Compito di Laboratorio di Programmazione C

A.A. 2005-2006

Docenti:

G.Campanile

A.Sterbini

S.Guerrini

Indice

1 Obiettivo 1
2 Suddivisione del compito 1
3 Cosa viene dato 1
4 Cosa deve essere fatto 1
5 Come deve essere fatto 2
5.1 Ambiente di sviluppo 2
5.2 Files sorgenti 2
5.3 Uso delle costanti 2
5.3.1 Path dei file 3
5.3.2 Richiamo di comandi 3
5.3.3 Richiamo di funzioni 3
5.3.4 Gestione dell'errore 3
6 Organizzazione del progetto 3
6.1 Albero generale 4
6.2 Directory principale 4
6.3 Modulo1 5
6.4 Modulo2 5
6.5 Modulo3 6
6.6 Modulo4 7
7 Come viene valutato 8
7.1 Voto Insufficiente 9
7.2 Voto Sufficiente 9
7.3 Voto Buono 9
7.4 Voto Ottimo 10
7.5 Sintesi 10
7.6 Voto finale 10

1 Obiettivo

Il compito si propone di implementare un database relazionale basato su file system.

Queste sono le caratteristiche principali:

- il database è single-user;
- ogni tabella è un file;
- ogni tabella può contenere un certo numero di campi, di diverso tipo e lunghezza;
- possono essere aperte più tabelle contemporaneamente, ma ogni tabella una sola volta;
- è possibile aggiungere un record, cancellarlo, modificarne i valori;
- è possibile effettuare ricerche di un valore sequenzialmente;

2 Suddivisione del compito

Il compito è suddiviso in 4 moduli. Ogni modulo deve essere costruito sulla base del modulo precedente, quindi per poter sviluppare correttamente ad esempio il secondo modulo è necessario disporre del primo.

3 Cosa viene dato

Per ogni modulo vengono fornite sia specifiche di interfaccia che specifiche interne necessarie affinché tutti i moduli siano interoperabili e testabili allo stesso modo, indipendentemente dalla loro struttura interna.

Viene quindi fornito:

- All'inizio, valido per tutti i moduli:
  - questo file di documentazione
  - un file di include contenente
    - prototipi delle funzioni
    - strutture da utilizzare
    - costanti necessarie
  - il sorgente della funzione di gestione errore
- Per ogni Modulo
  - le specifiche di interfaccia in formato Web e rtf sulle funzioni da sviluppare
  - le specifiche interne formato Web e rtf per "come" sviluppare le funzioni
  - le specifiche di test e il sorgente del programma di test
  - il makefile per compilare il modulo

4 Cosa deve essere fatto

Il compito consiste nello sviluppare le funzioni indicate, in modo che il programma di test fornito non dia errori.

Il programma di test verifica generalmente:

- controllo degli errori nei parametri passati;
- controllo della "chiamabilità" della funzione (ad. es. non è possibile cancellare un database se è in uso);
- correttezza dello svolgimento della funzione;

Quello che dovrà essere consegnato sarà quindi il modulo in formato sorgente, cioè i files .c e .h necessari.

In aggiunta per ogni modulo dovrà essere consegnato un file di testo con le indicazioni dello studente e delle caratteristiche del sistema utilizzato.

5 Come deve essere fatto

Le specifiche devono essere scrupolosamente osservate dagli studenti, in modo da sviluppare moduli congruenti e verificabili. Non rispettare una specifica equivale ad un compito errato, anche se il programma gira correttamente. I programmi di test forniti, e non modificabili, verificheranno che le specifiche siano osservate.

I programmi devono essere sviluppati in C ANSI standard, senza alcun riferimento al sistema operativo (ad.es. *windows.h* utilizzato in Windows) o al tipo di compilatore utilizzato (ad.es. *conio.h* utilizzato in DEVC++).

Qui di seguito vengono date altre regole da seguire per lo sviluppo.

5.1 Ambiente di sviluppo

Gli studenti potranno utilizzare qualunque ambiente di sviluppo su qualunque tipo di macchina. Il codice prodotto però dovrà essere compatibile ANSI C, quindi dovranno essere evitate le funzioni particolari non standard ANSI. Dovranno essere anche evitate particolarità legate al sistema operativo, quali ad esempio riferimenti a strutture di directories e così via.

