# Esame di Laboratorio di Fisica Computazionale

**12 luglio 2013, ore 13.30**

Il voto finale deriva dalla somma dei voti in shell scripting, `Mathematica` e `C++`.

Raggiunta la soglia di sufficienza in ciascuna delle tre parti il voto è pari a 18. La sufficienza si ottiene: nello shell scripting rispondendo alla prima domanda; in `Mathematica` ottenendo almeno 5 punti, secondo quanto indicato nel testo dei due esercizi; in `C++` risolvendo correttamente almeno i primi due punti ed impostando correttamente il terzo.

I punti rimanenti vengono assegnati attribuendo 1 punto allo shell shell scripting, 6 a `Mathematica` e 6 al `C++`. Per ottenere il punto di shell scripting bisogna risolvere la seconda domanda. Per ottenere 6 punti in `Mathematica` bisogna raggiungere un totale di 14 punti, secondo quanto indicato negli esercizi. Per ottenere 6 punti in `C++` bisogna rispondere correttamente alle prime 8 domande.

## shell scripting

1. A partire dal file `pth-h104-top.dat` se ne scriva uno nuovo in cui è stato cambiato il segno dei valori della seconda colonna.

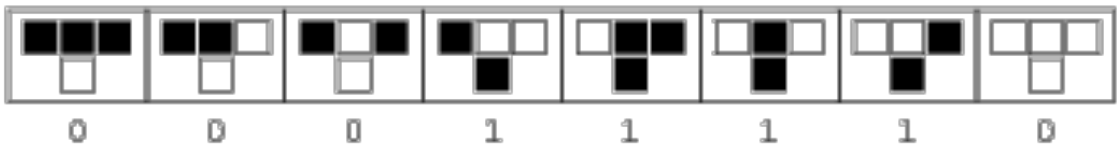
2. A partire dal file `pth-h104-top.dat` se ne scriva uno nuovo in cui è stato cambiato il segno dei valori della seconda colonna minori di 0.001.

# Mathematica

## Automa cellulare

- Si scriva una funzione che inizializza una lista contenente $2n + 1$ elementi, tutti zero con l'eccezione dell'elemento $n + 1$, che deve essere posto pari a 1.
  *(Si consideri l'utilizzo combinato di* `ReplacePart` *e di* `Table`*).*
  (1.5 punto)

- Si scriva una lista di regole di sostituzione che realizzi le trasformazioni rappresentate graficamente in figura.
  *(Si consideri per la singola sostituzione una regola del tipo* $\{a, b, c\} \to d$*)*
  (0.5 punto)




- La trasformazione di una lista esistente in una nuova, secondo il meccanismo dell'automa cellulare, richiede di applicare a ciascuna terna di elementi consecutivi della lista il set di regole di sostituzione del punto precedente. Per semplicità, il primo e l'ultimo elemento della lista possono essere ricopiati da una riga all'altra, mentre si devono calcolare tutti gli elementi nuovi dal secondo al penultimo.
  *Si utilizzino i comandi* `Table` *per generare i nuovi elementi, dal secondo al penultimo, e poi* `Prepend` *e* `Append` *per ricopiare il primo e l'ultimo.*
  (3 punti)

- Si generi una lista di 20 righe a partire da quella iniziale. Si visualizzi la matrice risultante con i comandi `ArrayPlot` e `Show`.
  (2 punti)

**Life**

- Si consideri il seguente sistema di equazioni differenziali

$$\begin{cases} \frac{dx(t)}{dt} &= [a - b\,y(t)]\,x(t) \\ \frac{dy(t)}{dt} &= [c\,x(t) - d]\,y(t) \end{cases} \tag{1}$$

  Si determini per quali valori e/o combinazioni dei parametri $a, b, c, d$ le due soluzioni sono costanti: $x(t) = x_0$ e $y(t) = y_0$. *(Si esprimano $(x_0, y_0)$ in funzione di $a, b, c, d$).*
  (2 punti)

- Si risolvano numericamente le equazioni 1, con condizioni al contorno $x(0) = 0.5, y(0) = 0.5$ e con $a = b = c = d = 1$, nell'intervallo $t \in [0, 20]$.
  (1 punto)

- Si disegnino $x(t)$ e $y(t)$ in funzione del tempo. Si disegnino nel piano $(x, y)$ le soluzioni al variare del tempo.
  (1 punto)

