



Corso di Laurea in Informatica e Tecnologie per la Produzione del Software (Track B) - A.A. 2017/2018

Laboratorio di Informatica

Testing

docente: Veronica Rossano

veronica.rossano@uniba.it

Slides ispirate ai contenuti proposti
dal prof. Corrado Mencar, Grade.

Testing

- **Testing** = fase di **verifica** **sistematica** della **correttezza** di un software
- Parte integrante dei processi di sviluppo del software

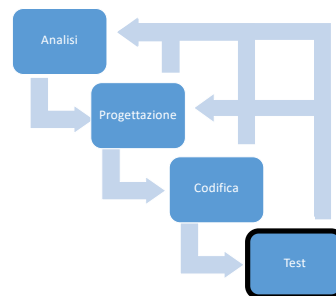
13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

2

Testing

- **Testing** = fase di **verifica** **sistematica** della **correttezza** di un software
- Parte integrante dei processi di sviluppo del software



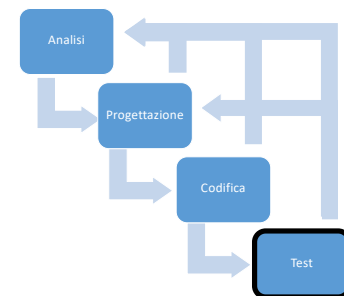
13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

3

Testing vs. Debugging

- **Testing** = fase di **verifica** **sistematica** della **correttezza** di un software
- **Debugging** = attività di rimozione di eventuali problemi **emersi dopo il testing**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

4

Testing

- **La verifica della correttezza** non è l'unico aspetto si può valutare con il testing del programma

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

5

Testing

- **La verifica della correttezza** non è l'unico aspetto si può valutare con il testing del programma
- **Esistono altre metodologie** di testing finalizzate a valutare
 - **Prestazioni:** il codice svolge i suoi compiti in un lasso di tempo sufficiente?
 - **Usabilità:** l'interazione con l'utente avviene in modo chiaro ed adeguato?
 - **Portabilità del Codice:** il codice è eseguibile su macchine diverse con minimo sforzo?
 - **Accettazione:** il codice risponde perfettamente alle richieste dell'utente finale?
- In questa sede ci concentreremo **solo sulle metodologie per la verifica della correttezza**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

6

Definizione dei Casi di Test

- Il problema principale che si affronta nel **testing** del programma è la **definizione dei casi di test**
 - Situazioni in cui il programma può presentare degli errori

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

7

Definizione dei Casi di Test

- Il problema principale che si affronta nel **testing** del programma è la **definizione dei casi di test**
 - Situazioni in cui il programma può presentare degli errori
- **Complessità del problema**
 - **Teorema di Howden:** Non esiste un algoritmo che, dato un programma P qualsiasi, generi un piano di test ideale e finito.
 - **Tesi di Dijkstra:** Il test di un programma può rilevare la presenza di malfunzionamenti, **ma non dimostrarne l'assenza.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

8

Metodologie di Testing

- Le metodologie di testing si dividono a grandi linee in due classi
- **Test Black Box (test funzionale)**
- **Test White Box (test strutturale)**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

9

Metodologie di Testing

- Le metodologie di testing si dividono a grandi linee in due classi
- **Test Black Box (test funzionale)**
 - Verifica la correttezza del programma **valutando l'output prodotto, senza entrare nel merito** del come il risultato è ottenuto.
- **Test White Box (test strutturale)**
 - Verifica la correttezza del programma **analizzando la strutturazione del codice sorgente**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

10

Metodologie di Testing

- Le metodologie di testing si dividono a grandi linee in due classi
- **Test Black Box (test funzionale)**
 - Verifica la correttezza del programma **valutando l'output prodotto, senza entrare nel merito** del come il risultato è ottenuto.
- **Test White Box (test strutturale)**
 - Verifica la correttezza del programma **analizzando la strutturazione del codice sorgente**

Nel primo caso, la selezione dei casi di test dipende **dai possibili valori di input/output**, nel secondo **dipendono dalla struttura del programma**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

11

Testing Black Box

- **Test Black Box (test funzionale)**
 - Verifica la correttezza del programma **valutando l'output prodotto, senza entrare nel merito** del come il risultato è ottenuto.
 - Si parla di «**verifica dei risultati attesi**»
 - La scelta dei casi di test dipende dai possibili input / output del programma

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

12

Testing Black Box

- **Test Black Box (test funzionale)**
 - Verifica la correttezza del programma **valutando l'output prodotto, senza entrare nel merito** del come il risultato è ottenuto.
 - Si parla di «**verifica dei risultati attesi**»
 - La scelta dei casi di test dipende dai possibili input / output del programma
- **Esistono tre tecniche principali**
 - Verifica delle **condizioni limite**
 - Definizione di **Classi di Equivalenza**
 - **Error Guessing**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

13

Testing Black Box – Verifica delle Condizioni Limite

- La maggior parte dei bug si verificano **in corrispondenza dei limiti**
 - **Cicli**: cosa succede se non entra nel ciclo? Esce correttamente dal ciclo?
 - **Array**: cosa succede se l'array è vuoto? Cosa succede se inserisco un indice dell'array non valido?
 - **Input**: cosa succede se l'input è nullo? Cosa succede se è zero? Cosa succede se non è valido?
 - **Stream**: Cosa succede se il file non esiste? Se il disco è pieno?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

14

Testing Black Box – Verifica delle Condizioni Limite

- La maggior parte dei bug si verificano **in corrispondenza dei limiti**
 - **Cicli**: cosa succede se non entra nel ciclo? Esce correttamente dal ciclo?
 - **Array**: cosa succede se l'array è vuoto? Cosa succede se inserisco un indice dell'array non valido?
 - **Input**: cosa succede se l'input è nullo? Cosa succede se è zero? Cosa succede se non è valido?
 - **Stream**: Cosa succede se il file non esiste? Se il disco è pieno?
- Prima di verificare le condizioni limite bisogna analizzare il programma e **capire quali sono i valori limite!**
 - **Occorre immaginare le condizioni limite e documentarle**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

15

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Cosa fa questo programma?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

16

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Cosa fa questo programma?

