

Progetto d'esame

Progetto

- ***Redazione e discussione di un progetto***
- Si tratta di realizzare un'**applicazione dotata di interfaccia grafica**
 - Preferibilmente per il sistema operativo Linux
- L'esame orale consiste nella discussione (individuale) del progetto

Partecipazione al progetto

- Il progetto può essere realizzato da soli o in gruppi *di non più di due persone*
- Oltre alla qualità del progetto, per ciascun studente del gruppo, in sede di discussione si verificherà:
 - la **padronanza delle conoscenze** utilizzate per realizzare il progetto stesso
 - la partecipazione attiva ad ogni parte del progetto

Specifiche

- **Massima libertà** di scelta sul progetto da implementare
- A patto di **non** usare soluzioni già pronte e/o motori di giochi e applicazioni esistenti
 - non è difficile scoprire chi riutilizza codice trovato in rete
- Possibile invece appoggiarsi su **librerie grafiche di base** esistenti
 - Es.: GDK/GTK+, Qt, OpenGL, Cairo, Allegro, etc.

Spazio alla fantasia!

- **Gestori/database** con interfaccia grafica
 - sull'esempio del gestore liste visto a lezione (ma deve essere significativamente differente!)
 - come ispirazione, vedere altri widget di GTK+
- **Giochi**
 - dama, scacchi, poker, risiko, monopoli, pacman, space invaders, asteroid, frogger, tetris, ruzzle, ...

Qualità e modularità

- Nel progetto si devono mettere in pratica i principi conosciuti in merito alla **qualità del software**
 - *Semplicità, chiarezza, robustezza, configurabilità, defensive programming, usabilità dell'interfaccia*
- Il programma deve essere organizzato in **moduli e sviluppato su più file**
- La **compilazione** deve essere effettuabile in modo **automatico** attraverso il comando **make**

Testing e debugging

- Il programma va **collaudato** adottando i vari metodi visti nel corso
 - Il fallimento del programma durante l'esame non è motivo di fallimento dell'esame stesso
 - A patto che sappiate spiegare come mai nei vostri test il fallimento non si è verificato, e perché le *asserzioni* che avete inserito nel codice non hanno catturato l'errore
- Si deve inoltre mettere in piedi una semplice architettura per il **tracing** ed utilizzarla per il **debugging**

Documentazione

- Il progetto va documentato opportunamente mediante **Doxygen**
- A partire dalla **pagina principale** la documentazione deve guidare il lettore nella comprensione del funzionamento del programma

Pagina principale

La pagina principale deve contenere:

- Una **descrizione generale** di cosa fa il programma
- Una prima descrizione generale di come lo fa, nonché la suddivisione in **moduli** del programma stesso
- Un rimando alla funzione (probabilmente la funzione **main**) la cui documentazione continuerà a guidare il lettore nell'approfondimento della conoscenza del programma

Quantità e qualità del lavoro

- Si raccomanda di mantenere il progetto il **più semplice possibile** e di non imbarcarsi in lavori più grandi di voi
- Si tratta solo di un esame, ed è meglio non perdere più tempo del necessario
- Sarà valutata la **qualità** del progetto e della documentazione
- Naturalmente, progetti più ambiziosi avranno un bonus sulla valutazione finale
 - a patto che funzionino!

'Consegna' del software

- Una volta implementato e verificato, il software va **rilasciato** agli utenti finali
- L'operazione di rilascio non è banale
 - Portabilità su diverse architetture: necessita spesso di un processo di compilazione ed installazione (dove metto i diversi file?)
 - La consegna deve prevedere/includere, oltre al software stesso, altre informazioni necessarie, prima fra tutte la documentazione

Distribuzione del software

- **Distribuzione del software:** è il processo tramite il quale viene costruito un formato di “**archivio**” contenente il software (e tutte le informazioni corredate)
- Tale processo consta di più fasi
 - Creazione di una gerarchia di directory
 - Popolazione della gerarchia di directory
 - Creazione dell'**archivio finale**
 - Distribuzione (pubblicazione) dell'**archivio finale**

