Modalità d'esame

L'esame è composto di due parti in successione: un progetto per il modulo di laboratorio e una prova orale per il modulo di teoria. Ciascuna parte è valutata dal docente del relativo modulo. Il docente di teoria è il titolare del corso, definisce e registra il voto complessivo. Le due parti si possono svolgere in appelli separati, in base a regole definite dal docente di teoria.

Condizioni per il progetto

Un progetto è rifiutato se:

1. non è svolto **individualmente**;
2. non è svolto **interamente**;
3. non è svolto **in C89**;
4. non è consegnato **entro la data e l'ora stabilite**;
5. non ha ricevuto **conferma di ricezione** dal docente;
6. alla consegna non si **dichiara di accettare queste condizioni**.

Organizzazione

Circa un mese prima di ciascun appello d'esame, sulla pagina web del modulo di laboratorio viene pubblicata la descrizione di un problema. Per prima cosa, si deve modellare questo problema concreto in termini di oggetti matematici astratti. Poi, si devono definire gli algoritmi opportuni per risolvere il problema e scegliere in base a questi le strutture dati opportune per rappresentare gli oggetti astratti. Quindi, si devono tradurre algoritmi e strutture dati in un programma scritto nel linguaggio di programmazione C. Infine, si deve stendere una relazione. Per lo svolgimento del progetto sono concesse circa tre settimane.

Il testo fornisce indizi chiari, ma in genere non espliciti, sugli oggetti matematici per modellare il problema, sugli algoritmi per risolverlo e sulle strutture per rappresentare dati e risultati. In casi più rari, li indica esplicitamente: se si è tentati di seguire strade diverse, si consiglia di discuterne col docente, dato che in genere i suggerimenti sono ben motivati.

Con il testo vengono forniti dati di esempio e soluzioni. Di solito, ottenere soluzioni diverse da quelle indicate è un indice di errore. Può accadere che esistano soluzioni corrette molteplici (in genere, questo è segnalato nel testo) o che le soluzioni pubblicate siano errate (in tal caso, segnalandolo verranno corrette al più presto). Ottenere le soluzioni pubblicate è un segno confortante, ma non una garanzia di correttezza.

Valutazione

Durante la valutazione, relazione e codice vengono scorsi in parallelo, verificando che corrispondano direttamente. Il codice viene eseguito su molti esempi diversi. Relazioni ostiche o scorrelate dal codice e codici che reagiscono a modo loro incagliano il processo e rendono più difficile mantenere un metro equanime di valutazione. I criteri di valutazione sono i seguenti.

1. Completezza della relazione Il lettore della relazione non sa nulla del problema: vuole sapere com'è e come è stato risolto; sa tutto dei corsi di programmazione e di algoritmi. Quindi, la relazione soddisfa il criterio di completezza quando descrive sinteticamente:

1. il problema a livello concettuale (senza dettagli tecnici come i formati di ingresso e uscita);
2. il modello astratto adottato e i motivi di tale scelta;
3. gli algoritmi e le strutture dati adottate, i motivi di tali scelte e i costi temporali e spaziali (senza i dettagli trattati nel corso).

2. Correttezza della relazione La relazione soddisfa il criterio di correttezza quando, scorrendola in parallelo al codice, si verifica pagina dopo pagina che le operazioni descritte e le analisi di complessità corrispondono all'implementazione e ai risultati della teoria.

Le analisi di complessità sono riferite agli indici che descrivono la dimensione del problema, non a un generico indice n. Le semplificazioni che cancellano indici vanno motivate, e la validità dei motivi verrà valutata.

3. Struttura della relazione La relazione soddisfa il criterio di struttura se l'esposizione è chiara, l'aspetto estetico apprezzabile e soprattutto ha un'organizzazione "top-down" parallela a quella del codice. Presenta prima il problema generale e poi i sottoproblemi particolari, prima i concetti astratti poi la realizzazione concreta. Fornisce un colpo d'occhio su ogni algoritmo e poi i dettagli organizzati in sezioni e sottosezioni. Allo stesso modo, il codice ha un programma principale con poche procedure di livello superiore, che chiamano poche procedure di livello inferiore, e così via. Leggendo le sezioni e procedure di livello superiore si capisce che cosa fa l'algoritmo senza dove subito leggere le sezioni e procedure di livello inferiore. Quindi la relazione si legge una volta sola dal principio alla fine, senza dover saltare avanti e indietro fra sezioni diverse. La relazione non descrive il codice istruzione per istruzione e il codice non è una singola procedura con centinaia di istruzioni allo stesso livello.

L'analisi di complessità temporale e spaziale di ogni funzione rispecchia la descrizione della struttura dell'algoritmo e la segue immediatamente, non a distanza di pagine.

I dettagli tecnici di programmazione (funzioni di libreria, costrutti, tipi di variabili, nomi di file e procedure) in genere sono inutili e danneggiano la leggibilità, a meno che non condizionino la correttezza o le prestazioni dell'algoritmo.

