SCJ2013 Data Structure & Algorithms

Queue – Linked List Implementation

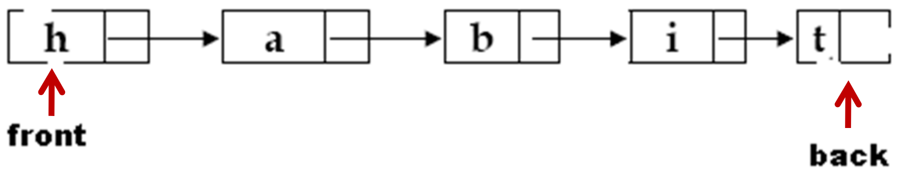
Nor Bahiah Hj Ahmad

Queue Implementation Link List

Pointer-Based Implementation

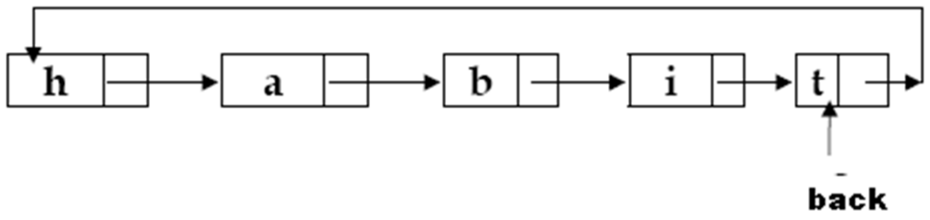
- Can be implemented using linear linked list or circular linked list.
  - Linear linked list

    Need two external pointer (front and back)



  - Circular linked list

    Need onle one pointer, that point at back.



Queue Implementation Link List

Need 2 structure

- Declaration of the node

```
     struct nodeQ {
          char item;
          nodeQ * next;
     }
```

- Declaration of the queue

```
 class queue
 {public:
          nodeQ *backPtr, * frontPtr;
          // operations for queue
 };
```

Queue
frontPtr *backPtr*

createQueue()
destroyQueue()
isEmpty();
enQueue();
deQueue();
getFront();
getRear();

Queue Implementation Link List

createQueue()

```
backPtr = Null; frontPtr = NULL;
```

destroyQueue()

Destroy the whole nodes in the queue

```
nodeQ *temp = frontPtr;
while (temp){
frontPtr = temp->next;
delete temp; temp=frontPtr; }
```

isEmpty()

```
backPtr == Null && frontPtr == NULL
```

Insert to a linear queue

Inserting a new node at the back needs 3 pointer changes

1. Change next pointer in the new node
2. Change the next pointer in the back node
3. Change the external pointer

- Special case:
    - If the queue is empty

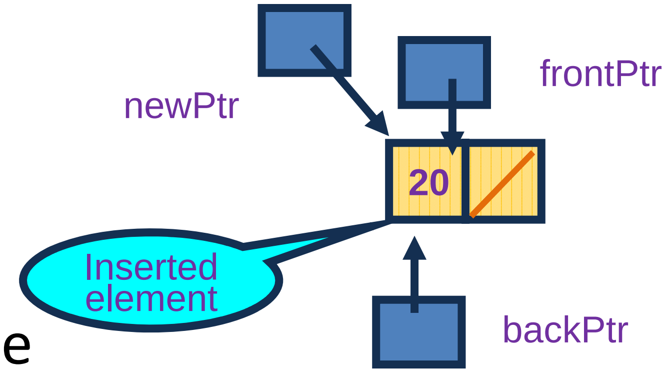
Queue Implementation: Linear Linked List

