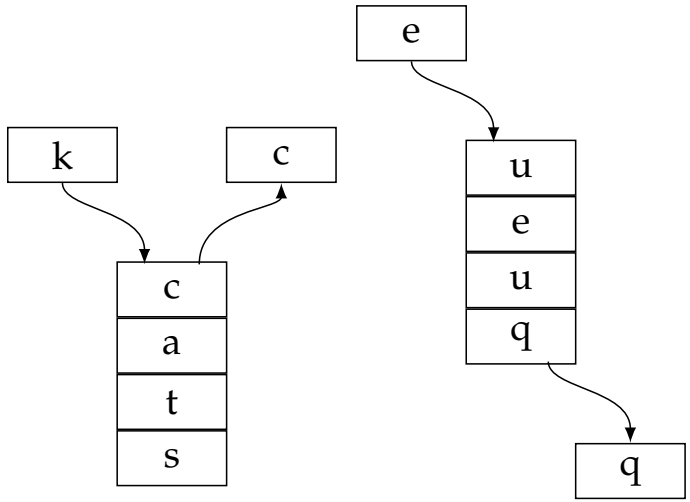
CSCI 2270: Data Structures

Lecture 11: Stacks and Queues

Ashutosh Trivedi

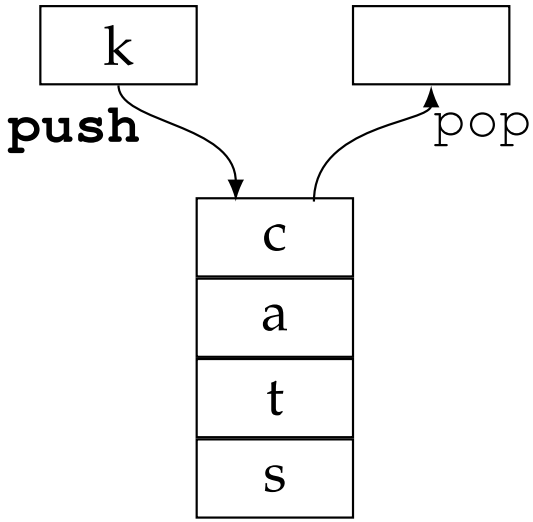


Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

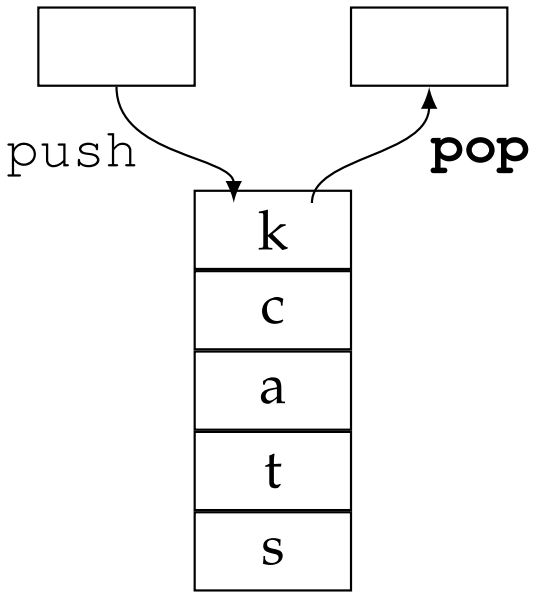
Introduction to Stacks

Introduction to Queues

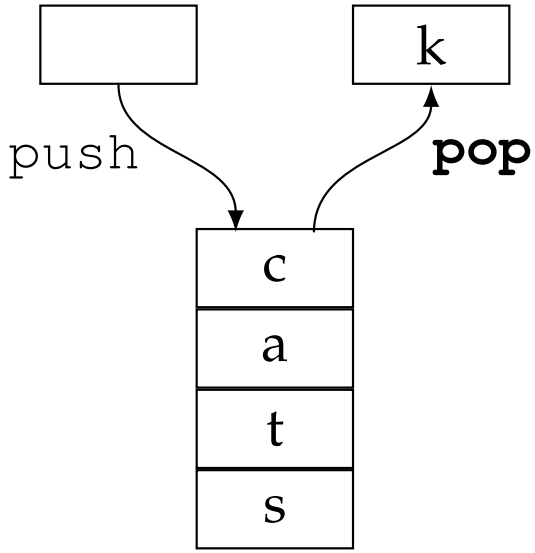
