Prima esercitazione per casa: 13 gennaio 2009

In questa esercitazione si deve usare i file e la gestione delle eccezioni. Queste nozioni sono spiegate con un esempio nel testo che segue. L'esercitazione consiste di un esercizio che riguarda la conversione di decimali in binario. Si richiede di scrivere pre e post condizioni e gli invarianti dei cicli.

esercizio

Si tratta di leggere da file una sequenza di caratteri cifra da '0' a '9' terminata dal carattere ';'. La sequenza va interpretata come un valore decimale. Quindi se si legge '3' '2' '8' ';' va costruito il valore `double` .328. Dopo aver costruito questo valore reale, si deve produrre la sua rappresentazione binaria. Visto che la rappresentazione binaria potrebbe richiedere anche un numero non finito di bit, si chiede di calcolare al massimo i primi 23 bit della rappresentazione (o meno quando bastano meno di 23 bit). Per .328 la risposta è 01010011111101111100111 che consiste di 23 bit. I bit della rappresentazione binaria vanno scritti come caratteri numerici sul file di output senza lasciare spazi tra un carattere ed il successivo. I bit vanno scritti nell'ordine "normale", cioè dal piú significativo al meno significativo.

Esempi di input/output corretti:

input	output
328;	01010011111101111100111
5;	1
2667;	01000100010001100111001

File ed eccezioni

Nell'esercizio appena assegnato si richiede di fare operazioni nuove, come leggere da file e scrivere su file. In realtà la cosa è molto semplice come mostra il seguente frammento di programma:

```
#include<iostream>
#include<fstream>
using namespace std;
main()

  try // apertura file
    {
      ifstream IN("input");
      ofstream OUT("output");
      if(! IN || ! OUT)
       throw (0);
```

```
..... resto del main
 }
  catch(int x){cout<<"problemi con i file"<< x <<endl;}
```

Per usare i file, basta aggiungere il comando di `#include<fstream>`, e, dentro al main, inserire le 2 dichiarazioni di `ifstream` e `ofstream` che nel nostro esempio dichiarano che volete accedere in input al file "input" (che si deve trovare nella stessa directory in cui "gira" il programma) e che volete accedere in output al file "ouput" (di nuovo nella directory attuale). Le dichiarazioni associano a questi file le variabili di tipo ifstream e ofstream chiamati IN e OUT, rispettivamente. Questi nomi serviranno ad eseguire le letture/scritture sui file. Chiariamo una volta per tutte che i nomi dei file così come quelli degli stream sono arbitrari e indipendenti tra loro.

Queste dichiarazioni si dice che **aprono** i file, ma può succedere che l'apertura fallisca. In particolare, può fallire l'apertura del file in input perché presuppone che il file prescelto ("input" nell'esempio) esista e sia nella directory corrente. Se il file in quella directory non c'è, l'operazione fallisce. Il *sistema* segnala il fallimento dell'operazione assegnando alla variabile IN il valore 0 (che rappresenta anche `false`). Quindi è facile, testando se IN è falso, controllare se l'operazione è riuscita o no.

L'apertura di un file di output è diversa: ha successo anche se il file prescelto ("output" nell'esempio) non esiste e in questo caso il file viene creato. Se invece esisteva già il suo contenuto viene eliminato in modo da prepararlo a ricevere il nuovo output. È quindi più difficile che questa operazione fallisca, ma può comunque succedere nel caso il sistema non abbia sufficiente spazio per la creazione del file. Il test si effettua nello stesso modo appena visto per l'input.

Nel frammento di programma dato prima, viene testato che l'apertura dei 2 file abbia successo. Se questo non fosse il caso, il programma non deve continuare. Questa è una situazione di **eccezione** (o di errore) che va gestita in modo appropriato ed il C++ ci offre delle istruzioni apposite per trattare in modo elegante le eccezioni. Le istruzioni sono descritte nel programma dato in precedenza e sono le seguenti: `try, throw` e `catch`. Il try delimita con la coppia di parentesi graffe che lo segue un blocco di istruzioni in cui potrebbero accadere eccezioni. Alla fine di questo blocco sta di sentinella l'istruzione `catch` (prendi) capace di "prendere" ogni eccezione sollevata nel blocco try che sollevi un valore intero. Infine l'istruzione di `throw` viene eseguita in caso di eccezione e solleva un valore che (sperabilmente) verrà preso da qualche `catch` che aspetta un valore di quel tipo. Nell'esempio, `catch(0);` viene eseguita se uno dei 2 file non si apre. In questo caso, l'intero 0 viene preso dal `catch` in fondo al programma che aspetta un intero e che stampa un messaggio d'errore appropriato. In questo esempio il valore 0 sollevato non interessa veramente dato che abbiamo un solo `throw` che solleva sempre 0. Comunque nell'output contenuto nel corpo del `catch` stampiamo anche `x` per mostrare che in questo blocco esiste una variabile `x` che avrà come R-valore il

valore sollevato dalla `throw` e catturato dalla `catch` (sempre 0 nel nostro esempio, ma in generale molti diversi valori saranno possibili). Dopo la stampa eseguita nel corpo del `catch`, l'esecuzione arriva alla fine del main e quindi il programma termina. Ci fossero altre istruzioni, esse verrebbero eseguite.

Maggiori dettagli sul trattamento delle eccezioni saranno dati nella dispensa. La vostra soluzione di entrambi gli esercizi deve usare il frammento di programma dato prima.

Supponendo che i file si aprano in modo regolare, resta solo da usarli. Vogliamo leggere caratteri e scrivere caratteri. Queste operazioni si possono realizzare nel seguente modo: data la dichiarazione `char X;` allora:

- per leggere da IN possiamo usare `IN >> X;` o equivalentemente `X=IN.get();`
- per scrivere su OUT possiamo usare `OUT << X;` o OUT.put(X);.

Una volta usati i file vanno **chiusi**. L'operazione mette a disposizione i file ad altri utenti. L'operazione di chiusura è semplicemente `IN.close();` e lo stesso per OUT.

Consegna

La data di consegna è il 19 gennaio 2009 (ore 23:59) ed il comando per consegnare è: `consegna prog1`. Questo comando va eseguito in una shell avendo come directory corrente la directory che contiene (SOLO) il file col programma sorgente da consegnare. Questo file si deve chiamare "esercizio.cpp" e deve leggere e scrivere dai file, rispettivamente, "input" ed "output" esattamente come descritto nella parte precedente. La pre- e la post-condizione del programma, ed anche l'invariante di ogni ciclo vanno scritti come commenti nel testo del programma.