Questo programma tenta di leggere una sequenza di caratteri da un file e li memorizza in un array fino a quando viene letta una newline o si raggiunge la dimensione massima MAX>0

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

17

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

18

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

- Tipicamente le condizioni limite corrispondono a:
- Input vuoto
- Input «pieno» (numero massimo di caratteri disponibili, es.)
- Input errato
- Condizioni di uscita dai cicli

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

19

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto («\n») (input nullo)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

20

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

21

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)
3. La lunghezza dell'input è uguale a MAX (*input massimo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

22

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)
3. La lunghezza dell'input è uguale a MAX (*input massimo*)
4. La lunghezza dell'input è maggiore o uguale di MAX+1 (*minimo input errato*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

23

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)
3. La lunghezza dell'input è uguale a MAX (*input massimo*)
4. La lunghezza dell'input è maggiore o uguale di MAX+1 (*minimo input errato*)
5. L'input non contiene '\n' (*assenza di condizione di uscita dal ciclo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

24

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Quali sono le condizioni limite?

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)
3. La lunghezza dell'input è uguale a MAX (*input massimo*)
4. La lunghezza dell'input è maggiore o uguale di MAX+1 (*minimo input errato*)
5. L'input non contiene \n (*assenza di condizione di uscita dal ciclo*)

Prima di testare il programma è necessario pensare a quali possono essere le condizioni limite per quel programma

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

25

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Testiamo il programma nei casi limite:

1. L'input è vuoto (*input nullo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

26

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```



Testiamo il programma nei casi limite:

1. L'input è vuoto (*input nullo*)

BUG

In caso di input vuoto («\n») non entrerebbe nel ciclo e avremmo un errore sull'indice del vettore (i=-1)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

27

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Testiamo il programma nei casi limite:

2. Max==1 (*input minimo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

28

Verifica delle Condizioni Limite - Esempio

```

int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';

```

Testiamo il programma nei casi limite:

2. Max=1 (*input minimo*)

BUG

Al primo ciclo **MAX-1** sarebbe uguale a 0, dunque la condizione $i < 0$ non sarebbe verificata e non entrerebbe nel ciclo. **Quindi non può leggere file contenenti un solo carattere.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

29

Verifica delle Condizioni Limite - Esempio

```

int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';

```

Testiamo il programma nei casi limite:

3. La lunghezza dell'input è uguale a MAX (*input massimo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

30

Verifica delle Condizioni Limite - Esempio

```

int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';

```

Testiamo il programma nei casi limite:

3. La lunghezza dell'input è uguale a MAX (*input massimo*)

BUG

La condizione di uscita dal ciclo ci fa uscire troppo presto, **quindi perdiamo l'ultimo carattere**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

31

Verifica delle Condizioni Limite - Esempio

```

int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';

```

Testiamo il programma nei casi limite:

4. La lunghezza dell'input è superiore a MAX (*minimo input errato*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

32

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Testiamo il programma nei casi limite:

4. La lunghezza dell'input è superiore a MAX (*minimo input errato*)

BUG

Ovviamente vengono persi tutti i caratteri in eccesso.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

33

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Testiamo il programma nei casi limite:

5. L'input non contiene \n (*assenza di condizione di uscita dal ciclo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

34

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Testiamo il programma nei casi limite:

5. L'input non contiene \n (*assenza di condizione di uscita dal ciclo*)

BUG

Ci possono essere problemi in uscita dal ciclo, perché uscirebbe dal ciclo solo quando $i \geq \text{MAX}$

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

35

Verifica delle Condizioni Limite - Esempio

```
int i = 0;
char s[MAX];
while ((s[i] = fgetc(file)) != '\n' && i < MAX-1)
    i++;
s[--i] = '\0';
```

Prima di testare il programma è necessario **pensare a quali possono essere le condizioni limite per quel programma, e valutarle tutte.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

36

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

37

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

Verifichiamo nuovamente le condizioni limite

1. L'input è vuoto (*input nullo*)
2. Max==1 (*input minimo*)
3. La lunghezza dell'input è uguale a MAX (*input massimo*)
4. La lunghezza dell'input è maggiore o uguale di MAX+1 (*minimo input errato*)
5. L'input non contiene \n (*assenza di condizione di uscita dal ciclo*)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

38

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

Verifichiamo nuovamente le condizioni limite

1. L'input è vuoto (*input nullo*)

Entra nel ciclo (perché **stop==0**) ed esce subito dopo, perché **stop** diventa pari a 1 (condizione **s[i] == '\n'**)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

39

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

Verifichiamo nuovamente le condizioni limite

2. Max==1 (*input minimo*)

Entra correttamente nel primo ciclo (**i < MAX == true**)
E legge il primo carattere
Esce dal ciclo
Sostituisce il carattere letto con il terminatore

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

40

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

Verifichiamo nuovamente le condizioni limite
 3. La lunghezza dell'input è uguale a MAX (*input massimo*)
 Entra correttamente nel primo ciclo ($i < MAX == \text{true}$)
 Legge tutti i caratteri e incrementa i ad ogni passaggio
 Esce dal ciclo quando $i == MAX$
 Inserisce il terminatore come ultimo carattere

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

41

Verifica delle Condizioni Limite - Esempio

Riscrivendo il codice usando uno stile più leggibile, si eviterebbe l'errore del codice precedente

```
int i = 0;
char s[MAX];
int stop = 0;
while (!stop && i < MAX) {
    s[i] = fgetc(file);
    stop = (s[i] == '\n');
    i++;
}
s[--i] = '\0';
```

Verifichiamo nuovamente le condizioni limite
 I casi 4 e 5 continuano a presentare errori.

In questo caso è normale che ciò avvenga perché sono errori dipendenti dall'input e non da errata implementazione. Si può però utilizzare la programmazione difensiva per gestire correttamente anche quei casi.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

42

Verifica delle Condizioni Limite - Esempio

- **Testare prima:**

- Le parti (componenti/casi) più semplici
- Le parti più frequentemente utilizzate

- **Esempio: Ricerca binaria**

- Ricerca in un array vuoto
- Ricerca in un array con un solo elemento
- Ricerca in un array che contiene l'elemento cercato
- Ricerca in un array che NON contiene l'elemento cercato
- Ricerca in un array dove l'elemento cercato è il primo
- Ricerca in un array dove l'elemento cercato è l'ultimo
- etc.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

43

Testing Black Box – Classi di Equivalenza

- **Possiamo testare un algoritmo con tutti i possibili valori?**

- **Ovviamente no**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

44

Testing Black Box – Classi di Equivalenza

- Possiamo testare un algoritmo con tutti i possibili valori?
 - Ovviamente no
- In questo particolare contesto, **una classe di equivalenza è un insieme di valori di input per i quali l'algoritmo si comporta in modo analogo**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

