Laboratorio di Elementi di Architetture e Sistemi Operativi

Soluzioni degli esercizi del 23 Maggio 2012

Esercizio 1.

1. *Scaricare dalla pagina web del corso l'archivio* Lab 11 - Materiale aggiuntivo per gli esercizi.
2. *Estrarre i file dal pacchetto usando il comando* `unzip`.
   ```
   $ unzip matdid848902.zip
   ```
3. *Creare un* `Makefile` *per compilare il codice di esempio con il seguente contenuto:*
   ```
   main: main.o f1.o f2.o
           gcc main.o f1.o f2.o -o main
   main.o: main.c f1.h f2.h
           gcc -c main.c
   f1.o: f1.h f1.c
           gcc -c f1.c
   f2.o: f2.h f2.c
           gcc -c f2.c
   clean:
           rm *.o main
   ```
4. *Qual'è il risultato dell'esecuzione del comando* `make`*?*
   ```
   $ make
   gcc -c main.c
   gcc -c f1.c
   gcc -c f2.c
   gcc main.o f1.o f2.o -o main
   ```
   Il comando compila i file sorgente e genera l'eseguibile `main`.
5. *Eseguire il comando* `touch f1.c`*. Cosa succede quando si esegue nuovamente il comando* `make`*?*
   ```
   $ touch f1.c
   $ make
   gcc -c f1.c
   gcc main.o f1.o f2.o -o main
   ```
   In questo caso viene ricompilato solo il file `f1.c` ed eseguito nuovamente il linking per generare l'eseguibile.
6. *Eseguire il comando* `touch f1.h`*. Cosa succede se si esegue il comando* `make`*? Quali sono le differenze rispetto al punto precedente?*
   ```
   $ touch f1.h
   $ make
   gcc -c main.c
   gcc -c f1.c
   gcc main.o f1.o f2.o -o main
   ```
   In questo caso, poiché anche `main.o` dipende da `f1.h`, vengono ricompilati i file `f1.c` e `main.c`, oltre ad effettuare nuovamente il linking per generare l'eseguibile.
7. *Eseguire il comando* `make clean` *e descriverne il risultato.*
   Il comando cancella i file oggetto `.o` e l'eseguibile del programma.
8. *Modificare il* `Makefile` *sostituendo le regole per* `main.o`*,* `f1.o` *e* `f2.o` *con la regola implicita*
   ```
   %.o : %.c
           gcc -o $@ -c $<
   ```
9. *Eseguire nuovamente* `make` *e verificarne il funzionamento.*

```
$ make
gcc -o main.o -c main.c
gcc -o f1.o -c f1.c
gcc -o f2.o -c f2.c
gcc main.o f1.o f2.o -o main
```
Il comando `make` ha lo stesso effetto del punto 4 e genera correttamente l'eseguibile.

10. *Eseguire il comando `touch f1.h`. Cosa succede se si esegue il comando `make`? Il comportamento del `make` è corretto?*
```
$ touch f1.h
$ make
make: 'main' is up to date.
```
In questo caso il `make` afferma che non c'è nulla da fare perché `main` è aggiornato. Questo non è corretto: la modifica di un file header deve provocare la ricompilazione di tutti i file che lo includono.

11. *Modificare il `Makefile` in modo che le modifiche agli header file causino la corretta ricompilazione dei file oggetto.*
In questo caso è necessario includere esplicitamente le dipendenze dai file header, per esempio aggiungendo le seguenti regole nel `Makefile`:
```
main.o: f1.h f2.h
f1.o: f1.h
f2.o: f2.h
```

Esercizio 2. *Si consideri il codice per l'esercizio 2 della lezione scorsa. Si scriva un `Makefile` che esegua le seguenti operazioni:*

