Queues and Stacks

Sandra Batista, Mark Redekopp, and David Kempe

1.1–1.2

Ordered collection of items,

- Each item has an index and there is a front and back (start and end)
- Duplicates allowed (i.e. in a list of integers, the value 0 could appear multiple times)
- Accessed based on their position (list[0], list[1], etc.)

What are some operations you perform on a list?

List Operations

Operation	Description	Input(s)	Output(s)
insert	Add a new value at a particular location shifting others back	Index	
remove	Remove value at the given location	Index	Value at location
get	Get value at given location	Index	Value at location
set	Changes the value at a given location	Index & Value	
empty	Returns true if there are no values in the list		bool
size	Returns the number of values in the list		Size_t
push_back	Add a new value to the end of the list	Value	
find	Return the location of a given value	Value	Index

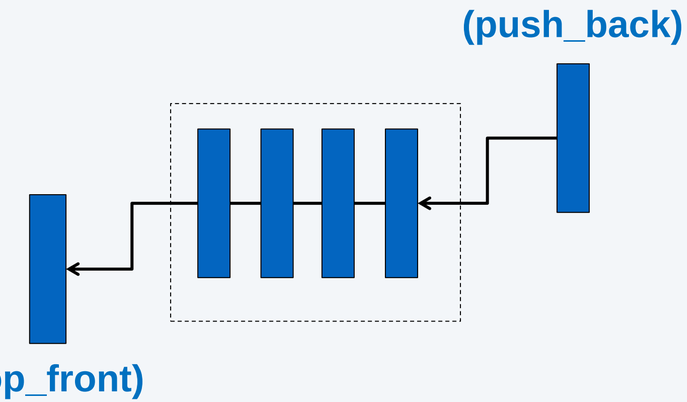
Queues and Stacks

Two specialized List ADTs

Queue (FIFO)

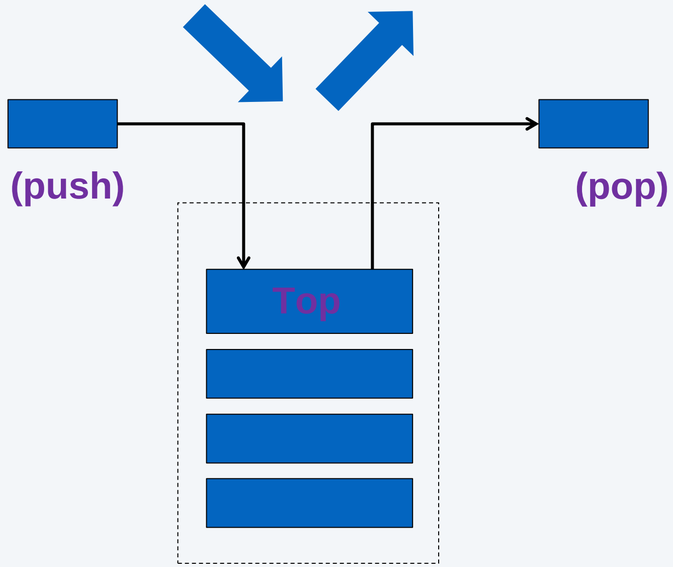
Items leave from the front (pop_front)

Items enter at the back (push_back)



Stack (LIFO)

Items enter and leave from the same side (i.e. the top)



Queue & Stack Operations

Queues

Operations Relative to Lists	Notes
~~insert~~	
~~remove~~	
front	Can only get front item
set	
empty	
size	
push_back	Add to one side
pop_front	Remove from the other

Stacks

Operations Relative to Lists	Notes
~~insert~~	
~~remove~~	
top	Can only get top item
set	
empty	
size	
push	Add to one size
pop	Remove from the same

First-In, First-Out (FIFOs)

