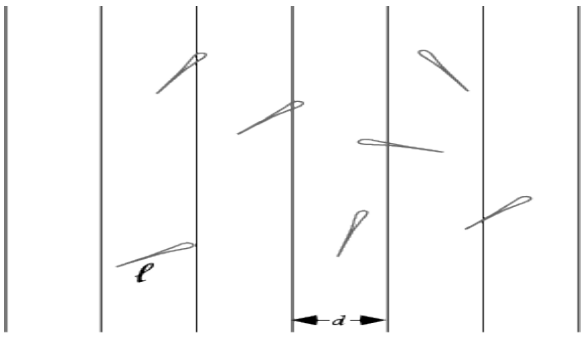
ΠΑΝΕΠΙΣΤΗΜΙΟ ΙΩΑΝΝΙΝΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ C

Προγραμματιστικές Ασκήσεις, Φυλλάδιο 1

Εκφώνηση: 9/3/2012 **Παράδοση: 5/4/2012, 11.59**

Άσκηση 0η : *Το πρόβλημα της βελόνας του Buffon*

Θέμα της εργασίας είναι να αναπαραχθεί με τη χρήση τυχαίων αριθμών ένα στιγμιότυπο του γνωστού προβλήματος της *βελόνας του Buffon* και το οποίο οδηγεί προσεγγιστικά σε εκτίμηση του αριθμού π = 3.14159….
Το πρόβλημα αυτό διατυπώθηκε αρχικά το 1733 από τον Γάλλο φυσιογνώστη και μαθηματικό George-Louis Lecrerc de Buffon (1707-1788) και παρουσιάστηκε με λύση από τον ίδιο το 1777. Το αξιοπρόσεκτο αποτέλεσμα είναι ότι η πιθανότητα αυτή σχετίζεται άμεσα με την τιμή του π. Το πρόβλημα περιλαμβάνει την ρίψη μιας βελόνας μήκους ℓ σε μια επιφάνεια με ισοδιάστατες παράλληλες ευθείες με απόσταση d και τον υπολογισμό της πιθανότητας η βελόνα να τέμνει μια από τις παράλληλες γραμμές.



Αυτό που έχει αποδειχτεί από τον Buffon είναι ότι αν το μήκος της βελόνας δεν είναι μεγαλύτερο από την απόσταση των παράλληλων γραμμών, τότε η πιθανότητα μια βελόνα που αφήνεται τυχαία στο πάτωμα να τέμνει μια από τις ευθείες είναι:

$$P = \frac{2\,\ell}{\pi\, d}$$

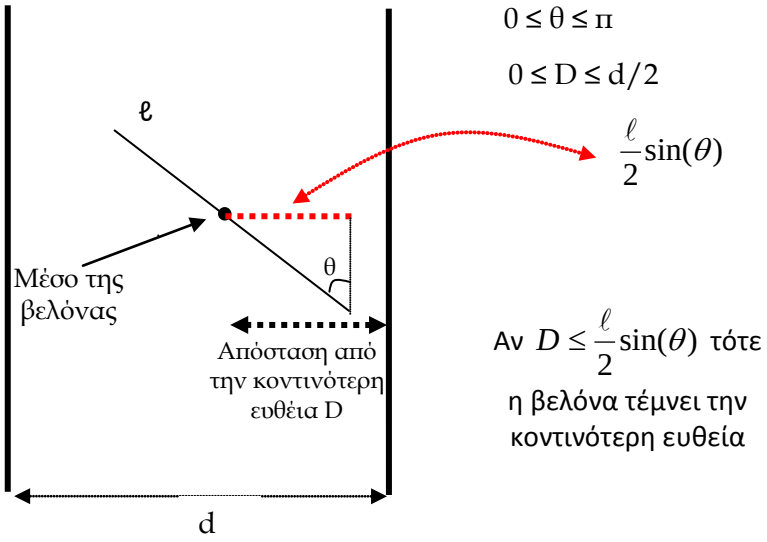
Αυτό σημαίνει, αντίστροφα, ότι αν στην πραγματικότητα ρίξουμε *k* βελόνες μήκους ℓ σε ένα πάτωμα με παράλληλες ευθείες που απέχουν d με $\ell \le d$ και διαπιστώσουμε ότι οι *m* από αυτές τέμνουν κάποια από τις ευθείες τότε θα ισχύει:

$$\left.\begin{array}{l} P \approx \dfrac{\#\,\text{βελόνες τέμνουν τις γραμμές}}{\#\text{βελονες συνολικά}} = \dfrac{m}{k} \\ P = \dfrac{2\,\ell}{\pi\, d} \end{array}\right\} \Rightarrow \frac{m}{k} \approx \frac{2\,\ell}{\pi\, d} \Rightarrow \pi = \frac{2k\,\ell}{m\, d}$$