45

Testing Black Box – Classi di Equivalenza

- Possiamo testare un algoritmo con tutti i possibili valori?
 - Ovviamente no
- In questo particolare contesto, **una classe di equivalenza è un insieme di valori di input per i quali l'algoritmo si comporta in modo analogo**
 - **Esempio:** l'algoritmo che ci dice se l'esame è stato superato o meno, si comporta in modo analogo per tutti i valori compresi tra 18 e 30 → **analogo = restituisce lo stesso risultato**
 - Allo stesso modo l'insieme dei valori compresi tra 0 e 17 si comporta in modo altrettanto analogo.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

46

Testing Black Box – Classi di Equivalenza

- Possiamo testare un algoritmo con tutti i possibili valori?
 - Ovviamente no
- In questo particolare contesto, **una classe di equivalenza è un insieme di valori di input per i quali l'algoritmo si comporta in modo analogo**
 - **Esempio:** l'algoritmo che ci dice se l'esame è stato superato o meno, si comporta in modo analogo per tutti i valori compresi tra 18 e 30 → analogo = restituisce lo stesso risultato
 - Allo stesso modo l'insieme dei valori compresi tra 0 e 17 si comporta in modo altrettanto analogo.
 - **L'algoritmo ha (almeno) due classi di equivalenza.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

47

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

48

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```

Classe 1
valori «validi»

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

49

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```

Classe 1
valori «validi»

Classe 2
valori «validi» per
esame non superato

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

50

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```

Classe 1
valori «validi»

Classe 2
valori «validi» per
esame non superato

Classe 3
valori «non validi»

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

51

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```

Classe 1
valori «validi»

→ [18 ... 30]

Classe 2
valori «validi» per
esame non superato

→ [0 ... 17]

Classe 3
valori «non validi»

→ Valori > 0
Valori > 30
Valori non
numerici

13/05/19

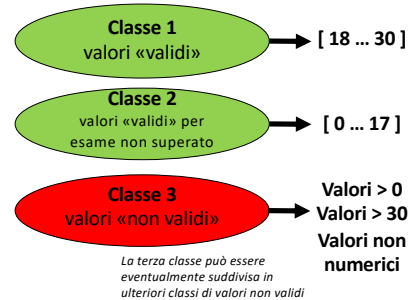
Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) – Università degli Studi di Bari – A.A. 2018/2019

52

Testing Black Box – Classi di Equivalenza

```
int main ( ) {
    int voto = 0;
    printf("Inserisci voto:");
    scanf("%d",&voto);

    if ( voto > 18 )
        printf("Esame Superato");
}
```



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

53

Testing Black Box – Classi di Equivalenza

- Come progettare i test basati su classi di equivalenza?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

54

Testing Black Box – Classi di Equivalenza

- Come progettare i test basati su classi di equivalenza?
- Si definiscono le classi di equivalenza (e magari si assegna un nome, esempio «**Superato**» e «**Non Superato**»)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

55

Testing Black Box – Classi di Equivalenza

- Come progettare i test basati su classi di equivalenza?
- Si definiscono le classi di equivalenza (e magari si assegna un nome, esempio «**Superato**» e «**Non Superato**»)
- Per ogni classe di equivalenza, si definisce un valore di esempio che appartiene al range
 - Ad esempio, per la classe «Superato», 22 è un valore valido mentre 10 è un valore valido per la classe «Non Superato»
 - Si testa l'algoritmo con un valore valido e con un valore non valido, e si **verifica che l'esito sia quello atteso.**
 - **Se l'esito non è quello atteso, bisogna capire quale sia il bug.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

56

Testing Black Box – Classi di Equivalenza

Classe	Descrizione	Valore di Test	Esito del Test (OK/NO)
1	Valori validi (esame superato)	22	OK
2	Valori validi (esame non superato)	16	OK
3a	Valori non validi (valori > 30)	40	NO (Stampa «Esame Superato»)
3b	Valori non validi (valori < 0)	-5	NO (Non Stampa Messaggio di Errore)
3c	Valori non validi (valori non numerici)	a	NO (Crash)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

57

Testing Black Box – Classi di Equivalenza

Classe	Descrizione	Valore di Test	Esito del Test (OK/NO)
1	Valori validi (esame superato)	22	OK
2	Valori validi (esame non superato)	16	OK
3a	Valori non validi (valori > 30)	40	NO (Stampa «Esame Superato»)
3b	Valori non validi (valori < 0)	-5	NO (Non Stampa Messaggio di Errore)
3c	Valori non validi (valori non numerici)	a	NO (Crash)

Normalmente siamo abituati a testare il codice SOLTANTO con i valori validi!
E' fondamentale invece verificare il comportamento del programma davanti ai valori non validi, perché tipicamente è proprio lì che sono presenti i bug!

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

58

Programmazione difensiva

L'utilizzo della programmazione difensiva è un metodo efficace per gestire i casi limite e i comportamenti anomali. **Capire prima i casi limite e scrivere del codice che prevenga eventuali valori errati può aiutare a limitare i bug del programma**

```
if (vote < 0 || vote > MAX_VOTE) {
    exam = NOT_VALID;
} else if (vote <= 18) {
    exam = NOT_PASSED;
} ...
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

59

Esercizio

```
/** Restituisce la media aritmetica di un array.
 * @param a double di len_a elementi;
 * @return media aritmetica degli elementi
 dell'array */

double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
    return sum / len_a;
}
```

- Quali sono i casi limite?
- Quali sono le classi di equivalenza?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

60

Esercizio

```
/** Restituisce la media aritmetica di un array.
 * @param a double di len_a elementi;
 * @return media aritmetica degli elementi
 dell'array */

double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
    return sum / len_a;
}
```

- Quali sono i casi limite?

- **len_a=0** (array vuoto)

- Produrrebbe divisione per zero

(pensare ad altri casi limite)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

61

Esercizio

