Alcuni errori tipici

Assegnamento 1

rand() ed srand()....

- Per generare davvero una sequenza casuale
  - La **`srand()`** va chiamata una sola volta **in tutto il programma** per impostare il primo valore della sequenza (quindi **non** nelle funzioni che possono essere chiamate più volte, magari nel main)
  - Tutti i valori successivi vanno generati chiamando la **`rand()`**
  - Quindi il modo corretto di procedere per generare una sequenza casuale lunga N e'

```
srand(42); //ben posizionato, ad es in main
for (i= 0; i < N; i ++)
{... x = rand(); .... }
```

Attenzione agli indici!!!

- Alcuni hanno acceduto a caselle inesistenti
  - Quindi sono usciti dai bound della matrice
  - Il comportamento è **imprevedibile.** potremmo ottenere:
    - valori casuali (quello che c'è in memoria)
    - **segmentation fault** (se accediamo a celle vietate)
  - Se gli array sono allocati sullo heap potete usare valgrind per verificare i bound (valgrind ./eseguibile)
  - Attenzione su cose come **a[i-k][j+h]**

```
/* gli indici corretti qui sono da 0 a N-1,
scrivere a[N] e' un errore */
int * a = malloc(N*sizeof(int));
```

Attenzione agli indici!!!

- Il controllo di validita degli indici della matrice:

```
/* sbagliato */
if( i<=0 || j<= 0 || i>L || j>L )
printf("Errore!");

/* giusto */
if( i<0 || j<0 || i>=L || j>=L )
printf("Errore!");
```

Commenti

- Non devono ripetere passo passo quello che è evidente dal codice!
- è preferibile:
  - scrivere un commento esauriente all'inizio o alla fine del file, magari spiegando a parole l'algoritmo utilizzato
  - commentare sinteticamente all'inizio di un blocco di 5-15 istruzioni per spiegare cosa sta per succedere
  - commentare bene le variabili globali (inclusa la politica di modifica)
  - commentare brevemente ogni variabile locale **dal significato non ovvio** (non serve 'i è un indice')

Costanti esplicite nel codice

- *è **PESSIMA*** norma usare costanti numeriche o caratteri cablate nel codice

  Es:

```
char a[37], b[35], c[33];

for (i=0 ; i< 32; i++)
      a[i] = ' ';
```

  Questo rende difficile leggere e manutenere il codice (i valori sono legati fra di loro o scollegati ? Quel'è il loro uso ?)

Costanti esplicite nel codice

- *è **PESSIMA*** norma usare costanti numeriche o caratteri cablate nel codice

  Es:

```
#define N 37
char a[N], b[N-2], c[N-4];
for (i=0 ; i< N-5; i++)
      a[i] = ' ';
```

  Bisogna fattorizzare le costanti intere e stringa con opportune **`#define`**, dando loro nomi significativi ed usare tali nomi consistentemente nel codice per esplicitare i legami

  – Ad esempio ORIZ e VERT per le direzioni orizzontale e verticale

Costanti esplicite nel codice

- *è una **PESSIMA IDEA** usare direttamente gli interi dei codici ASCII invece delle costanti char*

  Es:

```
for (i=0 ; i< N-5; i++)
      a[i] = 32;
```

  – Bisogna usare invece la versione con la costante, più leggibile agli umani

```
for (i=0 ; i< N-5; i++)
      a[i] = ' ';
```

Costanti esplicite nel codice

- *è **TERRIBILE** norma* NON usare le costanti globali definite, o usarle a tratti

Es:

```
#define EMPTY ' '
#define ORIZ 7

if(d == ORIZ)...    // ok.
if(d == 7)...       // e se domani decido
                    // di
  cambiare ORIZ ??
```

Non includere il .h

- *Le dipendenze che usiamo vanno incluse!*
  - Se non includiamo il .h a inizio file (e.g. **#include “bubble.h”**) non possiamo usarne i contenuti
  - se siamo fortunati il compilatore ci darà un errore

    error: ‘board’ undeclared (first use in this function)
  - se siamo sfortunati, il codice potrebbe compilare ma stiamo usando un’altra variabile omonima

Organizzazione del codice

- Alcuni hanno scritto un codice lungo esplicitando diversi casi
  - Ad esempio, i controlli verso su, giu, destra, e sinistra in pop_bubble
  - E' utile (e a volta fondamentale) **FATTORIZZARE** il codice replicato raggruppando i casi uguali e NON copiare il codice replicandolo in tutti i punti che servono (ad esempio, raggruppandolo in un altra funzione)

Organizzazione del codice

- Variabili GLOBALI: non c'era nessuna necessità di usare delle variabili globali nell'assegnamento
  - E' importante abituarsi ad usare le globali solo quando non abbiamo altra scelta, altrimenti il codice scritto sara' caotico e confuso!
  - Prima di tutto cerchiamo di risolvere usando le locali, e solo come ultima ratio le static e globali

Miscellanea

- Il codice va INDENTATO correttamente se no si legge malissimo e non va farcito di commenti lunghissimi
- La convenzione C e' che le macro usino nomi maiuscoli e le variabili minucoli.
  - Seguiamola – questo rende il codice più leggibile a voi ed agli altri