Il progetto consegnato verrà ricompilato utilizzando il makefile che viene fornito (vedi dopo) quindi, indipendentemente dall'ambiente utilizzato per lo sviluppo, è consigliato di verificare che la compilazione tramite makefile si svolga correttamente.
Il makefile fornito utilizza il compilatore GNU gcc.

5.2 Files sorgenti

In ogni Modulo devono essere sviluppate le funzioni indicate come interfaccia. Per ogni funzione è indicato il file che la contiene; in questo file non è ammesso inserire altre funzioni o variabili globali.
Tutte le funzioni che lo studente deve sviluppare per completare il compito, vanno inserite in un sorgente di nome

LPC_Common.c.

Tutte le dichiarazioni di costanti e di strutture specifiche di un modulo vanno inserite in un file di nome

LPC_M<n>_Include.h

dove <n> è il numero del modulo. Questo file è nella directory di include di ogni modulo e includerà a sua volta il file generale *LPC_Include.h*.

Una descrizione dettagliata dei files sorgenti è nel capitolo 6..

5.3 Uso delle costanti

Nei sorgenti .c non si deve far uso di valori costanti, ma questi devono essere definiti in un apposito file contenuto nella directory include.

In particolare i codici di errore devono essere sicuramente costanti; i codici di errore forniti coprono gran parte dei possibili errori, ma se ne possono definire altri. Questo dovrà essere fatto senza alterare il file di costanti generale, ma utilizzando il file di include specifico del modulo.

5.3.1 Path dei file

In nessun caso nei file sorgenti o nei file di include deve essere specificato un path assoluto. In particolare i file che vengono inclusi dalla direttiva *#include* devono apparire con il loro nome, senza alcun path, neanche relativo.

5.3.2 Richiamo di comandi

Non è ammesso il richiamo di comandi di sistema, quali ad esempio "cp" per copiare un file o altri. Più in generale non è ammesso l'utilizzo della funzione *system()* o di funzioni di *exec()* per eseguire comandi.

5.3.3 Richiamo di funzioni

E' ammesso richiamare funzioni dello stesso modulo o di moduli precedenti, anzi in alcuni casi è necessario. Quando non è consentito viene detto esplicitamente, quindi è necessario leggere sempre attentamente le specifiche interne dei vari moduli.

5.3.4 Gestione dell'errore

Ogni modulo deve testare tutti gli errori possibili. Questo vuol dire che deve testare almeno la correttezza dei parametri di input e i return code di **ogni** eventuale altra funzione invocata, comprese funzioni di sistema quali *calloc()* o *fopen()* o altro.

Se una funzione intercetta un errore, deve chiamare la funzione di errore *LPC_GestioneErrore()* fornita e ritornare un proprio codice errore al livello superiore. Prima di ritornare, deve liberare tutte le variabili da lei allocate , invocando le opportune funzioni (ad.es. la *free*()).

La funzione di errore è fornita e documentata, e stampa l'errore su stderr. Non va modificata ma solo compilata e inserita nella directory DBF/obj.

Nella documentazione delle funzioni sono specificati alcuni errori che la funzione ritorna. Questa è solo un'indicazione di massima, che è emersa dallo sviluppo di test effettuato dai docenti. L'insieme degli errori che una funzione ritorna è determinato dallo sviluppo dello studente, purché questi rientrino nell'insieme definito in *LPC_Include.h*.

6 Organizzazione del progetto

Anche se la consegna riguarderà solo i file in forma sorgente, viene qui fornita la struttura complessiva del progetto, in modo che ogni studente possa ricreare le condizioni nelle quali verrà valutato il proprio lavoro.

I files di ogni progetto saranno i seguenti:

- sorgenti C del progetto
- sorgenti C di test
- files di include
- oggetti
- librerie

- eseguibili di test
- makefile

Saranno suddivisi in diverse sottodirectories descritte nei capitoli seguenti. Per ognuno dei files viene indicato se è fornito inizialmente oppure se è da consegnare da parte dello studente. Se non è indicato nulla, si tratta di file che vengono prodotti nella fase di compilazione e che non devono essere consegnati.

Alcuni files saranno forniti: gli studenti **NON DEVONO MODIFICARE** questi files.

6.1 Albero generale

Il progetto è suddiviso nel seguente albero di directories:

- DBF
  - doc
  - src
  - obj
  - include
  - Modulo<n>
    - doc
    - src
    - include
    - obj
    - lib
    - test

Dove Modulo<n> sarà Modulo1, Modulo2, Modulo3 e Modulo4.

