ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Tecnico superiore per i metodi e le tecnologie per lo sviluppo di sistemi software

- BACKEND SYSTEM INTEGRATOR

Unità Formativa (UF): **FONDAMENTI DI PROGRAMMAZIONE**

Docente: **VALENTINO ARMANDO**

Titolo argomento: **CIN E CONTROLLO DELLO STATO**

Input Standard in C++

In C++ l'input standard si realizza utilizzando l'oggetto cin:

in generale:

- **cin è** lo STANDARD INPUT;
  tipicamente (di default) quest'oggetto è associato alla **TASTIERA** del terminale da cui il programma è mandato in esecuzione; esso è di classe *istream* e appartiene al namespace *std*.

- **cout** è lo STANDARD OUTPUT e **cerr**, lo STANDARD ERROR; questi oggetti tipicamente (di default) sono collegati al terminale (**MONITOR**) da cui il programma è mandato in esecuzione; entrambi sono di classe *ostream* e appartengono al namespace *std*.

➢ L'oggetto *cin* preleva i caratteri dalla tastiera e i converte nel tipo richiesto dall'input saltando i CARATTERI SPECIALI, quali *spazi*, *tab* e *newline*

Esempio:

```
int a, c;
float b;
cin>>a>>b>>c;
cout<<"a="<<a<<endl;
cout<<"b="<<b<<endl;
cout<<"c="<<c<<endl;
cout<<"a+b="<<a+b<<endl;
```

se l'input è il seguente:
10 SPAZIO SPAZIO 12.67 TAB SPAZIO 13

L'output sarà:
a=10
b=12.67
c=13
a+b=22.67

➢ quando in uno stream di input ci sono dei caratteri alfanumerici, se per leggerli si utilizzano delle variabili di tipo char e/o stringa, i caratteri e le stringhe vengono prelevati saltando i CARATTERI SPECIALI, quali *spazi*, *tab* e *newline*.

Esempio:

```
char c;
string s1, s2, s3;
cin>>c>>s1>>s2>>s3;
cout<<"c="<<c<<endl;
cout<<"s1="<<s1<<endl;
cout<<"s2="<<s2<<endl;

cout<<"s3="<<s3<<endl;
```

se l'input è il seguente:
Mario SPAZIO SPAZIO Rossi TAB 01/05/1985

L'output sarà:
M
ario
Rossi
01/05/1985

Il problema degli errori di input e lo STREAM STATE

Quando l'utente inserisce un input numerico inaspettato, costituito da un valore non corrispondente al tipo della variabile con cui si legge, il programma va incontro a malfunzionamenti.
Vediamolo con un semplice esempio:

```
#include <iostream>
using namespace std;
int main(){
    int a;
    double b;
    cout<<"INSERISCI UN NUMERO INTERO: ";
    cin>>a;
    cout<<"INSERISCI UN NUMERO REALE: ";
    cin>>b;
    cout<<endl<<"a="<<a<<endl;
    cout<<"b="<<b<<endl;
    return 0;
}
```

Durante l'esecuzione del programma, ad esempio:

- se l'utente inserisce come primo input il numero reale **12.5**, subito dopo l'INVIO la seconda operazione di input da parte dell'utente viene saltata e si ottiene di visualizzare a video:

```
INSERISCI UN NUMERO REALE:
a=12
b=0.5
```

Questo succede perché nello stream di input, la prima *cin* legge fino all'ultimo carattere interpretabile come un intero (ossia il **2**), lasciando gli altri caratteri (ossia **.5**) nello stream di input;
subito dopo va in esecuzione la seconda *cin* che converte nel numero reale **0.5** i caratteri lasciati nello stream dalla *cin* precedente e successivamente vengono eseguite le operazioni di output con le due *cout*.
Sicuramente non è quello che volevamo!

- se l'utente, invece, inserisce come primo input la sequenza di caratteri: **12r**, dopo l'invio si ottiene di visualizzare a video:

```
INSERISCI UN NUMERO REALE:
a=12
b=0
```

Questo perché la prima *cin*, per lo stesso motivo di prima, legge fino al carattere **2**. Subito dopo va in esecuzione la seconda *cin* che non legge correttamente in quanto non trova corrispondenza di tipo fra quello che c'è nello stream (ossia il carattere **r**) e la variabile utilizzata per leggere con la *cin*. Sicuramente non è quello che volevamo!

