STACKS AND QUEUES

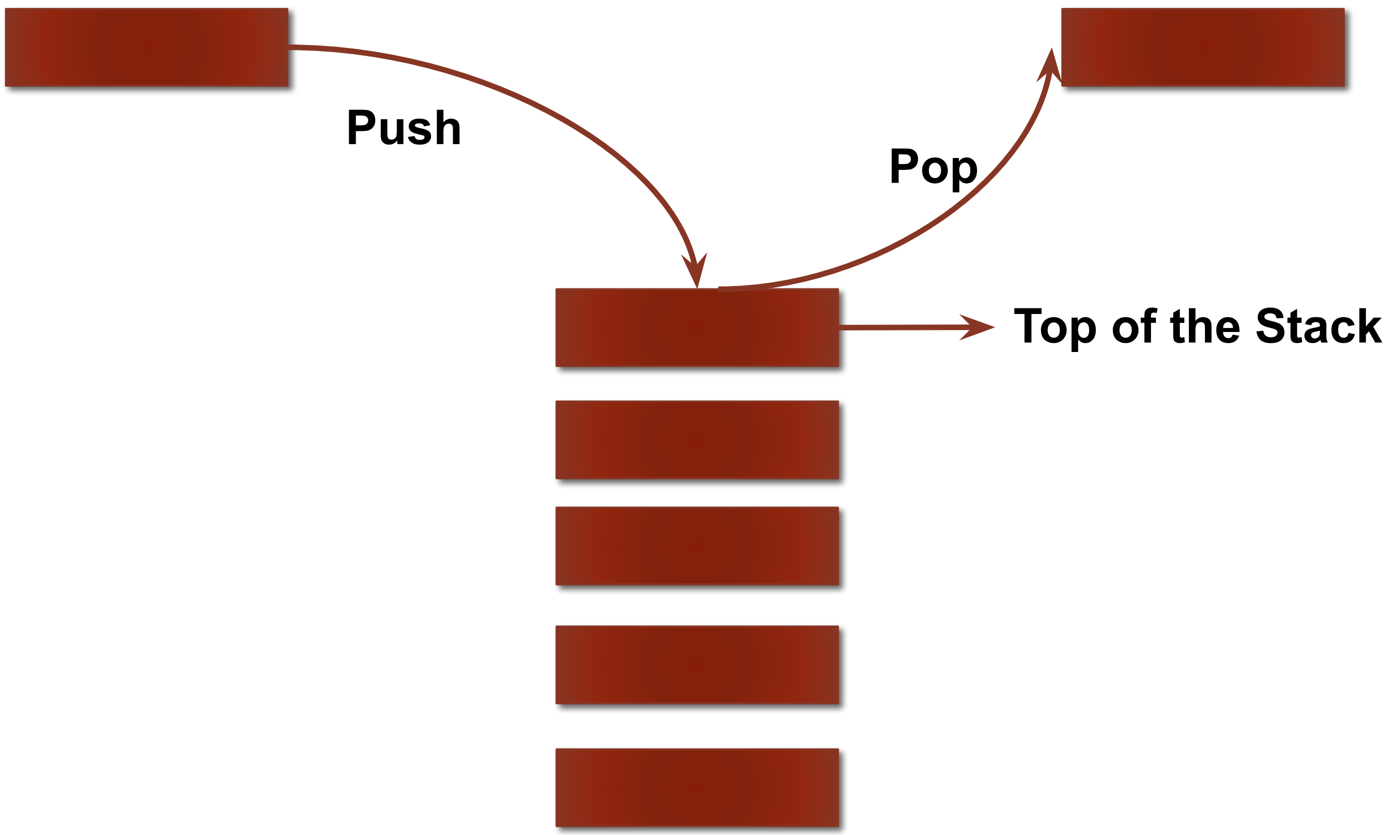
CMP-410-3: Data Structures and Algorithms, Fall 2016
Waheed Iqbal

Punjab University College of Information Technology (PUCIT)
University of the Punjab, Lahore, Pakistan.

Stack

- Stack is a data structure that allows access to items in a last in first out (LIFO) style
- Main Stack operation:
    - push(object): insert an element to the stack
    - pop(): return the last inserted element and remove it
- Auxiliary stack operations:
    - top() / peek(): return the element on top of the stack (last inserted element)
    - size(): return the number of elements stored
    - isEmpty(): return a boolean value indicating elements are store or not in the stack

Stack (Cont.)