και συνεπώς ο λόγος $\dfrac{2k\,\ell}{m\,d}$ μπορεί να χρησιμοποιηθεί ως προσέγγιση του π. Ο παρακάτω πίνακας Ο παρακάτω πίνακας δείχνει ενδεικτικά τις προσπάθειες μερικών ερευνητών να υπολογίσουν την τιμή του π (Απόσταση παραλλήλων d=1)

Ερευνητής	Μήκος βελόνας	Ρίψεις	Επιτυχίες	Τμή του π
Wolf (1856)	0,8	5000	2,532	3,15955
De Morgan	1	600	382	3,14136
Smith (1855)	0,6	3204	1218	3,15665
Fox	0,75	1030	489	3,1595
Lazzarini (1901)	0,83	3408	1808	3,12902

Για να προσομοιώσουμε τη ρίψη μιας βελόνας μήκους *ℓ* σε ένα πάτωμα από κάθετες ευθείες που απέχουν *d* θα ακολουθήσουμε την εξής στρατηγική:
«Πρώτα ορίζω τυχαία το μέσο της βελόνας και συγκεκριμένα την απόστασή D του μέσου από την κοντινότερη ευθεία στο πάτωμα. Χωρίς βλάβη της γενικότητας, θεωρώ ότι πάντα η κοντινότερη ευθεία στο μέσο της βελόνας είναι δεξιά της. Στην συνέχεια ορίζω τον προσανατολισμό της βελόνας που καθορίζεται από την γωνία θ μεταξύ της βελόνας και της κοντινότερης κάθετης. Η γωνία αυτή μπορεί να παίρνει τιμές στο διάστημα [0, 180] μοίρες. Αν η ποσότητα $\frac{\ell}{2}\sin\theta$ *είναι μεγαλύτερη από D έχω τομή (βλ. οριζόντια διακεκομμένη). Συνεπώς για να ορίσω τη θέση και την γωνία μιας βελόνας σε σχέση με το πάτωμα χρειάζομαι δυο τυχαίους αριθμούς. Ο ένας θα αντιπροσωπεύει πόσο κοντά είναι η βελόνα σε μια ευθεία και ο άλλος ποια κλίση θα έχει η βελόνα σε σχέση με την ευθεία αυτή»*



Καλείστε να υλοποιήσετε σε C τον αλγόριθμο που παρατίθεται σε ψευδοκώδικα παρακάτω, όπου random(a, b) γεννήτρια τυχαίων αριθμών στο διάστημα [a, b]:

```
Διάβασε: d, ℓ, arithmos_velones
temnoun = 0;
pp = 3.1415912;
for (i=1; i<arithmos_velones; i++)
{
      D = random(0, d/2)
      theta = random(0, pp);
      if (D <= (ℓ/2) *sin(theta))
          temnoun++;
}
proseggisi_PI = (2.0* arithmos_velones*l)/( temnoun*d);
```

Αφού υλοποιήσετε τον παραπάνω αλγόριθμο, καλείστε να απαντήστε στα παρακάτω:

1. Αν υποθέσουμε ότι «ρίχνουμε» 10000 βελόνες. Με ποιους από τους παρακάτω συνδυασμούς πέρνουμε καλύτερη προσέγγιση για το π:
    a. d = ℓ=1
    b. d=1, ℓ=0.8
    c. d=1, ℓ=0.6
    d. d=1.2, ℓ=1.0
2. Αν υποθέσουμε ότι d = ℓ=1 ,πόσες βελόνες πρέπει να χρησιμοποιήσουμε προσεγγίζουμε το π με 4 δεκαδικά ψηφία.
3. Συμπληρώστε τον παρακάτω πίνακα:

Απόσταση παραλλήλων	Μήκος βελόνας	Ρίψεις	Επιτυχίες	Τμή του π
1	0,8	5000		
1	1	600		
1	0,6	3204		
1	0,75	1030		
1	0,83	3408		

Άσκηση 1η (Connect Four)

Σκοπός της εργασίας είναι να δημιουργήσετε το κλασικό παιχνίδι στρατηγικής «*Connect Four*» το οποίο θα παίζεται από 2 παίχτες. Στο «*Connect Four*» ο κάθε παίχτης ρίχνει εναλλάξ ένα δίσκο σε μια κατακόρυφη πλατφόρμα με 6 γραμμές και 7 στήλες. Οι δίσκοι του κάθε παίχτη έχουν διαφορετικό χρώμα. Νικητής του παιχνιδιού είναι ο παίχτης που θα σχηματίσει μια ακολουθία από 4 δίσκους του ίδιου χρώματος οριζόντια, κάθετα ή διαγώνια (και προς τις δυο κατευθύνσεις).

Μια τυπική οθόνη του παιχνιδιού σε περιβάλλον κονσόλας θα μπορούσε να είναι:

```
   New Board – 10 Moves
1 |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |
3 |   | X |   |   |   |   |   |
4 |   | O |   | X |   |   |   |
5 |   | O |   | O |   |   |   |
6 | X | O | X | X | O |   |   |
  |---|---|---|---|---|---|---|
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   ---------------------------
Next player: Player 1 (O)
Input column > 3
```

```
    New Board – 11 Moves
1 |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |
3 |   | X |   |   |   |   |   |
4 |   | O |   | X |   |   |   |
5 |   | O | O | O |   |   |   |
6 | X | O | X | X | O |   |   |
  |---|---|---|---|---|---|---|
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   ---------------------------
Next player: Player 2 (X)
Input column > 1
```

```
    New Board – 12 Moves
1 |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |
3 |   | X |   |   |   |   |   |
4 |   | O |   | X |   |   |   |
5 | X | O | O | O |   |   |   |
6 | X | O | X | X | O |   |   |
  |---|---|---|---|---|---|---|
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   ---------------------------
Next player: Player 1 (O)
Input column > 5
```

```
    New Board – 13 Moves
1 |   |   |   |   |   |   |   |
2 |   |   |   |   |   |   |   |
3 |   | X |   |   |   |   |   |
4 |   | O |   | X |   |   |   |
5 | X | O | O | O |   |   |   |
6 | X | O | X | X | O |   |   |
  |---|---|---|---|---|---|---|
  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
   ---------------------------
Player 1 wins
```

Οδηγίες υλοποίησης:

1. Ορίστε έναν πίνακα δυο διαστάσεων:
   `int board[ROWS][COLS];`
   Ο πίνακας αυτός θα οριστεί στη main και θα περνάει σαν παράμετρος σε όλες τις συναρτήσεις που απαιτείται. **Σε κάθε στιγμή, ο πίνακας αυτός αποθηκεύει την κατάσταση του παιχνιδιού**. Οι μεταβλητές ROWS και COLS θα είναι κατάλληλα δηλωμένες με εντολή #define.
2. Ορίστε έναν πίνακα μιας διάστασης:
   `int skyline[COLS];`
   Ο πίνακας αυτός θα αποθηκεύει την επόμενη ελεύθερη θέση (από 0 έως 5) για μια συγκεκριμένη στήλη. Για παράδειγμα στην πάνω αριστερή εικόνα της τυπικής οθόνης ο πίνακας skyline θα είχε τις τιμές: skyline = {4, 1, 4, 2, 4, 5, 5}
   Για την αμέσως δεξιά εικόνα ο πίνακας θα ήταν: skyline = {4, 1, 3, 2, 4, 5, 5}
   Σε περίπτωση που μια στήλη είναι γεμάτη βάζουμε στον πίνακα skyline την τιμή -1.
3. Ορίστε δυο τύπους από δίσκους που θα αντιστοιχούν και στους παίχτες:
   ```
   #define BLACK 0
   #define WHITE 1
   ```
   BLACK θα είναι ο πρώτος παίχτης (0) και θα έχει σαν σχήμα στην οθόνη το 'X' και WHITE θα είναι ο δεύτερος παίχτης (1) που θα έχει σαν σχήμα το 'O'. Επίσης μπορείτε το κενό να το συμβολίσετε με
   ```
   #define EMPTY -1
   ```
4. Ορίστε μια συνάρτηση:
   `void initBoard(int board[][COLS], int numrows, int skyline[], int numcols);`
   η οποία θα αρχικοποιεί τον πίνακα board που αναπαριστά την κατακόρυφη πλατφόρμα και τον πίνακα skyline.
5. Ορίστε μια συνάρτηση:
   `int playerMove(int skyline[], int numcols, int player, int human);`

