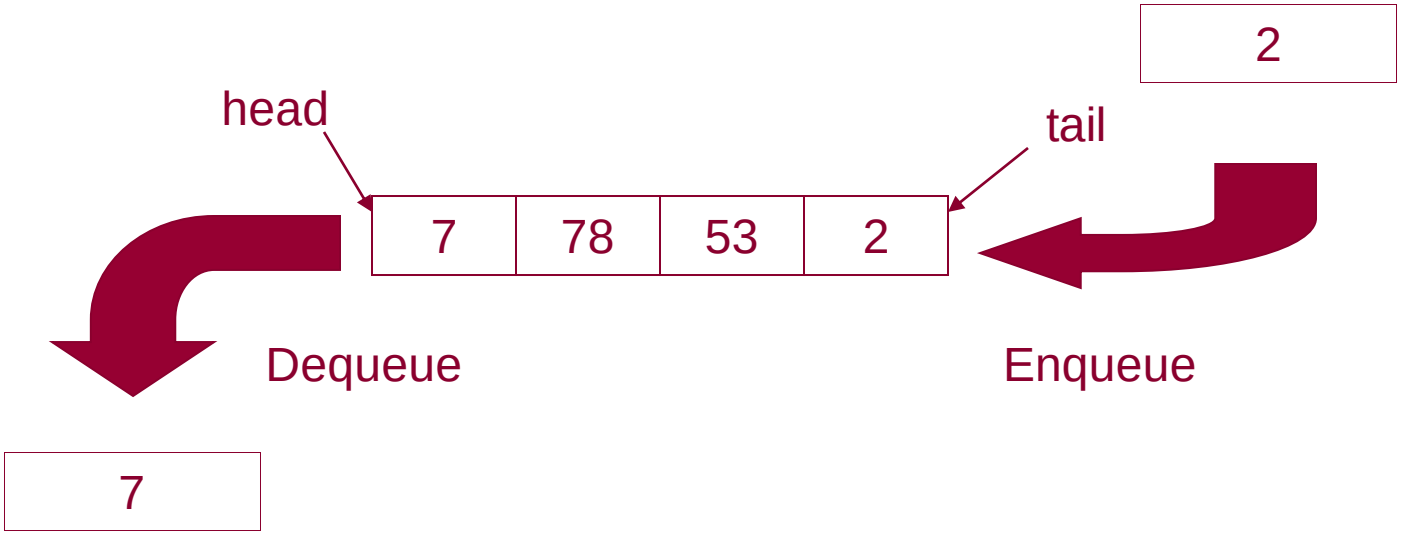
(6-2) Basics of a Queue

Instructor - Andrew S. O'Fallon
CptS 122 (February 14, 2024)
Washington State University

What is a Queue?

- A linear data structure with a finite sequence of nodes, where nodes are removed from the front or head and nodes are inserted at the back or tail
- A queue is referred to as a first-in, first-out (FIFO) data structure
  - Consider a grocery store line; as the line forms, people enter at the back or tail of the line; the person at the front or head of the line is always serviced before the others; once the front person is serviced, he/she leaves and the next in line is helped
- A queue is also considered a restricted or constrained list
- We will focus most of our attention on linked list implementations of queues

Typical Representation of Queue of Integers



Implementation of Queues in C

- The following slides will show how to implement Queues in C
- We will implement them in C++ during lecture

Struct QueueNode

- For these examples, we'll use the following definition for `QueueNode`:

```
typedef struct queueNode
{
    char data;
    // self-referential
    struct queueNode *pNext;
} QueueNode;
```


Initializing a Queue in C (1)

- Our implementation:

```
void initQueue (QueueNode **pHead,
                QueueNode **pTail)
{
    // Recall: we must dereference a
    // pointer to retain changes
    *pHead = NULL; // Points to front of queue
    *pTail = NULL; // Points to back of queue
}
```


Initializing a Queue in C (2)

- The `initQueue()` function is elementary and is not always implemented
- We may instead initialize the pointers to the front and back of the queue with `NULL` within `main()`

```
int main (void)
{
    QueueNode *pHead = NULL; // points to front
    QueueNode *pTail = NULL; // points to back
    …
}
```


Initializing a Queue in C (3)

- We can combine the two pointers (pHead and pTail) of a queue into a single struct called `Queue`

```
typedef struct queue
{
    QueueNode *pHead;
    QueueNode *pTail;
} Queue;
```