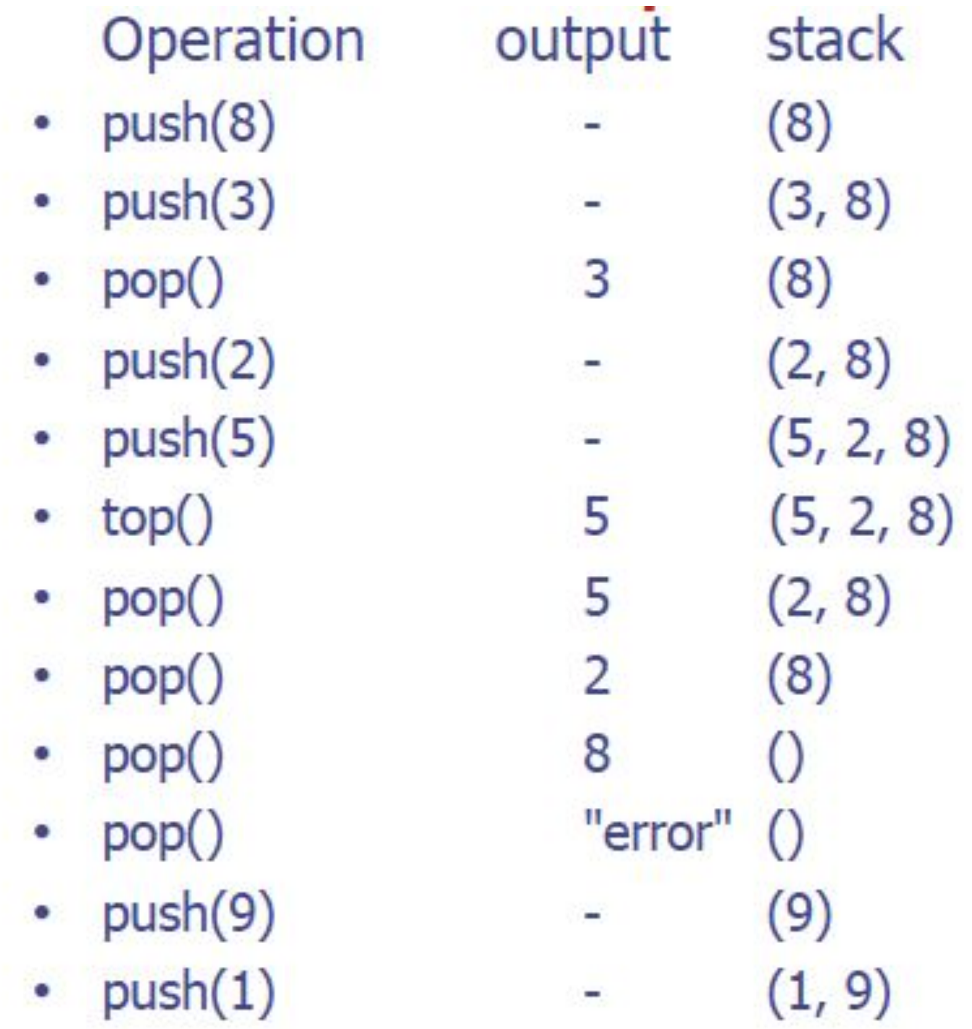


Push

Pop

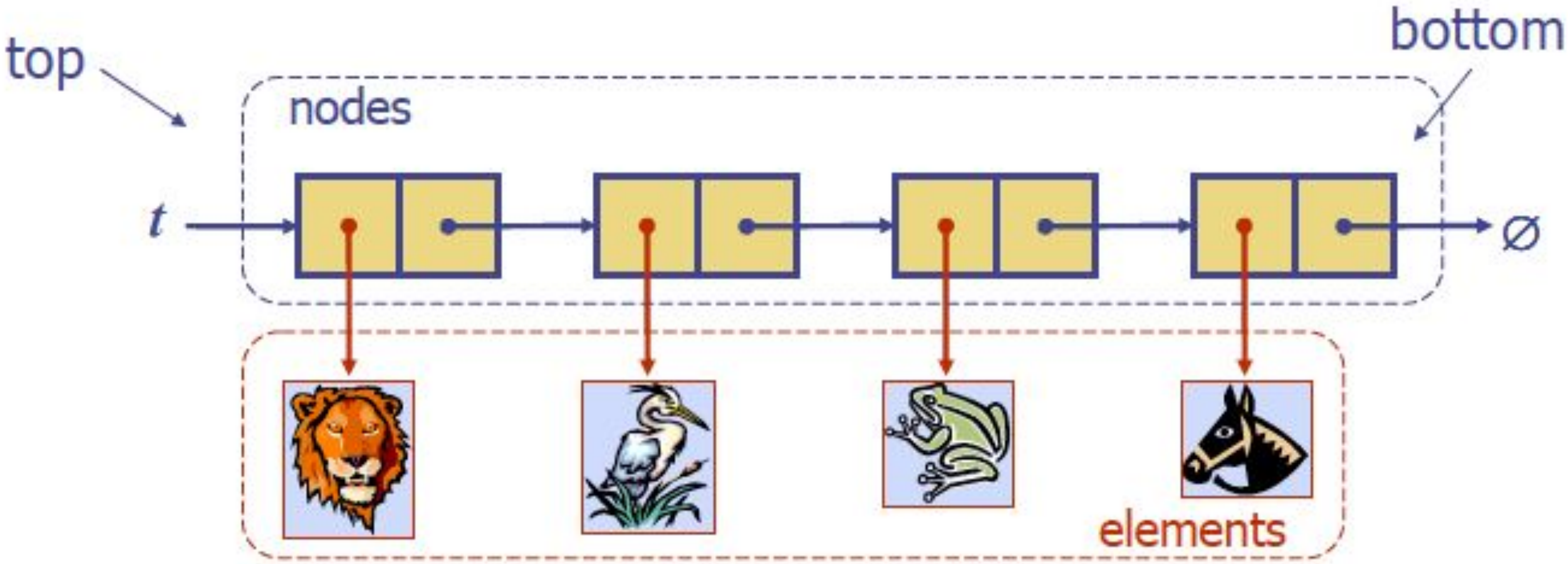
Top of the Stack

Stack Example



Operation	output	stack
• push(8)	-	(8)
• push(3)	-	(3, 8)
• pop()	3	(8)
• push(2)	-	(2, 8)
• push(5)	-	(5, 2, 8)
• top()	5	(5, 2, 8)
• pop()	5	(2, 8)
• pop()	2	(8)
• pop()	8	()
• pop()	"error"	()
• push(9)	-	(9)
• push(1)	-	(1, 9)

Stack Implementation Using Linked List



Stack Implementation Using Linked List

Consider the following classes:

```
class MyStack
{
public:
        MyStack();
        void push(int element);
        int pop();
        bool isEmpty();
        void display();
private:
           //some variable you may need …
};
```

```
class node
{
public:
    int data;
    node* next;
};
```

