Laboratorio su sistemi Linux
Linguaggio C, gcc, Makefile, utilities varie (ldd, nm, lsof, ps, tar wget)

Dalla sesta lezione in laboratorio in poi …

NOTA BENE: E INSISTO !!!!!

Usare il comando **man 3** nomefunzionedilibreria per ottenere informazioni sull'uso di una specifica funzione di libreria standard del linguaggio C
Ad esempio, per ottenere informazioni sull'uso della funzione coseno:

man 3 cos

Lezione 6 parte 3 in laboratorio usare moduli, man, gcc, Makefile

- Il risultato del primo programma in C eseguito da Kernighan e Ritchie

Prima di cominciare questa lezione, occorre spiegare anche come si usa find con l'argomento exec

usare moduli, man, gcc, Makefile

Esercizio quadrato_dimezza: scrivere programma con più moduli e Makefile.

In una propria directory, scrivere un programma costituito da 3 moduli: main.c quadrato.c dimezza.c e da due file di intestazioni, quadrato.h e dimezza.h.

Il modulo dimezza.c implementa una funzione che prende un argomento double e restituisce un double pari alla meta' del COSENO dell'argomento passato. **man 3 cos** spiega come usare coseno e come linkare libreria. Prima di restituire il risultato, la funzione copia il risultato in una variabile double globale **nel** modulo dimezza.c, variabile chiamata **salva**. Il file dimezza.h contiene il prototipo della funzione dimezza.

Il modulo quadrato.c implementa una funzione che prende un argomento double e restituisce un double pari al quadrato dell'argomento passato. Prima di restituire il risultato, la funzione copia il risultato in una variabile double globale **nel** modulo quadrato.c, variabile chiamata **salva**. Il file quadrato.h contiene il prototipo della funzione quadrato.

Il modulo main.c contiene il main del programma e utilizza le funzioni dimezza e quadrato per calcolare il quadrato della meta' del coseno di 13.17. Il main deve scrivere il risultato in una variabile double globale nel modulo main.c, variabile chiamata **salva**. Dopo avere salvato il valore, il main deve stamparlo a video.

Scrivere anche il Makefile del progetto.

Infine, spostare i due file .h in una directory separata e modificare il Makefile per riuscire a compilare tutto.

soluzioni esercizio più moduli e Makefile (1)

caso tutti i file in stessa directory

dimezza.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <math.h>
static double salva=0.0;
double dimezza( double x )
{
        salva = cos( x ) / 2.0;
        return( salva );
}
```

dimezza.h

```
#ifndef __DIMEZZA_H__
#define __DIMEZZA_H__
extern double dimezza( double x );
#endif
```

quadrato.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>

static double salva=0.0;
double quadrato( double x )
{
        salva = x*x;
        return(salva);
}
```

quadrato.h

```
#ifndef __QUADRATO_H__
#define __QUADRATO_H__
extern double quadrato( double x );
#endif
```

main.c

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include "dimezza.h"
#include "quadrato.h"
static double salva=0.0;
int main(void)
{
        salva = quadrato( dimezza( 13.17 ) );
        printf( "risultato = %f \n", salva );
        return(0);
}
```

Makefile

```
CFLAGS=-ansi -Wpedantic -Wall -Werror
LFLAGS=

all:            main.exe

main.exe:       main.o dimezza.o quadrato.o
                gcc ${LFLAGS} -o main.exe  main.o dimezza.o quadrato.o -lm

main.o:         main.c  dimezza.h  quadrato.h
                gcc -c ${CFLAGS} main.c

dimezza.o:      dimezza.c
                gcc -c ${CFLAGS} dimezza.c

quadrato.o:     quadrato.c
                gcc -c ${CFLAGS} quadrato.c

.PHONY:     clean

clean:
                rm -f main.exe main.o dimezza.o quadrato.o  *~  core
```

soluzioni esercizio più moduli e Makefile (2)

caso header file in directory separata

ipotizzo che ci sia una directory principale del progetto in cui ho due sottodirectory SRC e INCLUDE.

- La sottodirectory SRC contiene i moduli .c ed il **Makefile**.

- La sottodirectory INCLUDE contiene gli header files dimezza.h e quadrato.h

Il make deve essere eseguito stando nella directory SRC.

Il tal caso i sorgenti sono come quelli del caso in cui ho tutto in una stessa directory.

Il Makefile invece è cosi:

Makefile