Queste situazioni di errore andrebbero gestite e per farlo può essere utile leggere lo **stream state**. Ciascun (i/o)stream si trova in uno STATO memorizzato in un insieme di flag (valori booleani) all'interno dell'oggetto associato a quello stream. Gli errori e le condizioni non standard di uno stream, possono essere gestite in diversi modi testandone opportunamente lo stato.

L'oggetto *cin* utilizzato con l'operatore di lettura (>>) fornisce un valore di ritorno booleano:

- FALSE, quando l'operazione di input NON ha avuto successo: formato dell'input completamente errato (ad esempio nel caso di una *cin* su una variabile di tipo numerico, quando già il primo carattere dello stream non corrisponde ad un carattere valido, cioè interpretabile come parte di un input numerico) o è stato incontrato EOF, per cui la prossima operazione di input fallirà.
- TRUE, quando l'operazione di input ha avuto successo.

Pertanto, le istruzioni *cin>>…* e *!(cin>>…)* possono essere utilizzate dove è atteso un valore booleano, ad esempio in una *if()* o in un *while(),* per controllare e gestire eventuali errori di input non validi (vedi l'esempio in fondo all'articolo).

Tutti gli oggetti di un (i/o)stream, mettono a disposizione i seguenti metodi per recuperare il loro stato interno (stream state):

- **fail()** restituisce TRUE se si è verificato un errore nell'apertura dello stream, FALSE altrimenti;
- **good()** restituisce TRUE se l'operazione effettuata sullo stream è andata a buon fine, FALSE altrimenti;
- **bad()** restituisce TRUE se l'operazione effettuata sullo stream ha prodotto un errore irreversibile, FALSE altrimenti;
- **is_open()** restituisce TRUE se il file è aperto, FALSE altrimenti.

NOTA BENE !!
Attenzione, una volta che uno stream si porta in uno stato di errore, esso ci rimane finché i flag non sono esplicitamente resettati e in quel frattempo le operazioni di input su quello stream sono operazioni nulle. Quindi, prima di effettuare la successiva operazione di input, bisogna *resettare lo stato* dello stream utilizzando il metodo **clear()**.
Bisogna, inoltre, tener presente che quando un'operazione di input fallisce, dallo stream di input non viene rimosso alcun carattere, pertanto bisogna ripulire anche lo stream richiamando il metodo **ignore()**. Vediamo un semplice esempio:

```
#include <iostream>
using namespace std;
int main(){
    double n;
    cout<<"Inserisci un numero: ";
    while(!(cin>>n)){
        //si entra se cin incontra un input "completamente" non valido, es. la stringa
"a21"
        cin.clear();
        cin.ignore(80, '\n');
        cout<<"Errore, formato non valido. Devi inserire un numero"<<endl;
        cout<<"Inserisci un numero: ";
    }
    return 0;
}
```

P.S. (IMPORTANTE)
Nell'esempio precedente nel *while* non si entra in corrispondenza, per esempio, di un input di questo tipo: *21a*.
In questo caso, infatti, la *cin* leggerebbe il numero 21, restituendo TRUE e lasciando il catattere 'a' nello stream. Pertanto nell'esempio precedente in realtà **non si gestisce completamente il problema di un input numerico non valido**.
Una soluzione a questo problema viene proposta di seguito:

se si crea una funzione che legge lo stato dello stream possiamo verificare in quali condizioni per la gestione dell'input.

```
void StampaStatoStream(){
	cout << "STATO DELLO STREAM:"<<endl;
	cout <<"fail() = "<<cin.fail()<<endl;		//True se c'è un errore nell'apertura dello stream altrimenti FALSE
	cout <<"good() = "<<cin.good()<<endl;		//True se l'operazione dello stream è andata a buon fine altrimenti FALSE
	cout <<"bad() = "<<cin.bad()<<endl;		//True se l'operazione dello stream ha prodotto errore altrimenti FALSE
	cout <<"gcount() = "<<cin.gcount()<<endl<<endl;  // conteggia i caratteri ignorati con cin.ignore()
}

int main(){
 char car;
double n;
bool r;
// controllo input numerico con cin
  do{
	 cout << "inserisci  un numero reale: ";
	 r=true;
	cin >> n;
	cout <<"Senza reset dello stream "<<endl;
	StampaStatoStream();
       if(cin.fail() ){ // caso di input che è errato
		cout << "Errore! inserimento errato !!!! \n";
		cin.clear(); // resetta lo stato dello stream
			// uso di ignore() cancella dallo steam input 1000 caratteri fino a \n;
			//se lo uso senza parametri, es:cin.ignore(), cancella solo un carattere
		cin.ignore(1000,'\n');
		cout << "dopo il reset "<<endl;
		StampaStatoStream();
		r=false;
	}
		  }while (r == false);
  cout << endl <<"n= "<< n<<endl;
  printf("\n\n");
}
```

L'esecuzione del programma permette di vedere che a fronte di input completamente errati come *dfgh* oppure *d23,5* il programma gestisce bene l'errore, mentre a fronte di input parzialmente errato come *23.5dfg* il programma non gestisce bene l'errore.

Questo programma risolve tutti i problemi

```
int main(){
     char car;
     double n;
     bool r;
       // controllo input numerico con cin
    do{
          cout << "inserisci  un numero reale: ";
          r=true;
          cin >> n;
          if(cin.fail() ){ // caso di input che è errato
                cout << "Errore! inserimento errato !!!! \n";
                cin.clear(); // resetta lo stato dello stream
                // uso di ignore() cancella dallo stream input per 1000 caratteri fino a \n;
                //se lo uso senza parametri, es:cin.ignore(), cancella solo un carattere
                cin.ignore(1000,'\n');
                r=false;
          }else{     // caso di input parzialmente errato ma che cin lo prende corretto, es: 45.5abc prende 45.5, ma l'input è
errato
                // permette di ignorare i caratteri errati dopo l'input, es: 123abc, ignora i caratteri abc;
                cin.ignore(1000,'\n');
                // gcount() conteggia i caratteri ignorati es: 123abc, conteggia i caratteri dopo il numero 123
                if((cin.gcount()>1)){
                    cout << "Errore! inserimento errato !!!! \n";
                    r=false;
                }
          }
      }while (r == false);
       cout << endl <<"n= "<< n<<endl
}
```

L' if() controlla l'input completamente errato come per esempio: *asdf* oppure *12,5* oppure *df12,5*
L'else controlla l'input parzialmente errato come *12.5abc*
Quindi, se l'input e completamente errato viene eseguito l'if, se è parzialmente errato viene eseguito l'else.
Nel caso di input corretto comunque si passa da else, ma gcount() non è maggiore di 1 e non viene eseguito l'errore.

CONTROLLO INPUT NUMERO COME FUNZIONE

```
//controllo di inserimento di un numero
#include <iostream>
#include <string>
using namespace std;
bool controlloInput(){
		bool r=true;
		if(cin.fail() ){ // caso di input che è completamente errato
			cout << "Errore! inserimento errato !!!! \n";
			cin.clear(); // resetta lo stato dello stream
			// uso di ignore() cancella dallo steam input 1000 caratteri fino a \n;
			//se lo uso senza parametri, es:cin.ignore(), cancella solo un carattere
			cin.ignore(1000,'\n');
			r=false;
		}else{		// caso di input parzialmente errato ma che cin lo prende corretto,
				es: 123abc prende 123, ma l'input è errato
				// permette di ignorare i caratteri errati dopo l'input, es:
				123abc, ignora i caratteri abc;
			cin.ignore(1000,'\n');
				// gcount() conteggia i caratteri ignorati es: 123abc, conteggia i
				caratteri dopo il numero 123
			if((cin.gcount()>1)){
				cout << "Errore! inserimento errato !!!! \n";
				r=false;
			}
		}
		return r;
}

int main(){

		int n;
		do{
			// controllo input numerico con cin
		    cout << "inserisci  un numero intero: ";
		    cin >> n;
		}while (controlloInput() == false);
		cout << endl <<"Hai inserito n= "<< n<<endl;
}
```