Programmazione e Strutture Dati (PR&SD)

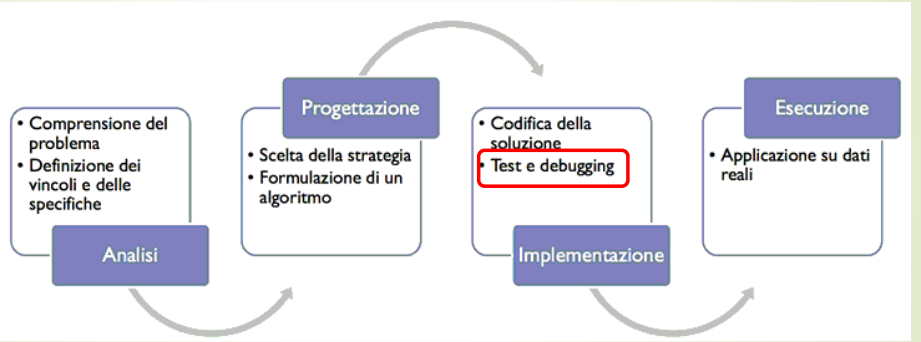
I° ANNO – Informatica UniSa

Testing dei programmi

… e uso dei file

Prof. Vittorio Fuccella – Ph.D.

Fasi dello sviluppo



Testing
Definizioni

- ***Testing***: esercitare il programma con dati di test per verificare che il suo comportamento sia conforme a quello atteso (definito nella specifica)
  - ✓ ***Oracolo***: è l'***output atteso***, quello che ci si aspetta il programma produca
  - ✓ ***Malfunzionamento***: comportamento del programma diverso da quello atteso
- In generale è impossibile dimostrare la correttezza di un qualunque programma …
  - ***Obiettivo del testing: individuare malfunzionamenti***

Testing
Definizioni

- Un malfunzionamento di un programma è causato da un difetto (errore, bug) nel codice
  - L'errore può essere introdotto in fase di analisi e specifica, di progettazione o di codifica
- ***Debugging***: Individuazione e correzione del difetto che ha causato il malfunzionamento
  - Più alta è la fase in cui si introduce il difetto, maggiore è la difficoltà nel rimuoverlo
  - La ricerca di un difetto può essere fatta inserendo nel codice sorgente punti di ispezione dello stato delle variabili
- NB: una volta corretto il difetto, rieseguire tutti i casi di test …

Testing
Classi di dati

- Testare il programma con tutti i possibili dati di test è impraticabile …
- **Obiettivo**: individuare classi di dati di test, selezionare un caso di test da ogni classe ed evitare casi di test ridondanti
- In questo corso non vedremo tecniche per scegliere i casi di test in maniera sistematica, ma useremo il buon senso …
- Documentare i casi di test e i risultati del testing …
- ***Test suite***: un insieme di casi di test per un programma

Testing
Esempio: ordinamento di un array

- Aspetti da tener conto nella scelta dei casi di test:
  - Il numero `n` di elementi dell'array:
    - Caso generale: array con n>1
    - Caso particolare n=1
  - La disposizione degli elementi:
    - Caso generale: array non ordinato
    - Array già ordinato in modo crescente
    - Array già ordinato in senso decrescente

Ordinamento Array

Test suite

- Test case 1 – TC1 (un solo elemento)
  - Array di input: 5
  - Oracolo: 5
- Test case 2 – TC2 (input ordinato in maniera crescente)
  - Array di input: 1 2 3 4 5 6 7 8 9
  - Oracolo: 1 2 3 4 5 6 7 8 9
- Test case 3 – TC3 (input ordinato in maniera decrescente)
  - Array di input: 10 9 8 7 6 5 4 3 2 1
  - Oracolo: 1 2 3 4 5 6 7 8 9 10
- Test case 4 – TC4 (non ordinato)
  - Array di input: 5 8 2 9 10 1 4 7 3 6 12 11
  - Oracolo: 1 2 3 4 5 6 7 8 9 10 11 12

Automatizzare il testing

- Per automatizzare il test si possono usare i file per leggere dati di input e scrivere i dati di output

Flussi (stream)

- In C, il termine ***stream*** indica una sorgente di input o una destinazione per l'output
- Molti programmi (piccoli) ottengono il loro input da uno stream (ad es. la tastiera) e lo inviano ad un altro stream (ad esempio il video)
- Programmi più grandi possono avere necessità di usare più stream
- Gli stream…
  - … rappresentano file memorizzati da qualche parte (hard disk o altri tipi di memoria a lungo termine)
  - … sono associati a periferiche (schede di rete, stampanti, etc)

Aprire (e chiudere) un File

```
FILE *fopen(const char *filename, const char *mode)
int fclose(FILE *stream)
```

- `filename` è il nome del file da aprire.
  - Può contenere informazioni riguardanti la sua posizione, come il drive o il percorso.
- `mode` è una "stringa di modalità" che specifica quali operazioni abbiamo intenzione di compiere sul file.
- *fclose()* ritorna 0 in caso di successo, EOF in caso di fallimento