Linear linked list with 2 external pointers

1. Create a new node -> newPtr
2. Insert to an empty queue

   newPtr -> next = NULL

   `frontPtr=backPtr = newPtr`

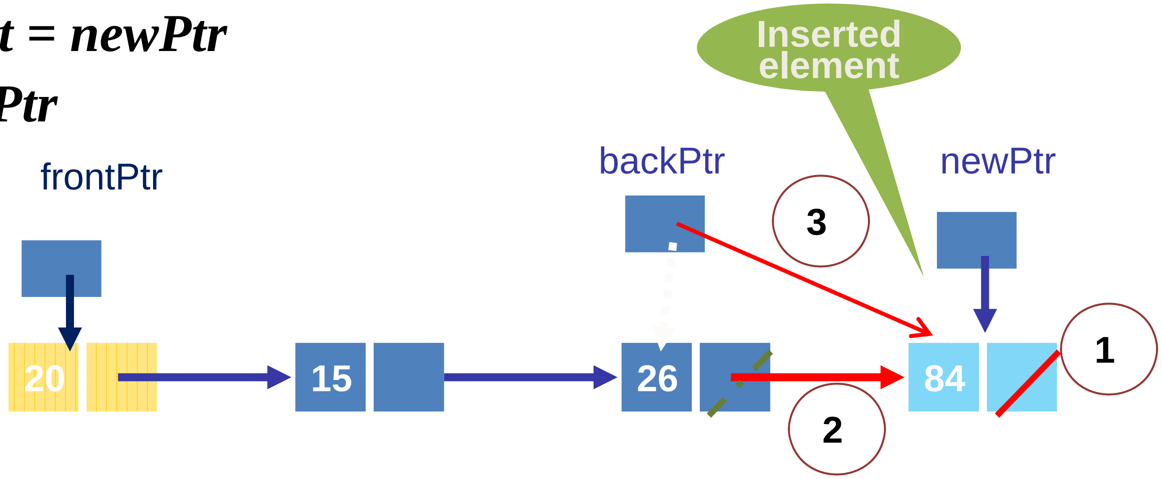


3. Insertion to a non-empty queue

   1 ***newPtr -> next = NULL***

   2 ***backPtr -> next = newPtr***

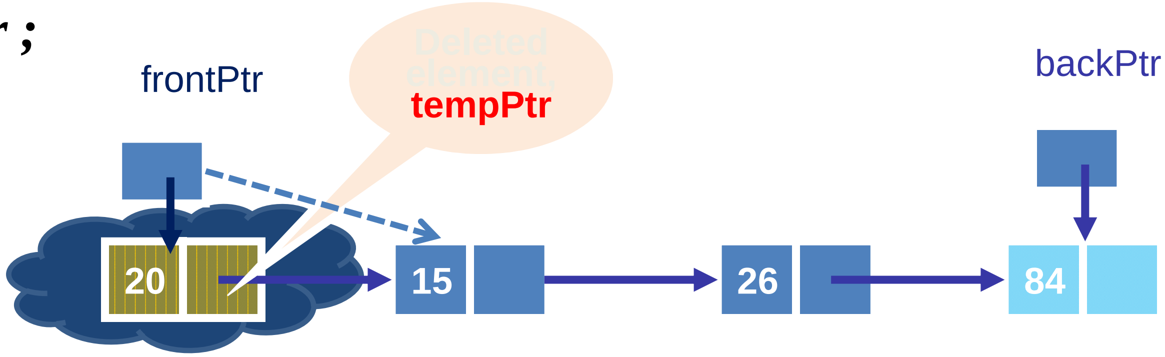
   3 ***backPtr = newPtr***



Delete from Linear queue

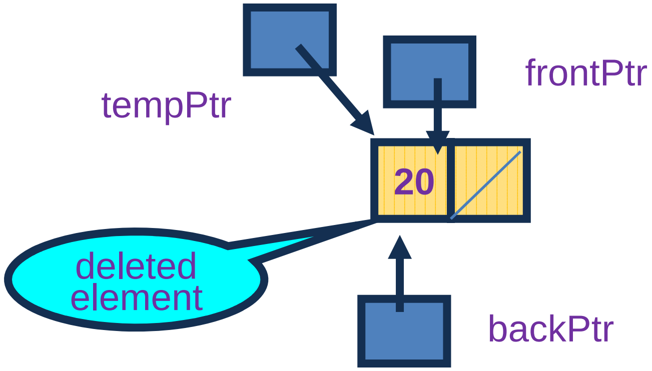
- Deletion
  - Delete from the Front
  - Only one pointer change is needed
  - Special case:
    - If the queue contains one item only
- Deletion Code

```
tempPtr = frontPtr
frontPtr = frontPtr -> next
tempPtr -> next = NULL
delete tempPtr ;
```

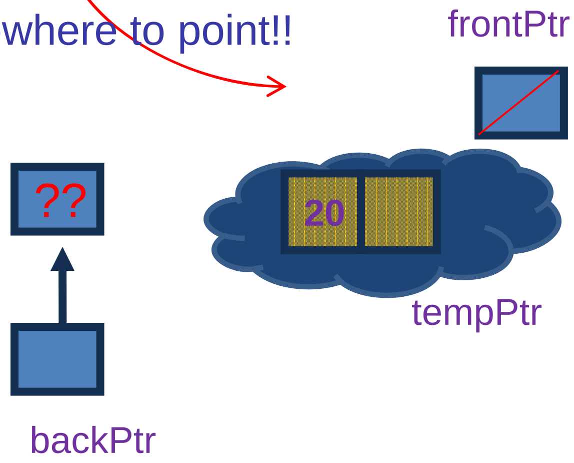


Delete from Linear queue

If the queue contains one item only,



After deletion, backPtr has nowhere to point!!



