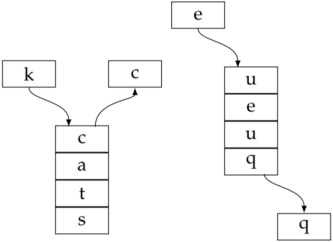
CSCI 2270: Data Structures

Lecture 13: Stacks and Queues: Implementations

Ashutosh Trivedi



Department of Computer Science
UNIVERSITY OF COLORADO BOULDER

Stacks

Array Implementation

Linked-List Implementation

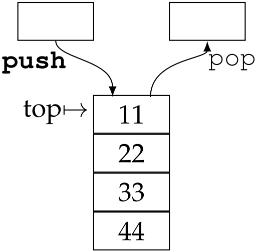
Queues

Linked-List Implementation

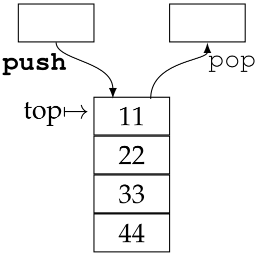
Array Implementation

Circular Array Implementation

Stack implemented as an Array



Stack implemented as an Array



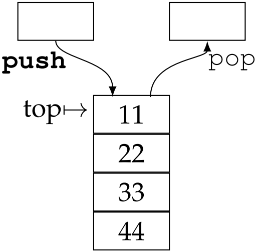
Option 1.

0	1	2	3	4	5	6	7
44	33	22	11	*B*	*B*	*B*	*B*

Option 2.

0	1	2	3	4	5	6	7
11	22	33	44	*B*	*B*	*B*	*B*

Stack implemented as an Array



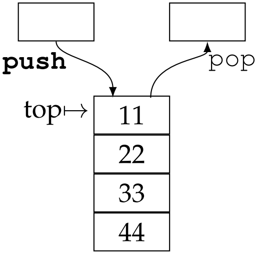
Option 1. push: $O(1)$ **and pop:** $O(1)$

0	1	2	3	4	5	6	7
44	33	22	11	B	B	B	B

Option 2. push: $O(n)$ **and pop:** $O(n)$

0	1	2	3	4	5	6	7
11	22	33	44	B	B	B	B

Stack implemented as an Array



Option 1. push: $O(1)$ **and pop:** $O(1)$

0	1	2	3	4	5	6	7
44	33	22	11	B	B	B	B

Option 2. push: $O(n)$ **and pop:** $O(n)$

0	1	2	3	4	5	6	7
11	22	33	44	B	B	B	B

ArrayStack ADT

```
#pragma once

#define DEFAULT_SIZE 5

class ArrayStack {
private:
  int capacity;
  int top;
  int* items;

public:
  ArrayStack();  /* Constructor with capacity DEFAULT_SIZE */
  ArrayStack(int cap); /* Constructor with capacity */
  ˜ArrayStack(); /* Destructor */

  bool isEmpty(); /* True, if stack is empty */
  bool isFull();  /* True, if stack is full  */
  void push(int element); /* Push an element to the stack */
  int pop(); /* Pop an element from the stack */
  int peek(); /* Return the top element of the stack */
  void prettyPrint(); /* print the stack */
};
```

ArrayStack: Push

```
void ArrayStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    top = top + 1;
    items[top] = element;
  }
}
```

Push “55” to the stack.

0	1	2	top	4	5	6	7
44	33	22	11	B	B	B	B

ArrayStack: Push

```
void ArrayStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    top = top + 1;
    items[top] = element;
  }
}
```

Push “55” to the stack.

0	1	2	3	top	5	6	7
44	33	22	11	B	B	B	B

ArrayStack: Push

```
void ArrayStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    top = top + 1;
    items[top] = element;
  }
}
```

Push “55” to the stack.

0	1	2	3	top	5	6	7
44	33	22	11	55	B	B	B

ArrayStack: Pop

```
int ArrayStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[top];
    top = top - 1;
    return result;
  }
}
```

Pop the stack.