Modalità di apertura

- Stringhe `mode` per file di testo:

Stringa	*Significato*
`"r"`	Apre il file in lettura (il file deve esistere)
`"w"`	Apre il file in scrittura (non è necessario che il file esista)
`"a"`	Apre il file in accodamento (non è necessario che il file esista)
`"r+"`	Apre il file in lettura e scrittura (il file deve esistere)
`"w+"`	Apre il file in lettura e scrittura - tronca il file se esiste
`"a+"`	Apre il file in lettura e scrittura - accoda se il file esiste

I/O da stream

```
char *fgets(char *s, int size, FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
int fprintf(FILE *stream, const char *format, ...)
```

- *fgets()* legge da *stream* fino al carattere newline (o finché *size-1* caratteri sono stati letti) e memorizza in *s*
  - Ritorna s in caso di successo, NULL in caso di errore o se si raggiunge la fine del file senza aver letto alcun carattere.
- *fscanf()* legge da stream fino ad un carattere di spazio e non lo memorizza
  - Restituisce il numero di dati letti e scritti con successo.
- *fprintf()* scrive su stream
  - Restituisce il numero di caratteri scritti; -1 in caso di errore

I/O su Stringhe

```
int sprintf(char *restrict buffer, const char *restrict format, ... );
int sscanf(const char *str, const char *format, ...)
```

- Le funzioni di sopra possono leggere e scrivere dati usando una stringa come se fosse un flusso.
  - `sprintf()` scrive caratteri in una stringa. Restituisce il numero di caratteri memorizzati
  - `sscanf()` legge i caratteri da una stringa (puntata dal primo argomento). Restituisce il numero di dati letti e scritti con successo.

Funzioni di Input

- Un esempio che usa `fgets` per ottenere una riga dell'input, poi passa la linea alla `sscanf` per ulteriori elaborazioni:

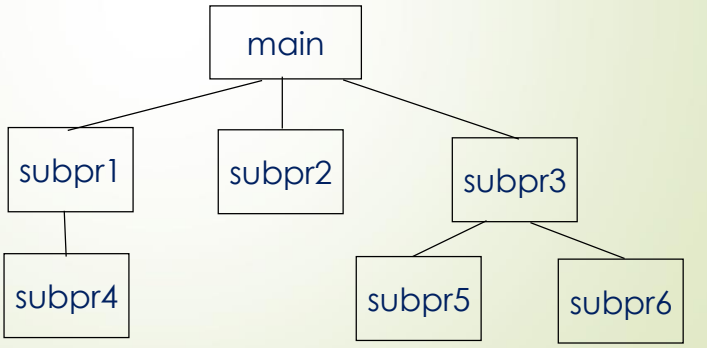
```
fgets(str, sizeof(str), stdin);
  /* legge una riga dell'input */
sscanf(str, "%d%d", &i, &j);
  /* estrae due interi */
```

Testing
Programmi con più funzioni

- ***Strategia big-bang***: integra il programma con tutti i sottoprogrammi e lo verifica nel suo insieme
  - Pessima strategia per programmi grandi: difficile localizzare la funzione contenente il difetto in caso di malfunzionamento …
- ***Strategie incrementali***: testare e integrare un sottoprogramma alla volta, considerando la struttura delle chiamate tra sottoprogrammi (architettura del programma)
  - **Bottom-up**
  - **Top-down**
  - **Sandwich**
  - **…**

Programmi con più funzioni
Strategia bottom-up

- verificare prima i sottoprogrammi terminali (più in basso) e poi via via quelli di livello superiore …
  - **un sottoprogramma può essere verificato se tutti i sottoprogrammi che usa (chiama) sono stati verificati**



Strategia bottom-up e driver

- Per ogni sottoprogramma da verificare è necessario costruire un programma main (detto ***driver***) che:
  - acquisisce i dati di ingresso necessari al sottoprogramma;
  - invoca il sottoprogramma passandogli i dati di ingresso e ottenendo i dati di uscita;
  - visualizza i dati di uscita del sottoprogramma
- … usare la specifica del sottoprogramma per individuare i casi di test …

… Approfondimenti più avanti nel corso …

Automatizzare il test

Esempio: ordinamento array

- Usiamo 2 file per dati di input
  - Un file "input.txt" contenente gli elementi dell'array di input
  - Un file "oracle.txt" contenente gli elementi dell'array ordinato
- … e in output:
  - Un file "output.txt" risultante dall'esecuzione del programma (output effettivo)
  - … oltre ad una indicazione dell'esito del test (PASS / FAIL)

Dati di test

input.txt

```
5
1 2 3 4 5 6 7 8 9
10 9 8 7 6 5 4 3 2 1
5 8 2 9 10 1 4 7 3 6 12 11
```

oracle.txt

```
5
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10 11 12
```