Laboratorio di algoritmi e strutture dati

Docente: Violetta Lonati

Esercizi sulle funzioni in C*

1 Esercizi introduttivi

1.1 Garibaldi fu ferito

Considerate il codice contenuto nel file `garibaldi.c`. Analizzate il codice sorgente e rispondete per iscritto alle seguenti domande. Se avete dubbi, potete testarlo, eseguendolo su casi di input significativi e modificandolo.

```c
#include <stdio.h>

char garibaldi( char a, char b ){
    if ( b == 'a' || b == 'e' || b == 'i' || b == 'o' || b == 'u' )
        return a;
    else return b;
}

int main() {
    char c, vocale;

    vocale = getchar();
    getchar();

    while ( ( c = getchar() ) != '.' )
        printf( "%c", garibaldi( vocale, c ) );

    printf( "\n" );
    return 0;
}
```

1. Senza eseguire il programma al computer, simulatene l'esecuzione su carta e stabilite cosa stampa il programma quando riceve da standard input la sequenza di caratteri:

   `u garibaldi fu ferito, fu ferito in una gamba.`

2. Descrivete a parole cosa fa la funzione `garibaldi`.
3. Riassumete a parole cosa fa il programma.

1.2 Primo

Scrivete una funzione con parametro un intero n che stabilisca se n è un numero primo. Scrivete la funzione partendo dal programma che avete scritto per l'esercizio 5 della scheda "L01-lab".

*Ultima modifica 21 ottobre 2021

1.3 Potenza

Scrivete una funzione ricorsiva avente due parametri interi b ed e che calcoli la potenza b^e.

1.4 Sequenze di Collatz

Considerate la seguente regola: dato un numero intero positivo n, se n è pari lo si divide per 2, se è dispari lo si moltiplica per 3 e si aggiunge 1 al risultato. Quando n è 1 ci si ferma.

Questa semplice regola permette di costruire delle sequenze: la sequenza che si costruisce a partire dal numero n è detta *sequenza di Collatz di n*. Ad esempio, la sequenza di Collatz di 7 è:

```
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
```

E' un noto problema aperto stabilire se ogni sequenza di Collatz termina (cioè, se arriva a 1).

Scrivete innanzitutto una funzione che, dato un numero, dia il successivo in una sequenza di Collatz. Quindi, inseritela in un programma che chiede all'utente un numero e mostra la sequenza di Collatz del numero (con tanto di lunghezza).

Esempi di funzionamento

```
Numero: 7
7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Lunghezza: 17
```

```
Numero: 9
9 28 14 7 22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1
Lunghezza: 20
```

2 Ricorsione e iterazione

Considerate questa funzione ricorsiva `f_rec`

```
1 unsigned long f_rec(int n){
2   if ( n == 1 || n == 2 ){
3     return 1;
4   }
5   return f_rec( n - 1 ) + f_rec( n - 2 );
6 }
```

Senza eseguire la funzione al computer, rispondete alle seguenti domande:

1. Cosa restituisce la funzione `f(7)`?
2. Perché `n` è dichiarato come intero mentre il valore restituito è di tipo `unsigned long`?
3. Riassumete a parole cosa restituisce la funzione se riceve come argomento un intero positivo n maggiore di 0.

Considerate ora le due funzioni `f_iter1` e `f_iter2`

```c
unsigned long f_iter1(int n){
  unsigned long f, f1 = 1, f2 = 1;
  if ( n == 2 | n == 1 ) {
    return 1;
  }
  while (n-- >= 3){
    f = f1 + f2;
    f1 = f2;
    f2 = f;
  }
  return f;
}
```

```c
unsigned long f_iter2(int n){
  unsigned long f, f1 = 1, f2 = 1, i;
  if ( n == 2 | n == 1 ) {
    return 1;
  }

  for ( i = 2; i <= n; i++ ) {
    f = f1 + f2;
    f1 = f2;
    f2 = f;
  }
  return f;
}
```

Senza eseguire la funzione al computer, rispondete alle seguenti domande:

4. Considerando solo il valore restituito, le due funzioni sono equivalenti? (ovvero: restituiscono sempre lo stesso valore?)
5. Le due funzioni sono equivalenti alla funzione `f_rec`?
6. Modificate (se necessario) le funzioni `f_iter1` e `f_iter2` in modo che risultino essere equivalenti a `f_rec`.
7. Stimate il numero di operazioni che si svolgono durante l'esecuzione di `f_rec`, `f_iter1` e `f_iter2`: sono paragonabili?