6.2 Directory principale

Nella directory principale saranno contenuti le seguenti directories e *files*:

- DBF
  - doc
    - *CompitoLabProgC.pdf* questo documento (**fornito**)
  - src
    - *LPC_Err.c* (sorgente funzione di errore) (**fornito**)
  - obj
    - *LPC_Err.o* (compilato della funzione di *LPC_ERR.c*)
  - include
    - *LPC_Include.h* (include files generale) (**fornito**)
  - Modulo1 home dir del Modulo1 (vedi dopo)
  - Modulo2 home dir del Modulo2 (vedi dopo)
  - Modulo3 home dir del Modulo3 (vedi dopo)
  - Modulo4 home dir del Modulo4 (vedi dopo)

La funzione di errore dovrà essere linkata ai vari programmi di test, insieme alle opportune librerie. Non è consentito copiare il modulo di gestione errore nelle directory sorgenti dei vari moduli.

6.3 Modulo1

Nella directory DBF/Modulo1 saranno contenuti le seguenti directories e *files*:

- doc
  - html
    - Files html della documentazione del modulo. Home page è *index.html*. (**tutto fornito**)
  - rtf
    - *refman.rtf* documentazione del modulo in formato rtf. (**fornito**)
- include
  - *LPC_M1_Include.h* file di include del progetto.
    Lo studente dovrà inserire qui tutte le definizioni delle costanti, delle strutture etc. che svilupperà per il Modulo1. Dovrà inserire anche i prototipi delle funzioni contenute in *LPC_Common.c*. (**da consegnare**);
- lib
  - *Modulo1.a* libreria contenente i moduli oggetto
- obj
  - *LPC_Common.o* compilato del file *src/LPC_CommonDBF.c* ;
  - *LPC_CreateDBF.o* compilato del file *src/LPC_CreateDBF.c* ;
  - *LPC_DeleteDBF.o* compilato del file *src/LPC_DeleteDBF.c* ;
- src
  - *LPC_CreateDBF.c* file della funzione *CreateDatabase()* (**da consegnare**);
  - *LPC_DeleteDBF.c* file della funzione *DeleteDatabase()* (**da consegnare**);
  - *LPC_Common.c* file delle funzioni aggiuntive sviluppate dallo studente (**da consegnare**);
- *makefile* makefile del modulo (**fornito**).
  Contiene le regole per costruire i seguenti target:
  - all: compila tutto il modulo e il file di test;
  - module: compila solo la libreria del modulo;
  - test: compila solo il programma di test;
  - clean: pulisce per una nuova compilazione;
- *Readme.txt* File di testo così strutturato: (**da consegnare**);
  Autore:
  <nome dello studente>
  Versioni sistema operativo
  <Versioni del (dei) sistema su cui il progetto è stato testato>
  Compilatore
  <Versione del compilatore con cui il progetto è stato testato>
  Note
  <Eventuali note da segnalare>
- test
  - *TestM1.c* sorgente di test per il Modulo 1 (**fornito**)
  - *TestM1* eseguibile di test per il Modulo1

6.4 Modulo2

Nella directory DBF/Modulo2 saranno contenuti le seguenti directories e *files*:

- doc
  - html

        - Files html della documentazione del modulo. Home page è *index.html*. (**tutto fornito**)
    - rtf
        - *refman.rtf* documentazione del modulo in formato rtf. (**fornito**)
- include
    - *LPC_M2_Include.h* file di include del progetto.
      Lo studente dovrà inserire qui tutte le definizioni delle costanti, delle strutture etc. che svilupperà per il Modulo2. Dovrà inserire anche i prototipi delle funzioni contenute in *LPC_Common.c*. (**da consegnare**)
- lib
    - *Modulo2.a* libreria contenente i moduli oggetto
- obj
    - *LPC_CloseDBF.o* compilato del file *src/LPC_CloseDBF.c* ;
    - *LPC_Common.o* compilato del file *src/LPC_CommonDBF.c* ;
    - *LPC_DBFInfo.o* compilato del file *src/LPC_DBFInfo.c* ;
    - *LPC_FieldInfo.o* compilato del file *src/LPC_FieldInfo.c* ;
    - *LPC_IsDBF.o* compilato del file *src/LPC_IsDBF.c* ;
    - *LPC_OpenDBF.o* compilato del file *src/LPC_OpenDBF.c* ;
- src
    - *LPC_CloseDBF.c* file della funzione *CloseDatabase()*(**da consegnare**);
    - *LPC_Common.c* file delle funzioni aggiuntive sviluppate dallo studente(**da consegnare**);
    - *LPC_DBFInfo.c* file della funzione *GetDBFInfo()*(**da consegnare**);
    - *LPC_FieldInfo.c* file della funzione *GetFieldInfo()*(**da consegnare**);
    - *LPC_IsDBF.c* file della funzione *IsDBFHandle ()*(**da consegnare**);
    - *LPC_OpenDBF.c* file della funzione *OpenDatabase()*(**da consegnare**);
- *makefile* makefile del modulo (**fornito**).
  Contiene le regole per costruire i seguenti target:
  - all: compila tutto il modulo e il file di test;
  - module: compila solo la libreria del modulo;
  - test: compila solo il programma di test;
  - clean: pulisce per una nuova compilazione;
- *Readme.txt* File di testo così strutturato: (**da consegnare**)
  Autore
  <nome dello studente>
  Versioni sistema operativo
  <Versioni del (dei) sistema su cui il progetto è stato testato>
  Compilatore
  <Versione del compilatore con cui il progetto è stato testato>
  Note
  <Eventuali note da segnalare>
- test
    - *TestM2.c* sorgente di test per il Modulo 2 (**fornito**)
    - *TestM2* eseguibile di test per il Modulo 2