```
/** Restituisce la media aritmetica di un array.
 * @param a double di len_a elementi;
 * @return media aritmetica degli elementi
 dell'array */

double avg(double a[], int len_a) {
    int i;
    double sum = 0.0;
    for (i=0; i < len_a; i++) {
        sum += a[i];
    }
    return sum / len_a;
}
```

- Quali sono le classi di equivalenza?

- Le classi di equivalenza dipendono a grandi linee dai valori limite, in questo caso definiamo due classi di equivalenza legate alla lunghezza dell'array (**Lunghezza_Valida** e **Lunghezza_Non_Valida**)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

62

Testing Black Box – Random Guessing

- **Non è una tecnica vera e propria**
 - Dipende dall'intuito di chi effettua i test
 - Tradotto: «**azzeccare a caso**»
- **Influenzata dall'esperienza**
 - I bug e i casi limite che possono verificarsi molto spesso si ripetono, anche in programmi molto diversi

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

63

Testing White Box

- **Test White Box (test strutturale)**
 - Verifica la correttezza del programma **analizzando la strutturazione del codice sorgente**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

64

Testing White Box

- **Test White Box (test strutturale)**
 - Verifica la correttezza del programma **analizzando la strutturazione del codice sorgente**
 - Sono basati sul concetto di «**copertura**»
 - La selezione dei casi di test non dipende dai valori di input/output, ma serve a verificare il corretto funzionamento del programma **in tutti i possibili «percorsi» che possono verificarsi**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

65

Testing White Box

- **Test White Box (test strutturale)**
 - Verifica la correttezza del programma **analizzando la strutturazione del codice sorgente**
 - Sono basati sul concetto di «**copertura**»
 - La selezione dei casi di test non dipende dai valori di input/output, ma serve a verificare il corretto funzionamento del programma **in tutti i possibili «percorsi» che possono verificarsi**
 - Esempio: c'è una istruzione **if...else**, devo verificare che il programma funzioni correttamente in entrambi i rami del flow chart. Oppure verificare tutti i casi di una istruzione **switch**
 - **Esistono dei criteri per selezionare solo alcuni tra i possibili percorsi.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

66

Testing - Accorgimento Generale

- A prescindere dalla metodologia scelta, procedere con la cosiddetta «**verifica incrementale**»
- **Test di pari passo con l'implementazione**
- **Test di unità elementari**
 - Una procedura o funzione
 - **Un blocco significativo di procedura**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

67

Introduzione a CUnit

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

68

CUnit

- Framework di **unit test** per il linguaggio C
- Home page: <http://cunit.sourceforge.net/index.html>
- Libreria che va inclusa in ogni progetto Eclipse CDT che intende avvalersene
- Guida di installazione <http://tiny.cc/cunit-installazione>

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

69

Unit test

- **Cos'è?**
 - Tecnica di progetto e sviluppo del software
- **A cosa serve?**
 - A ottenere evidenza che le singole unità software sviluppate siano corrette e pronte all'uso
- **Unità software?**
 - un programma, una funzione, una procedura
- **Come si fa?**
 - Si scrivono degli **unit test (o casi di test)** rappresentanti una sorta di “contratto scritto” che la porzione di codice testata deve assolutamente soddisfare

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

70

xUnit test framework

- In principio fu **JUnit per Java**
 - Creato da Kent Beck e Erich Gamma
- E' stato portato verso innumerevoli altri linguaggi (C/C++, C#, PHP, Python, JavaScript, Ruby, ...) dando vita all'ecosistema dei framework di tipo xUnit
- Ha dato vita al Test-Driven Development (TDD – sviluppo guidato dal test)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

71

Struttura del framework CUnit

- Il framework è basato su due concetti
 - **Test Registry**
 - **Test Suite**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

72

Struttura del framework CUnit

- Il framework è basato su due concetti
 - **Test Registry**
 - Insieme di tutti le test suite che vogliamo effettuare (il cosiddetto «piano di test»). Include una o più test suite
 - **Test Suite**
 - Insieme di tutti i test che riguardano un singolo metodo o una funzione del programma

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

73

Struttura del framework CUnit

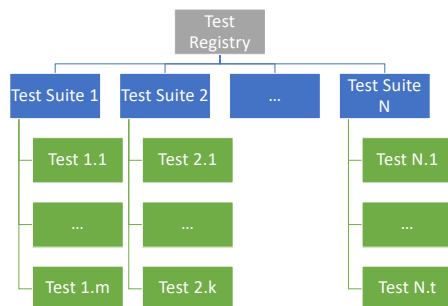
- Il framework è basato su due concetti
 - **Test Registry**
 - Insieme di tutti le test suite che vogliamo effettuare (il cosiddetto «piano di test»). Include una o più test suite
 - **Test Suite**
 - Insieme di tutti i test method che riguardano un singolo metodo o una funzione del programma
- **Riassumendo**
 - Dato un programma, definiamo i test che vogliamo effettuare. **Inseriamo i singoli test in una «test suite»** e inseriamo tutte le test suite **nel test registry**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

74

Struttura del framework CUnit

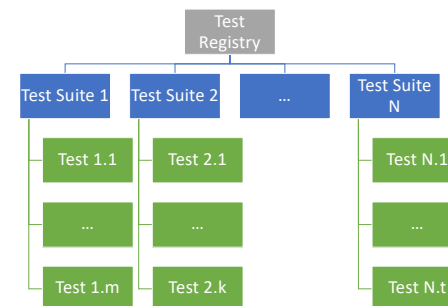


13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

75

Struttura del framework CUnit



13/05/19

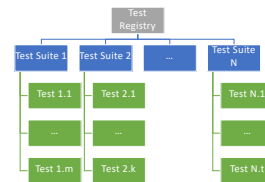
Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

76

L'ordine ha importanza!
 Le test suite sono eseguite
 nello stesso ordine di
 inserimento nel registry
 I test method sono eseguiti
 nello stesso ordine di
 inserimento nella suite

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

77

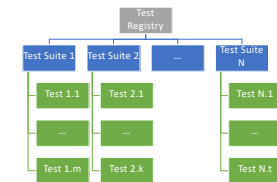
Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite definite**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

78

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

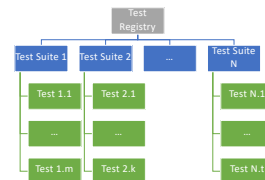
3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite definite**

5. Se necessario, ripeti i passi 3-4 per un'altra suite

6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

79

CUnit – Recap

Cosa sappiamo

- E' una libreria scritta in C per il testing dei progetti
- E' basata sulla definizione di vari **test method**, che si uniscono in una **test suite** (ogni test suite è legata a un «aspetto» che vogliamo testare) che a sua volta è legata a un **test registry**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

80

CUnit – Recap

Cosa sappiamo

- E' una libreria scritta in C per il testing dei progetti
- E basata sulla definizione di vari **test method**, che si uniscono in una **test suite** (ogni test suite è legata a un «aspetto» che vogliamo testare) che a sua volta è legata a un **test registry**

Cosa manca?

Come integrare CUnit nei nostri progetti?

Come scrivere i test method?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

81

Installazione CUnit

Credits: Fabio Calefato URL: <http://collab.di.uniba.it/fabio/guide/>

• Installazione CUnit test framework.

- Download: <https://tinyurl.com/cunit-labinf>
- Scaricate lo zip e scompattatelo in una cartella a scelta
 - es. C:\CUnit-2.1-0-winlib\CUnit-2.1-0
- All'interno della sottocartella include troverete un'altra cartella chiamata *CUnit*. Prendetela e copiatela all'interno della cartella C:\MinGW\include.
- Andate nella cartella C:\CUnit-2.1-0-winlib\CUnit-2.1-0\lib. Vi troverete due file, *libcunit.a* e *libcunit.dll.a*, che dovrete copiare in C:\MinGW\lib.
- Copiate il file libcunit.dll che si trova in C:\CUnit-2.1-0-winlib\CUnit-2.1-0\bin all'interno della cartella C:\MinGW\bin.


13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

82

Installazione CUnit

Credits: Fabio Calefato URL: <http://collab.di.uniba.it/fabio/guide/>

- Poiché la nostra applicazione avrà bisogno di eseguire casi di test in CUnit, è necessario informare il compilatore su dove trovare la libreria esterna.
- Per far questo, selezionate la cartella del progetto, premete il tasto destro e quindi scegliete **“Proprietà”**. Dalla finestra delle proprietà, scegliete **“C/C++ Build > Settings”**. Dopodiché, nel tab **“Settings”** scegliete la voce **“Libraries”** sotto la voce **“MinGW C Linker”**.
- Quindi cliccate sul tasto  per aggiungere la libreria. Per includere le librerie **“libcunit.a”** e **“libcunit.dll.a”** dovete soltanto scrivere **“cunit”** e **“cunit_dll”**.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

83

Step 0: Includere CUnit nel progetto

Quando preparate il modulo con i metodi di test ricordate di includere i file header in questo modo:

```
...
#include <stdio.h>
#include <stdlib.h>
#include "CUnit/Basic.h"
...
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

84

Ciclo di Esecuzione di uno Unit Test



1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

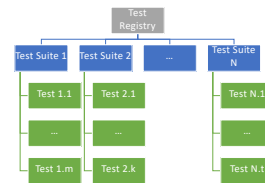
3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite** definite

5. Se necessario, ripeti i passi 3-4 per un'altra suite

6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

85

Step 1: Scrivere i test method

- Un metodo di test in CUnit si presenta sempre nella forma di **procedura senza parametri**
 - **Convenzione di nomenclatura**
 - `void test_<nomeFunzione>(void)`
 - **Esempio:** `void test_xyz(void)`
 - xyz è il nome del metodo che vogliamo testare
 - **Esempio:** voglio testare una funzione che calcola la media, chiamata «media»
 - `test_media(void)`

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

86

Step 1: Scrivere i test method

- Un metodo di test in CUnit si presenta sempre nella forma di **procedura senza parametri**
 - `void test_xyz(void)`
 - xyz è il nome del metodo che vogliamo testare
 - Es. voglio testare una funzione che calcola la media, chiamata «media» → `test_media(void)`
- Cosa inseriamo nei test method?
 - Ciascun metodo di test **contiene al suo interno delle asserzioni**
 - **Cosa è una asserzione?**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

87

Step 1: Scrivere i test method

- Ciascun metodo di test **contiene al suo interno delle asserzioni**
 - **Cosa è una asserzione?**
 - Un'asserzione in un linguaggio di programmazione **è una funzione che verifica una condizione logica e restituisce:**
 - **Vero**, se l'asserzione è rispettata
 - **Falso**, altrimenti

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

88

Step 1: Scrivere i test method

- Ciascun metodo di test **contiene al suo interno delle asserzioni**
 - Cosa è una asserzione?**
 - Un'asserzione in un linguaggio di programmazione **è una funzione che verifica una condizione logica e restituisce:**
 - Vero**, se l'asserzione è rispettata
 - Falso**, altrimenti

- Sintassi**

- CU_ASSERT(espressione)

• **Esempio:** CU_ASSERT(0==0) = **true**

• **Esempio:** CU_ASSERT(0==1) = **false**



Esempio di **Asserzione**
(Equivale a dire «asserisco che zero è uguale a zero»)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

89

Step 1: Scrivere i test method

- CUnit** include numerose tipologie di asserzioni. **Ad esempio.**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

90

Step 1: Scrivere i test method

- CUnit** include numerose tipologie di asserzioni. **Ad esempio.**
 - CU_ASSERT_EQUAL(espressione, valore) → Restituisce **true** se il valore calcolato è quello che indichiamo
 - CU_ASSERT_NOT_EQUAL(espressione, valore) → Restituisce **true** se il valore calcolato NON E' quello che indichiamo

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

91

Step 1: Scrivere i test method

- CUnit** include numerose tipologie di asserzioni. **Ad esempio.**
 - CU_ASSERT_EQUAL(espressione, valore) → Restituisce **true** se il valore calcolato è quello che indichiamo
 - CU_ASSERT_NOT_EQUAL(espressione, valore) → Restituisce **true** se il valore calcolato NON E' quello che indichiamo
- A cosa ci servono queste asserzioni?**
 - Le asserzioni vengono usate per verificare che le funzioni che implementiamo **restituiscano esattamente i valori corretti**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

92

Step 1: Scrivere i test method

- **CUnit** include numerose tipologie di asserzioni. **Ad esempio.**
 - `CU_ASSERT_EQUAL(espressione, valore)` → Restituisce **true** se il valore calcolato è quello che indichiamo
 - `CU_ASSERT_NOT_EQUAL(espressione, valore)` → Restituisce true se il valore calcolato NON E' quello che indichiamo
- **Esempio**
 - Supponiamo di avere una funzione `calcolaBMI` che calcola il BMI
 - `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25)`
 - Equivale a dire «Asserisco che il BMI di un individuo con altezza pari a 200 e peso pari a 100 è uguale a 25»