0	1	2	3	top	5	6	7
44	33	22	11	55	*B*	*B*	*B*

ArrayStack: Pop

```
int ArrayStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[top];
    top = top - 1;
    return result;
  }
}
```

Pop the stack.

0	1	2	3	top	5	6	7
44	33	22	11	55	B	B	B

ArrayStack: Pop

```
int ArrayStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[top];
    top = top - 1;
    return result;
  }
}
```

Pop the stack.

0	1	2	top	4	5	6	7
44	33	22	11	55	*B*	*B*	*B*

Stacks

Array Implementation

Linked-List Implementation

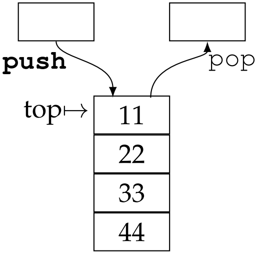
Queues

Linked-List Implementation

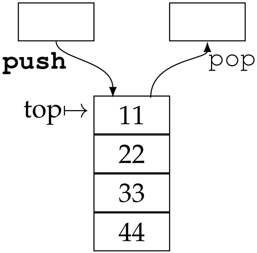
Array Implementation

Circular Array Implementation

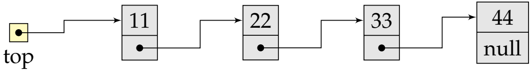
Stack implemented as a Linked List



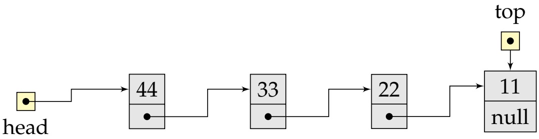
Stack implemented as a Linked List



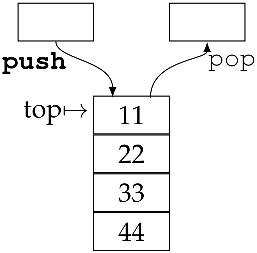
Option 1.



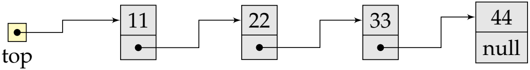
Option 2.



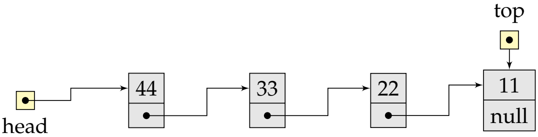
Stack implemented as a Linked List



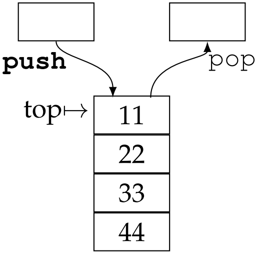
Option 1. push: $O(1)$ **and pop:** $O(1)$



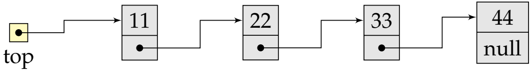
Option 2. push: $O(1)$ **and pop:** $O(n)$



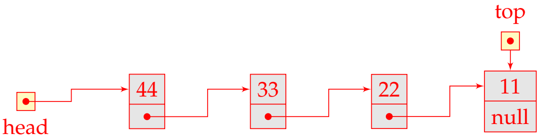
Stack implemented as a Linked List



Option 1. push: $O(1)$ and pop: $O(1)$



Option 2. push: $O(1)$ and pop: $O(n)$



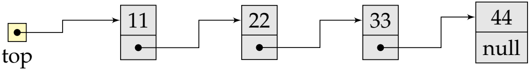
LinkedListStack ADT

```
struct StackNode {
  int item;
  StackNode* next;

  StackNode() {item = -1; next = 0;}
  StackNode(int element) {item = element; next = 0;}
};

class LinkedStack {
private:
  int capacity; /* capacity of the stack */
  StackNode* top; /* pointer to the top node of the stack */
  int size; /* number of elements in the stack */

public:
  LinkedStack();  /* Constructor with capacity DEFAULT_SIZE */
  LinkedStack(int cap); /* Constructor with capacity */
  ~LinkedStack(); /* Destructor */

  bool isEmpty(); /* True, if stack is empty */
  bool isFull();  /* True, if stack is full  */
  void push(int element); /* Push an element to the stack */
  int pop(); /* Pop an element from the stack */
  int peek(); /* Return the top element of the stack */
  void prettyPrint(); /* print the stack */
};
```