Need to add this statement:

```
If (!frontPtr)
    backPtr = NULL;
```

Circular Queue Implementation

Circular linear linked list with one external pointer

– Insertion

- Into an empty queue

  NewPtr -> Next = NewPtr

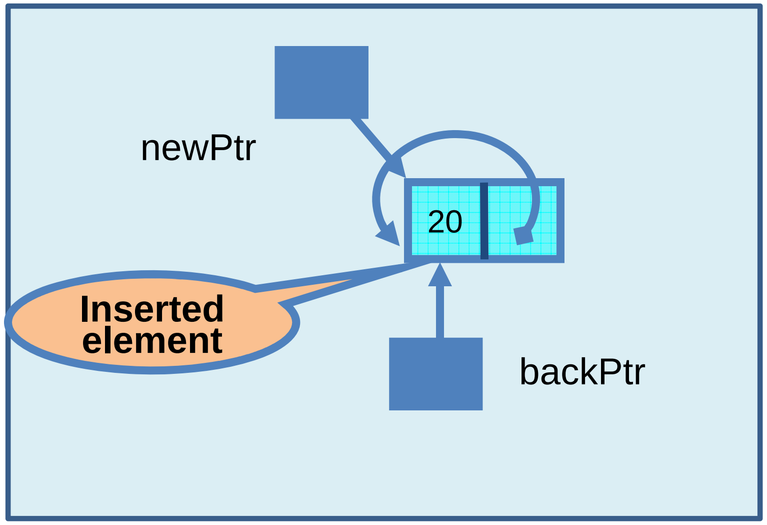
  BackPtr = NewPtr

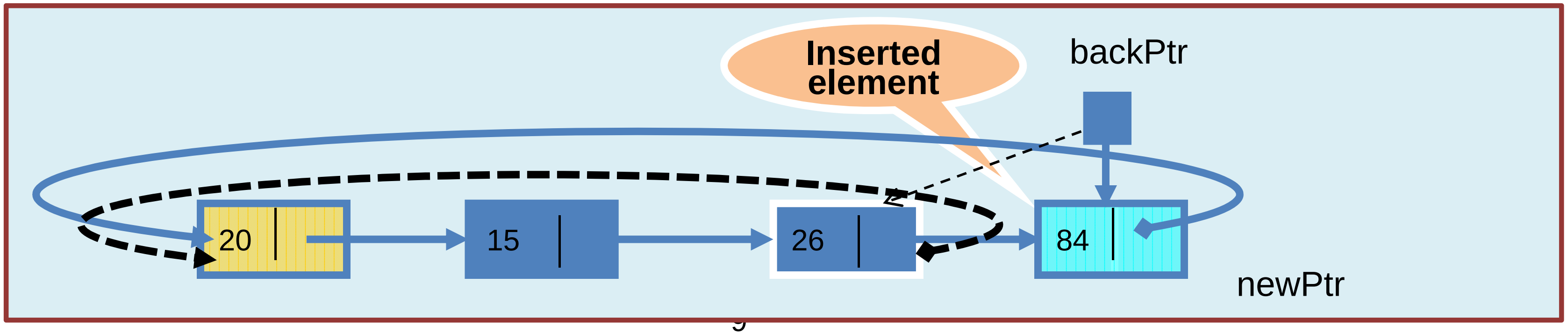
- Into a non-empty queue

  NewPtr -> Next = BackPtr-> Next

  BackPtr -> Next = NewPtr

  BackPtr = NewPtr



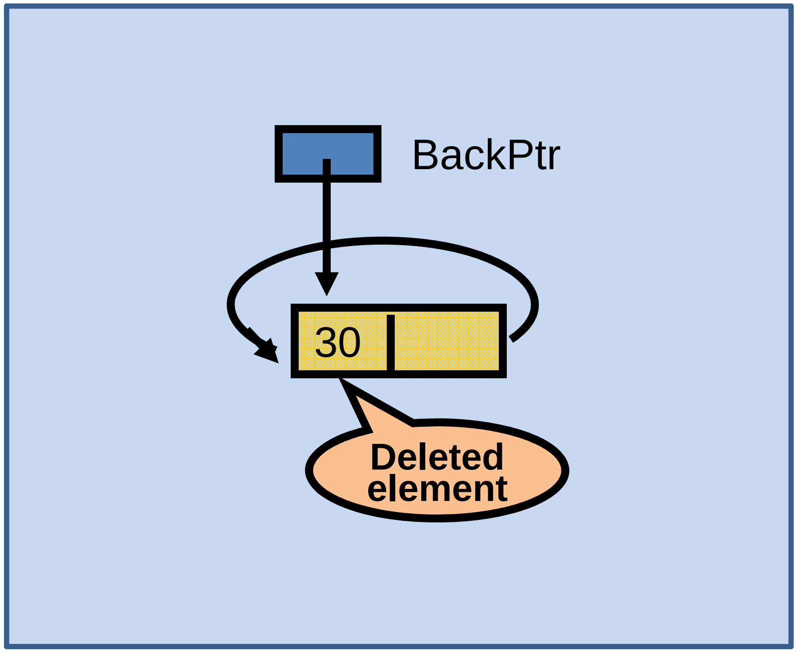


Circular Queue Implementation

Deletion

- From a one-node (one item) queue

```
deletePtr  = BackPtr -> Next
If (deletePtr = BackPtr)
        BackPtr = NULL
delete deletePtr
```