```
INCLUDEDIR=../INCLUDE
CFLAGS=-ansi -Wpedantic -Wall -Werror
LFLAGS=
LIBRARIES=-lm

all:            main.exe

main.exe:       main.o dimezza.o quadrato.o
                gcc ${LFLAGS} -o main.exe  main.o dimezza.o quadrato.o ${LIBRARIES}

main.o:         main.c   ${INCLUDEDIR}/quadrato.h   ${INCLUDEDIR}/dimezza.h
                gcc -c ${CFLAGS} -I${INCLUDEDIR} main.c

dimezza.o:      dimezza.c
                gcc -c ${CFLAGS}  dimezza.c

quadrato.o:     quadrato.c
                gcc -c ${CFLAGS}  quadrato.c

.PHONY:     clean

clean:
                rm -f main.exe main.o dimezza.o quadrato.o  *~  core
```

Macro, Linguaggio C (puntatori e vettori)

Esercizio 2 es2_ALLOCAVETTORE:

Scrivere una macro in C che alloca spazio in memoria per un vettore di 10 interi con segno a 32 bit e che mette in un puntatore ad intero, passato come argomento alla macro, l'indirizzo del blocco allocato. Se l'allocazione di memoria fallisce la macro deve mettere NULL nel puntatore. Se l'allocazione va a buon fine, la macro deve riempire gli elementi del vettore allocato con i valori crescenti da -1000 e fino a -991.

La macro deve essere scritta in un file separato dal file che contiene il main. Questione su cui ragionare: Bisogna scrivere la macro in un file con estensione .c oppure in uno con estensione .h ?

Scrivere un main che contiene il puntatore ad intero, e che chiama la macro, e che controlla la corretta allocazione controllando il valore inserito nel puntatore dalla macro. Il main dopo la chiamata alla macro cambia il valore degli elementi del vettore allocato mettendo valori compresi tra -19 e -10.

Scrivere un Makefile che genera l'eseguibile.

Provare il tutto.

Esercizio 3 es3_macroALLOCAVETTORE :

Modificare il precedente esercizio, in modo da non usare il file che contiene la macro. **Scrivere la stessa macro** non in un file .h o .c ma **mettendola come argomento della chiamata al gcc usata per compilare il main**. Modificare il main, se prima includeva qualcosa. Ovviamente modificare il Makefile.

Soluzioni

Esercizio 2 es2_ALLOCAVETTORE:

main.c

```
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdio.h>
#include "macro1.h“

int main(void) {
        int32_t *p;
        int i;

        ALLOCAVETTORE(p);
        for(i=0;i<10;i++)
                p[i]=-19+i;
        for(i=0;i<10;i++) {
                printf("%d ", p[i] );
                fflush(stdout);
        }
        printf("\n");
        return(0);
}
```

macro1.h