Lets try to implement these methods!

```
private:
    static const int SIZE = 100; // Maximum
stack size
    int arr[SIZE];              // Array to hold
stack elements
    int top;                    // Index of the top
element

// Constructor
MyStack::MyStack()
{
    top = -1; // Stack is empty
}
bool MyStack::isEmpty()
{
    return top == -1;
}

int MyStack::pop()
{
    if (isEmpty())
    {
        cout << "Stack Underflow!" <<
endl;
        return -1; // or throw exception
    }
    return arr[top--];
}
```

```
void MyStack::push(int element)
{
    if (top >= SIZE - 1)
    {
        cout << "Stack Overflow!" <<
endl;
        return;
    }
    arr[++top] = element;
}


void MyStack::display()
{
    if (isEmpty())
    {
        cout << "Stack is empty." <<
endl;
        return;
    }

    cout << "Stack elements (top to
bottom): ";
    for (int i = top; i >= 0; i--)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

Applications of Stack

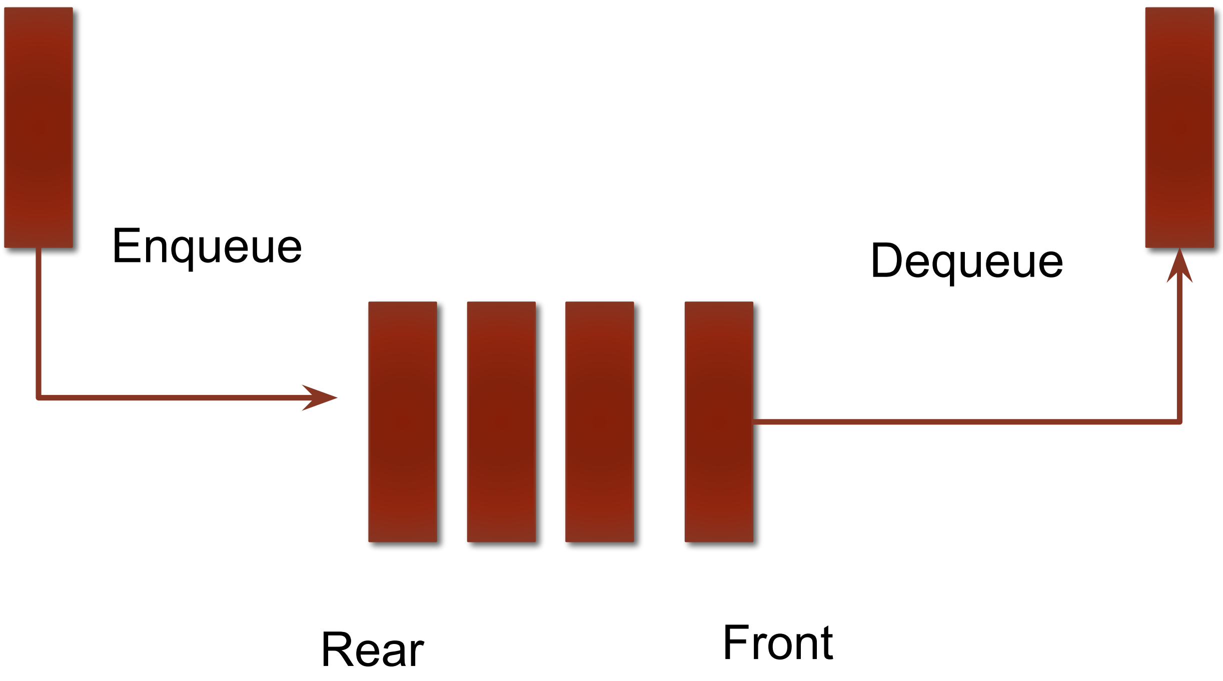
- Reversing data
- Detecting unmatched parentheses
- Page-visited history in a Web browser
- Undo sequence in a text editor
- Implementing recursion

Many other you may need to explore!

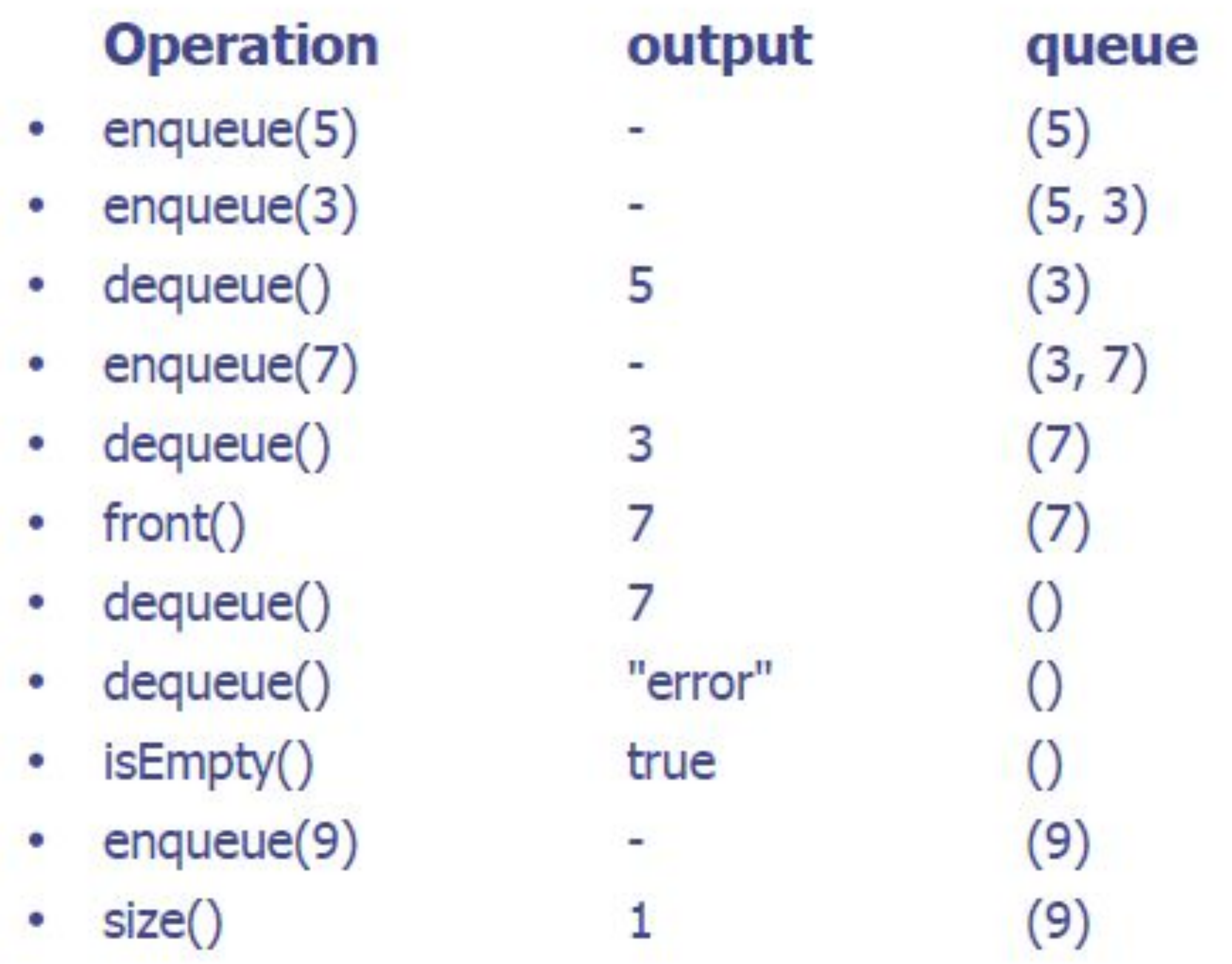
Queue

- Queue is a data structure that allows access to items in a first in, first out style (FIFO)
- Main Operations:
  - **enqueue (item)**: add to the queue)
  - **dequeue ():** remove the *oldest* item in the queue
- Auxiliary Operations:
  - **front()**: returns the element at the front without removing it
  - **size():** returns the number of elements stored
  - **isEmpty():** returns a Boolean value indicating whether no elements are stored

Queue (Cont.)



Queue Example



Operation	output	queue