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

93

Step 1: Scrivere i test method

- **Esempi di Asserzioni**
 - `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25) == true`
 - `CU_ASSERT_EQUAL(media (3,5,7) , 5) == true`
 - `CU_ASSERT_NOT_EQUAL(media (3,5,7) , 0) == true`
 - `CU_ASSERT_EQUAL(maggiorenne (14,18) , 1) == false`
 - `CU_ASSERT_EQUAL(maggiorenne (20,18) , 1) == true`
 - `CU_ASSERT_EQUAL(promosso (18, 18) , 1) == ???`
 - `CU_ASSERT_NOT_EQUAL(promosso (15, 18) , 1) == ???`

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

94

Step 1: Scrivere i test method

- **Esempi di Asserzioni**
 - `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25) == true`
 - `CU_ASSERT_EQUAL(media (3,5,7) , 5) == true`
 - `CU_ASSERT_NOT_EQUAL(media (3,5,7) , 0) == true`
 - `CU_ASSERT_EQUAL(maggiorenne (14,18) , 1) == false`
 - `CU_ASSERT_EQUAL(maggiorenne (20,18) , 1) == true`
 - `CU_ASSERT_EQUAL(promosso (18, 18) , 1) == true`
 - `CU_ASSERT_NOT_EQUAL(promosso (15, 18) , 1) == true`

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

95

Step 1: Scrivere i test method

- **Esempi di Asserzioni**
 - `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25) == true`
 - In generale, data una funzione che vogliamo testare, dei valori di input e un valore atteso, e il formato per le asserzioni sarà il seguente
 - `CU_ASSERT_EQUAL(nomeFunzione (parametri) , valoreAtteso) == true`
 - I parametri con cui testare la funzione **sono le condizioni limite e le classi di equivalenza** individuate in precedenza
 - Il valore atteso è il comportamento che ci aspettiamo!

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

96

Step 1: Scrivere i test method

Asserzione	Significato
<code>CU_ASSERT(int espressione)</code> <code>CU_TEST(int espressione)</code>	Asserisce che espressione è TRUE (diverso da 0)
<code>CU_ASSERT_TRUE(valore)</code>	Asserisce che valore è TRUE (diverso da 0)
<code>CU_ASSERT_FALSE(valore)</code>	Asserisce che valore è FALSE (uguale a 0)
<code>CU_ASSERT_EQUAL(reale, atteso)</code>	Asserisce che reale == atteso
<code>CU_ASSERT_NOT_EQUAL(reale, atteso)</code>	Asserisce che reale != atteso
<code>CU_ASSERT_STRING_EQUAL(reale, atteso)</code>	Asserisce che le stringhe reale e atteso coincidono
<code>CU_ASSERT_STRING_NOT_EQUAL(reale, atteso)</code>	Asserisce che le stringhe reale e atteso differiscono

Asserzioni di base (CUnit)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

97

Step 1: Scrivere i test method

```
#define MAXINT 1000
void test_max(void) {
    CU_ASSERT_EQUAL(max(-MAXINT, +MAXINT), +MAXINT);
    CU_ASSERT_TRUE(max(+MAXINT, -MAXINT) == +MAXINT);
    CU_TEST(max(0, 0) == 0);

    // questa asserzione è sbagliata e fallisce
    CU_ASSERT_TRUE(max(5, 6) == 2);
}
```

Esempio di test method per la funzione `max(a, b)`

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

98

Step 1: Scrivere i test method

```
void test_factorial(void) {
    CU_ASSERT_EQUAL(factorial(4), 12); // fallisce
    CU_ASSERT(factorial(0) == 1);
    CU_TEST(factorial(1) == 1);
}
```

Esempio di metodo di test per la funzione `factorial(x)`

```
n! = 1      se n = 0
     n(n-1)! se n > 0
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

99

Step 1: Scrivere i test method - Recap

- Data una funzione, viene scritto un **test method** che ne verifichi il corretto comportamento
- **Ogni test method** contiene al suo interno una serie di **asserzioni**
- **Una asserzione** è una funzione che restituisce **vero o falso**
- Viene utilizzata per verificare che l'output prodotto dalla funzione sia **uguale (o meno) a quello atteso**
- Come definire le asserzioni?

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

100

Step 1: Scrivere i test method - Recap

- Come definire **le asserzioni**?
- L'esperienza è un fattore importante. In generale bisogna però riapplicare **le tecniche per la definizione dei casi limite** e per **l'individuazione delle classi di equivalenza**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

101

Step 1: Scrivere i test method - Recap

- Come definire **le asserzioni**?
- L'esperienza è un fattore importante. In generale bisogna però riapplicare **le tecniche per la definizione dei casi limite** e per **l'individuazione delle classi di equivalenza**
 - Si definisce un caso di test che verifichi il **comportamento «normale»** della funzione (es. `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25)`)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

102

Step 1: Scrivere i test method - Recap

- Come definire **le asserzioni**?
- L'esperienza è un fattore importante. In generale bisogna però riapplicare **le tecniche per la definizione dei casi limite** e per **l'individuazione delle classi di equivalenza**
 - Si definisce un caso di test che verifichi il **comportamento «normale»** della funzione (es. `CU_ASSERT_EQUAL(calcolaBMI (200,100) , 25)`)
 - Si utilizzano le tecniche viste in precedenza per aggiungere delle asserzioni **legate ai casi limite o alle classi di equivalenza**
 - Es. `CU_ASSERT_FALSE (promosso(39))` ← classe di equivalenza

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

103

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari



2. Crea il **test registry**

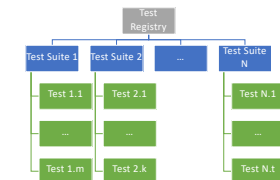
3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite** definite

5. Se necessario, ripeti i passi 3-4 per un'altra suite

6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

104

Step 2: Inizializzazione del Test Registry

L'inizializzazione del **test registry** è la prima operazione da effettuare

```

00
61 // *****
62 // TEST di UNITA'
63
64 int main() {
65     /* inizializza registro - e' la prima istruzione */
66     CU_initialize_registry();
67

```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

105

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

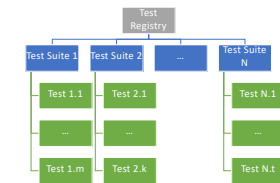
➔ 3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite** definite

5. Se necessario, ripeti i passi 3-4 per un'altra suite

6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

106

Step 3: Definizione della Test Suite