```
#define ALLOCAVETTORE(PTR)                                  \
   do {                                                     \
        PTR=(int32_t*)malloc(10*sizeof(int32_t));           \
        if ( PTR != NULL ) {                                \
            int j;                                          \
            for(j=0;j<10;j++) {                             \
                PTR[j]=-1000+j;                             \
            }                                               \
        }                                                   \
    } while(0)
```

Esercizio 3 es3_macroALLOCAVETTORE:

gcc -ansi -Wpedantic -Wall -c **-D'**ALLOCAVETTORE(PTR)**=**do { PTR=(int32_t*)malloc(10*sizeof(int32_t)); if(PTR!=NULL) { int j; for(j=0;j<10;j++) {PTR[j]=-1000+j; } } } while(0)**'** main.c

utilities

Esercizio 7 es7_find2livelli:

Utilizzare il comando find (guardare il man per le opzioni) per elencare tutti i file presenti nella directory /usr/ il cui nome termina con **i.h**

Devono essere visitate anche le sottodirectory fino ad un massimo di due livelli dentro a /usr/ (quindi ad esempio cercare in /usr/include/scsi ma non in /usr/include/scsi/fd

Esercizio 8 es8_findexecwc:

Estendere il comando precedente in modo che per ciascun file trovato non venga stampato a video solo il nome del file bensi' venga stampata una riga con il nome preceduto dal numero di righe di cui quel file e' composto. Si utilizzi, a tal scopo, il comando **wc** con opportuni parametri.

Esercizio 9 es9_tree:

Lanciare il comando tree –d /usr/lib/

e vedere cosa fa.

Se il comando non esiste allora installarlo, usando sudo, ed eseguendo i seguenti passi:

1) fare update con apt-get. 2) cercare con aptitude se esiste un pacchetto tree.

3) installare il pacchetto con apt-get.

Infine usare il comando come detto all'inizio tree –d /usr/lib/

Soluzioni

Esercizio 7 es7_find2livelli

```
find /usr/ -maxdepth 3 -type f -name "*i.h" -print
```

Esercizio 8 es8_findexecwc

```
find /usr/ -maxdepth 3 -type f -name "*i.h" -exec wc -l  '{}'   \;
```

Esercizio 9 es9_tree :

Nota Bene: quando necessario, usare password studente per user studente

```
sudo   apt-get update
aptitude search  tree
sudo   apt-get install  tree
```

Esercizi su moduli e funzioni di libreria

Esercizio 10 casuale.c: Scrivere un programma **casuale.c** in linguaggio ANSI C che non usa variabili globali e nemmeno variabili locali. Il main esegue un loop. All'inizio del loop viene generato un valore intero casuale. Se il resto modulo 10 del numero è uguale a 3 allora il main esce dal loop e termina.

Scrivere un Makefile per generare l'eseguibile. Obbligo: usare almeno i flag di compilazione **-ansi -Wall -Wpedantic -Werror**

Suggerimento: **usare** la funzione di libreria **rand** per generare i numeri casuali, **inizializzando preventivamente il generatore di numeri casuali chiamando la funzione srand**. **Guardare nel man (sezione 3) di rand** per capire come definire gli opportuni simboli per poter usare le funzioni suggerite.

Esercizio 11 dammi_il_precedente.c: Scrivere un programma in linguaggio ANSI C che **usa solo variabili definite dentro le funzioni**. Il main esegue un loop. All'inizio del loop viene generato un valore intero casuale che viene passato alla funzione unsigned int dammi_il_precedente(unsigned int). Il main ottiene il risultato intero della funzione e, se il resto modulo 10 del numero è uguale a 3, allora il main esce dal loop e termina.

La prima volta che viene chiamata la funzione dammi_il_precedente() restituisce zero. Le volte successive la funzione restituisce il numero che le è stato passato la volta precedente. **La funzione accetta il solo argomento intero e può usare solo variabili dichiarate al suo interno**. Scrivere un Makefile per generare l'eseguibile.

Obbligo: usare almeno i flag di compilazione **-ansi -Wall -Wpedantic -Werror**

Esercizi su moduli e funzioni di libreria - Soluzioni

Esercizio 10 casuale.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/EserciziRand/casuale.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/EserciziRand/Makefile

Esercizio 11 dammi_il_precedente.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/EserciziRand/dammi_il_precedente.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/EserciziRand/MakefileDammiilprecedente

Esercizio su disabilitazione di Wildcard (non banale il Punto3)

Punto1: Scrivere un comando che assegna alla variabile RIGA una sequenza di caratteri che contiene esattamente il carattere 1 seguito da uno spazio bianco seguito dal carattere * seguito da uno spazio bianco seguito dal carattere 2. Cioè 1 * 2

Fate attenzione all'espansione delle wildcard.

Punto2: Scrivere un comando che visualizza sullo stdout il contenuto della variabile RIGA precedentemente inizializzata. Fate attenzione all'espansione delle wildcard, voglio vedere gli asterischi a video.

Punto3: Scrivere uno script **Punto3.sh** che, al suo interno, inizializza come detto al Punto1 la variabile RIGA, poi separa le parole contenute in quella variabile RIGA e le visualizza una per una sullo stdout. Fate attenzione all'espansione delle wildcard, voglio vedere gli asterischi a video.

soluzioni in pagina successiva

Soluzioni Esercizio su disabilitazione di Wildcard

Punto1: RIGA='1 * 2'

Punto2: echo "${RIGA}"
sono invece sbagliati sia
ed anche

echo ${RIGA}
echo '${RIGA}'

Punto3:

```
#!/bin/bash
RIGA='1 * 2'
# itero fino a che la stringa ha qualcosa dentro
while [[ -n "${RIGA}" ]] ; do
        # copio prima parola, cioè quello che c'e' fino al primo bianco
        PAROLA="${RIGA%% *}"
        # Se a fine riga non c'e' spazio bianco allora
        # non sono riuscito ad estrarre la parola,
        # e ho preso tutta la riga.
        # Se ho preso tutta la riga vuol dire che sono alla fine.
        if [[ "${PAROLA}" == "${RIGA}" ]] ; then
                RIGA=
        else
                # rimuovo la prima parola e il successivo bianco
                RIGA="${RIGA#* }"
        fi
        echo "PAROLA is ${PAROLA}"
done
```

Esempio di soluzioni SBAGLIATE del Punto3

```
#!/bin/bash

RIGA='1 * 2'

for PAROLA in ${RIGA} ; do

        echo "PAROLA is ${PAROLA}"
done
```

errore: cosi' interpreta gli *
in particolare lo * viene sostituito dai nomi di file

errore: cosi' vede tutto come una sola unica parola

Esercizio su Script in background (complicato)

Scrivere uno script **estraiasterischi.sh** che attende 2 secondi e poi

- legge delle righe dallo standard input, le righe sono formate da parole e spazi,
- da queste righe estrae le sole parole che contengono il carattere *,
- tali parole vengono concatenate, separate da spazi bianchi, in una variabile TUTTO.
- Alla fine lo script stampa sullo standard output il contenuto della stringa TUTTO.

Scrivere poi un file di testo **input.txt** con almeno tre righe, in ciascuna riga devono esserci almeno tre parole, separate da spazi bianchi, e di queste almeno due parole devono contenere un *

Ad esempio:

i * due ciclisti procedevano *faticosamente

salendo il deso*lato sterrato che po*rta

da Monte*codruzzo ver*so Ciola Araldi

Infine scrivere uno script **lanciaestrazione.sh** che

- lancia in background lo script estraiasterischi.sh passandogli come standard input il contenuto del file input.txt
- dopo questo lancio, lo script manda in background il comando echo fatto
- ed infine attende che termini lo script lanciaestrazione.sh precedentemente lanciato

Soluzione di Script in background (1)

lanciaestrazione.sh

```
#!/bin/bash
 ./estraiasterischi.sh < input.txt   &
PIDLANCIATO=$!
echo   fatto   &
wait ${PIDLANCIATO}
```

Soluzione di Script in background (2)

estraiasterischi.sh **Attenzione ai " " e ai ' '**

```
#!/bin/bash
sleep 2
TUTTO=
while   read  RIGA ;   do
            # faccio parsing estraendo i pezzi e manipolo le stringhe con i quoting
            # per evitare problemi con gli asterischi e wildcard in genere.
            # Prima rimuovo un eventuale spazio banco iniziale
            RIGA="${RIGA# }"