Considerate infine la seguente funzione ricorsiva `f_riter`

```c
unsigned long f_riter(unsigned long a, unsigned long b, int n){
  if ( n == 2 ) {
    return a;
  }
  if ( n == 1 ) {
    return b;
  }
  return f_riter( a + b, a, n - 1 );
}
```

Senza eseguire la funzione al computer, rispondete alle seguenti domande:

8. Convincetevi che questa funzione può essere usata per calcolare `f_rec`. In particolare: con quali argomenti devo invocare `f_riter` per ottenere il valore restituito da `f_rec(n)`?
9. Rappresentate graficamente lo schema delle chiamate ricorsive definiti dall'invocazione `f_rec(7)` e dalla chiamata equivalente del tipo `f_riter(..., ..., ...)`.
10. Considerate il numero di chiamate ricorsive effettute da `f_rec(n)` e dalla chiamata equivalente del tipo `f_riter(..., ..., ...)`. Sono paragonabili?
11. Usate una variabile globale `counter` per tenere traccia del numero delle chiamate ricorsive; quindi scrivete un programma che invoca `f_rec` e `f_riter` e stampa, oltre al valore restituito, anche il numero di chiamate della funzione.

Una volta concluso l'esercizio potete usare il file `rec-iter.c` per fare degli esperimenti. Analizzate il codice per capire come usarlo e come interpretarne l'output!

3 Esercizi sulle funzioni ricorsive

I seguenti esercizi vanno risolti usando la ricorsione. Prima di cominciare è fondamentale che abbiate svolto gli esercizi precedenti per prendere familiarità con l'uso e l'implementazione delle funzioni in C.

3.1 Fiocco di Koch

Libreria `libpsgraph.c`

Questo esercizio fa uso di una libreria chiamata `libpsgraph.c` che consente di produrre dei semplici grafici in PostScript. Per usare questa libreria dovete procedere come segue:

- create una directory;
- salvate in essa i file `libpsgraph.c` e `libpsgraph.h`;
- scrivete la vostra applicazione, aggiungendo in testa la direttiva

```
#include "libpsgraph.h"
```

- compilate con il comando

```
gcc libpsgraph.c pippo.c -o pippo -lm
```

  dove al posto di `pippo.c` metterete il nome del vostro file sorgente, mentre al posto di `pippo` metterete il nome che volete che abbia il vostro eseguibile.

Funzioni fornite dalla libreria

La libreria vi permette di disegnare usando le funzioni della *turtlegraphics*; il disegno prodotto viene scritto in formato PostScript su un file (il cui nome dovete specificare quando iniziate a disegnare) e dopo aver eseguito il programma potete vedere il risultato aprendo il file con un opportuno viewer (`ggv` oppure `gv`, o un altro viewer che vi verrà segnalato dai tutor).

La libreria va usata così:

- per prima cosa, invocate la funzione `start(nomefile)` passandole il nome del file in cui volete che il grafico venga salvato, ad esempio `start("prova.ps");`
- a questo punto, per disegnare potete usare le seguenti funzioni (che prendono come argomento un double):
  - `draw(x)`: disegna un segmento lungo `x` millimetri;
  - `move(x)`: si sposta (senza disegnare) di un segmento lungo `x` millimetri;
  - `turn(x)`: si gira a destra di `x` gradi;
- alla fine, dovete invocare la funzione `end()`.

Ecco, ad esempio, un programma che disegna un quadrato:

```
#include "libpsgraph.h"

int main() {
start("quadrato.ps");
draw(50);
turn(90);
```

```
draw(50);
turn(90);
draw(50);
turn(90);
draw(50);
end();
return 0;
}
```

Curva di Koch

Realizzate una funzione che, data una lunghezza in millimetri x e un intero i, produce la curva di Koch di ordine i e di lunghezza x. Essa è definita come segue:

- se $i = 0$, è un segmento di lunghezza x;
- se $i > 0$, è ottenuta giustapponendo quattro curve di Koch di ordine $i - 1$ e di lunghezza $x/3$, come in Figura **??**.

Dopo averla realizzata, verificate che funzioni (scrivendo un main che la invoca e guardando il file risultante).

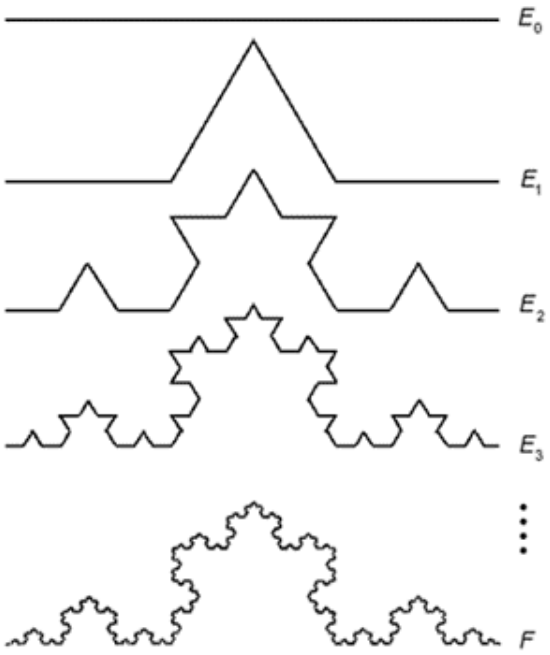

Figura 1: Costruzione di una curva di Koch (E_i è una curva di Koch di ordine i).