QUEUE ADT

Queue ADT

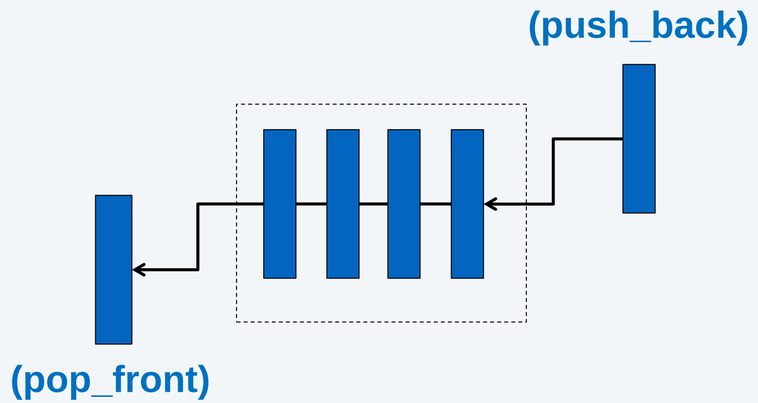
Queue – A list of items where insertion only occurs at the back of the list and removal only occurs at the front of the list

Queues are FIFO (First In, First Out)

- Items at the back of the queue are the newest
- Items at the front of the queue are the oldest
- Elements are processed in the order they arrive

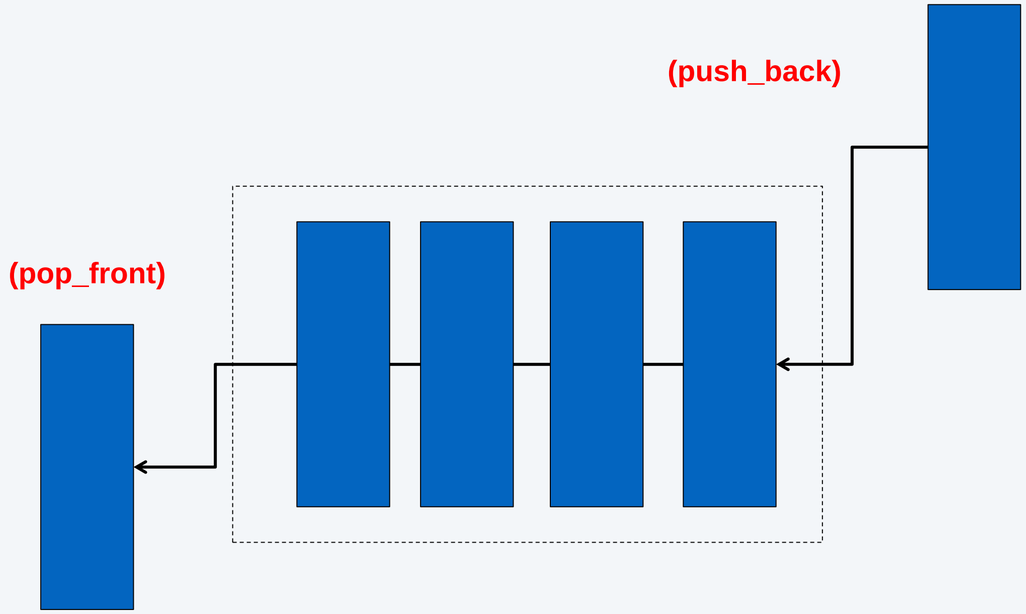
Examples from everyday life:

- Printing jobs in a printer
- Customers in line for cashier at store
- Diners waiting to be seated at restaurantcustomers
- Helpful to decouple producers and consumers



Queue Operations

- push_back(item) – Add an item to the back of the Queue
- pop_front() - Remove the front item from the Queue
- front() - Get a reference to the front item of the Queue (don't remove it though!)
- size() - Number of items in the Queue
- empty() - Check if the Queue is empty



A Queue Interface

This abstract Queue class specifies the Queue ADT operations

Any derived implementation must implement these public member functions to be instantiated

Queue Error Conditions

- **Queue Underflow** – The condition when pop_front is called on an empty Queue
- **Queue Overflow** – The condition when a queue has limited capacity and push_back is called when full

```
class IntQueue {

    public:
    virtual int front() =  0;
    virtual bool empty() = 0;
    virtual size_t size() = 0;
    virtual void push_back(int v) = 0;
    virtual void pop_front() = 0;
};
```

A STL List Queue Class

This class uses STL List to implement a queue.

It is an example of implementing a queue with a doubly linked list.

