CS102
C++ Stacks & Queues
Bill Cheng

http://merlot.usc.edu/cs102-s12
C++ Abstract Data Types

Topics to cover
- Vectors (Ch 22)
- Linked Lists (Ch 17)
- Templates (Ch 14)
- Stacks/Queues (Ch 18)
Stacks/Queues

- Templated lists are good for storing generic sequences of items, but they can be specialized to form other useful structures.

- What if we had a List, but we restricted how insertion and removal were done?
  - **Stack**: only ever insert/remove from the front of the list
  - **Queue**: only ever insert at the back and remove from the front of the list
Stacks

Stack: a list of items where insertion and removal only occurs at one end of the list
- Examples
  - A spring-loaded plate dispenser at a buffet
  - A stack of boxes where you have to move the top one to get to ones farther down
  - A PEZ dispenser

Stacks are LIFO (Last In, First Out)
- Items at the top of the stack are the newest
- Items at the bottom of the stack are the oldest
The Stack ADT

What member data does a Stack need?
- A list of items
- A length
- A maximum size (optional)

What member functions does a Stack have?
- \textit{push(item)} - Add an item to the top of the Stack
- \textit{pop()} - Remove the top item from the Stack
- \textit{top()} - Get a reference to the top item on the Stack (don’t remove it though!)
- \textit{size()} - Get the number of items in the Stack
- \textit{empty()} - Check if the Stack is empty
The Stack ADT

Push

Pop
What does the interface for a Stack look like?

```c++
template <typename T>
class Stack
{
    public:
        Stack();
        ~Stack();
        int size() const;
        void push(const T& value);
        void pop();
        T& top();
        bool empty() const;
};
```
Stack Declaration

How would you build a Linked List-based Stack?

template <typename T>
class Stack
{
    private:
        Node<T>* head;
        int length;
    public:
        Stack();    
        ~Stack();
        int size() const;
        void push(const T& value);
        void pop();
        T& top();
        bool empty() const;
};
Stack Declaration

How would you build a Linked List-based Stack?

You could also back the Stack with a vector

template <typename T>
class Stack
{
    private:
        T* date; // could also be vector<T>
        int length;
    public:
        Stack();
        ~Stack();
        int size() const;
        void push(const T& value);
        void pop();
        T& top();
        bool empty() const;
};
Stack Examples

Reversing the letters in a string

```cpp
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();
    }
}
```
Stack Examples

How would you check that this string has equal numbers of opening/closing parentheses?

\((( (3*4 + 1) 5) + 6 * (2-3) + 4 (1/5)) + 1 \) + 2

- If you see a "(" , use a **push**
- If you see a ")" , use a **pop**
- At the end of the string, your stack should be completely empty!
The call stack is used to keep track of function calls in C++

What happens when your code calls a function?

What happens when you return from a function?

How does your code keep track of which line it should return to when a function ends?

```c++
void A(int x, int y)
{
    int m=0;
    B(x);
}

void B(int x)
{
    int n=0;
    C();
}

void C()
{
    int p=0;
    cout << p;
}
```
What happens when a function is called?
- The address of the next line of code is pushed onto the stack (one line past the function call)
- A placeholder is put on the stack for the function’s return type
- Execution jumps to the function’s code
- All arguments to the function go on the stack
- The function begins executing
- All local variables to the function are pushed onto the call stack
```c
void C()
{
    int p = 0;
    cout << p;
}

void B(int x)
{
    int n = 0;
    C();
}

void A(int x, int y)
{
    int m = 0;
    B(x);
}
```
The Call Stack

What happens when a function returns?

- The return value is copied back into the placeholder that we made for it
- All local arguments and variables are popped off of the stack
- This is why we call them stack variables
- The return value is popped off the stack and assigned to a variable (if need be)
- The address of the next line of code is popped off the stack and executed
Inception is a perfect example of how the call stack works
- Dreams = Functions
- Dreaming = Calling a function
- Waking up = Returning from a function
Other Stack Details

How should you implement a Stack?
- Back it with an array
- Back it with a vector
- Back it with a linked list
- Inherit from linked list
- Which is best?

Stack Error Conditions
- **Stack Underflow**: the name for the condition where you call pop on an empty Stack
- **Stack Overflow**: the name for the condition where you call push on a full Stack (a stack that can’t grow any more)
Queues

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
- Like waiting in line for a cashier at a store

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
The Queue ADT

What member data does a Queue have?
- A list of items
- A length
- A maximum size (optional)

What member functions does a Queue have?
- `push(item)`: add an item to the back of the Queue
- `pop()`: remove the front item from the Queue
- `front() / back()`: get a reference to the front or back item of the Queue (don’t remove it though!)
- `size()`: get the number of items in the Queue
- `empty()`: check if the Queue is empty
The Queue ADT

Pop (Dequeue)

Push (Enqueue)
Queue Declaration

What does the interface for a Queue look like?

template <typename T>
class Queue
{
    public:
    Queue();
    ~Queue();
    int size() const;
    void push(const T& value); // enqueue
    void pop(); // dequeue
    T& front();
    T& back();
    bool empty() const;
};
Queue Declaration

What does a Linked List-based Queue look like?

template <typename T>
class Queue
{
    private:
        Node<T> *head, *tail;
        int length;
    public:
        Queue();
        ~Queue();
        int size() const;
        void push(const T& value); //enqueue
        void pop(); //dequeue
        T& front();
        T& back();
        bool empty() const;
};
Queue Declaration

What does an array-based Queue look like?

You could also back the Queue with a vector

```cpp
template <typename T>
class Queue
{
    private:
        T* data; //could also be vector<T>
        int length;
    public:
        Queue();    ~Queue();
        int size() const;
        void push(const T& value); //enqueue
        void pop(); //dequeue
        T& front();
        T& back();
        bool empty() const;
};
```

```
```
Queue Examples

How does a printer work?

- Multiple print jobs are sent in
- Click "Print" on the computer is much faster than actually printing (build a backlog)
- Each job is processed in the order it’s received (FIFO)

Why wouldn’t you use a "Print Stack" instead of a "Print Queue"?
Other Queue Examples

- Computer processor serving threads
- Serving customers at a restaurant (in the order they were seated)
- Valets parking cars at a busy restaurant (park them in the order they arrived)
- Anything that involves elements "waiting in line"
- How do you organize your closet?
Other Queue Details

How should you implement a Queue?
- Back it with an array
- Back it with a vector
- Back it with a linked list
- Inherit from a linked list
- Which is best?

Queue Error Conditions
- **Queue Underflow:** the name for the condition where you call `pop` on an empty Queue
- **Queue Overflow:** the name for the condition where you call `push` on a full Queue (a Queue that can’t grow any more)
Deques

**Deque:** a combination of a Stack and a Queue where you can insert or remove at either end of the list (but not the middle)

- Like books on a bookshelf
The Deque ADT

- Push Front
- Push Back
- Pop Front
- Pop Back
Implement Stack Using Deque

- Push Front
- Pop Front
- Push Back
- Pop Back
Implement Queue Using Deque

Push Front

Push Back

Pop Front

Pop Back