02393 C++ Programming Exercises

Week 3, February 15, 2016

To be handed in via CodeJudge, before February 22, 5pm
`https://dtu.codejudge.net/02393-f16/assignment/show/1240`

Histogram A *histogram* is a graphical way to display a distribution of a dataset into discrete intervals. Consider for instance a data set given by the numbers 100 95 47 88 86 92 75 89 81 70 55 80 and suppose we want build a histogram with 11 intervals $[0-10)$, $[10-19)$, $\ldots$, $[100-110)$ to be textually represented as

```
0: 0
10: 0
20: 0
30: 0
40: 1
50: 1
60: 0
70: 2
80: 5
90: 2
100: 1
```

The task is to write a program that reads the following values (in this order) from the standard input (cin):

- the number ℓ of intervals (e.g. 11 in the example)
- the size n of the data set (e.g. 12 in the example)
- and n non-negative integers.

and then outputs the histogram in the above format.

For simplicity, set the size k of the intervals to be the integer $\lceil \frac{m}{\ell} \rceil$, where m is the maximum number in the data set. That is, interval i should be $[(i-1) \times k \ldots i \times k)$. Function $\lceil \cdot \rceil$ is implemented as function ceil() in library math.h

As an example, the input for the previous histogram is:
`11 12 100 95 47 88 86 92 75 89 81 70 55 80`
and the output is as above, where one can see that k is $\lceil \frac{100}{11} \rceil = \lceil 9,0909\ldots \rceil = 10$, and thus the i-th interval starts at $(i-1)\times 10$. For instance the last interval (11-th) starts at $(11-1)\times 10 = 100$.

As another example: with the same data but interval size $\ell = 7$ we have the input
`7 12 100 95 47 88 86 92 75 89 81 70 55 80`
and the output is:

```
0: 0
15: 0
30: 0
45: 2
60: 1
75: 6
90: 3
```

where the size of the intervals is $\lceil \frac{100}{7} \rceil = \lceil 14,2857\ldots \rceil = 15$.

Random Walk In the lecture, we discussed the following piece of code

```
typedef enum {wood , stone} material;

typedef struct{
  int x,y;
  bool isWall;
  material type;
}field;

#define n 16
#define m 12
field playground[n][m];

int main(){
  for (int i=0; i<n; i++){
    for (int j=0; j<m; j++){
      playground[i][j].x=i;
      playground[i][j].y=j;
      playground[i][j].isWall=(i==0||i==(n-1)||(j==0&&i!=3) ||j==(m-1));
      if (playground[i][j].isWall && !(i==3 && j==0))
        playground [i][j].type=stone;
      else
        playground [i][j].type=wood;
    }
  }
}
```

We want to make a mini text-based game out of this.

- Define two global variables x and y that represent the position of a player in the playground, and let's have initially $x = y = 5$.
- Write a function that uses these global variables and displays the current situation on the standard output, writing a "`*`" for every field that is a wall, and a single space " " for every field that is empty, except the field that the player is on, that is denoted "`O`". (The function may assume that the player is always within the playground.) Thus, the initial state is displayed as follows:

```
*** ************
*            *
*            *
*            *
*            *
*   O        *
*            *
*            *
*            *
*            *
*            *
***************
```

- Obtain a character from the standard input. If the character is "`l`", then it means the player should make one step to the left if possible, i.e., if the field exists and is not a wall. If it is not possible, then the player remains in the same position. Similarly, we use characters "`r`" for right, "`u`" for up, and "`d`" for down. After each step, the playground is displayed again and the user can give the next command. The game ends when the user gives command "`q`".