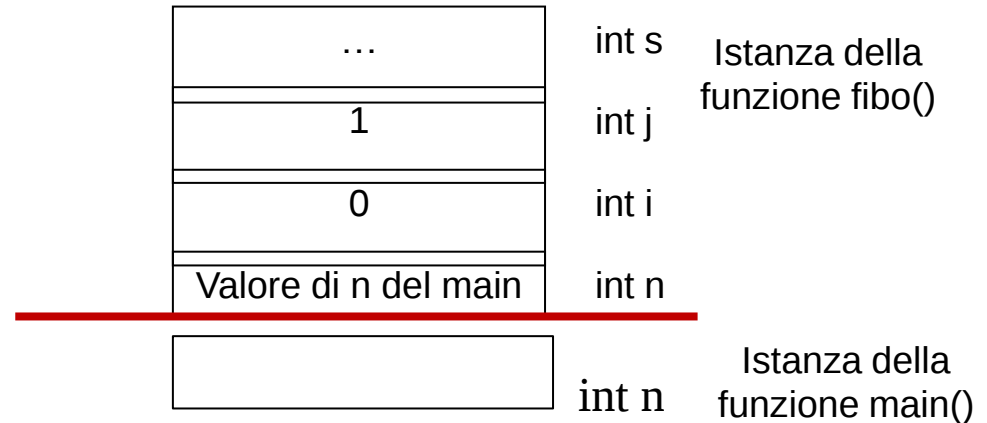
// Restituisce il massimo termine della successione di
// Fibonacci minore o uguale al dato intero positivo n

```
unsigned int fibo(unsigned int n)
```

```
{  
    unsigned int i = 0, j = 1, s;  
    for (;;)   
    {  
        if ((s = i + j) > n)  
            return j;  
        i = j;  
        j = s;  
    }  
}
```

```
int main()
```

```
{  
    unsigned int n;  
    cout << "Inserisci un numero intero " << endl;  
    cin >> n;  
    cout << "Termine successione Fibonacci: ";  
    cout << fibo(n) << endl;  
    return 0;}
```



7.3.1 Istruzione return (II)

// Controlla se un intero e' positivo, negativo o nullo

#include <cstdlib>

#include <iostream>

using namespace std;

enum val {N, Z, P};

val segno(int n)

{

if (n > 0)

return P;

if (n == 0)

return Z;

return N;

}

7.3.1 Istruzione return (II)

```
int main ()
{
    int i;
    // termina se legge un valore illegale per i
    while (cout << "Numero intero? " && cin >> i)
        switch (segno(i))
        {
            case N:
                cout << "Valore negativo" << endl;
                continue;
            case Z:
                cout << "Valore nullo" << endl;
                continue;
            case P:
                cout << "Valore positivo" << endl;
        }
    system("PAUSE");
    return 0;
}
```

7.3.1 Istruzione return (III)

// Controlla se un intero e' positivo, negativo o nullo

Numero intero? -10

Valore negativo

Numero intero? 3

Valore positivo

Numero intero? 0

Valore nullo

Numero intero? ^Z

Prima istanza funzione segno()

-10

int n

Seconda istanza funzione segno()

3

int n

Terza istanza funzione segno()

0

int n

7.4 Dichiarazioni di funzioni (I)

// Controlla se i caratteri letti sono lettere minuscole o numeri.

```
int main()
{
    char c;
    while (cout << "Carattere:? " << endl && cin >> c)
        if (!is_low_dig(c))           // ERRORE!
        {
            system("PAUSE");
            return 0;
        }
}

bool is_low_dig(char c)
{
    return (c >= '0' && c <= '9' ||
            c >= 'a' && c <= 'z') ? true : false;
}
```

``is_low_dig'` undeclared (first use this function)

ERRORE SEGNALATO IN FASE DI COMPILAZIONE

7.4 Dichiarazioni di funzioni (II)

```
// Controlla se i caratteri letti sono lettere minuscole o  
// numeri.
```

```
#include <cstdlib>  
#include <iostream>  
using namespace std;
```

```
bool is_low_dig(char c); // oppure bool is_low_dig(char);
```

```
int main()  
{  
    char c;  
    while (cout << "Carattere:? " << endl && cin >> c)  
        if (!is_low_dig(c))  
        {  
            return 0;  
        }  
}
```

7.4 Dichiarazioni di funzioni (II)

```
bool is_low_dig(char c)
{
    return (c >= '0' && c <= '9' ||
           c >= 'a' && c <= 'z') ? true : false;
}
```

Carattere:?

r

Carattere:?