It is basically a wrapper class for calls directly to STL List.

```
// This class implements the queue ADT interface using STL list.
// Remember STL list is a doubly linked list implementation.

class ListIntQueue : public IntQueue {
        public:
        int front() {
                // Add proper error handling
                if (queue.empty()) return -1;
                 return queue.front();
        }
        bool empty() {        return queue.empty();    }
        size_t size() {        return queue.size();    }
        void push_back(int value) {        queue.push_back(value);    }
        void pop_front() {
                  if (!empty()) {
                       queue.pop_front();
                  }
         }

         private:
         std::list<int> queue;

};
```

A STL Vector Queue Class

This class uses STL Vector to implement a queue.

It is an example of implementing a queue with a dynamically sized array.

It is basically a wrapper class for calls directly to STL Vector.

```
// This class implements the queue ADT interface using STL vector.
// Remember STL vector is a dynamically sized array


class  VectorIntQueue : public IntQueue {

        public:
            int front() {
                    // add appropriate error handling
                    if (this->empty()) { return -1;}
                    return queue.front();
             }

             bool empty() {        return queue.empty();    }
             size_t size() {        return queue.size();    }
             void push_back(int value) {        queue.push_back(v);    }
             void pop_front() {
                        if (!empty()) {
                            queue.erase(queue.begin());
                        }
             }

            private:
            std::vector<int> queue;
};
```

Last-In, First-Out (LIFOs)

STACK ADT

Stack ADT

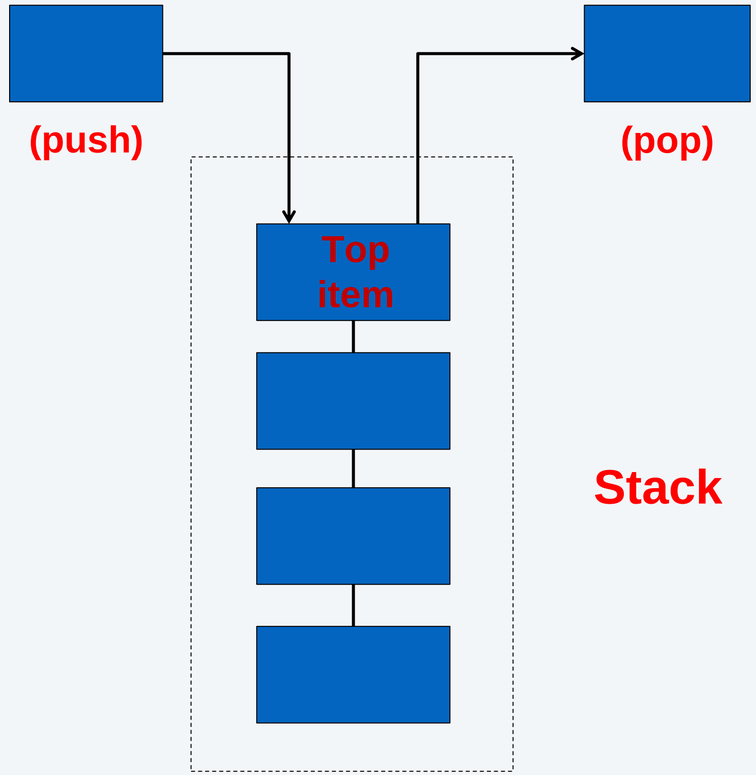
Stack: A list of items where insertion and removal only occurs at one end of the list

Everyday examples:

- A stack of boxes
- A PEZ dispenser
- Your e-mail inbox

Stacks are LIFO

- Newest item at top
- Oldest item at bottom

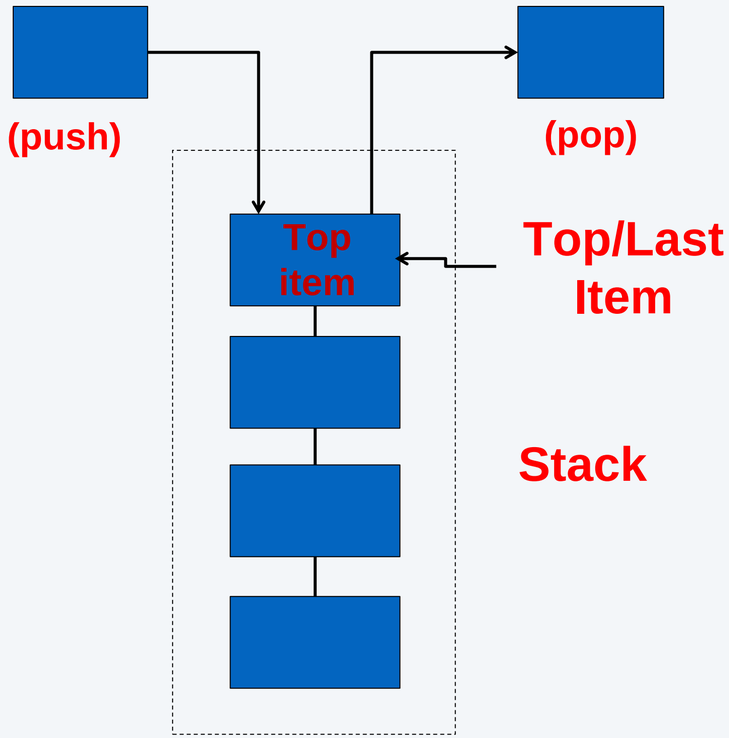


Stack Operations

- push(item) – Add an item to the top of the Stack
- pop() - Remove the top item from the Stack
- top() - View the top item on the Stack
- size() - Get the number of items in the Stack
- empty() –Check if stack has items

A stack implementation needs at least the following member data:

- A list of items
- Top Pointer/Index



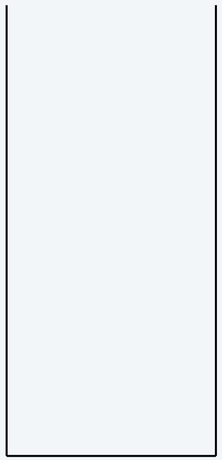
Stack Axioms

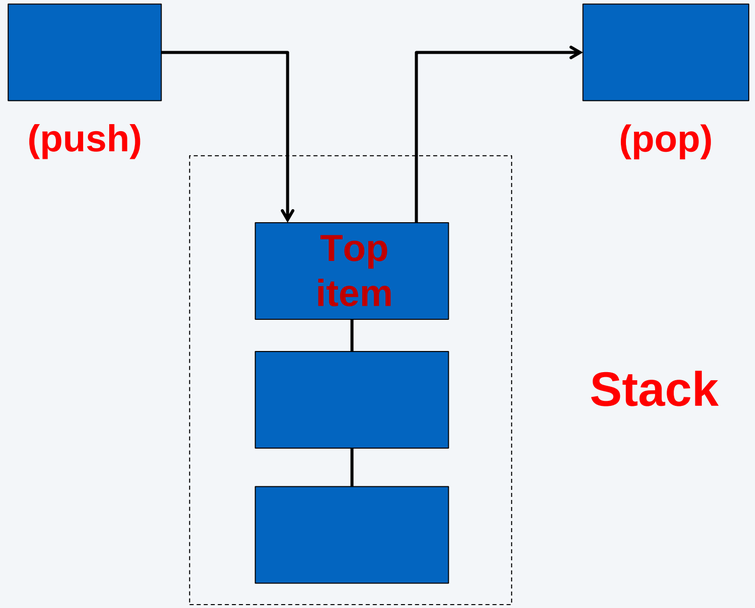
For all stacks, s:

- `s.push(item).top() = item`
- `s.push(item).pop() = s`

Let's draw the stack for these operations:

- `s.push(5).push(4).pop().top()`





A Stack Class

A sample class interface for a Stack

Stack Error Conditions

- Stack Underflow – The condition when pop is called on an empty stack
- Stack Overflow – The condition when push is called on a full stack

```
#ifndef STACKINT_H
#define STACKINT_H

class StackInt  {
 public:
  StackInt();
  ~StackInt();
  size_t size() const;
  bool empty() const;
  void push(const int& value);
  void pop();
  int const& top() const;
private:
  // Use a dynamically sized array
  // or linked list
   list<int> items;//or
  // vector<int> items
   size_t top_index;
};
#endif
```