6.5 Modulo3

Nella directory DBF/Modulo3 saranno contenuti le seguenti directories e *files*:

- doc
    - html

        - Files html della documentazione del modulo. Home page è *index.html*. (**tutto fornito**)
    - rtf
        - *refman.rtf* documentazione del modulo in formato rtf. (**fornito**)
- include
    - *LPC_M3_Include.h* file di include del progetto.
      Lo studente dovrà inserire qui tutte le definizioni delle costanti, delle strutture etc. che svilupperà per il Modulo3. Dovrà inserire anche i prototipi delle funzioni contenute in *LPC_Common.c*. (**da consegnare**)
- lib
    - *Modulo3.a* libreria contenente i moduli oggetto
- obj
    - *LPC_Common.o* compilato del file *src/LPC_CommonDBF.c* ;
    - *LPC_DeleteRec.o* compilato del file *src/LPC_DeleteRec.c* ;
    - *LPC_GotoRec.o* compilato del file *src/ LPC_GotoRec.c* ;
    - *LPC_InsertRec.o* compilato del file *src/LPC_InsertRec.c* ;
- src
    - *LPC_Common.c* file delle funzioni aggiuntive sviluppate dallo studente(**da consegnare**);
    - *LPC_DeleteRec.c* file della funzione *DeleteRecord()*(**da consegnare**);
    - *LPC_GotoRec.c* file della funzione *GotoRecord()*(**da consegnare**);
    - *LPC_InsertRec.c* file della funzione *InsertBlankRecord()*(**da consegnare**);
- *makefile* makefile del modulo (**fornito**).
  Contiene le regole per costruire i seguenti target:
  - all: compila tutto il modulo e il file di test;
  - module: compila solo la libreria del modulo;
  - test: compila solo il programma di test;
  - clean: pulisce per una nuova compilazione;
- *Readme.txt* File di testo così strutturato: (**da consegnare**)
  Autore
  <nome dello studente>
  Versioni sistema operativo
  <Versioni del (dei) sistema su cui il progetto è stato testato>
  Compilatore
  <Versione del compilatore con cui il progetto è stato testato>
  Note
  <Eventuali note da segnalare>
- test
    - *TestM3.c* sorgente di test per il Modulo 3 (**fornito**)
    - *TestM3* eseguibile di test per il Modulo 3

6.6 Modulo4

Nella directory DBF/Modulo4 saranno contenuti le seguenti directories e *files*:

- doc
    - html
        - Files html della documentazione del modulo. Home page è *index.html*. (**tutto fornito**)
    - rtf