3

Carattere:?

A

7.6 Argomenti e variabili locali (I)

```
// Somma gli interi compresi tra i dati interi n ed m,  
// estremi inclusi, con n <= m
```

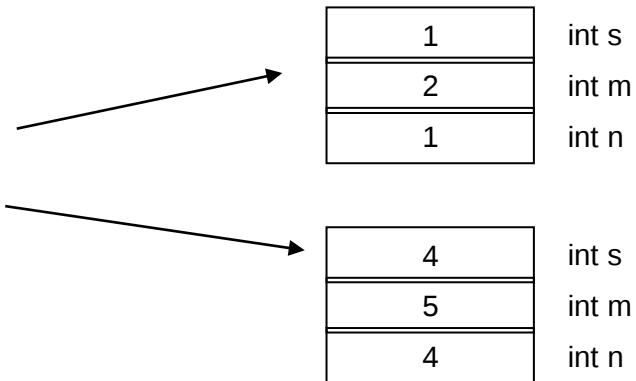
```
#include <cstdlib>  
#include <iostream>  
using namespace std;
```

```
int sommaInteri(int n, int m)  
{  
    int s = n;  
    for (int i = n+1; i <= m; i++)  
        s += i;  
    return s;  
}
```

7.6 Argomenti e variabili locali (I)

```
int main ()
{
    int a, b;
    while (cout << "Due interi? " && cin >> a >> b)
        // termina se legge un valore illegale per a, b
        cout << sommaInteri(a, b) << endl;
    system("PAUSE");
    return 0;
}
```

Due interi? 1 2
3
Due interi? 4 5
9
Due interi? s



7.6 Argomenti e variabili locali (II)

```
// Calcola il massimo fra tre numeri interi
#include <cstdlib>
#include <iostream>
using namespace std;
int massimo(int a, int b, int c)
{
    return ((a > b) ? ((a > c) ? a : c) : (b > c) ? b : c);
}
```

7.6 Argomenti e variabili locali (II)

```
int main()
{
    int i, j, k;
    cout << "Inserisci tre numeri: ";
    cin >> i >> j >> k;
    int m = massimo (i, j, k);
    cout << m << endl;
    /*...*/

    double x, y, z;
    cout << "Inserisci tre numeri: ";
    cin >> x >> y >> z;
    double w = massimo (x, y, z);
    cout << w << endl;
    return 0;
}
```

// Si applicano le regole standard
per la conversione di tipo.

```
Inserisci tre numeri: 3 4 5
5
Inserisci tre numeri: 3.3 4.4 5.5
5
```

7.7 Funzioni void

// Scrive asterischi

```
#void scriviAsterischi(int n)  
{  
    for (int i = 0; i < n; i++)  
        cout << '*';  
    cout << endl;  
}  
  
int main()  
{  
    int i;  
    cout << "Quanti asterischi? ";  
    cin >> i;  
    scriviAsterischi(i);  
  
    return 0;  
}
```

Quanti asterischi? 20

7.8 Funzioni senza argomenti

```
#include <cstdlib>
#include <iostream>
using namespace std;

const int N = 20;
void scriviAsterischi(void) // anche void scriviAsterischi()
{
    for (int i = 0; i < N; i++)
        cout << '*';
    cout << endl;
}

int main()
{
    scriviAsterischi();

    return 0;
}
```

```
*****
```

7.9 Funzioni ricorsive (I)

Funzione:

- può invocare, oltre che un'altra funzione, anche se stessa;
- in questo caso si ha una funzione *ricorsiva*.