            # itero fino a che la stringa ha qualcosa dentro
            while [[ -n "${RIGA}" ]] ; do
                        # copio prima parola, quello che c'e' fino al primo bianco
                        PAROLA="${RIGA%% *}"
                        # se a fine riga non c'e' spazio bianco non sono riuscito ad estrarre la parola,
                        # e ho preso tutta la riga
                        # se ho preso tutta la riga vuol dire che sono alla fine.
                        if [[ "${PAROLA}" == "${RIGA}" ]] ; then
                                    RIGA=
                        else
                                    # rimuovo la prima parola e il successivo bianco
                                    RIGA="${RIGA#* }"
                        fi

                        if     echo "${PAROLA}"   |  grep '*'  > /dev/null   ;  then
                                    TUTTO="${TUTTO}  ${PAROLA}"
                        fi
            done
done
echo  "${TUTTO}"                              notare che servono i " "
```

Alcune Soluzioni ERRATE di Script in background

estraiasterischi.ERRATO.sh **PROVOCA PROBLEMI CON GLI ***

```
#!/bin/bash
sleep 5
TUTTO=
while   read  RIGA ;   do

        for PAROLA in ${RIGA} ; do              errore: cosi' interpreta gli *

        for PAROLA in ${RIGA} ; do              errore: cosi' vede tutto come una sola parola

                if     echo "${PAROLA}"   |  grep '*'   ;  then
                        TUTTO="${TUTTO}  ${PAROLA}"
                fi
        done
done
echo  "${TUTTO}"                        notare che servono i " "
```

Lezione 7 in laboratorio

Makefile, Installazioni, Linguaggio C, Include, Librerie

- Occhio a quello che fate

Makefile

Esercizio 1 es1_tutto:

Scrivere un Makefile che permette di ottenere/rigenerare un file tutto.txt secondo le seguenti regole:

Il target **all** ordina la creazione del file tutto.txt

Il target **clean** ordina l'eliminazione dei file A1.txt e A2.txt

Il target **touch** aggiorna la data del file B1.txt , se questo esiste già.

Il file tutto.txt deve contenere il concatenamento dei contenuti aggiornati dei file B1.txt e B2.txt

Si ricorda che il comando ***date*** mette sullo stdout l'istante attuale.

Il file B1.txt deve contenere il contenuto del file A1.txt seguito da una riga di testo che specifica l'istante in cui il Makefile modifica il file B1.txt stesso.

Il file B2.txt deve contenere il contenuto del file A2.txt preceduto da una riga di testo che specifica l'istante in cui il Makefile modifica il file B2.txt stesso.

Se il file A1.txt non esiste deve essere creato inserendovi il contenuto "AAAA"

Se il file A2.txt non esiste deve essere creato inserendovi il contenuto "ZZZZ"

Provare il Makefile e i diversi target

Makefile : Soluzioni

Esercizio 1 es1_tutto: Makefile

```
SHELL=/bin/bash

all:    tutto.txt

tutto.txt :      B1.txt B2.txt
            cp B1.txt tutto.txt
            cat B2.txt >> tutto.txt