```

/* Aggiungi le suite al test registry */
CU_pSuite pSuite_A = CU_add_suite("Suite_A", init_suite_default, clean_suite_default);
CU_pSuite pSuite_B = CU_add_suite("Suite_B", init_suite_default, clean_suite_default);

```

- Una test suite è definita da:
 - Una descrizione testuale
 - Una procedura di inizializzazione (init)
 - Una procedura di pulizia (clean)
- Le test suite definite vengono aggiunte al test registry (l'ordine è rilevante!)

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

107

Step 3: Inizializzazione e Pulizia

• Le test suite devono essere **inizializzate** e **ripulite** prima e dopo l'uso

- I metodi non sono forniti da CUnit ma devono essere scritti dal programmatore

• **Perché?**

- Perché devono liberare le risorse allocate **specificatamente** per eseguire il caso di test
- Es. file, connessioni, etc.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

108

Step 3: Inizializzazione e Pulizia

```
// Alloca tutte le risorse necessarie all'esecuzione dei test
int init_suite_default(void) {
    return 0; // tutto ok
}

// dealloca tutte le risorse allocate all'inizializzazione
int clean_suite_default(void) {
    return 0; // tutto ok
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

109

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

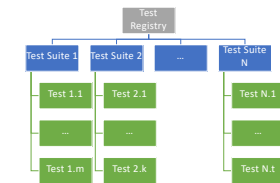
3. Crea la **test suite** e aggiungila al **test registry**

➔ 4. Aggiungi i **test method** alle **test suite** definite

5. Se necessario, ripeti i passi 3-4 per un'altra suite

6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

110

Step 4: Aggiungi i Test Method alle Test Suite

• Un **test method** viene aggiunto ad una **test suite** specificando:

- il puntatore **alla suite**
- una **descrizione testuale del test**
- il puntatore **al test method**

```
/* Aggiungi i test alle suite
 * NOTA - L'ORDINE DI INSERIMENTO E' IMPORTANTE
 */
CU_add_test(pSuite_A, "test of f1()", test_f1);
CU_add_test(pSuite_A, "test of f3()", test_f3);

CU_add_test(pSuite_B, "test of f4()", test_f4);
CU_add_test(pSuite_B, "test of f2()", test_f2);
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

111

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari

2. Crea il **test registry**

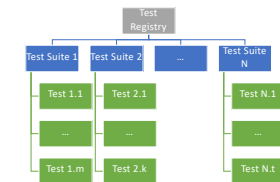
3. Crea la **test suite** e aggiungila al **test registry**

4. Aggiungi i **test method** alle **test suite** definite

5. Se necessario, ripeti i passi 3-4 per un'altra suite

➔ 6. Esegui il **test registry**

7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

112

Step 6: Esegui il Test Registry

- La procedura `CU_basic_run_tests` esegue tutte le suite del registry e mostra i risultati
- È possibile impostare il livello di “verbosità” dell’output

```
/* Esegue tutti i casi di test con output sulla console */
CU_basic_set_mode(CU_BRM_VERBOSE);
CU_basic_run_tests();
```

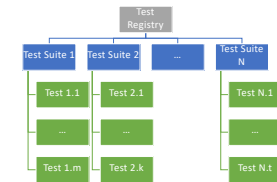
13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

113

Ciclo di Esecuzione di uno Unit Test

1. Scrivi tutti i **test method** necessari
2. Crea il **test registry**
3. Crea la **test suite** e aggiungila al **test registry**
4. Aggiungi i **test method** alle **test suite** definite
5. Se necessario, ripeti i passi 3-4 per un'altra suite
6. Esegui il **test registry**
7. Pulisci il **test registry**



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

114

Step 7: Pulisci il Test Registry

- Pulizia – dopo aver eseguito tutti i test nel registro
 - Procedura void `CU_cleanup_registry(void)`
 - Il `main()` termina con la return dell’eventuale codice di errore di CUnit

```
/* Pulisce il registro e termina lo unit test */
CU_cleanup_registry();

return CU_get_error();
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

115

Template – Utilizzo di CUnit

(1) Inclusione della libreria

```
#include <stdio.h>
#include "CUnit/Basic.h"
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

116

Template – Utilizzo di CUnit

(2) Metodi di
Inizializzazione e
Pulitura

```
/* Funzione di inizializzazione. */
int init_suite1(void) {
    return 0; // nessuna inizializzazione
}

/* Funzione di cleanup. */
int clean_suite1(void) {
    return 0; // nessun cleanup
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

117

Template – Utilizzo di CUnit

(3) Metodi di Test

```
/* Test method */
void test_fun1(void) {
    CU_FAIL("test method not implemented"); // metodi vuoti
}

/* Test method */
void test_fun2(void) {
    CU_FAIL("test method not implemented");
}

/* Test method */
void test_fun3(void) {
    CU_FAIL("test method not implemented");
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

118

Template – Utilizzo di CUnit

(4) Main del
programma

```
int main()
{
    CU_initialize_registry();
    CU_pSuite pSuite = CU_add_suite("Suite_1", init_suite1, clean_suite1);
    CU_add_test(pSuite, "test1", test_fun1);
    CU_add_test(pSuite, "test2", test_fun2);
    CU_add_test(pSuite, "test3", test_fun3);
    ...
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

119

Template – Utilizzo di CUnit

(5) Main del
programma

```
int main()
{
    ...
    /* Run all tests using the CUnit Basic interface */
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return CU_get_error();
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

120

Documentazione della libreria CUnit

<http://cunit.sourceforge.net/doc/introduction.html>

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

121

Esercizio 11.1 - Guidato

- Riutilizzare il codice dell'esercitazione che implementa la libreria delle funzioni matematiche
 - `int succ(int a)`
 - `int pred(int a)`
 - `int sum(int a, int b)`
 - `int product(int a, int b)`
 - `int max(int a, int b)`
 - `int min(int a, int b)`
- Definire (senza utilizzare CUnit) i **test methods** e le **asserzioni** per verificare la corretta implementazione di ciascuna delle funzioni

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

122

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000

int succ(int n){
    return ++n;
}

int pred(int n){
    return --n;
}

int min(int a, int b) {
    return (a < b) ? a : b;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}

int sum(int a, int b) {
    return a+b;
}

int product(int a, int b) {
    return a*b;
}
```