Funzione ricorsiva:

- naturale quando il problema risulta formulato in maniera ricorsiva;

- esempio (fattoriale di un numero naturale n):

fattoriale(n) = 1 se $n = 0$

n *fattoriale($n-1$) se $n > 0$

7.9 Funzioni ricorsive (I)

```
#include <cstdlib>
#include <iostream>
using namespace std;

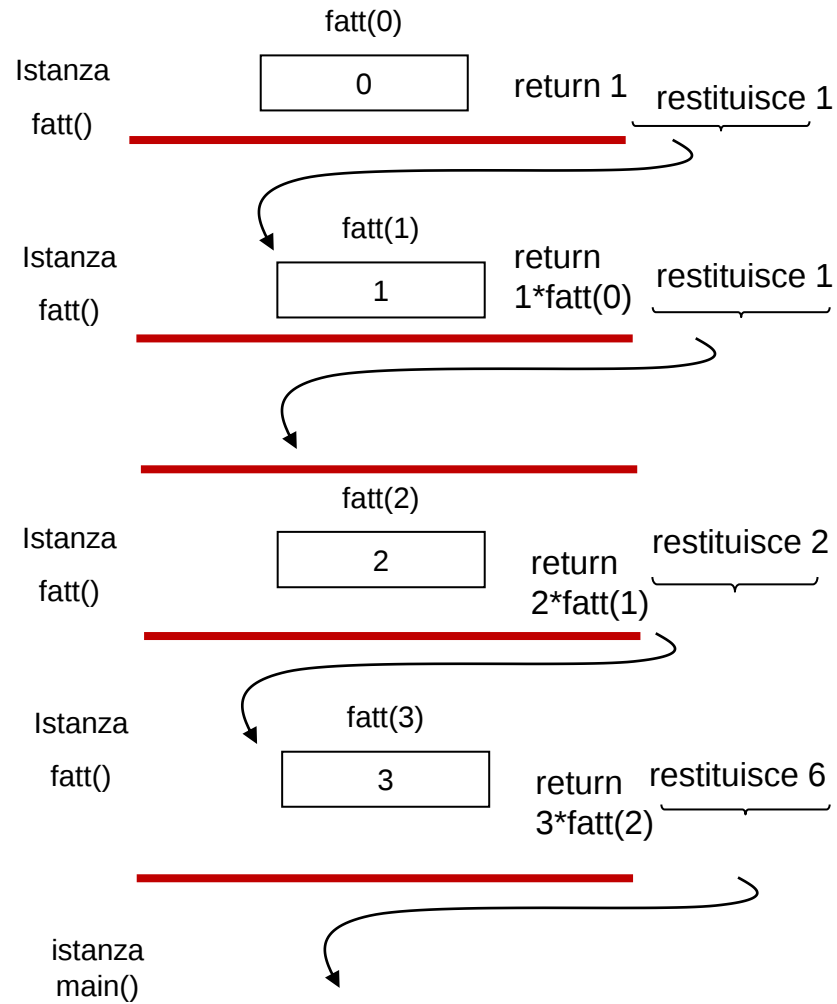
int fatt(int n)
{ if (n == 0) return 1;
  return n * fatt(n - 1);
}

int main()
{
    cout << "Il fattoriale di 3 e': " << fatt(3) << endl;

    return 0;
}
```

Il fattoriale di 3 e': 6

7.9 Funzioni ricorsive (II)



7.9 Funzioni ricorsive (III)

Massimo comun divisore:

```
int mcd(int alfa, int beta)
{
    if (beta == 0) return alfa;
    return mcd(beta, alfa % beta);
}
```

Somma dei primi n naturali:

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

7.9 Funzioni ricorsive (III)

Reale elevato a un naturale:

```
double pot(double x, int n)
{
    if (n == 0) return 1;
    return x * pot(x, n - 1);
}
```

Elementi della serie di Fibonacci:

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

7.9 Funzioni ricorsive (IV)

// Legge una parola terminata da un punto, e la scrive
// in senso inverso. Per esempio, "pippo" diventa "oppip".