Gli *schemi* o *tracce* di relazione riportati sulla pagina web del corso non sono *esempi* o *fac-simile*. Ci si accerti di conoscere le differenze tra questi vocaboli.

4. Correttezza del codice Il codice soddisfa il criterio di correttezza quando non contiene errori di programmazione o algoritmici.

Gli errori di programmazione più tipici sono l'uso di aree di memoria non allocate, lo sforamento di aree allocate, l'uso di variabili non inizializzate, lo sfruttamento di condizioni valide per gli esempi pubblicati, ma non indicate nel testo del progetto. Sono errori insidiosi perché alcuni compilatori li compensano o perché i dati pubblicati possono nasconderli, mentre il compilatore e i dati usati per la valutazione li rivelano[1].

[1] A puro titolo informativo (nessuna pubblicità), i progetti sono valutati compilandoli con Microsoft Visual Studio Express, che segnala o fa emergere durante l'esecuzione anche errori ignorati da `gcc`. Codici che funzionano sulle macchine degli studenti spesso non lo fanno sulla mia: c'è sempre dietro un errore.

Esempi di errori algoritmici sono la scelta di un algoritmo scorretto o diverso da quello indicato (se il testo ne indica uno), di criteri di ordinamento diversi da quelli indicati, di algoritmi che risolvono un problema diverso da quello indicato.

5. Efficienza del codice Il codice soddisfa il criterio di efficienza quando usa gli algoritmi e le strutture dati più efficienti rispetto al tempo e allo spazio, privilegiando il primo in caso di conflitti. Esempi sono la scelta di algoritmi di complessità superiore, la scelta di strutture dati inadatte, il ricalcolo o la ricerca di informazioni che sono disponibili o potrebbero esserlo usando strutture opportune.

6. Struttura del codice Il codice soddisfa il criterio di struttura quando rispetta i formati di ingresso e uscita specificati nel testo, è adeguatamente commentato, ben strutturato e portabile, cioè rispetta lo standard richiesto.

I dati vengono caricati attraverso una linea di comando da uno o più file nell'ordine indicato dal testo. I risultati vengono stampati a video rispettando il formato richiesto (compresi spazi, a capi, maiuscole e minuscole, ecc...). Un modo per verificare le soluzioni ottenute è redirigere le stampe da video in un file di testo accodando alla linea di comando `>` e il nome del file, e confrontare il file ottenuto con quello pubblicato. Ogni sistema operativo offre strumenti per confrontare file. Violare il formato richiede al docente di modificare a mano il codice e implica una penalizzazione.

Lo standard adottato nel corso è il C89. Ci sono diversi motivi tecnici per questa scelta. I principali sono la portabilità su altri compilatori e sistemi operativi, una più semplice analisi di complessità per gli studenti e una più semplice ricerca di errori per il docente. Conviene quindi compilare il codice con le opzioni `-std=c89` e `-Wall` e rimediare agli avvertimenti così ottenuti. Può essere utile l'opzione `-pedantic`, anche se genera avvertimenti un po' eccessivi.

Consegna del progetto

Relazione e codice vanno raccolti in un file compattato (in formato .ZIP, .RAR o .GZ) e spediti al docente di laboratorio per posta elettronica entro la mezzanotte del giorno indicato nel testo del progetto. Prima di spedire, si verifichi che l'allegato non contenga file eseguibili o ambigui, perché il server di posta cancella questi allegati. Nel giro di alcune ore si riceverà un messaggio di conferma. **Un progetto che non riceve conferma non è consegnato**[2]. Se il codice consegnato genera errori di compilazione o non rispetta lo standard C89, si riceveranno i messaggi di errore e un invito a spedirne una versione conforme entro la scadenza[3].

Domande utili

- Si possono sviluppare algoritmi e strutture dati diverse da quelle realizzate in laboratorio?

  Sì, in particolare quelli descritti nelle lezioni di teoria, ma non solo.

- Si possono usare e modificare le librerie sviluppate in laboratorio?

  Assolutamente sì: è il loro scopo.

- Si possono usare librerie scaricate dalla rete?

  No, il corso mira a rendere gli studenti in grado di realizzare algoritmi, non solo di usarli.

[2] Consegnare e andare a dormire è un rischio che ci si assume consapevolmente.
[3] Valgono le condizioni 3, 4 e 6 riportate al principio.

- Avendo a disposizione due algoritmi, nessuno dei quali abbia complessità strettamente migliore dell'altra, quale si deve scegliere?

  Se i due algoritmi sono sostanzialmente equivalenti, il problema non si pone. Se invece l'efficienza dipende dall'istanza (per esempio, $\Theta(nm)$ contro $\Theta(n \log n)$), si sceglie un algoritmo argomentando nella relazione i motivi, che verranno valutati. Non è corretto presentare la scelta come scontata.