Stacks



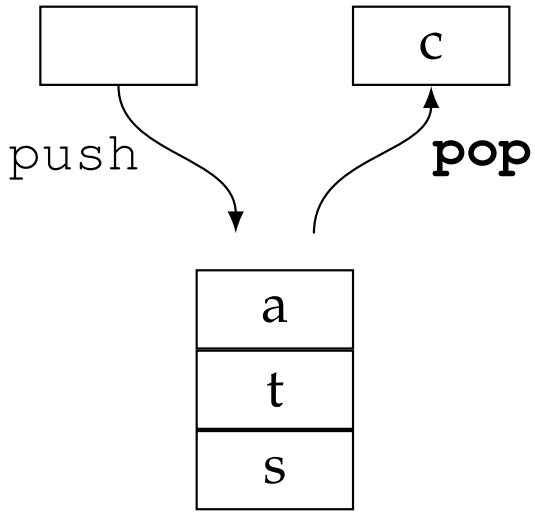
Stacks



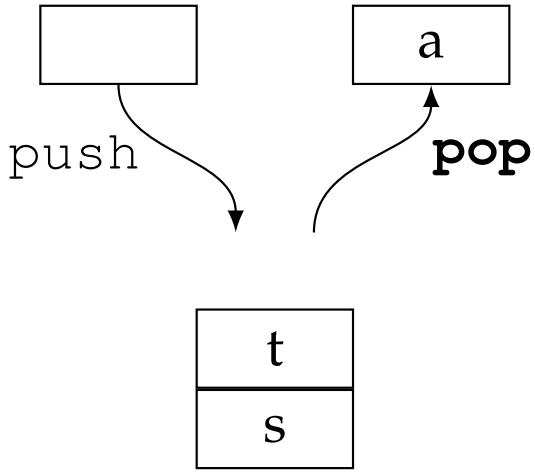
Stacks



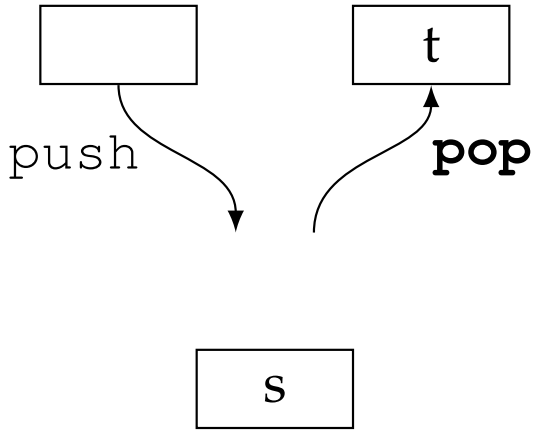
Stacks



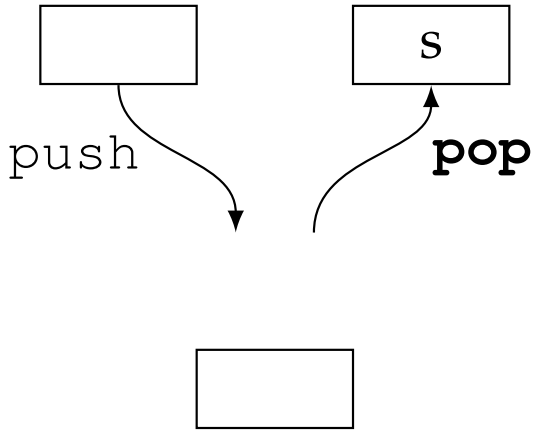
Stacks



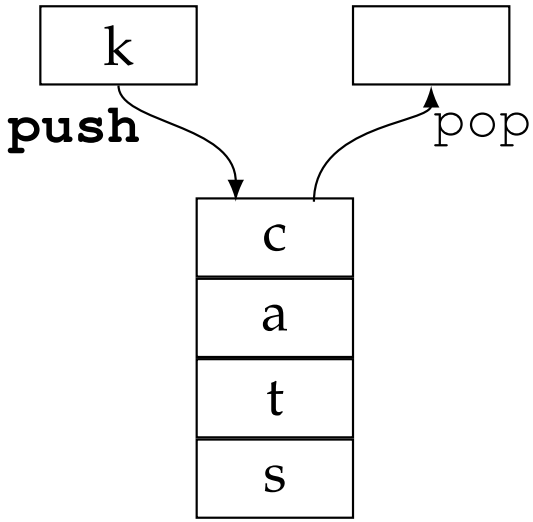
Stacks



Stacks

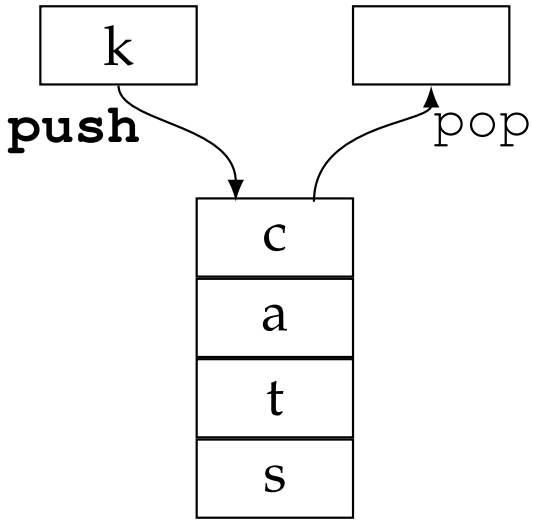