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

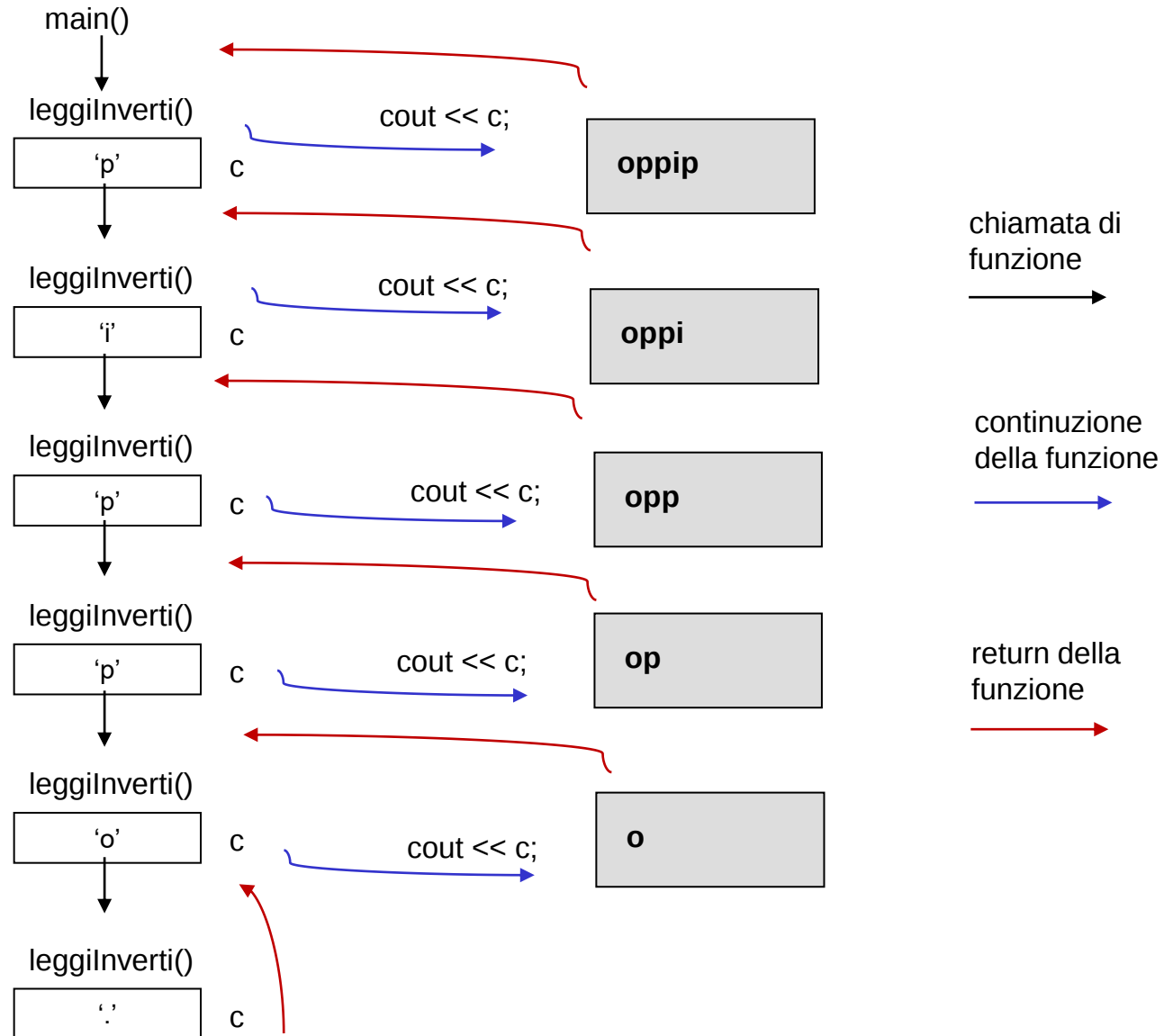
```
int main()
{
    leggiInverti();
    cout << endl;

    return 0;
}
```

pippo
oppip.

7.9 Funzioni ricorsive (V)

ordine
delle
chiamate
alle
funzioni



7.9 Funzioni ricorsive (VI)

Formulazione ricorsiva di una funzione:

- individuazione di uno o più *casi base*, nei quali termina la successione delle chiamate ricorsive;
- definizione di uno o, condizionatamente, di più passi ricorsivi;
- ricorsione controllata dal valore di un argomento di controllo, in base al quale si sceglie se si tratta di un caso base o di un passo ricorsivo;
- in un passo ricorsivo, la funzione viene chiamata nuovamente passandole un nuovo valore dell'argomento di controllo;
- il risultato di questa chiamata, spesso ulteriormente elaborato, viene restituito al chiamante dell'istanza corrente;
- nei casi base, il risultato viene calcolato senza altre chiamate ricorsive.

