Esame di Metodi Computazionali della Fisica, I modulo
25 gennaio 2010, ore 15.00

C++

Nell'esercizio di C^{++} è prevista l'implementazione di una classe **Reticolo** per simulazioni di particelle cariche su un reticolo.

In ogni punto del reticolo (quadrato a dimensione fissata) può essere presente (o non presente) una particella carica. Ad ogni istante temporale, il sistema può evolvere creando (o distruggendo) coppie di particelle vicine con carica opposta oppure semplicemente spostando particelle da un sito ad un sito vicino libero.

Nella prima parte dell'esame sarà dunque necessario implementare una classe **Particella** che rappresenti una particella carica. Nella seconda parte verrà chiesto di implementare una classe **Reticolo** che dovrà contenere le particelle. La base della classe reticolo sarà una matrice (di dimensione fissata) di puntatori a particelle. Perché puntatori? Perché se nel sito (i, j) non è presente alcuna particella, l'elemento della matrice (i, j) conterrà un puntatore **NULLO** (ovvero $= 0$). Al contrario, se sarà presente una particella, l'elemento della matrice conterrà un indirizzo di memoria valido.

Ogni parte è suddivisa in punti i quali possono essere obbligatori o facoltativi. I punti obbligatori debbono essere svolti in sequenza per il corretto funzionamento della libreria mentre i punti facoltativi (se non esplicitamente indicato) possono essere svolti in qualsiasi ordine. Il codice deve essere **correttamente compilabile** e accompagnato dal rispettivo **Makefile** funzionante (se il Makefile è assente saranno decurtati dei punti). Non saranno accettati codici non compilabili (abbiate cura di commentare la parte di codice che non riuscite sviluppare. In caso di indecisione sul voto tale parte sarà comunque parzialmente valutata). Il codice deve rispondere correttamente alle richieste previste nel main. In sede di valutazione il codice sarà comunque testato per verificarne l'affidabilità.

I punti obbligatori sono:

I Parte (a.1), (a.2), (a.3), (a.4)

II Parte (b.1), (b.2), (b.3), (b.4), (b.5)

II Parte (c.1), (c.2)

1 I Parte: classe Particella

1.1 Definizione della classe Particella

(a) Costruttori/Distruttori:

(1 - Obb) Implementare il costruttore per la classe **Particella** che accetti in ingresso un intero (**int carica**) che rappresenta la carica della particella.

(2 - Obb) Implementare il costruttore di copia per la classe **Particella**.

(3 - Obb) Implementare il distruttore di default.

(4 - Obb) La variabile **carica** non può essere modificata direttamente dall'esterno. Implementare una funzione che restituisca all'esterno la carica della particella.

2 II Parte: classe Reticolo

(b) Classe Reticolo: La classe **Reticolo** si basa su una matrice di dimensione $SIZE * SIZE$ (usare $\#define\ SIZE$ 100 ad esempio) di puntatori a **Particella**.

(1 - Obb) Nella sezione privata dichiarare una matrice di dimensione $SIZE * SIZE$ di puntatori a **Particella** ed un intero che rappresenterà il numero di particelle contenute nel reticolo.

(2 - Obb) La variabile **n_particelle** non può essere modificata direttamente dall'esterno. Implementare una funzione che restituisca all'esterno il numero di particelle.

(3 - Obb) Nella sezione privata implementare una funzione **Check** che accetta in ingresso due interi x ed y, rispettivamente la righe e la colonna del reticolo, e restituisce $false$ se il relativo puntatore è $= 0$ (nessuna particella) e $true$ altrimenti (particella presente).

(4 - Obb) Nella sezione privata implementare una funzione **Add** che accetta in ingresso tre interi x, y ed e, rispettivamente le posizioni nel reticolo e la carica della particella e aggiunge una particella in tale posizione se il sito è vuoto. Se la particella è stata aggiunta correttamente restituire $true$, altrimenti $false$. Usare l'operatore **new** che crea l'oggetto e restituisce l'indirizzo ad una posizione di memoria valida. Ricordarsi di incrementare il numero di particelle.

(5 - Obb) Nella sezione privata implementare una funzione **Remove** che accetta in ingresso due interi x ed y, rispettivamente la riga e la colonna del reticolo, e rimuove la particella (se presente) in tale posizione. Se la particella è stata rimossa correttamente restituire $true$, altrimenti $false$. Usare l'operatore **delete** che distrugge l'oggetto. Ricordarsi di ridurre il numero di particelle e di settare l'elemento di matrice a zero.