Stack: Push and Pop



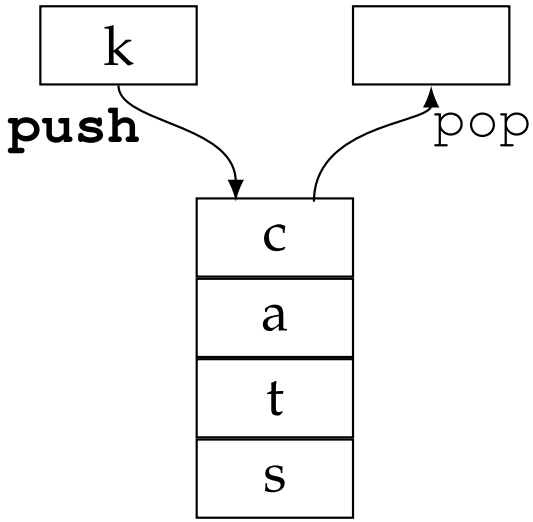
- A *stack* is a list with the last-in first-out (LIFO) structure

Stack: Push and Pop



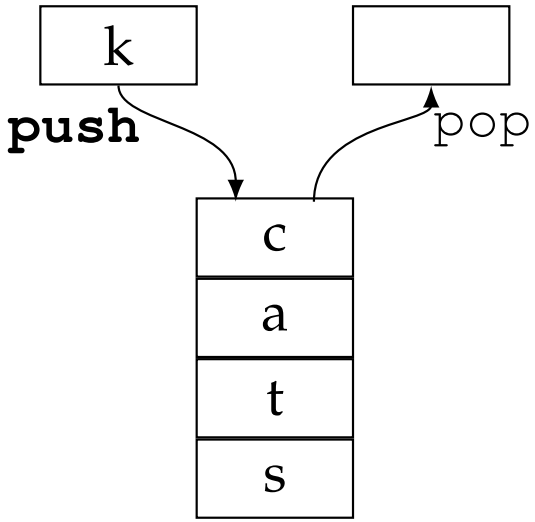
- A *stack* is a list with the last-in first-out (LIFO) structure
- When an element is added to a stack, it is "pushed" on to stack.