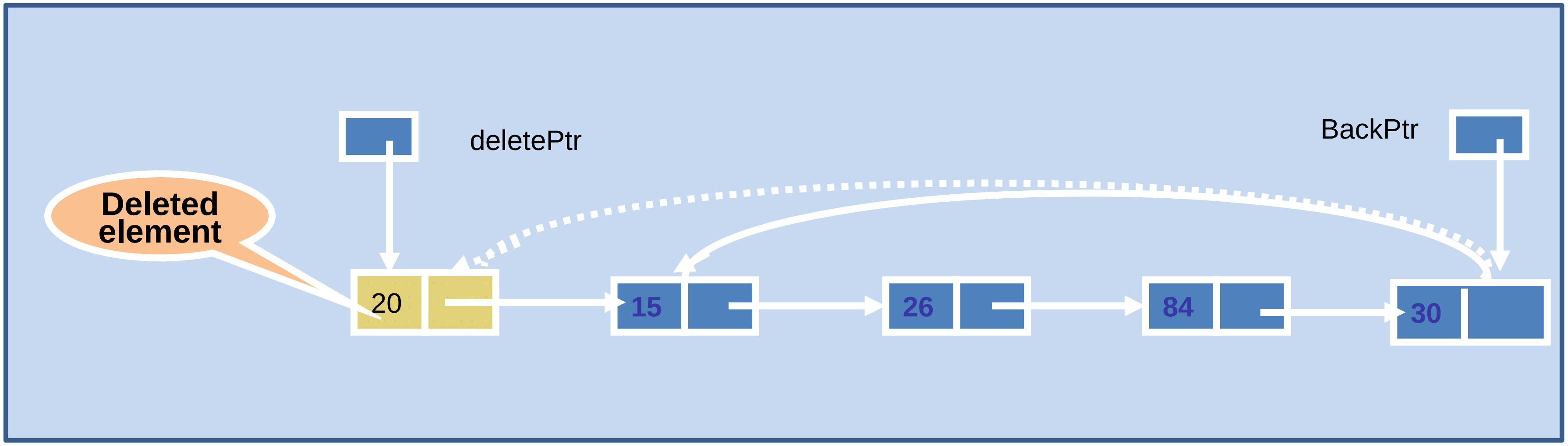


Circular Queue Implementation

Deletion

- From a non-empty, more than one item queue

```
deletePtr  = BackPtr -> Next
BackPtr -> Next =  deletePtr -> Next
delete deletePtr
```



Array Implementation vs Linked Lists Implementation

- Implementation
  - Array
    - Prevents the **enqueue** operation from adding an item to the queue if the array is full.
    - No overhead of pointer manipulation
  - Linked list
    - No size restriction on the **enqueue** operation
    - More efficient, and flexible
    - More complicated than ADT List

Summary of Queue

- Operations are defined in terms of position of data items
- Position is restricted to the front and back of the queue.
- Operations:
  - *create:*
    - Creates an empty ADT of the Queue type
  - *isEmpty:*
    - Determines whether an item exists in the ADT
  - *enqueue:*
    - Inserts a new item in the Back position
  - *dequeue:*
    - Deletes an item from the Front position
  - *peek:*
    - Retrieves the item from the Front position

Queue and Stack

- Stacks and queues are very similar
- Operations of stacks and queues can be paired off as
  - *createStack* and *createQueue*
  - Stack *isEmpty* and queue *isEmpty*
  - *push* and *enqueue*
  - *pop* and *dequeue*
  - Stack *getTop* and queue *getFront*

Summary and Conclusion

- Queue is a data structure that implement FOFO concept (First In First OUT).
- Queue can be implemented using array or linked list.
  - Queue linear array has rightward drift problem and can be solved using circular array implementation.
  - Queue linked list can be implemented linearly or circular. The advantage is the number of nodes are not limited to the queue size and can be created dynmically.
-

References

- Nor Bahiah et al. “*Struktur data & algoritma menggunakan C++”. Penerbit UTM. 2005.*
- Frank M. Carano, Janet J Prichard. “*Data Abstraction and problem solving with C++” Walls and Mirrors*. 5th edition (2007). Addision Wesley.