

Esercizi su

Funzioni

Testing

Tracing

Fasi di sviluppo

- Tracce extra
 - Sul sito del corso

- *funz_max.cc*
- Per casa:
 - *funz_fattoriale.cc*

Documentazione

- Il codice va documentato (commentato)
 - Leggibilità
 - Riduzione degli errori
 - Manutenibilità
- Documentare gli algoritmi utilizzati

Documentazione: funzioni

- Interfaccia della funzione
 - Valori in ingresso
 - Valori in uscita
- Algoritmo implementato

Collaudo (testing)

- Come si va alla ricerca degli errori?
- Innanzitutto occorre collaudare il programma
- Per entrare nell'argomento, partiamo dalla seguente domanda

Esautività 1/2

- Se un programma funziona correttamente per un valore di ingresso, si può affermare che sia corretto?

Esautività 2/2

- Ovviamente no
- Senza entrare in ulteriori dettagli, per questo corso diciamo solo che bisogna cercare di provare il programma per tutti gli ingressi possibili, o almeno per un alta percentuale degli ingressi possibili
- Quale logica e quale approccio usare?

Testing a scatola aperta

- Testing a scatola aperta (*white box*)
 - Mi metto nei panni del compilatore prima e soprattutto dell'esecutore dopo
 - Cerco di capire come vanno le cose al variare dei rami di codice eseguiti
 - I commenti nel programma aiutano

Testing a scatola chiusa

- Testing scatola chiusa (*black box*)
 - Si opera sui valori di ingresso supponendo di non sapere nulla di come funziona il programma
 - Si provano i valori agli estremi, nel mezzo, fuori dagli estremi degli intervalli consentiti

Fallimento 1/2

- Se troviamo un caso in cui il programma non si comporta correttamente, siamo di fronte ad un caso di **fallimento del programma**
- Vi sono fondamentalmente due tipi di fallimento:

Fallimento 2/2

- 1) Il programma viene terminato forzatamente dal sistema operativo
 - Esempio: divisione per zero
- 2) Il programma non viene terminato forzatamente, ma fornisce risultati scorretti

- Ed una volta scoperto che il programma fallisce?
- Vuol dire che il programma contiene un **errore**
 - Spesso si usa il termine **bug** o **baco**
- Il passo successivo è trovare l'errore

Analisi del codice

- La prima cosa che possiamo fare per trovare l'errore è rileggere con cura il codice
 - Cercare di capire dove sta l'errore
 - Facendosi guidare, se possibile dal tipo di fallimento
 - Spesso non è facile

Flusso di controllo 1/2

- Come faccio a capire dove e perché fallisce un programma?
- Cosa accade all'esecuzione di ciascuna istruzione?
 - Eventuale cambio del flusso di esecuzione in conseguenza di
 - **Lettura** di un qualche valore
 - **Scrittura** di un valore

Flusso di controllo 2/2

- Cosa guida l'esecuzione di un programma?
 - Il **valore** delle variabili
- Come posso guardare il valore delle variabili mentre il programma è in esecuzione?

- Stampandolo!
 - Si usa spesso il termine *tracing* per indicare la stampa di valori o in generale messaggi per capire cosa sta facendo esattamente un programma

- Inserire una `cout` in un ciclo può creare problemi?
- Cosa succede se il ciclo non termina più?
- Possibili soluzioni?
 - Inserire delle letture da `stdin` per controllare il ritmo delle iterazioni durante l'esecuzione

Collaudo e correzione errori

- D'ora in poi, ogni volta che si scrive un programma:
 - Collaudarlo sempre a scatola aperta e chiusa
 - Trovare e correggere autonomamente gli errori, eventualmente con l'aiuto del *tracing*
- Adottare questo approccio vi condurrà verso la professionalità
 - nonché verso un buon voto alla prova pratica ...