Definire (senza utilizzare CUnit) i **test methods** e le **asserzioni**

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

123

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000

int succ(int n){
    return ++n;
}

CU_ASSERT_EQUAL ( succ(INT_MIN) , INT_MIN+1 );
CU_ASSERT_EQUAL ( succ(INT_MAX) , INT_MAX+1 );
CU_ASSERT_EQUAL ( succ(-1) , 0 );
CU_ASSERT_EQUAL ( succ(0) , 1 );
```

Asserzioni di Esempio

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

124

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000
```

```
int pred(int n){
return --n;
}
```

```
CU_ASSERT_EQUAL ( pred(INT_MIN) , INT_MIN-1 );
CU_ASSERT_EQUAL ( pred(INT_MAX) , INT_MAX-1 );
CU_ASSERT_EQUAL ( pred(1) , 0 );
CU_ASSERT_EQUAL ( pred(0) , -1 );
```

Assertazioni di Esempio

Cerchiamo sempre di lavorare sui valori «limite»

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

125

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000
```

```
int min(int a, int b) {
return (a < b) ? a : b;
}
```

```
CU_ASSERT_EQUAL ( min(INT_MIN, INT_MAX), INT_MIN );
CU_ASSERT_EQUAL ( min(INT_MAX, INT_MIN), INT_MIN );
CU_ASSERT_EQUAL ( min(INT_MIN, 0) , INT_MIN );
CU_ASSERT_EQUAL ( min(0, INT_MAX), 0 );
CU_ASSERT_EQUAL ( min(INT_MAX, INT_MAX), INT_MAX );
CU_ASSERT_EQUAL ( min(INT_MIN, INT_MIN), INT_MIN );
CU_ASSERT_EQUAL ( min(0, 0), 0 );
ecc.
```

Testiamo tutte le combinazioni che coinvolgono i valori limite

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

126

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000
```

```
int max(int a, int b) {
return (a > b) ? a : b;
}
```

```
CU_ASSERT_EQUAL ( max(INT_MIN, INT_MAX), INT_MAX );
CU_ASSERT_EQUAL ( max(INT_MAX, INT_MIN), INT_MAX );
CU_ASSERT_EQUAL ( max(INT_MAX, 0) , INT_MAX );
CU_ASSERT_EQUAL ( max(0, INT_MAX), INT_MAX );
CU_ASSERT_EQUAL ( max(INT_MAX, INT_MAX), INT_MAX );
CU_ASSERT_EQUAL ( max(INT_MIN, INT_MIN), INT_MIN );
CU_ASSERT_EQUAL ( max(0, 0), 0 );
ecc.
```

Testiamo tutte le combinazioni che coinvolgono i valori limite

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

127

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000
```

```
int sum(int a, int b) {
return a+b;
}
```

```
CU_ASSERT_EQUAL ( sum(INT_MAX, INT_MIN) , 0 );
CU_ASSERT_EQUAL ( sum(INT_MIN, INT_MAX) , 0 );
CU_ASSERT_EQUAL ( sum(INT_MAX, 0), INT_MAX );
CU_ASSERT_EQUAL ( sum(0, INT_MAX), INT_MAX );
CU_ASSERT_EQUAL ( sum(0, INT_MIN), INT_MIN );
CU_ASSERT_EQUAL ( sum(0, 0), 0 );
ecc.
```

Testiamo tutte le combinazioni che coinvolgono i valori limite

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

128

Esercizio 11.1 - Guidato

```
#define INT_MAX 1000
#define INT_MIN -1000
```

```
int product(int a, int b) {
    return a*b;
}
```

Testiamo tutte le combinazioni che coinvolgono i valori limite

```
CU_ASSERT_EQUAL ( product(INT_MAX, INT_MIN) , product(INT_MIN, INT_MAX) );
CU_ASSERT_EQUAL ( product(INT_MAX, INT_MAX+1), product(INT_MAX, INT_MAX) + INT_MAX);
CU_ASSERT_EQUAL ( product(INT_MAX, 0), 0 );
CU_ASSERT_EQUAL ( product(0, INT_MAX), 0 );
CU_ASSERT_EQUAL ( product(0, INT_MIN), 0 );
CU_ASSERT_EQUAL ( product(0, 0), 0 );
```

ecc.

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

129

Codice Sorgente (solo main)

```
int main() {
    CU_initialize_registry();

    CU_pSuite pSuite_A = CU_add_suite("Suite_SUCC-PRED", init_suite_default, clean_suite_default);
    CU_pSuite pSuite_B = CU_add_suite("Suite_MIN-MAX", init_suite_default, clean_suite_default);
    CU_pSuite pSuite_C = CU_add_suite("Suite_SUM-PROD", init_suite_default, clean_suite_default);

    CU_add_test(pSuite_A, "test of succ()", test_succ);
    CU_add_test(pSuite_A, "test of pred()", test_pred);
    CU_add_test(pSuite_B, "test of min()", test_min);
    CU_add_test(pSuite_B, "test of max()", test_max);
    CU_add_test(pSuite_C, "test of sum()", test_sum);
    CU_add_test(pSuite_C, "test of product()", test_product);

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();

    return CU_get_error();
}
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

130

Output

CUnit - A Unit testing framework for C - Version 2.1-0
<http://cunit.sourceforge.net/>

```
Suite: Suite_SUCC-PRED
Test: test of succ() ... FAILED
1. ..\matematica.c:43 - CU_ASSERT_EQUAL(succ(INT_MAX),INT_MIN+1)
2. ..\matematica.c:44 - CU_ASSERT_EQUAL(succ(INT_MIN-1),INT_MIN+1)
Test: test of pred() ... passed
Suite: Suite_MIN-MAX
Test: test of min() ... passed
Test: test of max() ... passed
Suite: Suite_SUM-PROD
Test: test of sum() ... passed
Test: test of product() ... passed
```

```
--Run Summary: Type      Total   Ran   Passed   Failed
suites           3         3     n/a       0
tests            6         6       5       1
asserts          40        40      38       2
```

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

131

Esercizio 11.2

- Definire il piano di test e realizzare un programma che utilizzi CUnit per verificare la correttezza delle seguenti funzioni

- Calcolo della Media
- Calcolo del BMI
- Verifica Esame Superato oppure Maggiore Età

- Implementare il piano di test in un file con estensione .c che includa

- Metodi implementati
- Metodi di test
- Main che dichiara la test registry e la test suite

- Opzionalmente:** inviare il piano di test e il relativo output sulla piattaforma ADA

13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

132



13/05/19

Veronica Rossano - Testing Laboratorio di Informatica (INF, Track B) - Università degli Studi di Bari - A.A. 2018/2019

133