B1.txt :  A1.txt
            cp A1.txt B1.txt
            date >> B1.txt

B2.txt :  A2.txt
            date > B2.txt
            cat A2.txt >> B2.txt

   segue   ---------------------->
```

```
A1.txt :
            echo AAAA > A1.txt

A2.txt :
            echo ZZZZ > A2.txt

.PHONY :            clean touch

clean :
            rm   A1.txt   A2.txt

touch :
            if [[  -e  B1.txt  ]] ; then touch B1.txt ; fi
```

Installare e usare la libreria cmph (1)

Esercizio: installare la libreria cmph in una directory utente.

Nella home directory /home/studente creare una sottodirectory LIB

Nella home directory /home/studente creare una sottodirectory CMPH e andare lì.

Download del codice sorgente della libreria usando il comando wget (vedi man)

http://downloads.sourceforge.net/project/cmph/cmph/cmph-2.0.tar.gz

Usare il comando tar per spacchettare l'archivio scaricato (vedi man).

Spostarsi nella directory cmph-2.0 appena creata.

A) Bisogna compilare e installare sia la libreria dinamica (shared) che quella statica e, inoltre B) le librerie dovranno essere installate in /home/studente/LIB

Leggere il file INSTALL per capire come configurare la directory in cui installare la libreria. Guardare in particolare la parte "Installation Names".

Il software da installare fornisce **uno script configure** che **serve a creare i Makefile** per compilare ed installare la libreria. Lanciare il comando ./configure con i parametri necessari per creare i Makefile con le impostazioni necessarie per ottenere A) e B)

Lanciare il comando ./configure –help per capire come far compilare e installare sia la libreria dinamica (detta shared) che quella statica e nella directory specificata.

Continua in slide successiva**...**

Installare e usare la libreria cmph (2)

Lanciare make per compilare e linkare

Lanciare make check per verificare compilazione

Lanciare make install per installare

Verificare se nella directory /home/studente/LIB/lib ci sono le librerie

Verificare se nella directory /home/studente/LIB/include ci sono gli include *.h

Provare le librerie installate

Creare una directory /home/studente/SRC e spostarsi lì dentro.
Copiarvi dentro il file /home/studente/CMPH/cmph-2.0/examples/file_adapter_ex2.c

Il sorgente C deve essere compilato specificando al gcc, a riga di comando, dove si trovano i file di inclusione della libreria cmph.
Solo per questo esercizio, NON usare i flag **-**ansi e -Wpedantic poiché i sorgenti non sono ANSI C in quanto usano estensioni quali commenti // e keyword inline.

Il modulo deve poi essere linkato con la libreria cmph installata. Capire **come** dire al gcc dove si trova la libreria cmph

Scrivere un opportuno Makefile.
Generare l'eseguibile.
Nella directory dell'eseguibile creare un file keys.txt con qualche riga con una parola
Lanciare l'eseguibile generato.

Soluzioni su installare e usare libreria cmph (1)

Spacchettare l'archivio

```
tar  xvzf  cmph-2.0.tar.gz
```

Configurazione delle librerie

```
./configure   --prefix=/home/studente/LIB    --enable-static   --enable-shared
```

Compilazione e linking dell'esempio

```
gcc  -c  file_adapter_ex2.c  -I/home/studente/LIB/include

gcc  -o  file_adapter_ex2.exe    -Wl,-rpath,/home/studente/LIB/lib      \
         -L/home/studente/LIB/lib  file_adapter_ex2.o  -lcmph
```

Soluzioni su installare e usare libreria cmph (2)

il Makefile

```
LIBRARIES=-lcmph
RUNTIME_LIBPATH=/home/studente/LIB/lib
LINKTIME_LIBPATH=/home/studente/LIB/lib
INCLUDEPATH=/home/studente/LIB/include

all:                    file_adapter_ex2.exe

file_adapter_ex2.exe:   file_adapter_ex2.o
        gcc -o file_adapter_ex2.exe    -Wl,-rpath,${RUNTIME_LIBPATH}     \
                -L${LINKTIME_LIBPATH}    file_adapter_ex2.o      ${LIBRARIES}

file_adapter_ex2.o:     file_adapter_ex2.c
        gcc   -c   file_adapter_ex2.c   -I${INCLUDEPATH}