   Αν η παράμετρος human είναι ίση με 0, το πρόγραμμα θα τυπώνει κατάλληλο μήνυμα, θα διαβάζει από το πληκτρολόγιο την κίνηση (αριθμός στήλης) του παίχτη player και θα επιστρέφει τον αριθμό της στήλης. Η διαδικασία θα επαναλαμβάνεται έως ότου προκύψει μη γεμάτη στήλη.
   Αν η παράμετρος human είναι διάφορη από το 0 (και άρα πρέπει ο υπολογιστής να βρει μια κίνηση για να παίξει), το πρόγραμμα θα κάνει τα εξής για να βρει την επόμενη κίνηση του παίχτη player:
   Α) Θα ελέγχει αν υπάρχει στήλη που αν τοποθετήσει εκεί δίσκο ο παίχτης player κερδίζει.
   Β) Θα ελέγχει αν υπάρχει στήλη που αν ΔΕΝ τοποθετήσει εκεί δίσκο ο παίχτης player χάνει.
   Γ) Θα επιλέγει με χρήση γεννήτριας τυχαίων αριθμών μια στήλη που δεν είναι γεμάτη (προσοχή: αν η γεννήτρια δώσει μια στήλη που είναι γεμάτη, πρέπει να ξαναπροσπαθήσει).
6. Ορίστε μια συνάρτηση:
   `int putMoveOnBoard(int board[][COLS], int numrows, int skyline[], int numcols, int player, int column, int *move)`

   Η συνάρτηση θα παίρνει σαν είσοδο τη στήλη column που επιλέξει ο παίχτης player (και η οποία έχει ελεγχθεί αν είναι δυνατή) και θα εισάγει ένα δίσκο του παίχτη player, τροποποιώντας κατάλληλα τον πίνακα board και τον skyline. Επίσης θα αυξάνει κατά ένα τον συνολικό αριθμό κινήσεων move.
7. Ορίστε μια συνάρτηση:
   `void checkForEndOfGame(int board[][COLS], int numrows, int skyline[], int numcols)`

Η συνάρτηση αυτή θα ελέγχει αν το παιχνίδι με την παρούσα κατάσταση του board έχει τελειώσει. Η συνάρτηση θα ελέγχει αν κάποιος παίχτης έχει κερδίσει, είτε αν υπάρχει ισοπαλία. Σε κάθε περίπτωση τερματισμού θα εμφανίζει κατάλληλο μήνυμα και θα τερματίζει το πρόγραμμα.
Βοηθητικά ορίστε τις συναρτήσεις:

```
void countConsecutiveHorizontal(int board[][COLS], int numrows, int player,
int *max_horiz)
void countConsecutiveVertical(int board[][COLS], int numrows, int player,
int *max_vertic)
void countConsecutiveDiagonal(int board[][COLS], int numrows,  int player,
int *max_diagon)
```

οι οποίες θα υπολογίζουν το μέγιστο αριθμό ***συνεχόμενων*** δίσκων του παίχτη player στην πλατφόρμα στην οριζόντια, στην κατακόρυφη και στις δυο διαγώνιες διευθύνσεις[1]. Οι τιμές θα επιστρέφονται μέσω των μεταβλητών max_horiz, max_vertic και max_diagon αντίστοιχα.
Σχετικά με τον έλεγχο των διαγωνίων 1η περίπτωση (αριστερά προς δεξιά): Σας παραθέτουμε τη λογική αρίθμηση του πίνακα board.