LinkedListStack: Push

```
void LinkedStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    StackNode* curr = new StackNode(element);
    curr->next = top;
    top = curr;
    size = size + 1;
  }
}
```

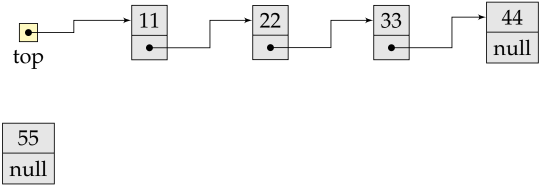
Push “55” to the stack.



LinkedListStack: Push

```
void LinkedStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    StackNode* curr = new StackNode(element);
    curr->next = top;
    top = curr;
    size = size + 1;
  }
}
```

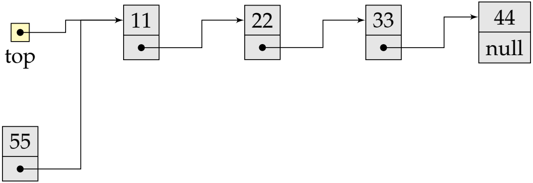
Push “55” to the stack.



LinkedListStack: Push

```
void LinkedStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    StackNode* curr = new StackNode(element);
    curr->next = top;
    top = curr;
    size = size + 1;
  }
}
```

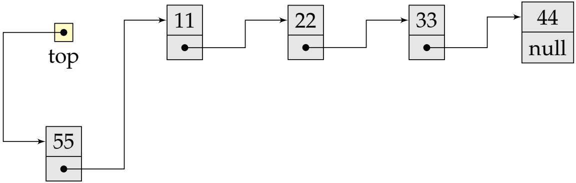
Push “55” to the stack.



LinkedListStack: Push

```
void LinkedStack::push(int element) {
  if (isFull()) {
    std::cerr << "Stack Overflow!! Push failed" << std::endl;
  }
  else {
    StackNode* curr = new StackNode(element);
    curr->next = top;
    top = curr;
    size = size + 1;
  }
}
```

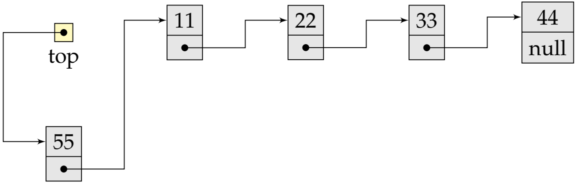
Push “55” to the stack.



LinkedListStack: Pop

```
int LinkedStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = top->item;
    StackNode* tmp = top;

    top = top->next;
    size = size -1;

    delete tmp;
    return result;
  }
}
```

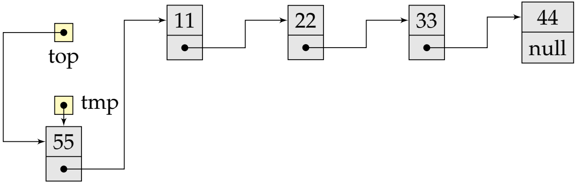
Pop the stack.



LinkedListStack: Pop

```
int LinkedStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = top->item;
    StackNode* tmp = top;

    top = top->next;
    size = size -1;

    delete tmp;
    return result;
  }
}
```

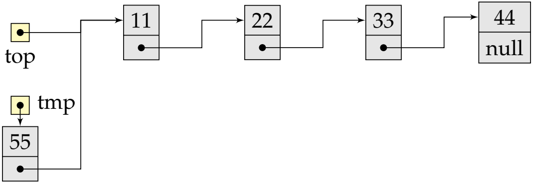
Pop the stack.