Stack: Push and Pop



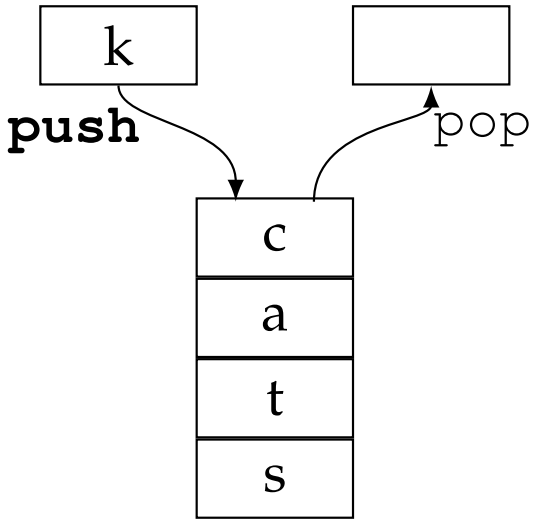
- A *stack* is a list with the last-in first-out (LIFO) structure
- When an element is added to a stack, it is "pushed" on to stack.
- When an element is removed from a stack, it is "popped" off the stack.

Stack: Push and Pop



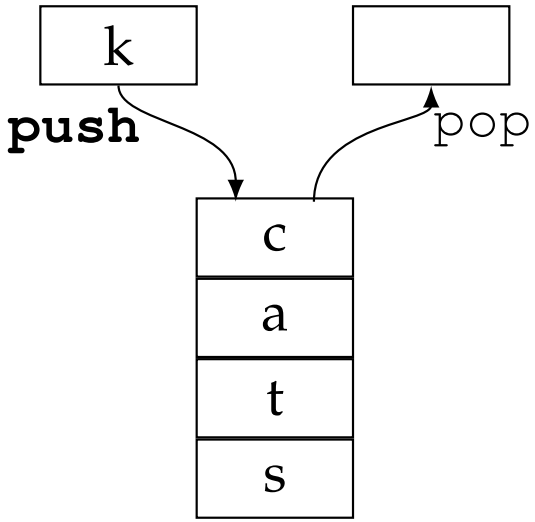
- A *stack* is a list with the last-in first-out (LIFO) structure
- When an element is added to a stack, it is "pushed" on to stack.
- When an element is removed from a stack, it is "popped" off the stack.
- It is common to keep a **maxSize** restriction on the stack.

Stack: Push and Pop



- A *stack* is a list with the last-in first-out (LIFO) structure
- When an element is added to a stack, it is "pushed" on to stack.
- When an element is removed from a stack, it is "popped" off the stack.
- It is common to keep a **maxSize** restriction on the stack.
- Attempting to push on a stack with elements equal to maxSize results in **stack overflow!**

Stack: Push and Pop



- A *stack* is a list with the last-in first-out (LIFO) structure
- When an element is added to a stack, it is "pushed" on to stack.
- When an element is removed from a stack, it is "popped" off the stack.
- It is common to keep a **maxSize** restriction on the stack.
- Attempting to push on a stack with elements equal to maxSize results in **stack overflow!**
- Attempting to pop an empty stack results in **stack underflow**!

