Technical Note

2229027 이지민

1

Problem : Replace linked stack code implemented with a simple linked list to use a doubly linked list.

Fixed part

Push function: When new data enters the stack, the push function implemented as a simple linked list assigns the existing head to the link of the new data and receives the head of LinkedStackedType s as the new data. On the other hand, the push function implemented as a doubly linked list must be coded separately for the case when the stack is empty and the case when there is more than one data. First, if the stack is empty, it assigns itself to rlink and llink to make them continue. If there is more than one data in the stack, update the new node's llink and rlink with the old top and the old top's rlink, respectively. Then update the last node that was connected to the first node by s->top->rlink->llink by assigning temp to the llink. End the connection by assigning temp to the rlink of the top node. Finally, update top with temp.

Pop function: We can think of it in two ways: when there is a single piece of data in the stack, and when there is more than one piece of data. First, we assign NULL to top because running the pop function when there is only one data will result in an empty stack. If we run the pop function when there is more than one data, we disconnect the temp node that holds the top node. Link the rlink of the node before top, connected by Temp->llink, with the last node assigned to temp->rlink. Connect the previous node of temp to the last node accessed by temp->rlink by accessing it with temp->llink. Then replace top with the previous node and free the dynamic memory allocated for temp.

Printed result

3
2
1

2

Problem: Increase the number of tellers from one to two in a bank simulation where customers are represented by Queues.

Fixed part

Remove_customer function: takes an int I as a parameter to know how many times the customer has been served by the teller.
Main function: use a for statement to represent the two tellers. Assign each teller a separate service_time and have them do their job in each timestamp.

Translated with www.DeepL.com/Translator (free version)

Printed result

```
Current time=1
Customer 0 comes in 1 minutes. Service time is 1 minutes.
```

```
Customer 0 starts service in 1 minutes. Wait time was 0 minutes. In bank_staff 1
Current time=2
Current time=3
Customer 1 comes in 3 minutes. Service time is 3 minutes.
Customer 1 starts service in 3 minutes. Wait time was 0 minutes. In bank_staff 1
Current time=4
Customer 2 comes in 4 minutes. Service time is 1 minutes.
Customer 2 starts service in 4 minutes. Wait time was 0 minutes. In bank_staff 2
Current time=5
Customer 3 comes in 5 minutes. Service time is 4 minutes.
Customer 3 starts service in 5 minutes. Wait time was 0 minutes. In bank_staff 2
Current time=6
Current time=7
Customer 4 comes in 7 minutes. Service time is 3 minutes.
Customer 4 starts service in 7 minutes. Wait time was 0 minutes. In bank_staff 1
Current time=8
Current time=9
Customer 5 comes in 9 minutes. Service time is 1 minutes.
Customer 5 starts service in 9 minutes. Wait time was 0 minutes. In bank_staff 2
Current time=10
Customer 6 comes in 10 minutes. Service time is 4 minutes.
Customer 6 starts service in 10 minutes. Wait time was 0 minutes. In bank_staff 1
====result====
Number of customers served = 7
Total wait time = 0 minutes
Average wait time per person = 0.000000 minutes
Number of customers still waiting = 0
```