board	0	1	2	3	4	5	6
0	(0, 0)	(0, 1)	(0, 2)	(0, 3)	(0, 4)	(0, 5)	(0, 6)
1	(1, 0)	(1, 1)	(1, 2)	(1, 3)	(1, 4)	(1, 5)	(1, 6)
2	(2, 0)	(2, 1)	(2, 2)	(2, 3)	(2, 4)	(2, 5)	(2, 6)
3	(3, 0)	(3, 1)	(3, 2)	(3, 3)	(3, 4)	(3, 5)	(3, 6)
4	(4, 0)	(4, 1)	(4, 2)	(4, 3)	(4, 4)	(4, 5)	(4, 6)
5	(5, 0)	(5, 1)	(5, 2)	(5, 3)	(5, 4)	(5, 5)	(5, 6)

	Δείκτες						Αλγόριθμος (i, j)
1η διαγώνιος	(2, 0)	(3, 1)	(4, 2)	(5, 3)			i=2, j=0 while(i<6){ i++;j++;}
2η διαγώνιος	(1, 0)	(2, 1)	(3, 2)	(4, 3)	(5,4)		i=1, j=0 while(i<6) {i++;j++;}
3η διαγώνιος	(0, 0)	(1, 1)	(2, 2)	(3, 3)	(4, 4)	(5, 5)	i=0, j=0 while(i<6) {i++;j++;}
4η διαγώνιος	(0, 1)	(1, 2)	(2, 3)	(3, 4)	(4, 5)	(5, 6)	i=0, j=1 while(j<7){ i++;j++;}
5η διαγώνιος	(0, 2)	(1, 3)	(2, 4)	(3, 5)	(4, 6)		i=0, j=2 while(j<7) {i++;j++;}
6η διαγώνιος	(0, 3)	(1, 4)	(2, 5)	(3, 6)			i=0, j=3 while(j<7) {i++;j++;}

Σχετικά με τον έλεγχο των διαγωνίων 2η περίπτωση (δεξιά προς αριστερά): *Εργαστείτε παρόμοια*

8. Ορίστε μια συνάρτηση:

[1] Προσοχή: Μας ενδιαφέρουν τα συνεχόμενα ίδια δισκία στη σειρά!!

int printBoard(int board[][COLS], int numrows, int moves);

που θα τυπώνει την κατακόρυφη πλατφόρμα του παιχνιδιού στην οθόνη καθώς και πόσες κινήσεις έχουν παιχτεί μέχρι τώρα. Για την εκτύπωση, και μόνο, προσθέτουμε +1 στον αριθμό της κάθε στήλης και της κάθε γραμμής.

Τέλος θα χρειαστείτε μια main στην οποία θα καλείτε όλες τις παραπάνω και θα προσομοιώνετε το παιχνίδι. Μια τυπική main του παιχνιδιού θα μπορούσε να είναι:

```
#define ROWS 6
#define COLS 7
#define BLACK 0
#define WHITE 1
#define EMPTY -1

#define PLAYER0 BLACK
#define PLAYER1 WHITE
#define HUMAN 0
#define COMPU 1

int main(int argc, char** argv)
{
  int board[ROWS][COLS];
  int skyline[COLS];
  int moves=0, numplayers, player0, player1;

  printf("Epelexe posoi paixtes 1/2 ");
  scanf("%d", &numplayers);
  player0 = HUMAN;
  if (numplayers == 1)
      player1 = COMPU;
  else if (numplayers >= 2)
      player1 = HUMAN;


  initBoard(board, ROWS, skyline, COLS);
  printBoard(board, ROWS, moves);

  while (1)
  {
  //O  Protos paixtis epilegei 1-7
   column0 = playerMove(skyline, COLS, PLAYER0, player0);
   putMoveOnBoard(board,  ROWS, skyline, COLS, PLAYER0, column0, &moves);
   printBoard(board, ROWS, moves);
   checkForEndOfGame(board, ROWS, skyline, COLS);

  // O Deuteros paixtis epilegei 1-7
   column1 = playerMove(skyline, COLS, PLAYER1, player1);
   putMoveOnBoard(board, ROWS, skyline, COLS, PLAYER1, column1, &moves);
   printBoard(board, ROWS, moves);
   checkForEndOfGame(board, ROWS, skyline, COLS);

  }
}
```