.PHONY:  clean

clean:
        rm  -f   file_adapter_ex2.exe    file_adapter_ex2.o
```

Generare codice (1/2)

Esercizio creaC.sh creaH.sh

Un programma scritto in ANSI C è formato da più moduli.

Il main.c include il file define.h il quale contiene una sola riga: #define NUM 1000

Un modulo variabiliglobali.c contiene le dichiarazioni di tante variabili globali intere quante indicate dal simbolo NUM, variabili aventi nome var1, var2, var3, ,, . var1000, inizializzate rispettivamente a 1,2,3, ... , 1000.

Il modulo variabiliglobali.c implementa una funzione int conta(void); che restituisce la somma dei valori di tutte le variabili intere varQualcosa.

Il file variabiliglobali.h contiene le dichiarazioni extern di quelle stesse variabili.

Il file variabiliglobali.h contiene anche il prototipo della funzione int conta(void);

Il main include anche il file variabiliglobali.h

Il main include gli header delle comuni funzioni di libreria standard del C.

Il main stampa a video il valore di NUM seguito dal valore calcolato dalla funzione int conta(void) seguito dal valore della variabile var1;

segue dopo -------->

Generare codice (2/2)

…. Continuazione Esercizio creaC.sh creaH.sh

Scrivere uno script **creaC.sh** che legge il file define.h estrae il valore di NUM e, usando quel valore, ricrea il file variabiliglobali.c

Scrivere uno script **creaH.sh** che legge il file define.h estrae il valore di NUM e, usando quel valore, ricrea il file variabiliglobali.h

Scrivere un **Makefile** che usa il file define.h come dipendenza di variabiliglobali.c e di variabiliglobali.h e che invoca i due script creaC.sh e creaH.sh se il file define.h è stato modificato più recentemente dei due files variabiliglobali.c e di variabiliglobali.h

il Makefile inoltre serve a compilare e linkare il programma costituito dai moduli main.c e variabiliglobali.c

Scrivere opportunamente il file **main.c** ed il file **define.h**

Verificare che il Makefile riesca a creare l'eseguibile.

Verificare che il Makefile riesca a rigenerare automaticamente l'eseguibile SE CAMBIAMO IL VALORE DI NUM SOSTITUENDO 1000 con 1010

Soluzioni Generare codice

Soluzioni Esercizio creaC.sh creaH.sh

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/CREA_C_H.tgz

Vari (1)

Esercizio alterna.sh

Scrivere uno script **alterna.sh** che prende come argomenti i percorsi assoluti o relativi di due file (che chiameremo file sorgenti) e che scrive in un fine out.txt le righe dei due file, alternandone una di un file ed una dell'altro e mettendo poi nel file le righe rimanenti del file più grande.

Scrivere inoltre un Makefile che stabilisce quali sono i due file da mixare e che controlla la generazione del file out.txt. Tale file viene creato/modificato solo se non esiste oppure se è più vecchio di almeno uno dei due file sorgenti. La rigenerazione del file out.txt comincia con l'eliminazione del file e la successiva esecuzione del file alterna.sh

Esercizio random09.sh

Scrivere uno script **random09.sh** che sfrutta la variabile bash RANDOM e scrive sullo standard output un numero casuale compreso tra 0 e 9.

Esercizio genera.sh

Scrivere uno script genera.sh che prende come argomento il nome di un file e che usa il precedente script random09.sh per aggiungere al file una riga contenente il numero casuale generato.

Esercizio lanciagenera.sh

scrivere uno script che elimina il file out.txt, se già esiste, poi lancia 10 volte lo script genera.sh facendogli aggiungere righe al file out.txt, infine somma i valori numerici contenuti nel file out.txt e stampa a video la somma.

Soluzioni Vari (1)'

Soluzione Esercizio alterna.sh

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/SCRIPT_ESERCIZI/alterna.sh

Soluzione Esercizio random09.sh

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/SCRIPT_ESERCIZI/random09.sh

Soluzione Esercizio genera.sh

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/SCRIPT_ESERCIZI/genera.sh

Soluzione Esercizio lanciagenera.sh

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/SCRIPT_ESERCIZI/lanciagenera.sh

TextProcessing (1)

Esercizio strippa1.sh

Scrivere uno script **strippa1.sh** che cerca tutte le sottodirectory di /usr/include/ e, per ciascuna di queste:

se esiste il 7° carattere del nome della directory (non del percorso, proprio il nome), lo aggiunge in una riga al file 7.txt, altrimenti prosegue,

se esiste l' 8° carattere del nome della directory (non del percorso, proprio il nome), lo aggiunge in una riga al file 8.txt, altrimenti prosegue.

Esercizio contadirettivenonripeture.sh

Scrivere uno script **contadirettivenonripeture.sh** che

cerca tutti i file con estensione .h nella directory /usr/include/ e in tutte le sue sottodirectory

considera di questi file le sole righe che contengono almeno un carattere #

conta il numero di queste righe per tutti i file, ma se ci sono delle righe uguali le conta una volta sola

suggerimento:

capire cosa fanno il comando **sort** ed il comando **uniq**

Soluzioni TextProcessing (1)'

Esercizio strippa1.sh