- Partendo dalla condizione iniziale $x(0) = 0.5, y(0) = 0.5$ e considerando degli intervalli temporali pari a $\Delta t = 0.01$, si consideri l'evoluzione del sistema definito dall'equazione 1 con $a = b = c = d = 1$; si calcolino i primi 2000 passi dell'evoluzione temporale discretizzata, usando per l'aggiornamento delle due funzioni la legge

$$\begin{cases} x(t+1) &= x(t) + [a - b\,y(t)]\,x(t)\Delta t \\ y(t+1) &= y(t) + [c\,x(t) - d]\,y(t)\Delta t \end{cases} \tag{2}$$

  Si disegni nel piano $(x, y)$ l'evoluzione del sistema, utilizzando `ListPlot` con l'opzione `PlotJoined` $\rightarrow$ `True`.
  (3 punti)

- Si disegnino separatamente $x(t)$ e $y(t)$ in funzione del tempo, sempre utilizzando `ListPlot` con l'opzione `PlotJoined` $\rightarrow$ `True`.
  (2 punti)

Si risolva l'esercizio proposto. Se è possibile, includere tutto in un unico file sorgente. Si consiglia di concentrarsi sulla pulizia del codice: pochi punti risolti in modo ordinato saranno valutati meglio di tanti punti trattati disordinatamente. La sufficienza è raggiunta risolvendo correttamente i primi due punti ed impostando correttamente il terzo.

# Esercizio

Si vuole scrivere una classe `Parabola` che descriva polinomi di grado due a coefficienti reali.

1. Si pongano come membri privati tre variabili corrispondenti ai coefficienti. Tra i membri pubblici, si scriva un opportuno costruttore che richieda come parametri i coefficienti (in modo che il primo sia quello di $x^2$, il secondo quello di $x$ e il terzo il termine noto).

2. Si implementino il costruttore di copie e l'operatore di assegnazione.

3. Tra i membri pubblici, si scriva l'operatore "+", che restituisce il polinomio somma di due polinomi, e l'operatore "$*$", che restituisce il polinomio prodotto tra un polinomio e un numero reale. Quest'ultimo funziona solo quando il prodotto è scritto in un dato ordine: se `p` è una `Parabola` e `d` è un `double`, funzionerà la sintassi `p*d`. Implementare una funzione ausiliaria (che richiami quella già scritta) in modo che funzioni anche la sintassi `d*p`.

4. Si vuole ora implementare la soluzione delle equazioni di secondo grado. Si scriva una classe `Result` che abbia tra i membri privati una variabile booleana `has_solution` che specifichi se l'equazione ha almeno una soluzione nel campo complesso, una variabile booleana `is_indeterminate` che specifichi se l'equazione ha infinite soluzioni, una variabile reale contenente il valore. Si scriva un opportuno costruttore che richieda tre parametri, con valori di default assegnati in modo che `has_solution` sia vero e `is_indeterminate` sia falso se non sono specificati.

5. Si implementi, all'interno della classe `Result`, una funzione void `print` che chieda in ingresso una referenza ad un oggetto di tipo `ostream` (nella libreria `iostream`), e stampi `'Nessuna soluzione'`, `'Infinite soluzioni'` oppure il valore, a seconda dei flag. Si implementi poi l'overloading dell'operatore `<<` per gli `ostream`, che dovrà servirsi della funzione appena scritta.

6. Si scriva, tra i membri pubblici della classe `Parabola`, una funzione `largest_root` che restituisca un `Result` contenente la parte reale della radice con parte reale più grande. (Tale funzione dovrà essere definita `virtual` per risolvere l'ultimo punto.) Si faccia in

modo che i flag vengano impostati opportunamente nel caso in cui l'equazione abbia infinite soluzioni (quando tutti i coefficienti sono nulli) oppure non ne abbia nessuna (quando solo il termine noto è non nullo).

7. Si verifichi il comportamento del codice con il seguente `main`:

```
#include <iostream>
using namespace std;

int main() {
        cout << Parabola(1,1,1).largest_root() << endl;
        cout << Parabola(0,1,1).largest_root() << endl;
        cout << Parabola(0,0,1).largest_root() << endl;
        cout << Parabola(0,0,0).largest_root() << endl;
}
```

8. Si scriva una nuova classe `Parabola_R`, che erediti pubblicamente da `Parabola`, e che, attraverso l'overriding della funzione `largest_root`, implementi la seguente funzionalità: il risultato dovrà avere il suo flag `has_solution` falso quando l'equazione non ha soluzioni reali.

9. Per verificare il comportamento polimorfico, si scriva un `main` in cui si istanzia un `vector` di puntatori a oggetti di tipo `Parabola` e lo si riempie con (puntatori a) una `Parabola` e una `Parabola_R`, entrambe inizializzate con il polinomio $x^2 + 1$. Si stampi a schermo per ogni elemento del vettore il risultato della funzione `largest_root`.