IC250 Laboratory Assignment-03

Aditya Nigam

August 28, 2016

IC250 Programming and Data Structure Practicum

Lab Assignment No. 03; Date : {30,31} Aug and 01 Sep , 2016

NOTE : In this assignment two problems are given. First one is *Hot potato* or *Josephus problem* and the second one is implementation of queue using stacks. Both of them are described as below.

Write both codes as follows :

- One file containing all the defined function ($my\_fun.c$)
- One header file containing all function prototype and global variables ($my\_library.h$).
- One file containing the main function that can call any function in (my_fun) by including ($my\_library$) named as ($my\_assignment.c$).
- Finally write a makefile that can compile the code automatically ($Makefile$).

The sample is already uploaded on moodle. Download it, understand it and code accordingly.

Introduction/Problem Context

This laboratory assignment focuses on use of queue data structure in the context of solving some general puzzles. This data structure can only allow to access a special element and that by its own specific routine. Also their insertion and deletions are also different.

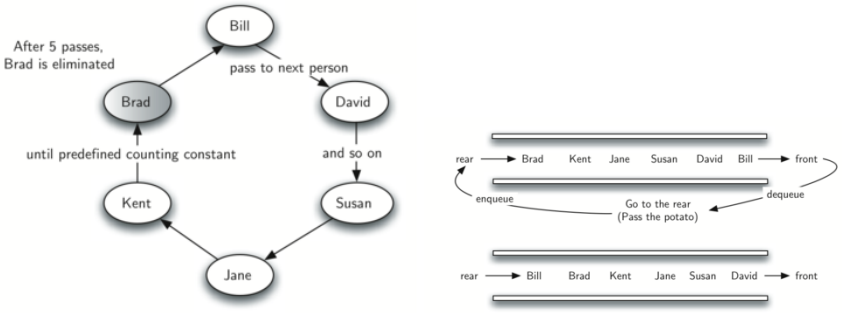
It assumes that you are familiar with static and dynamic data representation and C language features related to them. You may refer to the references given, if you required to refresh these topics.

Problem [A] : Hot Potato (Josephus problem)

In this game children line up in a circle and pass an item from neighbor to neighbor as fast as they can. At a certain point (say after a round) in the game, the action is stopped and the child who has the item (the potato) is removed from the circle. Play continues until only one child is left.

1. **Players :** Let there are n children.
2. **Elimination Rule :** Every i^{th} child will be removed after a round .

Devise a winning strategy, it mean that one can choose a strategy to be the last person left who will be declared as the winner.



(a) Six Children Playing Hot Potato (b) Queue Implementation

Figure 1: Hot Potato Setup

Task Description

You are required to write a C program which take the number of children (n) from user and output the winning strategy.

Pseudocode :

```
Hot_potato(n,i)

[1] Problem for collection of n elements (realize via queue).
[2] Associate a potato (token) to the front queue element.
[3] Store queue elements as their order around the circle.
[4] Now passing potato is equivalent to dequeuing an element.
    and immediately enqueuing it again.
[5] Repeat this process i times.
[6] Now Dequeue and discard front element.
[7] Do this untill only one element is left.
[8] Report the finally left element. (WINNER)
```

Input Data and Format

- Enter number of children : n
- Elimination Rule : i

Expected Output for Correct and Incorrect Inputs

Return: winning strategy *i.e.* a safe position s for some given values of n and i.

Example : If $n = 7$ and $i = 3$, then the safe position is $s = 4$. The persons at positions $3, 6, 2, 7, 5, 1$ are removed in order, and person at position 4 survives.

Sample Output :

```
$ hpotato

Please enter values of n and i : 5 2

The removal sequence is as follows -

[1] Firstly, the person at position 2 is removed.
[2] Then person at position 4 is removed.
[3] Then person at position 1 is removed.
[4] Finally, the person at position 5 is removed.

Hence the person at position 3 survives. (WINNER)
```

Problem [B] : Queue using Stack

1. Implement a basic queue using two linked list based user stacks. You are expected to use the code of your previous assignment in order to use the stack functionality. Write the code as follows :
   - One file containing all the defined function ($my\_fun.c$)
   - One header file containing all function prototype and global variables ($my\_library.h$).
   - One file containing the main function that can call any function in (my_fun) by including ($my\_library$) named as ($my\_assignment.c$).
   - Finally write a makefile that can compile the code automatically ($Makefile$).

   The sample is already uploaded on moodle. Download it, understand it and code accordingly.

Optional part for problem A

- Estimate the time complexity in terms so n, i.
- Try to solve it theoretically using recurrence relations.
- Try to solve it using dynamic programming in $O(n)time$, where n is the number of persons.
- **Extended Josephus problem definition :** There are n persons, numbered 1 to n, around a circle. We eliminate second of every two remaining persons until one person remains. Given the value of n, determine the number of x^{th} person who is eliminated.

Optional part for problem B

1. Implement a basic queue using one linked list based user stack and one system stack (*i.e.* function call stack).

References

- http://interactivepython.org/runestone/static/pythonds/BasicDS/queues.html
- https://en.wikipedia.org/wiki/Josephus_problem