LinkedListStack: Pop

```
int LinkedStack::pop() {
  if (isEmpty()) {
    std::cerr << "Stack Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = top->item;
    StackNode* tmp = top;

    top = top->next;
    size = size -1;

    delete tmp;
    return result;
  }
}
```

Pop the stack.



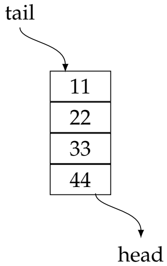
Stacks

- Array Implementation
- Linked-List Implementation

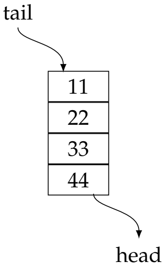
Queues

- Linked-List Implementation
- Array Implementation
- Circular Array Implementation

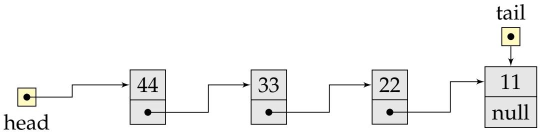
Queue implemented as a Linked List



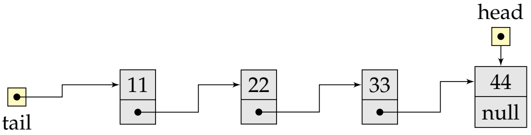
Queue implemented as a Linked List



Option 1.

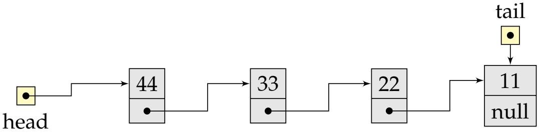


Option 2.

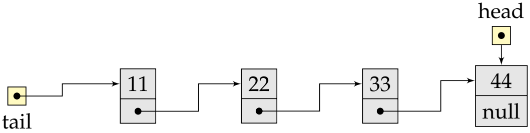


Queue implemented as a Linked List

Option 1. Enqueue: $O(1)$ **and Dequeue** $O(1)$

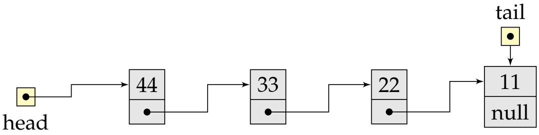


Option 2. Enqueue: $O(1)$ **and Dequeue** $O(n)$

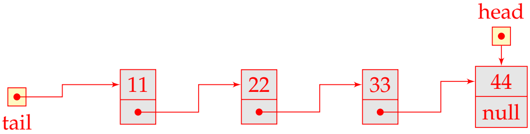


Queue implemented as a Linked List

Option 1. Enqueue: $O(1)$ **and Dequeue** $O(1)$



Option 2. Enqueue: $O(1)$ **and Dequeue** $O(n)$



LinkedListQueue: Enqueue

```
void LinkedQueue::enqueue(int element) {
  if (isFull()) {
    std::cerr << "Queue Overflow!! Enqueue failed" << std::endl;
  }
  else {
    QueueNode* tmp = new QueueNode(element);
    tmp->next = 0;

    if (head == 0) {
      head = tmp;
      tail = tmp;
      size = size + 1;
    }
    else {
      tail->next = tmp;
      tail = tmp;
      size = size + 1;
    }
  }
}
```

LinkedListQueue: Dequeue

```
int LinkedQueue::dequeue() {
  if (isEmpty()) {
    std::cerr << "Queue Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = head->item;
    QueueNode* tmp = head;

    if (tail == head) {
      // there is only one node in the queue
      tail = 0;
      head = 0;
    }
    else {
      head = head->next;
    }

    delete tmp;

    size = size - 1;
    return result;
  }
}
```

