Parte 14

Defensive programming

[Joan Miró – Carnival of Harlequin, 1925]

Definizione di robustezza

- Un **software** si definisce **robusto** se si **comporta "bene"** in **condizioni di lavoro non previste** dallo sviluppatore
- Che si intende per **"bene"**?
  - non va in crash, non sbaglia calcoli
- Che si intende per **"condizioni non previste"**?
  - carico di lavoro eccessivamente elevato, input non conforme allo standard
- **Fragilità**: la non robustezza di un software

Come aumentare la robustezza

- La robustezza viene aumentata grazie a:
  - **semplicità**: comprensibilità dell'intero design per un essere umano
  - **trasparenza**: comprensibilità immediata di ciascun aspetto del software
- Più i programmi sono semplici e trasparenti, più in generale sono robusti
- Come si ottengono trasparenza e semplicità?
  - Progettando il software in maniera **modulare**

Legge di riparazione (1)

- Il concetto di robustezza si deve applicare soprattutto al design del **sistema di gestione degli errori** di un software
- **Legge della riparazione**: il software deve essere trasparente non solo nelle sue operazioni normali, bensì anche nella **gestione degli errori**
  - Mantenere la consistenza dei dati (evitare corruzioni in ogni caso)
  - Cercare di “riprendersi” sempre in seguito ad un errore
  - In caso di fallimento, bloccarsi immediatamente

Legge di riparazione (2)

Motivazioni alla necessità di un blocco immediato

1. Il mancato immediato arresto in seguito ad un errore grave rende molto più **difficile il processo di scoperta di un errore**
2. Il mancato arresto in seguito ad un errore grave può essere **fonte di corruzione dei dati o di conseguenze più gravi**

Defensive programming

- Per garantire la robustezza del software si applicano tecniche di **bug prevention** e **defensive programming**
- In particolare, le tecniche di **defensive programming** sono utilizzate tipicamente nel caso di software che potrebbero essere usati da utenti malintenzionati e/o il cui uso inappropriato potrebbe portare ad effetti catastrofici

Tecniche di defensive programming

- Scrivere un software perfetto è virtualmente impossibile
- Tuttavia, **poche tecniche di defensive programming, applicate rigorosamente,** possono aiutare ad avvicinare il codice all'ideale

Invarianti e asserzioni

- **Invariante:** espressione che deve essere invariabilmente vera nel momento in cui compare nel codice
- Non è sempre possibile **prevenire** violazioni degli invarianti nel codice
  - A volte la violazione dipende dall'input dell'utente
- Utilizziamo **asserzioni** per verificare eventuali violazioni degli invarianti
- **Asserzione**: affermazione relativa ad una **condizione che deve essere vera**

C++ assert

- Il **C/C++** fornisce in una libreria standard (**assert.h** e **cassert**) la funzione

  assert(expr);
- Se **expr != 0** l'esecuzione continua ininterrotta
- Altrimenti:
  - Un **messaggio** contenente l'espressione violata viene **stampato** insieme con il nome del file sorgente e il numero di riga
  - **Il programma abortisce**

Osservazione

Usare invarianti ed asserzioni presenta **due aspetti molto positivi**:

- *E' semplice e pratico*
- *E' un approccio sempre vincente:*
  - Se nessuna **assert** fallisce siamo più confidenti nella correttezza del codice
  - Se una **assert** fallisce:
    - Se c'è un errore nel codice lo troviamo e lo correggiamo
    - Se non ci sono errori nel codice, è sbagliato l'invariante (errore concettuale)

#define NDEBUG

- E' possibile disabilitare/abilitare le asserzioni attraverso la definizione della macro **NDEBUG**
  - *Disabilitare*: **#define NDEBUG**
  - *Abilitare*: **#undefine NDEBUG**
- **NOTA**:
  - Le asserzioni sono disabilitate se al momento dell'inclusione di **<cassert>** la macro **NDEBUG** è già definita
    - **Reminder**: per definire una macro si può usare l'opzione **-D** di **compilatore/make**
  - **<cassert>** deve essere (re)incluso dopo aver (ri)definito o meno **NDEBUG**

Esempio

- Un esempio di **uso delle asserzioni** e della macro **NDEBUG** per disattivare le **assert** è riportato in

  progetto_1file_ass/GSeq_1file_ass.cc