Stack: ADT

```
struct Node {
  int data;
  Node* prev;

  Node() {          /* Default Constructor */
    data = -1;
    prev = 0;
  }
  Node(int data_){  /* Fills data field  */
    data = data_;
    prev = 0;
  }
  Node(int data_, Node* prev_){ /* Fills both fields */
    data = data_;
    prev = prev_;
  }
};

class Stack {
private:
  Node* top;

  int maxSize;
  int size;

public:
  Stack();       /* Constructor */
  Stack(int maxSize_);       /* Constructor */
  ~Stack();     /* Destructor */

  bool isFull();
  bool isEmpty();
  void push(int value);
  int pop();
  void show();
};
```

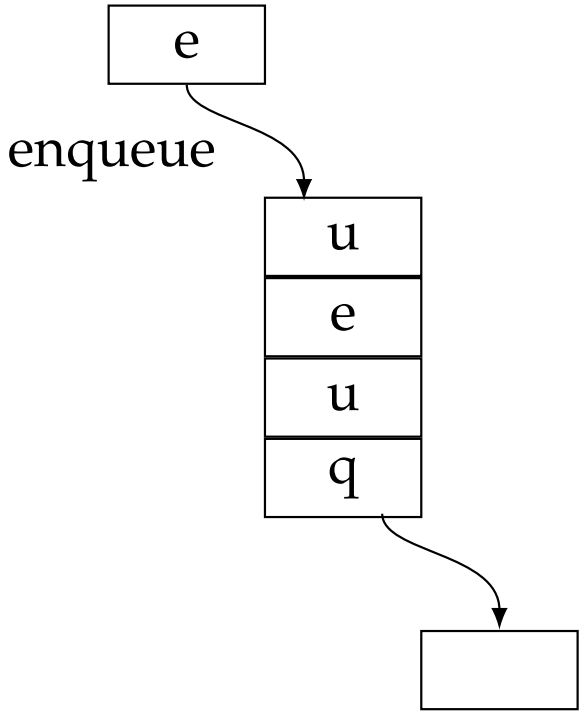
Discussions: Implementation

- Array versus LinkedList implementations
- Number of steps required for push and pop in array implementation.
- Number of steps required for push and pop in queue implementation.