- We can then modify our `initQueue()` to accept a `Queue` struct type

```
void initQueue (Queue *pQueue)
{
    pQueue -> pHead = NULL;
    pQueue -> pTail = NULL;
}
```


Checking for Empty Queue in C (1)

- Only need to check the head pointer to see if the queue is empty
- Our implementation:

```
int isEmpty (Queue q)
{
    // Condensed the code into
    // one statement; returns 1 if
    // pHead is NULL; 0 otherwise
    return (q.pHead == NULL);
}
```


Checking for Empty Queue in C (2)

- Note: we could substitute the `int` return type with an enumerated type such as Boolean

```
typedef enum boolean
{
   FALSE, TRUE
} Boolean;
```


Checking for Empty Queue in C (3)

- Our implementation with `Boolean` defined:

```
Boolean isEmpty (Queue q)
{
    Boolean status = FALSE;

    if (q.pHead == NULL) // Queue is empty
    {
        status = TRUE;
    }

    return status;
}
```


Printing Data in Queue in C

- A possible implementation using recursion:

```
void printQueueRecursive (QueueNode *pHead)
{
     if (pHead != NULL) // Recursive step
     {
            printf ("%c ->\n", (pHead) -> data);
            // Get to the next item
            pHead = (pHead) -> pNext;
            printQueueRecursive (pHead);
     }
     else // Base case
     {
            printf ("NULL\n");
     }
}
```


Inserting Data into Back of Queue with Error Checking in C (1)

- Let's modify our code so that we can check for dynamic memory allocation errors
- We'll start with `makeNode()`:

```
QueueNode * makeNode (char newData)
{
      QueueNode *pMem = NULL;

      pMem = (QueueNode *) malloc (sizeof (QueueNode));
      if (pMem != NULL)
      {
               // Initialize the dynamic memory
               pMem -> data = newData;
               pMem -> pNext = NULL;
      }
      // Otherwise no memory is available; could use else, but
      // it's not necessary

      return pMem;
}
```


Inserting Data into Back of Queue with Error Checking in C (2)

- Now let's add some error checking to `enqueue()`:

```
Boolean enqueue (Queue *pQueue, char newData)
{
        QueueNode *pMem = NULL;
        Boolean status = FALSE; // Assume can't insert a new node; out of memory

        pMem = makeNode (newData);

        if (pMem != NULL) // Memory was available
        {
                // Insert the new node into the back of the queue
                if (isEmpty (*pQueue)) // Inserting first node into queue
                {
                        pQueue -> pHead = pMem;
                }
                else // Already at least one node in queue; update tail only
                {
                        pQueue -> pTail -> pNext = pMem;
                }
                pQueue -> pTail = pMem;
                status = TRUE; // Successfully added a node to the queue!
        }

        return status;
}
```


Removing Data from Front of Queue in C (1)

- We will apply defensive design practices and ensure the queue is not empty
- This implementation of `dequeue()` returns the data in the node at the front of the queue

```
char dequeue (Queue *pQueue)
{
        char retData = '\0';
        QueueNode *pFront = NULL;

        if (!isEmpty (*pQueue)) // Stack is not empty; defensive design
        {
                pFront = pQueue -> pHead; // Temp storage of front of queue
                retData = pQueue -> pHead -> data;
                pQueue -> pHead = pQueue -> pHead -> pNext;
                if (pQueue -> pHead == NULL) // Queue is now empty; update tail
                {
                        pQueue -> pTail = NULL;
                }
                free (pFront); // Remove the front node
        }

        return retData;
}
```


Queue Applications

- Operating systems maintain queues of processes that are ready to execute
- Printers queue print requests; first-come, first-serve
- Simulations of real world processes, such as movie lines, grocery store lines, etc.

Closing Thoughts

- Can you build a driver program to test these functions?
- A queue is essentially a restricted linked list, where one additional pointer is needed to keep track of the back, tail, or rear of the queue
- You can implement a queue without using links; Hence, you can use an array as the underlying structure for the queue

References

- P.J. Deitel & H.M. Deitel, *C: How to Program* (7th ed.), Prentice Hall, 2013
- J.R. Hanly & E.B. Koffman, *Problem Solving and Program Design in C (7th Ed.)*, Addison-Wesley, 2013

Collaborators

- Jack Hagemeister