1. *generare la libreria statica per la gestione delle liste;*
2. *generare l'eseguibile del programma, linkato staticamente alla libreria;*
3. *fornire un target `install` che esegua le seguenti operazioni:*
   - *copiare l'eseguibile in `$HOME/bin`;*
   - *copiare la libreria statica in `$HOME/lib`;*
   - *copiare gli header file della libreria in `$HOME/include`;*
4. *fornire un target `archive` che crei un archivio `liste.zip` usando il comando `zip` che contenga solamente il `Makefile` ed i file `.c` ed `.h` con il codice;*
5. *fornire un target `clean` per rimuovere i file oggetto, l'eseguibile, le librerie e l'archivio.*

```
minmaxlist: minmaxlist.o liblistedoppie.a
	gcc -o $@ $< -L. -llistedoppie

liblistedoppie.a: listedoppie.o
	ar r $@ $<

%.o : %.c
	gcc -c -o $@ $<

clean:
	rm -f *.o *.a minmaxlist liste.zip

install: minmaxlist liblistedoppie.a listedoppie.h
	cp minmaxlist $(HOME)/bin
	cp liblistedoppie.a $(HOME)/lib
	cp listedoppie.h $(HOME)/include
```

```
archive: liste.zip

liste.zip: *.c listedoppie.h Makefile
        zip liste.zip $^

listedoppie.o: listedoppie.h
minmaxlist.o: listedoppie.h
```

Verificare il corretto funzionamento del `Makefile`:

- *verificare che generi correttamente l'eseguibile;*

```
$ make
gcc -c -o minmaxlist.o minmaxlist.c
gcc -c -o listedoppie.o listedoppie.c
ar r liblistedoppie.a listedoppie.o
ar: creating archive liblistedoppie.a
gcc -o minmaxlist minmaxlist.o -L. -llistedoppie
```

- *verificare che i target `clean` e `install` eseguano il loro compito correttamente;*

```
$ make clean
rm -f *.o *.a minmaxlist liste.zip

$ make install
gcc -c -o minmaxlist.o minmaxlist.c
gcc -c -o listedoppie.o listedoppie.c
ar r liblistedoppie.a listedoppie.o
ar: creating archive liblistedoppie.a
gcc -o minmaxlist minmaxlist.o -L. -llistedoppie
cp minmaxlist /Users/davide/bin
cp liblistedoppie.a /Users/davide/lib
cp listedoppie.h /Users/davide/include
```

- *verificare che l'archivio creato con `make archive` non contenga file oggetto, librerie od eseguibili;*

```
$ make archive
zip liste.zip listedoppie.c minmaxlist.c listedoppie.h Makefile
  adding: listedoppie.c (deflated 73%)
  adding: minmaxlist.c (deflated 49%)
  adding: listedoppie.h (deflated 59%)
  adding: Makefile (deflated 57%)
$ unzip -l liste.zip
Archive:  liste.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
     2118  05-29-2012 12:48   listedoppie.c
      529  05-29-2012 12:43   minmaxlist.c
      521  05-29-2012 12:48   listedoppie.h
      488  05-29-2012 12:47   Makefile
---------                     -------
     3656                     4 files
```

- *dopo aver estratto l'archivio in una directory separata, verificare se il comando* `make`*, eseguito in quella directory, genera correttamente il programma.*

```
$ mkdir prova
$ cd prova
$ unzip ../liste.zip
Archive:  ../liste.zip
  inflating: listedoppie.c
  inflating: minmaxlist.c
  inflating: listedoppie.h
  inflating: Makefile
$ make
gcc -c -o minmaxlist.o minmaxlist.c
gcc -c -o listedoppie.o listedoppie.c
ar r liblistedoppie.a listedoppie.o
ar: creating archive liblistedoppie.a
gcc -o minmaxlist minmaxlist.o -L. -llistedoppie
```

Nota: Nei makefile è possibile utilizzare tutti i costrutti degli shell script che abbiamo visto nella prima parte del corso. Per esempio, è possibile utilizzare il costrutto `if` per definire una versione più sofisticata del target `install`, che controlli l'esistenza delle directory dove copiare i file, e le crei se mancano.

Siccome il `make` passa i comandi alla shell *una riga alla volta*, i costrutti complessi devono essere scritti su una sola riga per essere eseguiti correttamente. Per migliorare la leggibilità del codice, si può utilizzare il simbolo speciale \, che posto alla fine di una riga indica che il comando continua anche sulla riga successiva.

```
install: minmaxlist liblistedoppie.a listedoppie.h
    if [ ! -e $(HOME)/bin ] ; then \
        mkdir $(HOME)/bin; \
    fi
    cp minmaxlist $(HOME)/bin;
    if [ ! -e $(HOME)/lib ] ; then \
        mkdir $(HOME)/lib; \
    fi
    cp liblistedoppie.a $(HOME)/lib
    if [ ! -e $(HOME)/include ] ; then \
        mkdir $(HOME)/include; \
    fi
    cp listedoppie.h $(HOME)/include
```