Stack Example: Reversing a string

Reverse a string trace:

```
#include <iostream>
#include <string>
#include <stack>
using namespace std;
int main()
{
  stack<char> s;

  string word;
  cout << "Enter a word: ";
  getline(cin,word);

  for(int i=0; i < word.size(); i++)
    s.push(word.at(i));

  while(!s.empty()){
    cout << s.top();
    s.pop();
  }
}
```

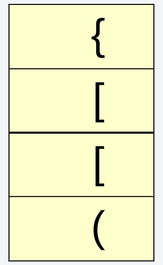
Type in: "hello"
Output: "olleh"

Stack Example: Checking Parentheses

- Check whether an expression is properly parenthesized with '(', '[', '{', '}', ']', ')'
  - Correct: (7 * [8 + [9/{5-2}]])
  - Incorrect: (7*8
  - Incorrect: (7*8]
- Note: The last parentheses started should be the first one completed
- Approach

(7 * [8 + [9/{5-2}]])

([[{ }]])



- Scan character by character of the expression string
- For each current character that is an open-paren '(', '[', '{' : **push** it on the stack.
- When current character is close-paren a ')', ']', '}':

  if **top** equals matching open paren, **pop from stack.**

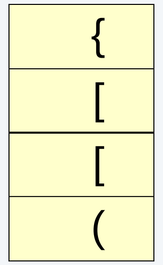
  otherwise ERROR!
- If no characters left in string and stack empty, accept.
- If no characters left in string and stack not empty, reject.

Stack Example: Checking Parentheses Trace

- Check whether an expression is properly parenthesized with '(', '[', '{', '}', ']', ')'
  - Incorrect: (7*8
  - Incorrect: (7*8]
- Let's trace for (7*8 and (7*8] below:

(7 * [8 + [9/{5-2}]])

([[{ }]])

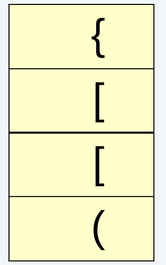


Stack Example: Checking Parentheses Trace

- Check whether an expression is properly parenthesized with '(', '[', '{', '}', ']', ')'
  - Correct: (7 * [8 + [9/{5-2}]])
- Let's trace for (7 * [8 + [9/{5-2}]])

(7 * [8 + [9/{5-2}]])

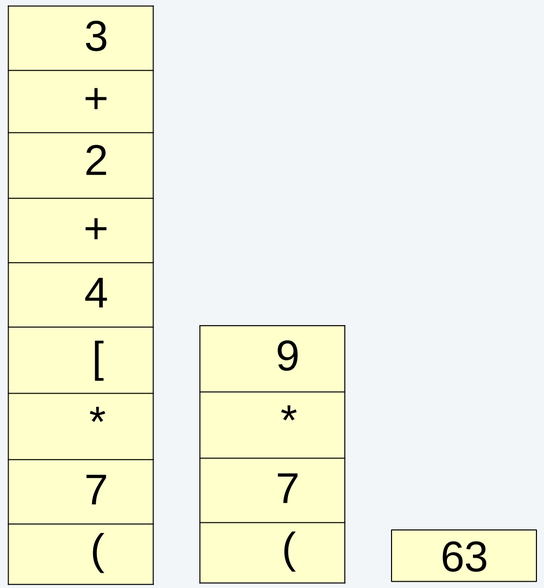
([[{ }]])



Stack Example: Evaluating mathematical expressions

- How do we modify our approach if we also want to evaluate the mathematical expression?
- Approach Modifications
  - Scan character by character of the expression string
  - **push all chars** on the stack
  - When you encounter a ')', ']', '}' , pop until find matching opening paren.
  - Evaluate expression popped and push value on stack.
  - (If malformed expressions permitted and no matching opening paren found, then reject expression.)

(7 * [4 + 2 + 3])



Queue with two stacks

To enqueue(x), push x on stack 1

To dequeue()

- If stack 2 empty, pop everything from stack 1 and push onto stack 2.
- Pop stack 2