```
for percorso in `find /usr/include/ -mindepth 1 -type f ` ; do
   name=${percorso##*/}
   if (( ${#name} >= 7 )) ; then  echo ${name:6:1} >> 7.txt
            if (( ${#name} >= 8 )) ; then  echo ${name:7:1} >> 8.txt ;  fi  ;   fi  ;  done
```

Esercizio contadirettivenonripeture.sh

```
for name in `find /usr/include/ -type f -name "*.h"` ; do grep '#' ${name} ; done | sort
| uniq |  wc -l
```

Ricompila al cambiare del Makefile (1)

Esercizio MakeAlVariareDelmake

Scaricare l'archivio simplemake.tgz che si trova a questo link http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/MAKEFILE_ESEMPINOTEVOLI/simplemake.tgz

Decomprimerlo e guardare il codice ed il Makefile.

Modificare il Makefile in modo tale che, se viene aggiornata la data di ultima modifica del Makefile, il comando make produce la ricompilazione e linking di tutto il codice.

Soluzioni Ricompila al cambiare del Makefile (1)'

Makefile

```
all :                main.exe

main.exe :           main.o  funzioni.o
                     gcc  -o main.exe main.o funzioni.o

main.o : main.c  funzioni.h strutture.h Makefile
                     gcc -c -ansi -Wpedantic -Wall main.c

funzioni.o :         funzioni.c  strutture.h Makefile
                     gcc -c -ansi -Wpedantic -Wall funzioni.c

clean:
                     -rm     main.exe *.o
```

Es MakefileRicorsiviELibrerie

MakefileRicorsiviELibrerie (1)

Creare la directory /home/studente/esMake/ e le sue sottodirectory Main LibA e libB

Nella directory LibA implementare un modulo A.c che contiene una funzione calcolaA che restituisce il coseno dell'argomento double passato alla funzione. Scrivere inoltre un file A.h che contiene il prototipo della funzione calcolaA. Realizzare inoltre un Makefile che genera la libreria dinamica libA.so che mette a disposizione quella funzione calcolaA. La libreria deve essere prodotta e mantenuta nella directory stessa.

Nella directory LibB implementare un modulo B.c che contiene una funzione calcolaB che restituisce la metà dell'argomento double passato alla funzione. Scrivere inoltre un file B.h che contiene il prototipo della funzione calcolaB. Realizzare inoltre un Makefile che genera la libreria dinamica libB.so che mette a disposizione quella funzione calcolaB. La libreria deve essere prodotta e mantenuta nella directory stessa.

Nella directory Main scrivere un file main.c che prende un argomento intero a riga di comando, calcola calcolaB di quel numero e usa il risultato come argomento di calcolaA, stampando a video il risultato di quest'ultima funzione. Scrivere un Makefile che compila e linka main e librerie per generare l'eseguibile main.exe.

Nella directory principale esMake scrivere un Makefile che fa eseguire i make di tutte le 3 sottodirectory per creare le librerie libA e libB e l'eseguibile main.exe

Ciascun Makefile deve prevedere un target fittizio clean per eliminare tutti i moduli oggetto, le librerie e gli eseguibili create nelle directory di propria competenza.

Anche il Makefile nella directory principale deve contenere un target clean che richiama i target clean di tutti gli altri Makefile.

Soluzioni MakefileRicorsiviELibrerie (1)'

ça va sans dire, tutto deve essere implementato in linguaggio ANSI C, usando i flag di compilazione più restrittivi e prudenziali

-ansi -Wall -Wpedantic -Werror

Soluzioni

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/MakefileRicorsiviELibrerie.tgz