Laboratorio di Fondamenti di Informatica 12/11/2015

Anno accademico 2015/2016

1. Scendesale, salescende, …

Si implementi una semplice funzione

```
void effe0( int n )
```

che visualizzi l'n-esima stringa della serie $F_0(n)$ definita dai seguenti esempi:

$F_0(0)$ → "."
$F_0(1)$ → "1.1"
$F_0(2)$ → "221.122"
$F_0(3)$ → "333221.122333"

Come estensione, si considerino poi le 5 varianti (effe1(int n), effe2(int n), effe3(int n), effe4(int n), effe5(int n)) in cui (ad esempio per n=3) si ha:

$F_1(3)$ = 111223.322111
$F_2(3)$ = 33322**1**22333
$F_3(3)$ = 11122**3**22111
$F_4(3)$ = 122**333**221
$F_5(3)$ = 322**111**223

2. Scambio lettere

Si codifichi una funzione **void scambioLettere(char *dest, char *lettere, int p_o_d)** che modifica la stringa destinazione (**dest**), sostituendo ordinatamente i caratteri che si trovano in posizione pari o dispari (a seconda del valore di **p_o_d**, 0 o 1 rispettivamente), con quelli contenuti nella stringa **lettere**, fino "ad esaurimento" di una delle due stringhe (possono avanzare caratteri in lettere o meno, a seconda della lunghezza delle due stringhe).

Esempi:

Se dest: “parolagigante” lettere: “PROVA”, p_o_d = 0
dest diventa: “p**P**r**R**l**O**g**V**g**A**nte”

Se dest: “parolagigantelunghissimapiugggrossa” lettere: “ABCDEF” p_o_d = 1
dest diventa: “**A**a**B**o**C**a**D**i**E**a**F**telunghissimapiugggrossa”

Se dest: “parolina” lettere: “TANTISSIMELETTEREUSATESOLOINPARTE” p_o_d = 0
dest diventa: “p**T**r**A**l**N**n**T**”

Se dest: “GHIAIE” lettere: “CIVILE” p_o_d = 1
dest diventa: “**C**H**I**A**V**E”

3. Le frazioni (1/3)

Si implementino alcune funzioni per un tipo di dato astratto che modella le *frazioni razionali* ($\pm n/d$, $n \in \mathbb{N}$, $d \in \mathbb{N}\backslash\{0\}$).

```
typedef struct( int segno, num, den; ) Frazione;
```

Il segno positivo è rappresentato dal valore +1, quello negativo dal valore -1. Una Frazione si dice *minima* se è ridotta ai minimi termini.

Una frazione rappresenta sempre un numero razionale, che si può *approssimare* con un numero decimale (float), e un numero decimale può sempre essere espresso in forma frazionaria. Due frazioni si dicono equivalenti se rappresentano lo stesso numero (ed hanno quindi identica forma minima).

Somma, prodotto, differenza, … sono da intendersi come normalmente definite in matematica.

Può essere comodo (ed elegante) avvalersi di funzioni ausiliarie (ad esempio, per stabilire se due numeri interi sono primi fra loro... e di sicuro non l'avete buttata!).

3. Le frazioni (2/3)

`Frazione costruisci( int s, int n, int d );`

restituisce una Frazione di segno s, numeratore n, denominatore d

`Frazione riduci( Frazione fr );`

restituisce una Frazione **equivalente a** fr, ma ridotta ai minimi termini

`void riduci_ind( Frazione * fr );`

modifica la frazione di cui riceve l'indirizzo, **riducendola** ai minimi termini

`float valuta( Frazione fr );`

restituisce il valore (inevitabilmente approssimato!) di fr, espresso come float

`void stampa( Frazione  fr );`

stampa fr in formato ± n / d (semplicemente ±n se d è 1) e omettendo il +

`int confronta( Frazione  fr1, Frazione fr2 );`

restituisce **1** se il valore di fr1 è maggiore del valore di fr2, **0** se sono equivalenti, **-1** altrimenti

`Frazione frangi( float val );` ***`/* Questa è difficile!?! */`***

restituisce la minima Frazione equivalente al numero razionale val

3. Le frazioni (3/3)

`Frazione inverso( Frazione fr );`

restituisce l'inverso di fr (a/b → b/a)

`Frazione opposto( Frazione fr );`

restituisce l'opposto di fr (a/b → -a/b)

`Frazione somma( Frazione fr1, Frazione fr2 );`

restituisce la somma di due frazioni

`Frazione prodotto( Frazione fr1, Frazione fr2 );`

restituisce il prodotto di due frazioni

`Frazione molt_scalare( int k, Frazione fr );`

restituisce il prodotto della frazione fr per lo scalare k

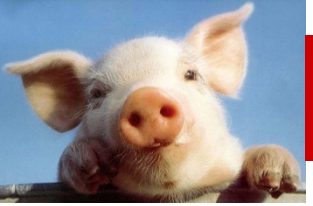
`Frazione differenza( Frazione fr1, Frazione fr2 );`

restituisce la differenza di due frazioni

`Frazione divisione( Frazione fr1, Frazione fr2 );`

restituisce il risultato della divisione di due frazioni

… e chi più ne sa più ne metta!!!