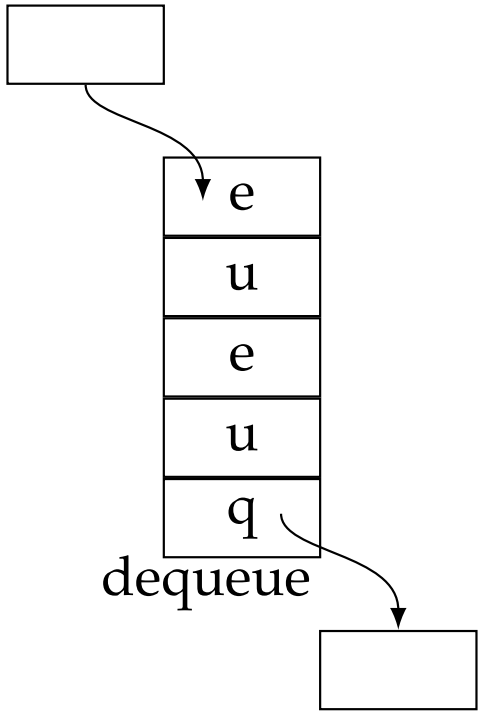
Introduction to Stacks

Introduction to Queues

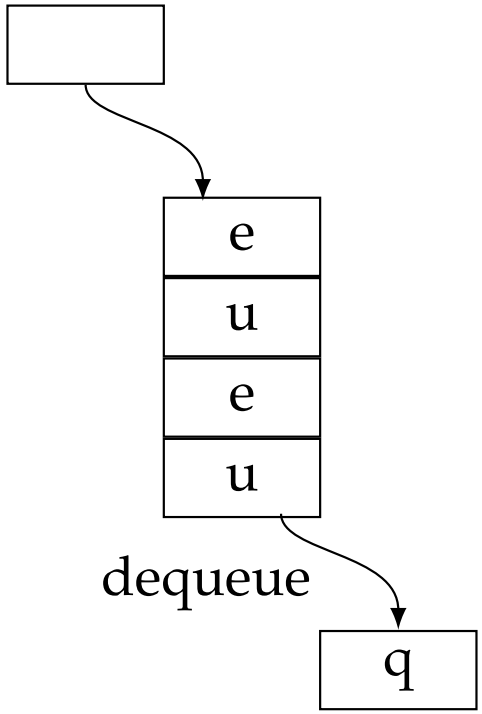
Queues



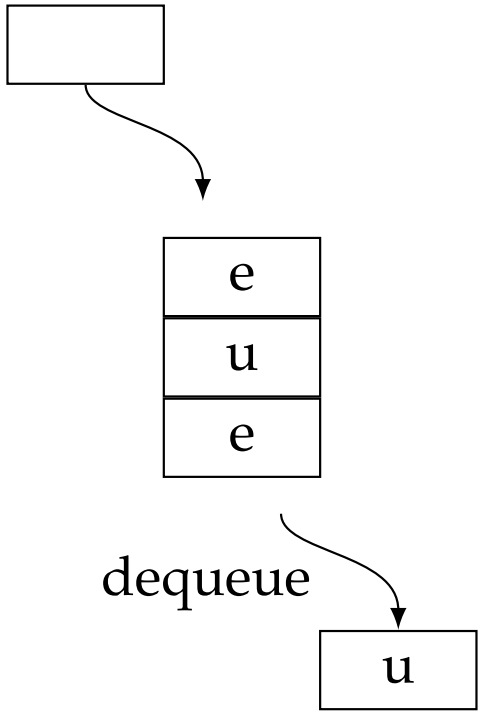
Queues



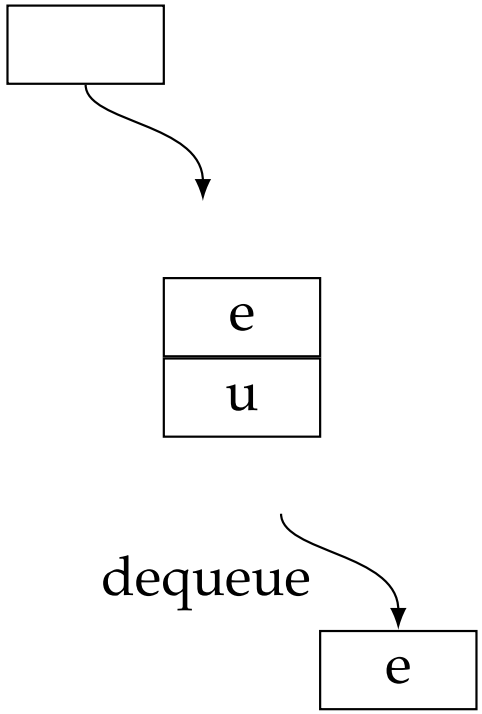
Queues



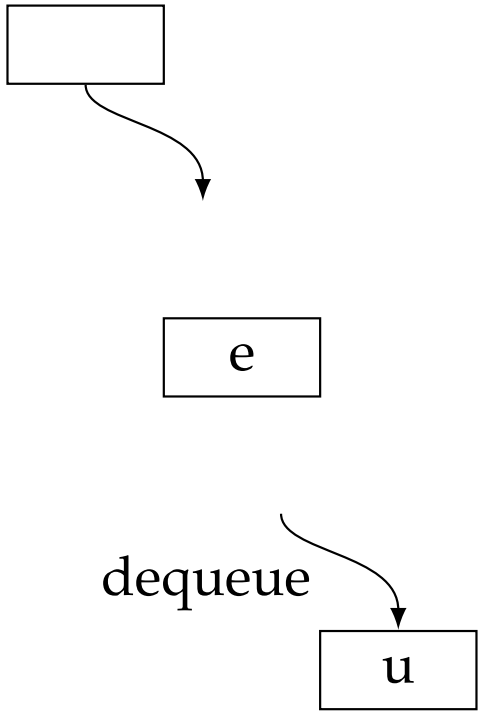
Queues



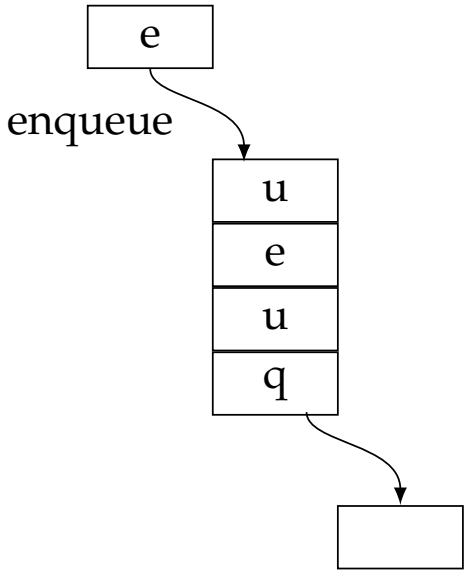
Queues



Queues

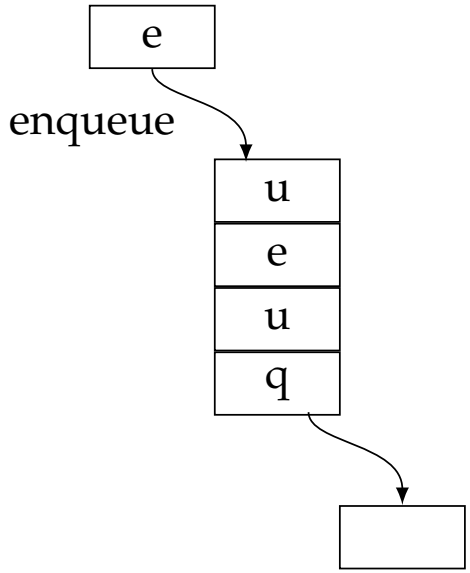