Fiocco di neve di Koch

Realizzate ora una funzione che, data una lunghezza in millimetri x e un intero i, produce il fiocco di neve di Koch di ordine i e di lunghezza x: esso si ottiene come un triangolo equilatero di lunghezza x i cui lati siano stati sostituiti con curve di Koch di ordine i e lunghezza x (vedi Figura **??**).

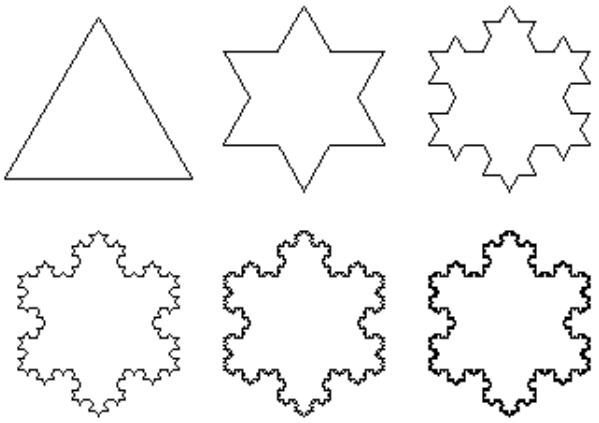

Figura 2: Fiocco di neve di Koch.

3.2 Torri di Hanoi

Il gioco delle torri di Hanoi (anche detto delle torri di Brahma) è stato inventato nel 1883 dal matematico francese Edouard Lucas. L'obiettivo è spostare da un paletto ad un altro un certo numero di dischi forati di dimensione crescente infilati sul paletto e appoggiati l'uno sull'altro. Le regole del gioco sono che si può spostare solo il disco che è in cima ad una pila e non si deve mai appoggiare un disco di dimensione più grande sopra uno più piccolo; per aiutarsi, è possibile usare un terzo paletto come appoggio ausiliario. Secondo una leggenda (forse inventata dal matematico stesso) alcuni monaci di un tempio Indù sono costantemente impegnati a spostare sessantaquattro dischi secondo le regole del gioco; la leggenda dice che quando i monaci completeranno il lavoro, il mondo finirà!

Obiettivo dell'esercizio è scrivere un programma in grado di giocare al gioco delle torri di Hanoi, ossia di specificare la sequenza di mosse da effettuare per risolvere il rompicapo data l'altezza $n > 0$ della pila. Potete assumere che i tre paletti siano numerati da 0 a 2 e che gli n dischi siano inizialmente impilati dal più piccolo (in cima alla pila) al più grande (sotto tutta la pila) sul paletto 0 e vadano spostati al paletto 2. Per semplicità, le mosse sono date semplicemente dall'indicazione del paletto da e verso cui si deve muovere il disco.

Ad esempio, una soluzione per una pila di altezza 3 è data dalla seguente sequenza di mosse:

```
0 -> 2
0 -> 1
2 -> 1
0 -> 2
1 -> 0
1 -> 2
0 -> 2
```

Questo vuol dire che il disco più piccolo va spostato dal paletto 0 al paletto 2, quindi il disco mediano, ora in cima al paletto 0, va spostato al paletto 1; a questo punto il disco più piccolo (rimasto sul paletto 2) va rimesso sopra il mediano (ora sul paletto 1) e, finalmente, il disco più grande va spostato dal paletto 0 al paletto 2, nella sua posizione finale. Le restanti tre mosse, spostando i due dischi rimanenti dal paletto 1 al paletto 2.

Scrivete una funzione `hanoi( int n, int from, int temp, int to );` che stampi le mosse per spostare `n` dischi dal paletto `from` al paletto `to` aiutandosi, se necessario, con il paletto ausiliario `temp`. Osservate che tale funzione, posto che siano state stampate le mosse per spostare $n-1$ dischi da `from` a `temp` (usando eventualmente `to` come paletto ausiliario) può stampare la mossa `from ->to` e quindi stampare le rimanenti mosse necessarie a spostare gli $n-1$ dischi da `temp` a `to` (usando eventualmente `from` come paletto ausiliario). Questa osservazione dovrebbe suggerirvi immediatamente una implementazione ricorsiva della funzione `hanoi`.

La fine del mondo. Modificate la funzione precedente perché restituisca soltanto il numero di mosse effettuate (invece che stamparle). Come cresce tale numero al crescere del numero di dischi? Per rendervi meglio conto del tasso di crescita, provate a considerare il logaritmo del numero di mosse, come cresce?

Vi sembra realistico che per spostare 64 dischi ci voglia un tempo pari alla durata del mondo?

Le torri in dettaglio Supponendo ora di chiamare i dischi dal più grande al più piccolo con le lettere `A`, `B`, `C`, ..., scrivete ora una versione più dettagliata della funzione appena sviluppata che stampi ad ogni passo del gioco il contenuto di ogni paletto (usando una linea per ogni passo e separando il contenuto dei tre paletti con una virgola). Ad esempio, se la pila iniziale ha altezza 3, ed è quindi data da `ABC`, la nuova funzione deve scrivere le seguenti mosse

```
ABC, ,
AB, , C
A, B, C
A, BC,
, BC, A
C, B, A
C, , AB
, , ABC
```