4. Siam tre piccoli typedef struct

La FPF Inc. (Fictional Pigs Farm) si dedica da sempre all'allevamento di maialini da palcoscenico più o meno famosi*, e conserva i dati dei suoi piccoli divi in un vettore (di cui solo una parte è utilizzata, pari a num_maialini: gli elementi di indice da num_maialini a N-1 non sono usati):

```
typedef struct { char nome[20]; Data datanascita; float peso; int popolarita } Maialino;
typedef struct { int num_maialini; Maialino pigs[N]; } Allevamento;
```

I maialini non sono elencati nell'Allevamento in un alcun ordine particolare, ma la ditta necessita di scandirli in ordine di ognuna delle loro caratteristiche (cioè in ordine di nome, di data di nascita, di popolarità, e di peso corporeo), a seconda delle diverse necessità applicative (rispettivamente: appello nominale, trattamento pensionistico, merchandising, utilizzo culinario qualora la loro popolarità cali bruscamente). A tal fine, si utilizzano quattro distinti **vettori di puntatori**, in cui ogni puntatore punta a un maialino specifico. In questo modo si rappresentano quattro diversi ordinamenti indipendenti degli elementi di uno stesso insieme (di maiali) senza dover replicare tutti i dati ad essi relativi, ma replicando solo i puntatori:

```
Maialino * ord_alfabetico[N], * ord_data[N], * ord_pop[N], * ord_peso[N];
```

Si codifichi un programma che, anche riutilizzando le definizioni e le funzioni dell'esercizio sulle date, costruisca correttamente i quattro vettori di puntatori. Si usino eventualmente, per esercizio, algoritmi di ordinamento diversi per ciascuno dei vettori. La slide successiva mostra l'inizializzazione di un allevamento.

Il programma deve dapprima stampare tutti i dati dei maialini, nell'ordine in cui si trovano nell'allevamento, poi ordinare i puntatori nei vari vettori, e visualizzare i maialini secondo i diversi ordinamenti.

[*] Un elenco parziale è ad esempio en.wikipedia.org/wiki/List_of_fictional_pigs

4. Siam tre piccoli typedef struct

```
#include <stdio.h>
#define N 100 // fino a 100 maialini in un allevamento

typedef struct { int giorno, mese, anno; } Data;
typedef struct { char nome[20]; Data datanascita;
                 float peso; int popolarita; } Maialino;
typedef struct { int num_maialini; Maialino pigs[N];} Allevamento;
int confronta( Data, Data );
int main () {

 Allevamento a = { 7,  "Porky Pig", {12,7,1936}, 33.50, 85,
                       "Miss Piggy", {17,12,1974}, 23.95, 170,
                       "Babe", {23,1,1996}, 18.80, 250,
                       "Pumbaa", {14,11,1994}, 79.99, 1690,
                       "Peppa", {31,5,2004}, 12.15, 8500,
                       "Hamm", {12,7,1968}, 19.05, 290,
                       "Piglet", {22,4,1926}, 9.30, 1260  };

 Maialino  * ord_alfabetico[N] = { NULL },   // inizialmente, per "sicurezza", tutti a NULL
           * ord_data[N] = { NULL },
           * ord_peso[N] = { NULL },
           * ord_pop[N] = { NULL };

  /* codice! */

  return 0;
}
```

Giochini

Robot programmabile (1/4)

Quesito:

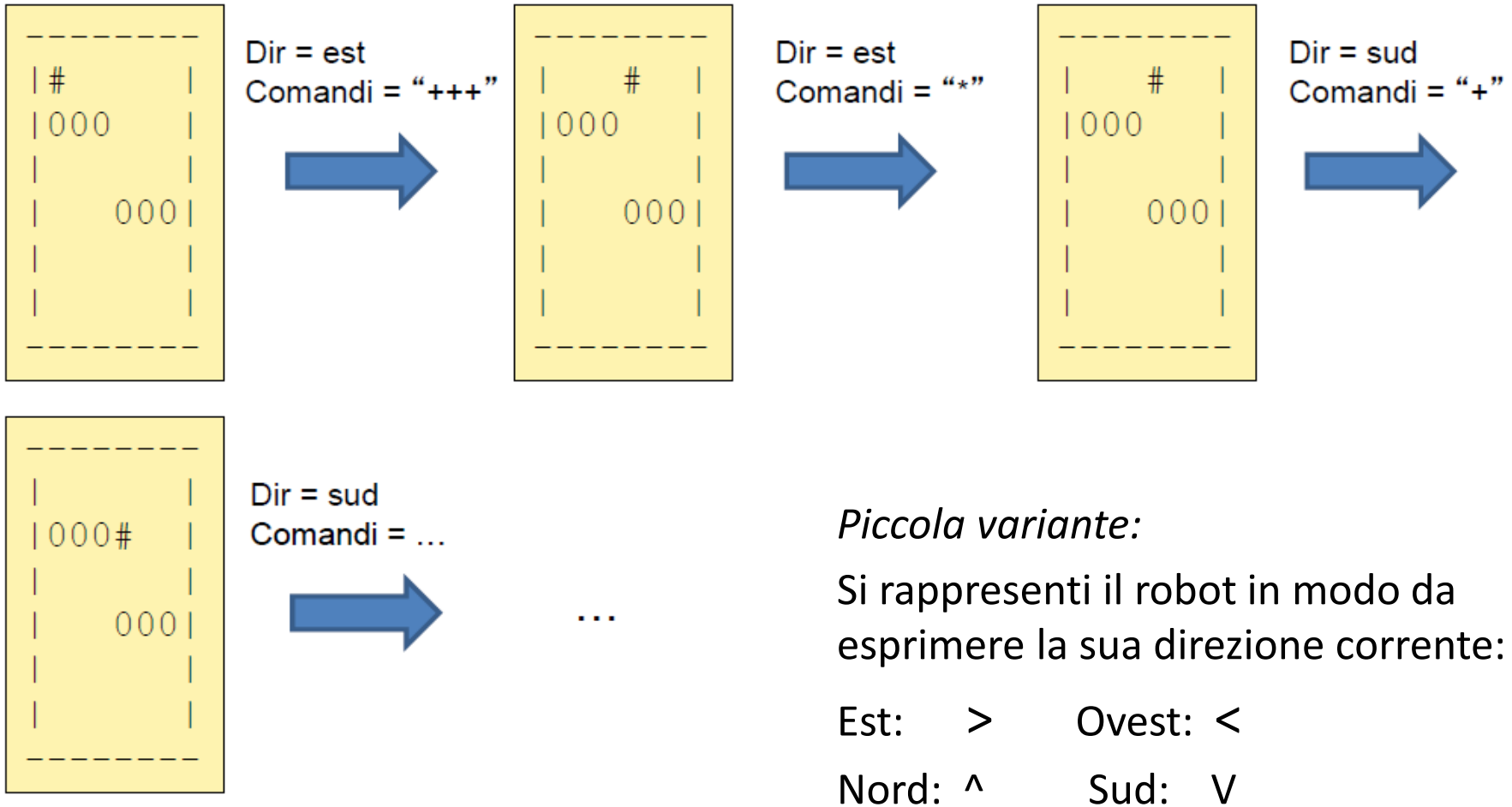
Sia dato un robot capace di compiere le seguenti azioni elementari, corrispondenti ad un comando (un **char** – tra parentesi):

- Ruotare di 90° a sinistra (‘/’)
- Ruotare di 90° a destra (‘*’)
- Avanzare di un passo (‘+’)
- Indietreggiare di un passo (‘-’)

Il robot (‘#’) si trova in un mondo quadrato 6x6, come quello rappresentato in figura nella prossima slide. Nel mondo possono trovarsi anche ostacoli insormontabili (‘0’), attraverso i quali il robot non può passare.

La posizione iniziale del robot corrisponde alla cella (0, 0) della matrice che rappresenta il mondo. Il robot è inizialmente rivolto ad est (un passo in avanti lo sposterebbe nella casella di riga 0, colonna 1).

Robot programmabile (2/4)



Piccola variante:

Si rappresenti il robot in modo da esprimere la sua direzione corrente:

Est: > Ovest: <

Nord: ^ Sud: V

Robot programmabile (3/4)

Scrivere un programma che:

- Implementi le azioni elementari del robot come funzioni
- Includa una funzione in grado di interpretare una stringa di comandi e di far eseguire al robot le azioni corrispondenti, mostrandone la posizione sulla mappa azione dopo azione

Successivamente:

- Interpretare la stringa "+++*++*+/+*++/++-*++", determinando la posizione finale del robot sulla mappa mostrata alla slide precedente
- Individuare una delle possibili stringhe di comandi che portino il robot nell'angolo in basso a destra della mappa, verificandola con l'aiuto del programma

Robot programmabile (4/4)

Suggerimenti:

- Definire un'enumerazione per la direzione del robot
- Dichiarare delle variabili globali per la direzione, riga e colonna correnti del robot e per la mappa del mondo
- Le funzioni "ruota a destra" e "ruota a sinistra" influenzano solo l'effetto dei successivi spostamenti (cambiano la direzione, ma non la posizione)
- Le funzioni "passo avanti" e "passo indietro" cambiano la posizione del robot a seconda della sua direzione attuale. Devono inoltre tenere presenti i confini della mappa e gli eventuali ostacoli (che impongono al robot di fermarsi)
- Scandire la stringa di comandi carattere per carattere, invocando l'azione corrispondente e ridisegnando la mappa sullo schermo ad ogni azione