• enqueue(5)	-	(5)
• enqueue(3)	-	(5, 3)
• dequeue()	5	(3)
• enqueue(7)	-	(3, 7)
• dequeue()	3	(7)
• front()	7	(7)
• dequeue()	7	()
• dequeue()	"error"	()
• isEmpty()	true	()
• enqueue(9)	-	(9)
• size()	1	(9)

Application of Queue

- Waiting lists e.g., customer checkout on a point of sale counter
- Access to shared resources e.g., printer

Queue Implementation Using Array

Palindromes

Palindromes are words which can be read same from forward and revers. Few examples are:

- Radar
- Mom
- Dad
- Stats
- Madam
- Wassamassaw

How we may use Stack and Queue to determine a given word is palindrome?

Palindrome simple recursive implementation

```
bool is_palindrome (int start, int end, string str)
{
  if (start >= end)
      return true;
  if (str[start] != str[end])
      return false;
   start++;
   end--;
  return is_pal(start, end, str);
}
```

Palindrome another recursive implementation

```
bool is_palindrome(string word)
{
     int length = word.length();
    string first = word.substr(0,1);
    string last = word.substr((length - 1), 1);
    if (first == last)
    {
        word = word.substr((0 + 1), (length - 2));
        if (word.length() <= 1)  return true;
         return palindrome(word);
     }
      else return false;
}
```

Palindromes

How we may use Stack and Queue to determine a given word is palindrome?

Credit

Some of the slides are adopted from official material of the book:

- Data Structures and Algorithms in C++ Goodrich, Tamassia and Mount (Wiley, 2004)

Palindrome implementation is taken from:

- http://stackoverflow.com/questions/22890946/finding-a-string-palindrome-with-a-recursive-function

- http://stackoverflow.com/questions/21298797/c-algorithmically-simple-recursive-palindrome-checker