Corso di laurea in fisica
Anno Accademico 2020/2021
Laboratorio di informatica 1

Esercitazione 9

Nota 1 Un programma per calcolatore può essere strutturato a diversi livelli. In questo esercizio ci concentriamo su due di questi livelli. Il primo poggia sul concetto di *funzione*, cioè di porzione di codice che svolge un compito ben definito e comunica con il resto del programma attraverso i suoi argomenti e il valore prodotto e restituito. Il secondo livello di strutturazione è dato dalla possibilità di ripartire il codice di un programma in più file.

In generale, questa ripartizione dovrebbe obbedire a criteri di separazione funzionale fra parti diverse di un programma, e permette di (a) nascondere i dettagli di realizzazione delle strutture dei dati e degli algoritmi alle parti del programma che possono prescindere da quei dettagli; (b) rendere quelle strutture e quegli algoritmi potenzialmente riutilizzabili anche in altri programmi. Consideriamo qui un caso molto semplice, ma tipico nella struttura, di ripartizione.

Il programma si compone di tre file, che trovate nella sezione *Materiale* del sito. Il file `parentesi.c` realizza un algoritmo che usa una pila per verificare se le parentesi che compaiono in un'espressione sono correttamente bilanciate. Il file `parentesi.c` non contiene informazioni sulla struttura interna di una pila e sul modo di realizzare le operazioni di base (*push*, *pop*, e così via), ma include il file `pila.h`. Questo file contiene la cosiddetta *interfaccia* della struttura "pila", cioè l'elenco delle operazioni con le quali si può maneggiare una pila. Di ogni operazione, è riportato il *prototipo*, cioè il nome della funzione, il tipo degli argomenti, e il tipo del valore eventualmente prodotto dalla funzione. Queste sono le informazioni sintattiche sufficienti per adoperare quelle operazioni.

L'effettiva definizione dei tipi di dati con i quali si realizza la pila (in questo esempio, una lista concatenata, ma si potrebbe adoperare un *array*, o altro) e delle funzioni corrispondenti alle operazioni di base e a quelle ausiliarie si trova nel file `pila.c`. Questo file non contiene un programma completo (manca il metodo `main`), e si può quindi usare solo come componente. D'altra parte, `pila.c` non è legato indissolubilmente al programma di verifica delle parentesi, ma si può adoperare ogni volta che occorra una pila. Esso forma perciò una *libreria* (traduzione maccheronica dell'inglese *library*) di funzioni: un componente *software* riutilizzabile in diversi contesti.

Per compilare il programma delle parentesi, scrivete

(1) `gcc -Wall -std=c99 -o parentesi parentesi.c pila.c`

I tre file devono trovarsi nella stessa *directory*. Verificate se il programma funziona, e confrontatelo con `eparentesi.c` per dedurre i principi di separazione del codice.

Nota tecnica: l'inclusione di `pila.h` *ha una forma leggermente diversa dal solito: il nome del file è racchiuso fra virgolette; per convenzione, in questo caso il compilatore cerca il file nella stessa directory in cui si trova il file che sta compilando.*

È possibile compilare separatamente pila.c; in questo caso non otterremo

un programma eseguibile, dato che `pila.c` non contiene la funzione `main`, ma il codice eseguibile delle sue funzioni, memorizzato in un file *oggetto*.

Il corrispondente comando di compilazione è il seguente:

(2) `gcc -Wall -std=c99 -c pila.c`

Il risultato sarà un file di nome `pila.o`. D'ora in poi, per compilare un programma completo che include `pila.h`, potremo sostituire, nel comando (1), `pila.c` con `pila.o`.

Esercizi avanzati: sapreste riscrivere `pila.c` in modo da realizzare la pila per mezzo di un array? Ancora più difficile: come si può modificare `pila.c` in modo che un programma possa dichiarare più variabili di tipo *Pila*?

1 Scrivete una funzione che riceve come parametri un numero intero positivo e un vettore di caratteri, e riempie il vettore con le cifre della rappresentazione dello stesso numero in base 2.

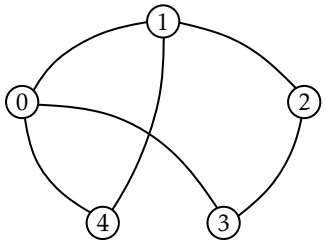
Scrivete una funzione che riceve come parametri: (1) una stringa di caratteri che rappresenta un numero intero in base 10; un vettore di caratteri. La funzione deve riempire il vettore con le cifre della rappresentazione dello stesso numero in base 2.

Scrivete una funzione che compie la conversione nel senso opposto.

2 Scrivete un programma che legga da file la descrizione di un grafo e ne costruisca una rappresentazione mediante matrice di incidenza. Il formato del file è il seguente:

```
5
0 1
1 2
3 2
1 4
0 3
0 4
0 0
```

La prima riga contiene il numero di vertici (nodi) del grafo (numerati a partire da 0); ogni riga successiva contiene una coppia di numeri corrispondente ai vertici sui quali incide un arco del grafo. La coppia `0 0` segnala la fine dei dati e non corrisponde a un arco del grafo. L'esempio precedente descrive il seguente grafo:



Dopo aver costruito la rappresentazione del grafo, il programma dovrà stampare l'istogramma dei gradi dei vertici (il *grado* di un vertice è il numero di

vertici adiacenti; l'istogramma va stampato in orizzontale).

3 I valori interi rappresentabili nella memoria di un calcolatore sono limitati. Supponiamo di voler sommare due numeri interi più grandi del massimo valore rappresentabile: possiamo pensare di memorizzare le cifre (decimali) dei due numeri in due vettori di, poniamo, dimensione 100. Scrivete una funzione che riceve come argomenti due numeri così rappresentati, ne calcola la somma e pone il risultato in un terzo vettore; scrivete una funzione analoga che moltiplica due numeri.

In entrambi i casi, riflettete sulla dimensione del vettore che ospita il risultato.