(6) Nella sezione privata implementare una funzione **MOD** che accetta in ingresso un intero i e restituisce i modulo $SIZE$ (in modo tale da rendere ciclico il reticolo). In particolare se $i = -3$ allora la funzione MOD deve restituire $SIZE-3$, e se $i = SIZE+2$ allora la funzione MOD deve restituire 2. Modificare i punti $2-4$ in modo da usare questa funzione.

TIP: Per creare una matrice di **type**, usare la sintassi
type $var[SIZE][SIZE]$;

TIP: Nel caso della funzione **MOD** assumere che l'intero in ingresso sia compreso tra $-SIZE \leq i \leq 2 * SIZE - 1$.

(c) Costruttore/Distruttore:

(1 - Obb) Implementare il costruttore che accetta in ingresso un intero che rappresenta il numero di particelle iniziali e settare ogni elemento del reticolo a zero. Usare la funzione **Add** per aggiungere in posizioni casuali (e con carica casuale) il numero di particelle richiesto.

(2 - Obb) Implementare il distruttore che distrugge tutte le particelle. Usare la funzione **Remove** implementata precedentemente.

TIP: Per estrarre un numero casuale intero tra $[0,\ N)$, includere la libreria **cstdlib** ed usare $int\ rand\ =\ lrand48()\ \%\ N$.

TIP: Se la funzione **Remove** è stata implementata correttamente è sufficiente scorrere tutti gli elementi del reticolo ..

(d) Funzioni Varie:

(1) Implementare l'operatore **operator()** che accetta in ingresso due interi e restituisce il puntatore del reticolo nella posizione indicata.

(2) Implementare una funzione **Evolve** che sceglie una posizione a caso nel reticolo e una posizione prima vicina e fa evolvere il sistema nel seguente modo:

a. Se entrambe le posizioni non contengono una particella, creare una coppia di particelle con carica opposta.
b. Se le posizioni contengono due particelle con carica opposta, eliminare le particelle.
c. Se solo una posizione è occupata, spostare la particella da una posizione all'altra.

TIP: Per la funzione del punto (2) usare la funzione **MOD**.

3 III Parte: Main

(e) Main:

(1) Creare un file main che crei un **Reticolo** di 10 particelle.

(2) Simulare l'evoluzione del reticolo (è necessario aver implementato il punto (d.2)).

Mathematica

- Si definisca una funzione `sol(h,v)` che contiene la soluzione del seguente sistema di equazioni differenziali, per dati h e v:

$$\begin{aligned} x'(t) &= (2h+1)x(t) - y(t) \\ y'(t) &= (1+2h+h^2+v)x(t) - y(t) \\ x(0) &= -1, \quad y(0) = -1 \end{aligned} \tag{1}$$

  Si definisca una funzione `gra(h,v)` che disegna nel piano xy, per dati h e v, la coppia di punti $(x(t), y(t))$ soluzione dell'equazione, con $t \in [0, 40]$ (*si ricordi di utilizzare* `Evaluate`).

- Si disegnino le curve definite nel punto precedente con le seguenti scelte di parametri: 1) $h = -0.1,\ v = 1$, 2) $h = 0.1,\ v = 1$, 3) $h = 0.1,\ v = -0.1$.

- Si scriva, senza l'ausilio di Mathematica, la matrice A associata al sistema di eq.(1) (ovvero la matrice che permette di scrivere il sistema nella forma $X'(t) = A.X(t)$ dove $X(t)$ è il vettore colonna $(x(t), y(t))$). Si definisca una funzione `pol(h,v)` che restituisce il polinomio caratteristico associato alla matrice, per dati h e v.
  Si definisca una funzione `eig1(h,v)` che calcola gli zeri del polinomio caratteristico della matrice A, per dati h e v.
  Si definisca una funzione `eig2(h,v)` che calcola gli autovalori della matrice A, per dati h e v.
  Si verifichi che i risultati di `eig1` e di `eig2` coincidono.

- Si disegni la superficie 3D parametrica il cui raggio è dato dalla funzione $r = |J_5(\theta)e^{i\phi}|$, facendo variare $\theta \in [0, \pi]$ e $\phi \in [0, 2\pi]$.
  ($J_5(\theta)$ è la funzione di Bessel di tipo J)