Schema di calcolo:

- parallelo a quello usato nelle computazioni iterative.

NOTA BENE:

- ogni funzione può essere formulata sia in maniera ricorsiva che in maniera iterativa;
- spesso, la formulazione iterativa è più conveniente, in termini di tempo di esecuzione e di occupazione di memoria.
- in diversi casi è più agevole (per il programmatore) esprimere un procedimento di calcolo in maniera ricorsiva;
- questo può riflettersi in una maggiore concisione e chiarezza del programma, e quindi una minore probabilità di commettere errori.

7.9 Funzioni ricorsive (Esempio)

Scrivere una funzione ricorsiva che stampi su uscita standard un triangolo rettangolo rovesciato composto di asterischi. I due cateti del triangolo contengono lo stesso numero N di asterischi. Nell'esempio seguente $N = 3$.

* * *

* *

*

7.9 Funzioni ricorsive (Esempio)

```
void scrivi(int n)
{
    if (n==0) return;
    for (int i=0; i<n; i++)
        cout << '*';
    cout << endl;
    scrivi(n-1);
}
```

Libreria:

- insieme di funzioni (sottoprogrammi) precompilate;
- formata da coppie di file;
- per ogni coppia un file contiene alcuni sottoprogrammi compilati ed uno contiene le dichiarazioni dei sottoprogrammi stessi (quest'ultimo è detto file di intestazione e il suo nome termina tipicamente con l'estensione *h*).

Utilizzo di funzioni di libreria:

- nella fase di scrittura del programma, includere il file di intestazione della libreria usando la direttiva `#include`;
- nella fase di collegamento, specificare la libreria da usare, secondo le convenzioni dell'ambiente di sviluppo utilizzato.

Esempio:

- programma contenuto nel file *mioprogram.cpp*, che usa delle funzioni della libreria matematica;
- deve contenere la direttiva:
`#include <cmath>`
- Alcune librerie C++ sono disponibili in tutte le implementazioni e contengono gli stessi sottoprogrammi.

3.11 Libreria standard

cstdlib

- ***abs(n)*** valore assoluto di *n*;
- ***rand()*** intero pseudocasuale compreso fra 0 e la costante predefinita **RAND_MAX**;
- ***srand(n)*** inizializza la funzione *rand()*.

cctype

Restituiscono un valore booleano

- ***isalnum(c)*** lettera o cifra;
- ***isalpha(c)*** lettera;
- ***isdigit(c)*** cifra;
- ***islower(c)*** lettera minuscola;
- ***isprint(c)*** carattere stampabile, compreso lo spazio;
- ***isspace(c)*** spazio, salto pagina, nuova riga, ritorno carrello, tabulazione orizzontale, tabulazione verticale;
- ***isupper(c)*** lettera maiuscola;

cmath

- funzioni trigonometriche (x è un double)
 - $\sin(x)$ seno di x ;
 - $\cos(x)$ coseno di x ;
 - $\tan(x)$ tangente di x ;
 - $\operatorname{asin}(x)$ arcoseno di x ;
 - $\operatorname{acos}(x)$ arcocoseno di x ;
 - $\operatorname{atan}(x)$ arcotangente di x
- funzioni esponenziali e logaritmiche
 - $\exp(x)$ e elevato alla x ;
 - $\log(x)$ logaritmo in base e di x ;
 - $\log_{10}(x)$ logaritmo in base 10 di x ;

3.11 Libreria standard

- altre funzioni (anche y è un double)
 - *fabs(x)* valore assoluto di x;
 - *ceil(x)* minimo intero maggiore o uguale a x;
 - *floor(x)* massimo intero minore o uguale a x;
 - *pow(x, y)* x elevato alla y;
 - *sqrt(x)* radice quadrata di x;