Gerarchia di directory

- L'**archivio finale** altri non è che la rappresentazione compatta di un insieme di file organizzati in una gerarchia di directory
- Directory principali:
 - **src**
 - **doc**
 - **media**

Contenuto delle directory

- **src:**
 - albero dei sorgenti
 - *Makefile*
- **media:**
 - contiene eventuali file audio, immagini, video necessari per l'esecuzione del software
- **doc:**
 - contiene documentazione relativa al progetto
 - descrizione della struttura del software
 - manuale di uso del software

Creazione archivio

Creazione di un **archivio compresso**

Comando

- **tar zcvf** <nome-progetto.tar.gz> <nomeprogetto>

Opzioni

- **z**: uso compressione
- **c**: create
- **v**: verbosity
- **f**: nome dell'archivio

Utilizzo archivio da parte dell'utente finale

- Scaricamento archivio compresso
- Scompattazione di archivio compresso:
tar zxvf <nome-progetto.tar.gz> <nomeprogetto>
 - **z**: uso compressione
 - **x**: estrazione
 - **v**: verbosity
 - **f**: nome dell'archivio
- Fase di compilazione del progetto
cd nome-progetto; make depend; make

Ingegneria del codice

Applicare i concetti visti nel corso di *Programmazione I* relativi all'**ingegneria del codice**:

- Assegnare **nomi significativi** a variabili e funzioni
- **Formattare** in modo opportuno il codice
- **Evitare duplicazioni** del codice
- **Evitare eccessive nidificazioni** delle istruzioni

http://algogroup.unimore.it/people/paolo/courses/programmazione_I/materiale_1213/lezioni/Lez_09-Ingegneria_del_codice.pdf

Qualità del software

Tenere presenti alcuni concetti basilari di qualità del software

- **Modularità**
- **Chiarezza**
- **Robustezza**
- **Semplicità**

Modularità

- Scrivere **software semplici** che comunicano tra loro in maniera semplice
 - Modalità di comunicazione: **interfaccia di moduli/funzioni**
- La fase più tediosa del processo di sviluppo è la **correzione** degli errori
 - Semplifichiamo il processo di correzione spezzando i problemi complessi in più **sottoproblemi**

Chiarezza

- Rendiamo più semplice il processo di correzione degli errori scrivendo **programmi immediatamente chiari (leggibili)** all'occhio umano
- La chiarezza si esplica in due modalità
 - **Estetica** (*commento e formattazione del codice*)
 - **Design** (*scelta di algoritmi e strutture dati semplici*)

Robustezza

- Un software si dice robusto se si comporta bene in condizioni **non previste**
 - Parametri di ingresso sbagliati
 - Condizioni di carico non previsti
- Come si rende robusto un programma?
 - **Semplicità e modularità**
- Utilizzare **asserzioni** per gestire i casi non previsti/voluti

Semplicità

- Progettate avendo sempre in mente la **semplicità**
- Un algoritmo molto sofisticato può dare enormi difficoltà di manutenzione/correzione degli errori, a fronte di un lieve aumento di prestazioni

Note finali

- Non abbiate paura ad usare altre librerie open source non viste a lezione
 - A patto che siano provviste di una buona documentazione
- Sono a vostra disposizione per dubbi su
 - validità di un'idea di progetto da implementare
 - librerie utilizzabili
- **Non sono a vostra disposizione invece per correggere progetti non consegnati**

Per qualsiasi cosa...

- Scrivetemi pure via email
 - tenendo presente che siete quasi 100...
- Evitate di scrivere per problemi banali (tipo: “prof, non mi compila...”)
- Provate prima a risolvere il problema da soli, cercando in rete problemi simili o consultando la documentazione

Buon lavoro!