CS19001: Programming and Data Structures Laboratory

Assignment No. 10A (Linked Lists and Abstract Data Types)

Date: 25-October-2019

Problem Statement:

In the well-connected city of Wonderland, the train stoppages are marked using integers as IDs. The heart of the city is reached via railway junction (stoppage) numbered 0, where the palace of the King-of-Hearts is located[1]. Also the trains may start from various source points (again demarcated by integer numbers), but follow a pattern in while they give stoppages. In particular, if a train starts or previously stops at junction numbered n, then its next stoppage will be $(n - \lfloor\sqrt{n}\rfloor)$. This means that every train runs from a source point into monotonically decreasing stoppage numbers until it reaches the destination point, i.e. 0. This means that we can build a list of monotonically decreasing integers marking the stoppages of a train. The list terminates after the node containing the value 0. For example, consider two trains `T1` and `T2` are starting their journey from stoppages 22 and 20, respectively. Their trajectory of stoppages are given as follows:

$$\texttt{T1} \Rightarrow 22 \rightarrow 18 \rightarrow 14 \rightarrow 11 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$$
$$\texttt{T2} \Rightarrow 20 \rightarrow 16 \rightarrow 12 \rightarrow 9 \nearrow$$

It may be noted that, the stoppages of these 1two trains are common from stoppage numbered 6. We call that these stoppages of these two trains *intersect* at stoppage 6.

Your task is to develop a C-program which outputs the trajectories of two trains from their given starting points and find the intersecting stoppage. In order to do so, you may follow the following specifications:

- Define a node in the list in the usual way:

```
typedef struct node_type {
        int data;
        struct node_type *next;
} node;
```

- Write a C function to create a single list like `T1` as explained above. The list starts with a supplied integer value n, and subsequently uses the above formula for the subsequent nodes.

```
node *genSeq1 ( int n )
```

- Create another list to be headed by `T2` (as shown in previous figure). This second list starts with another value (like 20), and contains nodes storing integer values satisfying the same formula used in the first list. After some iterations, two lists must encounter a common value (6 in the example figure). From this node onwards, the second list follows the same nodes and links as the first list (you must not generate these redundant nodes and links in your second list, but point them into the first list).
  - Write a C function to create the second list. The header `T1` to the first list is passed to this function. Also, starting value n for the second list is passed.

```
node *genSeq2 ( node *T1, int n )
```

  - Write a C function that, given the headers `T1` and `T2` as input, returns a pointer to the first common node in these two lists.

```
node *getIntersection ( node *T1, node *T2 )
```

- Write a C-program main routine that –
  - Asks from user two integer values $n1$ and $n2$.
  - Call `genSeq1`() and `genSeq2`() functions to build two lists `T1` and `T2`.
  - Prints the two lists, `T1` and `T2`.
  - Calls `genIntersection`() function to find the common node.
  - Prints the common node information.

[1]For your information, Alice's house in Wonderland city is near the railway junction/stop 2 which is two stoppages apart ($2 \rightarrow 1 \rightarrow 0$) from the King-of-Hearts palace. More importantly, all the trains that start from railway stop numbered higher than 2 gives stoppages in the Alice's home stoppage 2 (why?).

Example Inputs/Outputs:

Sample-1:

```
Enter Starting Stoppage Number (ID) for Train-1: 22
Enter Starting Stoppage Number (ID) for Train-2: 20

Stoppages (ID) for Train-1:
T1 --> [ 22 ] --> [ 18 ] --> [ 14 ] --> [ 11 ] --> [ 8 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 20 ] --> [ 16 ] --> [ 12 ] --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 6 ]
```

Sample-2:

```
Enter Starting Stoppage Number (ID) for Train-1: 14
Enter Starting Stoppage Number (ID) for Train-2: 21

Stoppages (ID) for Train-1:
T1 --> [ 14 ] --> [ 11 ] --> [ 8 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 21 ] --> [ 17 ] --> [ 13 ] --> [ 10 ] --> [ 7 ] --> [ 5 ] --> [ 3 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 2 ]
```

Sample-3:

```
Enter Starting Stoppage Number (ID) for Train-1: 12
Enter Starting Stoppage Number (ID) for Train-2: 24

Stoppages (ID) for Train-1:
T1 --> [ 12 ] --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 24 ] --> [ 20 ] --> [ 16 ] --> [ 12 ] --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 12 ]
```

Sample-4:

```
Enter Starting Stoppage Number (ID) for Train-1: 28
Enter Starting Stoppage Number (ID) for Train-2: 15

Stoppages (ID) for Train-1:
T1 --> [ 28 ] --> [ 23 ] --> [ 19 ] --> [ 15 ] --> [ 12 ] --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 15 ] --> [ 12 ] --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 15 ]
```

Sample-5:

```
Enter Starting Stoppage Number (ID) for Train-1: 9
Enter Starting Stoppage Number (ID) for Train-2: 9

Stoppages (ID) for Train-1:
T1 --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 9 ] --> [ 6 ] --> [ 4 ] --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 9 ]
```

Sample-6:

```
Enter Starting Stoppage Number (ID) for Train-1: 2
Enter Starting Stoppage Number (ID) for Train-2: 1

Stoppages (ID) for Train-1:
T1 --> [ 2 ] --> [ 1 ] --> [ 0 ]--> X

Stoppages (ID) for Train-2:
T2 --> [ 1 ] --> [ 0 ]--> X

Intersect Stoppage ID --> [ 1 ]
```

Submit a single C source file. Do not use global/static variables.