Somma di quadrati

- *somma_quadrati.cc*
 - Mettere in pratica quanto appreso sul *tracing* se ci si imbatte in casi di fallimento

Valori di ritorno ed eccezioni

- Ritornare
 - -1 oppure in generale
 - un valore fuori dall'intervallo di valori di output attesi

in caso di errore è una buona norma?
- Soluzione migliore: meccanismo delle eccezioni del C++ (non lo vedremo in questo corso)

- Abbiamo visto
 - Chiamata di funzione con due parametri
 - Suddivisione di un ciclo annidato tra il *main* ed una funzione
 - Utilizzo delle *cout*<< per il *tracing*

Difficoltà del debugging

- A questo punto dovremmo aver acquisito abbastanza esperienza da aver capito fino in fondo che
 - correggere gli errori è faticoso
- In merito c'è un problema molto serio:
 - Se introduciamo un secondo errore prima di esserci accorti del precedente, il debugging diviene molto più difficoltoso e lungo

Combinazione errori

- Se ne introduciamo anche un terzo siamo in guai seri
- In sintesi, la difficoltà ed il tempo di debugging aumentano esponenzialmente col numero di errori
 - perché gli errori possono **combinare i loro effetti**

Aggiunta codice ed errori

- Ma la nostra esperienza dovrebbe già averci insegnato che a peggiorare la situazione c'è anche il fatto che
 - **Ogni** riga di codice che si aggiunge ad un programma può introdurre nuovi errori
- Detto tutto questo, come facciamo a sviluppare il nostro programma tenendo al minimo l'attività di debugging?

Ciclo di sviluppo 1/3

- Un approccio estremamente efficace è il seguente:
 - Dato l'insieme di linee di codice che si dovrebbero scrivere per aggiungere una certa funzionalità ad un programma (o per scrivere il programma da zero)
 - **Non scrivere tutto il codice subito**
 - Non iniziare a revisionare, compilare, collaudare solo dopo aver finito di scrivere tutto il codice previsto

Ciclo di sviluppo 2/3

- Al contrario, seguire sempre il seguente ciclo di sviluppo
 - Dividere la scrittura in *micro-fasi* di scrittura&compilazione successive:
 - Aggiungere una quantità minima di nuovo codice, tale che il programma dovrebbe perlomeno compilarci
 - Analizzare **subito** il codice aggiunto
 - Provare a compilare
 - Se compila procedere con la successiva micro-fase, altrimenti correggere gli errori

Ciclo di sviluppo 3/3

- Ogni volta che, dopo un certo numero di micro-fasi successive, si è aggiunto/modificato abbastanza codice da avere realizzato una nuova funzionalità o un nuovo meccanismo del programma, allora
 - anche se ancora non si è arrivati alla versione completa del programma,
 - **collaudare subito la nuova versione parziale**

Quantità minima di codice

- Qual è la quantità minima di codice per ogni micro-fase di scrittura&compilazione e per ogni collaudo intermedio?
 - Non vi è una risposta precisa
 - Dipende dal problema e dalla confidenza che il programmatore ha nel codice che sta scrivendo
- In generale, l'errore tipico di un programmatore inesperto è quello di scrivere troppo prima di provare

Approccio vincente

- D'ora in poi adottare sempre questo approccio nello sviluppo dei programmi
- Al contrario, non adottarlo permette di scrivere il codice più velocemente e senza interruzioni, ma
 - quasi sempre allunga il tempo necessario per arrivare ad un programma funzionante
 - aumenta la probabilità che rimangano errori nel programma
 - rende estremamente più spiacevole lo sviluppo del programma

Generazione numeri primi

- **Esercizio per casa**
 - *gen_primi.cc*
- **Nella soluzione vedrete:**
 - Invocazione di funzioni all'interno di funzioni diverse dal *main*
 - Uso dell'istruzione vuota

Compiti per casa

- In ordine di difficoltà:
 - *gen_primi_gemelli.cc*
 - *congettura_goldbach.txt*
 - *funz_quadrato_pieno.cc*
 - *verifica_data.cc*
 - *funz_pot_pos_overflow.txt*
 - *ricevimento_iter.cc*