        - *refman.rtf* documentazione del modulo in formato rtf. (**fornito**)
- include
    - *LPC_M4_Include.h* file di include del progetto.
      Lo studente dovrà inserire qui tutte le definizioni delle costanti, delle strutture etc. che svilupperà per il Modulo4. Dovrà inserire anche i prototipi delle funzioni contenute in *LPC_Common.c*. (**da consegnare**)
- lib
    - *Modulo4.a* libreria contenente i moduli oggetto
- obj
    - *LPC_Common.o* compilato del file *src/ LPC_CommonDBF.c* ;
    - *LPC_EditRect.o* compilato del file *src/ LPC_EditRec.c* ;
    - *LPC_LocateRec.o* compilato del file *src/ LPC_LocateRec.c* ;
    - *LPC_ReadRec.o* compilato del file *src/ LPC_ReadRec.c* ;
- src
    - *LPC_Common.c* file delle funzioni aggiuntive sviluppate dallo studente(**da consegnare**);
    - *LPC_EditRec.c* file della funzione *EditRecord()*(**da consegnare**);
    - *LPC_LocateRec.c* file della funzione *LocateRecord()*(**da consegnare**);
    - *LPC_ReadRec.c* file della funzione *ReadRecord()*(**da consegnare**);
- *makefile* makefile del modulo (**fornito**).
  Contiene le regole per costruire i seguenti target:
  - all: compila tutto il modulo e il file di test;
  - module: compila solo la libreria del modulo;
  - test: compila solo il programma di test;
  - clean: pulisce per una nuova compilazione;
- *Readme.txt* File di testo così strutturato: (**da consegnare**)
  Autore
  <nome dello studente>
  Versioni sistema operativo
  <Versioni del (dei) sistema su cui il progetto è stato testato>
  Compilatore
  <Versione del compilatore con cui il progetto è stato testato>
  Note
  <Eventuali note da segnalare>
- test
    - *TestM4.c* sorgente di test per il Modulo 4 (**fornito**)
    - *TestM4* eseguibile di test per il Modulo 4

7 Come viene valutato

I Moduli vengono valutati sia per la correttezza del risultato ma soprattutto per l'aderenza alle specifiche.
Sono forniti tutti gli strumenti (analisi, makefile, specifiche di test, programmi di test) perché uno studente possa verificare la correttezza del suo compito prima di consegnarlo.

Ci sono 4 voti possibili:

- Insufficiente;
- Sufficiente;
- Buono;
- Ottimo.

Il calcolo dei test superati viene effettuato dall'apposito programma di test di ogni Modulo. Attenzione: alcuni test sono più importanti di altri e il programma di test ne tiene conto: al termine del test viene stampato il punteggio ottenuto per il modulo. Questo punteggio è quello che viene utilizzato per valutare il Modulo, come descritto nei paragrafi seguenti.

Il voto finale dell'esame viene calcolato sulla media dei voti di ogni Modulo.

7.1 Voto Insufficiente

Questi sono i criteri per cui un Modulo è giudicato insufficiente:

- Il programma non viene consegnato o è consegnato oltre i termini previsti;
  oppure
- Il programma è stato copiato;
  oppure
- Alcuni dei moduli non compilano e/o non si riesce a linkare il relativo programma di test;
  oppure
- Compilazione con Warning e 0% <= [punteggio test Modulo] <= 60%;
  oppure
- Compilazione senza Warning e 0% <= [punteggio test Modulo] <= 50%;

ATTENZIONE:
Basta che **uno solo** di questi punti sia verificato perché il voto venga giudicato insufficiente.

7.2 Voto Sufficiente

Un Modulo è giudicato sufficiente se:

- Compila/Linka con Warning
  e
- 60% < [punteggio test Modulo] <=80%

Oppure

- Compila/Linka senza Warning
  e
- 50% < [punteggio test Modulo] <=70%

7.3 Voto Buono

Un Modulo è giudicato buono se:

- Compila/Linka con Warning
  e
- 80% < [punteggio test Modulo] < 100%

Oppure

- Compila/Linka senza Warning
  e
- 70% < [punteggio test Modulo] <= 90%

7.4 Voto Ottimo

Un Modulo è giudicato ottimo se:

- Compila/Linka con Warning
  e
- [punteggio test Modulo] = 100%

Oppure

- Compila/Linka senza Warning
  e
- 90% < [punteggio test Modulo] <= 100%

7.5 Sintesi

	Punteggio del test												
	0		50		60		70		80		90		100
Compilazione con Warning	Insufficiente				Sufficiente				Buono				Ottimo
Compilazione senza Warning	Insufficiente		Sufficiente				Buono				Ottimo		

7.6 Voto finale

Il voto finale è determinato sulla base delle votazioni ottenute nei 4 Moduli del compito. Anche se la valutazione non è strettamente aritmetica, è stabilito che:

- 3 insufficienze non permettono di superare l'appello, anche se il 4 voto è ottimo;
- 2 insufficienze possono essere recuperate solo con 2 voti ottimo;