CS19001: Programming and Data Structures Laboratory

Assignment No. 10B (Linked Lists and Abstract Data Types)

Date: 25-October-2019

Problem Statement:

It has been observed that the peace in the city of Wonderland was getting increasingly disturbed day-by-day due to the turmoils created by the rebels and the traitors. To take control of such a situation, the King-of-Hearts captures n such traitors (as convicts) and put them in prison. Unfortunately, the Knave-of-Hearts was also selected as a convict by mistake by the King-of-Hearts. Now, the King-of-Hearts appoints the Made-Hatter as the prosecutor, who was smart enough to immediately recognize the wrongly chosen Knave-of-Hearts as convicts and thought of a procedure by which he can save the Knave-of-Hearts. His strategy was as follows:

The Mad-Hatter called all n convicts, randomly numbered then as $1, 2, \ldots, n$ and told them to sit in a round table in the same order as numbered. The Mad-Hatter keeps on circling around the table in the order $1, 2, \ldots, n$ and then back to 1. He starts his rotation at convict 1 and kills (and removes) every m-th convict he encounters (the Mad-Hatter chooses m by his own). After $(n-1)$ iterations, only one convict survives and is set free. Now, the only trick the Mad-Hatter did here is to assign appropriate number (say, k) to the Knave-of-Hearts such that this k-th convict becomes the survivor at the end by his chosen m-th elimination mechanism. As an example, consider $n = 10$ and $m = 3$ and let the Mad-Hatter assigns 4 to the Knave-of-Hearts. Now, the convicts are killed/prosecuted in the order $3, 6, 9, 2, 7, 1, 8, 5, 10$, and it can be concluded that the convict 4 (which is the Knave-of-Hearts) survives at the end.

Having known this strategy, now you are in a position to write a C-program that, given n and m, determines the survivor (that is k which had been pre-computed by the Mad-Hatter in his mind while numbering the convicts). You are asked to use a circular linked list which is like an ordinary linked list with the only exception that the pointer of the last node points to the first node (instead of being NULL). The following program first creates a circular list on n persons and then simulates the prosecutor by traversing around the circular list. The program finally prints the survivor. No dummy node is maintained at the head. However, in order to make deletion easy, the running pointer points to the node immediately before the node to be deleted.

Example Inputs/Outputs:

Sample-1:

```
Enter n: 10
Enter m: 3
Convict 3 Killed!
Convict 6 Killed!
Convict 9 Killed!
Convict 2 Killed!
Convict 7 Killed!
Convict 1 Killed!
Convict 8 Killed!
Convict 5 Killed!
Convict 10 Killed!
==> Convict 4 Survives!
```

Sample-2:

```
Enter n: 20
Enter m: 20
Convict 20 Killed!
Convict 1 Killed!
Convict 3 Killed!
Convict 6 Killed!
Convict 10 Killed!
Convict 15 Killed!
Convict 4 Killed!
Convict 13 Killed!
Convict 7 Killed!
Convict 19 Killed!
Convict 18 Killed!
```

```
Convict 5 Killed!
Convict 12 Killed!
Convict 9 Killed!
Convict 14 Killed!
Convict 11 Killed!
Convict 8 Killed!
Convict 17 Killed!
Convict 16 Killed!
==> Convict 2 Survives!
```

Sample-3:

```
Enter n: 45
Enter m: 13
Convict 13 Killed!
Convict 26 Killed!
Convict 39 Killed!
Convict 7 Killed!
Convict 21 Killed!
Convict 35 Killed!
Convict 4 Killed!
Convict 19 Killed!
Convict 34 Killed!
Convict 5 Killed!
Convict 22 Killed!
Convict 38 Killed!
Convict 10 Killed!
Convict 28 Killed!
Convict 45 Killed!
Convict 18 Killed!
Convict 40 Killed!
Convict 14 Killed!
Convict 33 Killed!
Convict 11 Killed!
Convict 32 Killed!
Convict 12 Killed!
Convict 37 Killed!
Convict 17 Killed!
Convict 44 Killed!
Convict 27 Killed!
Convict 9 Killed!
Convict 43 Killed!
Convict 30 Killed!
Convict 23 Killed!
Convict 15 Killed!
Convict 6 Killed!
Convict 3 Killed!
Convict 8 Killed!
Convict 20 Killed!
Convict 29 Killed!
Convict 42 Killed!
Convict 25 Killed!
Convict 16 Killed!
Convict 24 Killed!
Convict 41 Killed!
Convict 1 Killed!
Convict 2 Killed!
Convict 31 Killed!
==> Convict 36 Survives!
```

Sample-4:

```
Enter n: 1
Enter m: 5
==> Convict 1 Survives!
```

Submit a single C source file. Do not use global/static variables.