Fondamenti di Programmazione A
Appunti per le lezioni – *Gianfranco Rossi*

Controllo dati di input

Come indicato nel cap. 2.7.1 della dispensa,
il dato letto tramite l'operatore di estrazione `s >> v`, con `s` stream di input e `v` variabile di tipo `t`, deve essere una costante di tipo `t` sintatticamente corretta.
Altrimenti, il dato viene rifiutato (alla variabile `v` non viene assegnato alcun valore) e lo stream `s` viene impostato allo stato `failed`.
E' possibile verificare la presenza di questo stato tramite la funzione di libreria `fail`, che funziona nel modo seguente:

```
s.fail()
```

restituisce `true` se lo stream `s` si trova in uno stato `failed`, `false` altrimenti.

Esempio:

```
cout << "inserire un numero intero";
int n;
cin >> n;
if (cin.fail())
     cout << "errore nel dato di input" << endl;
else
     cout << "il dato inserito e' corretto" << endl;
```

Una volta individuato un errore nell'inserimento del dato, è frequente richiedere all'utente di inserire nuovamente il dato. Per far questo bisogna però provvedere prima a ripristinare lo stato corretto dello stream di input `s` tramite la funzione di libreria `clear`, nel modo seguente

```
s.clear()
```

Purtroppo questo non è ancora sufficiente. Infatti sullo stream di input è presente ancora il dato che l'operatore `>>` non è riuscito a leggere. Prima di ripetere la lettura bisogna pertanto rimuovere questo dato, altrimenti la lettura fallirebbe nuovamente (andando quindi in loop). Un modo comodo per ottenere lo “svuotamento” dello stream di input è tramite la funzione di libreria `ignore`, che funziona nel modo seguente:

```
s.ignore(n,delim)
```

estrae i caratteri dallo stream di input `s` e li scarta; l'estrazione termina quando sono stati estratti e scartati `n` caratteri o quando si è trovato il carattere delim (dipende da quale delle due condizioni avviene per prima); nell'ultimo caso anche il carattere delimitatore viene estratto.

Esempio:

```
cout << “inserire un numero intero”;
int n;
do  {
    cin >> n;
    if (cin.fail()) {
            cout << "errore - ripetere" << endl;
            cin.clear();
            cin.ignore(256,'\n');
    }
    else break;
}
while (true);
cout << “il dato inserito e' corretto” << endl;
```

(n.b. 256 è un numero arbitrario; in questo modo si fa l'ipotesi di poter leggere e scartare, per ogni lettura che sia fallita, al massimo 256 caratteri consecutivi).

Modi alternativi per il test dello stream.

Per verificare lo stato di uno stream è possibile usare direttamente il nome dello stream come condizione booleana.

Esempio:

```
...
int n;
cin >> n;
if (cin)
     cout << "il dato inserito e' corretto" << endl;
else
     cout << "errore nel dato di input" << endl;
```

(Questo è possibile perchè la classe `istream` fornisce una funzione che può convertire un oggetto di tipo `istream`, come ad es. `cin`, in un valore booleano; questa funzione viene chiamata quando lo `stream` (ad es. `cin`) appare in una posizione in cui è richiesta un'espressione boolena, come ad esempio nella condizione di un `if` o di un `while`.)

Un altro modo alternativo per ottenere lo stesso risultato è di controllare direttamente il risultato della valutazione dell'operatore `>>`. Siccome questo operatore restituisce come suo risultato uno stream (precisamente, lo stream modificato a seguito dell'estrazione del dato di input), è possibile testare lo stream risultante esattamente come nella prima soluzione alternativa.

Esempio:

```
...
int n;
if (cin >> n)
     cout << "il dato inserito e' corretto" << endl;
else
     cout << "errore nel dato di input" << endl;
```

L'uso della funzione `fail` risulta semanticamente più chiaro, ma spesso si preferiscono le soluzioni alternative perché più sintetiche

(a rigore l'uso degli stream come condizione di test risulta un po' più generale dell'uso del `fail` perchè permette di testare anche altre possibili cause di fallimento, come ad esempio errori di disco).