Queue: Enqueue and Dequeue



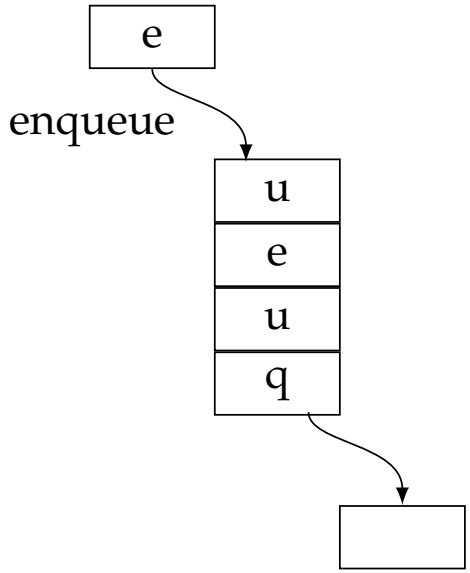
– A *queue* is a list with the first-in first-out (FIFO) structure

Queue: Enqueue and Dequeue



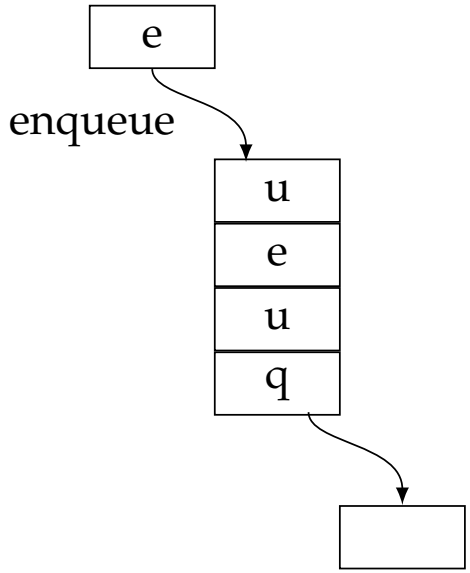
- A *queue* is a list with the first-in first-out (FIFO) structure
- When an element is added to a queue, it is "enqueued" to the "tail" of to the queue.