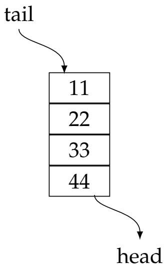
Stacks

- Array Implementation
- Linked-List Implementation

Queues

- Linked-List Implementation
- Array Implementation
- Circular Array Implementation

Queue implemented as an Array



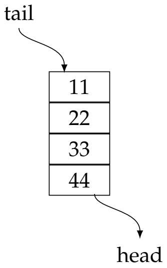
Option 1.

head	1	2	tail	4	5	6	7
44	33	22	11	B	B	B	B

Option 2.

tail	1	2	head	4	5	6	7
11	22	33	44	B	B	B	B

Queue implemented as an Array



Option 1. Enqueue: $O(1)$ **and Dequeue:** $O(n)$

head	1	2	tail	4	5	6	7
44	33	22	11	B	B	B	B

Option 2. Enqueue: $O(n)$ **and Dequeue:** $O(1)$

tail	1	2	head	4	5	6	7
11	22	33	44	B	B	B	B

ArrayQueue: Enqueue

```
void ArrayQueue::enqueue(int element) {
  if (isFull()) {
    std::cerr << "Queue Overflow!! Enqueue failed" << std::endl;
  }
  else {
    if (head == -1) {
      head = 0;
      tail = 0;
      items[tail] = element;

    }
    else {
      tail = tail + 1;
      items[tail] = element;
    }
  }
}
```

ArrayQueue: Dequeue

```
int ArrayQueue::dequeue() {
  if (isEmpty()) {
    std::cerr << "Queue Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[head];

    for (int i = 0; i < tail; i++) {
      items[i] = items[i+1];
    }
    tail = tail-1;

    if (tail == -1) head = -1;

    return result;
  }
}
```

Stacks

- Array Implementation
- Linked-List Implementation

Queues

- Linked-List Implementation
- Array Implementation
- Circular Array Implementation

Queue implemented as a Circular Array

head	1	2	tail	4	5	6	7
44	33	22	11	*B*	*B*	*B*	*B*

- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array

head	1	2	3	tail	5	6	7
44	33	22	11	55	B	B	B

- **enqueue(55);**
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array

0	head	2	3	tail	5	6	7
B	33	22	11	55	B	B	B

- enqueue(55);
- **dequeue();**
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array

0	head	2	3	4	5	6	tail
B	33	22	11	55	66	77	88

- enqueue(55);
- dequeue();
- **enqueue(66); enqueue(77); enqueue(88);**
- dequeue();
- enqueue(99);

Queue implemented as a Circular Array

0	1	head	3	4	5	6	tail
B	*B*	22	11	55	66	77	88

- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- **dequeue();**
- enqueue(99);

Queue implemented as a Circular Array

head	1	head	3	4	5	6	7
99	*B*	22	11	55	66	77	88

- enqueue(55);
- dequeue();
- enqueue(66); enqueue(77); enqueue(88);
- dequeue();
- **enqueue(99);**

CircularArrayQueue: Enqueue

```
void CircularArrayQueue::enqueue(int element) {
  if (isFull()) {
    std::cerr << "Queue Overflow!! Enqueue failed" << std::endl;
  }
  else {
    if (head == -1) {
      //first element to insert
      head = 0;
      tail = 0;
      items[tail] = element;
    }
    else {
      if (tail == capacity-1) {
  items[0] = element;
  tail = 0;
      }
      else {
  tail = tail + 1;
  items[tail] = element;
      }
    }
  }
}
```

CircularArrayQueue: Dequeue

```
int CircularArrayQueue::dequeue() {
  if (isEmpty()) {
    std::cerr << "Queue Empty!! Returning garbarge" << std::endl;
    return -1;
  }
  else {
    int result = items[head];

    if (head == tail) {
      // Only one element in the queue
      head = -1;
      tail = -1;
    }
    else {
      if (head == capacity -1) {
  head = 0;
      }
      else {
  head = head + 1;
      }
    }
    return result;
  }
}
```