Brevi note sul testing

Introduzione e scelta dei dati.
Un programma è corretto quando si comporta conformemente alla sua specificazione
La certezza sulla correttezza di un programma si ottiene solo dimostrandola con tecniche di porva analoghe a quelle utilizzate per dimostrare un teorema della matematica. Purtroppo la complessità di queste prove è tale che non possono essere automatizzate, e, se realizzate "a mano" o anche con il supporto di mezzi di calcolo, risultano molto onerose, in termini di risorse umane e di tempo. Nell'ambito di un'area di ricerca denominata "Metodi formali per la verifica dei programmi", si sono messi a punto e si continuano a studiare metodi e tecniche per ottenere strumenti sempre più affidabili e utilizzabili in pratica per provare, anche magari solo in parte, la correttezza dei programmi.
L'approccio che consente di ridurre drasticamente i tempi di verifica dei programmi consiste nell'anticipare il più possibile i controlli di correttezza alle prime fasi di progettazione dei programmi, in modo da produrre programmi corretti per costruzione. Comunque questi metodi per ora non hanno eliminato la necessità di compiere controlli di correttezza dei programmi attraverso un'attività di testing, cioè un'attività di controllo del codice prodotto attraverso la sua esecuzione su dati di input per i quali si conosce l'output, e quindi per i quali si può controllare il corretto comportamento del programma. Dunque con il testing non si prova che il programma è corretto, ma se ne controlla il comportamento in un certo numero di casi.
Se i casi di controllo sono ben scelti, possiamo dire che il programma si comporta bene in tutti i casi "assimilabili" a quelli su cui abbiamo eseguito il programma.
Difficilmente quindi un'attività di testing basata su dati casuali potrà darci delle indicazioni su altri dati che non su quelli su cui abbiano fatto il controllo, e per questo che non è la scelta consigliata. Inoltre i dati utilizzati per il testing devono essere conservati in modo da poter ripetere gli stessi test in caso di modifica del programma, per essere sicuri che le modifiche apportate non abbiamo introdotto nuovi errori. E' anche evidente che si debba automatizzare il più possibile la fase di testing, visto che in programmi un po' complessi ci si può trovare nella necessità di compiere migliaia o anche milioni di verifiche che possono essere effettuate in poche ore da un calcolatore.
Distinguiamo due approcci fondamentali al testing: il testing a scatola nera o funzionale, basati solo sulle specifiche e il testing basato sul codice. Nel primo caso la scelta dei dati di input significativi è basata solo sulla specifica della funzione nel secondo caso sono invece scelti sulla base del codice, secondo vari criteri, per esempio facendo in modo che ogni istruzione del programma sia eseguita almeno una volta.

Organizzazione del testing

Vediamo come possiamo organizzare i test seguendo per ora l'approccio a scatola nera.

Il test può esssere organizzato in modi diversi a seconda del tipo di applicazioni:

- utilizzando le funzioni del programma
- organizzando tabelle (vettori) per i dati di input
- preparando i dati di input su files

Molti esempi del primo tipo sono stati forniti nelle lezioni precedenti.

Un esempio del secondo tipo è stato fornito quando abbiamo introdotto la classe dizionario:

```
#define NUM_TESTS  sizeof(tests)/sizeof(struct casi_test)

struct casi_test{
  char *parola;
  int in;} tests[ ] = {"tu", 1}, {"noi", 0},{"verde",1},{"citta",1} };

/*mettiamo 1 se la parola sarà presente nel file, 0 altrimenti*/
```

Il principale vantaggio nell'utilizo del file rispetto alla tabella di dati è che il file è esterno al programma e può essere modificato con un semplice editor. La lettura del testo è un po' più complicata rispetto all'accesso alla tabella, ma questo è un tipico esempio di "trade-off" tra complessità di stesura e facilità di uso.