Queue: Enqueue and Dequeue



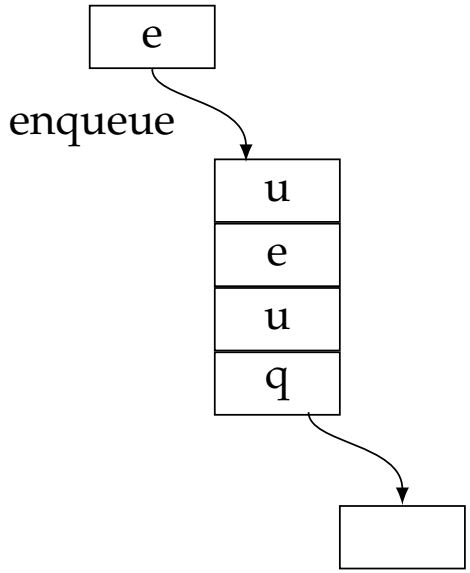
- A *queue* is a list with the first-in first-out (FIFO) structure
- When an element is added to a queue, it is "enqueued" to the "tail" of to the queue.
- When an element is removed from a queue, it is "dequeued" off the "head" of the queue.

Queue: Enqueue and Dequeue



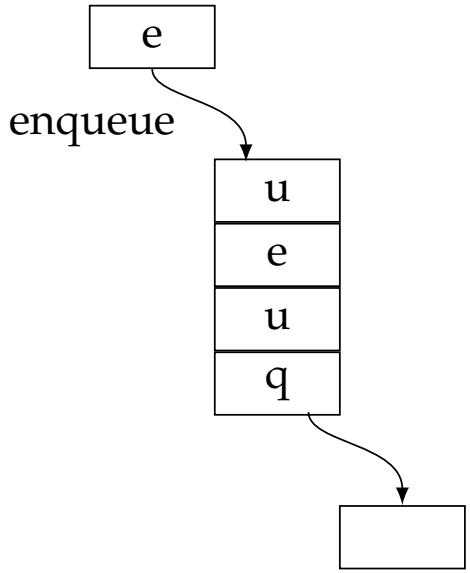
- A *queue* is a list with the first-in first-out (FIFO) structure
- When an element is added to a queue, it is "enqueued" to the "tail" of to the queue.
- When an element is removed from a queue, it is "dequeued" off the "head" of the queue.
- It is common to keep a **maxSize** restriction on the queues.

Queue: Enqueue and Dequeue



- A *queue* is a list with the first-in first-out (FIFO) structure
- When an element is added to a queue, it is "enqueued" to the "tail" of to the queue.
- When an element is removed from a queue, it is "dequeued" off the "head" of the queue.
- It is common to keep a **maxSize** restriction on the queues.
- Attempting to enqueue to a queue with elements equal to maxSize results in **queue full!**

Queue: Enqueue and Dequeue



- A *queue* is a list with the first-in first-out (FIFO) structure
- When an element is added to a queue, it is "enqueued" to the "tail" of to the queue.
- When an element is removed from a queue, it is "dequeued" off the "head" of the queue.
- It is common to keep a **maxSize** restriction on the queues.
- Attempting to enqueue to a queue with elements equal to maxSize results in **queue full!**
- Attempting to dequeue an empty queue results in **queue empty**!

Queue: ADT

```
struct Node {
  int data;
  Node* next;

  Node() {          /* Default Constructor */
    data = -1;
    next = 0;
  }
  Node(int data_){  /* Fills data field  */
    data = data_;
    next = 0;
  }
  Node(int data_, Node* next_){ /* Fills both fields */
    data = data_;
    next = next_;
  }
};

class Queue {
private:
  Node* head;
  Node* tail;

  int maxSize;
  int size;

public:
  Queue();       /* Constructor */
  Queue(int maxSize_);       /* Constructor */
  ~Queue();     /* Destructor */

  bool isFull();
  bool isEmpty();
  void enqueue(int value);
  int dequeue();
  void show();
};
```

Discussions: Implementation

- Array versus LinkedList implementations
- Number of steps required for enqueue and dequeue in an array implementation.
- Number of steps required for enqueue and dequeue in linked list implementation.