Lecture 7
Stacks

Slides modified from © 2010 Goodrich, Tamassia

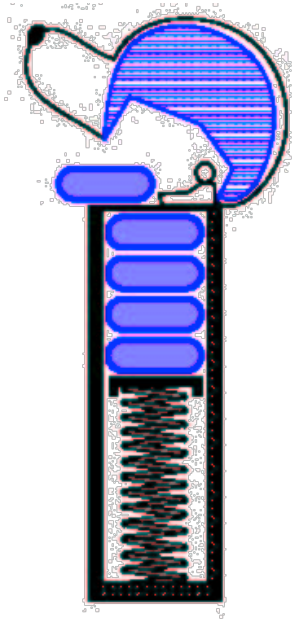
How would you store the visited sites in a browser!

Design a data structure to store lines of a Text Editor!

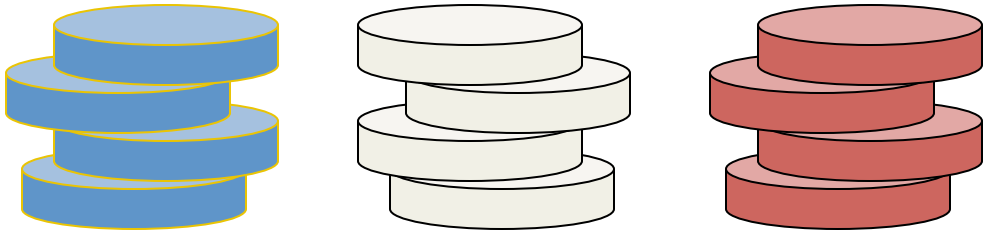
frequent “undo” operations!

What is the common requirement?

Organize data in Last-In First-Out fashion!

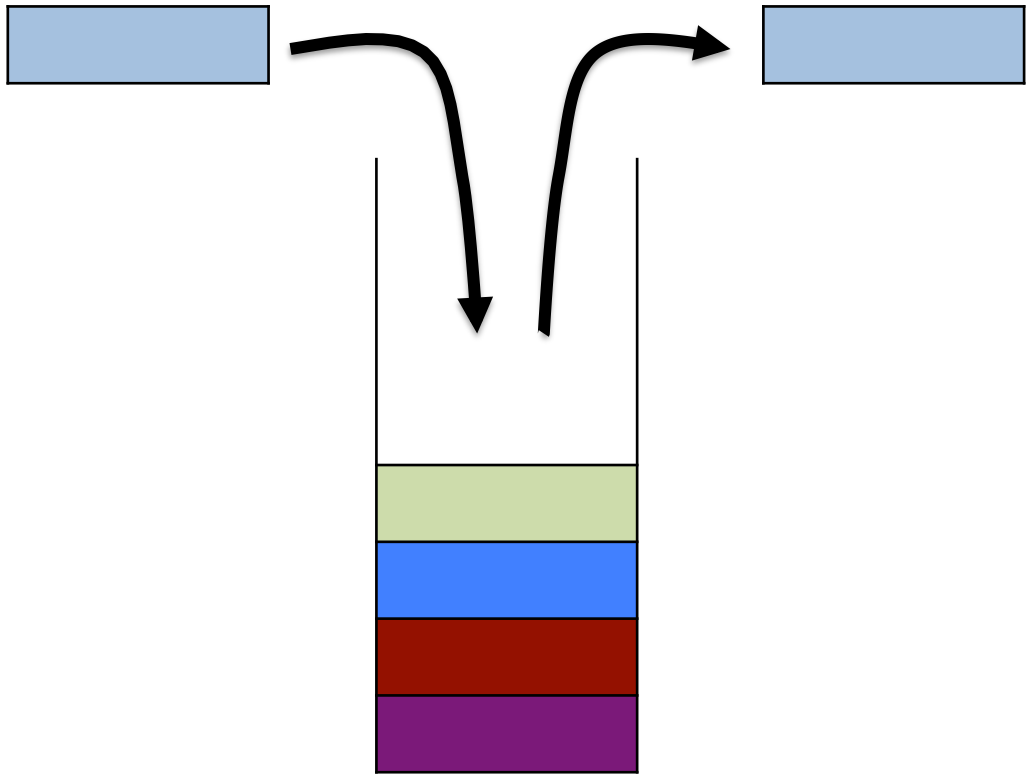


Stacks



Main Operations

- Push
- Pop



Auxiliary Operations

- object top(): returns the last inserted element without removing it
- integer size(): returns the number of elements stored
- boolean empty(): indicates whether no elements are stored

Stack Interface

```
template <typename E>
class Stack  {
public:
  int size() const;
  bool empty() const;
  const E& top();
  void push(const E& e);
  void pop();
}
```

How to implement a stack ADT?

Array-based Implementation

- add elements left to right
- keep track of index

```
template <typename E> class ArrayStack {
  enum { DEF CAPACITY = 100 };
  public:
  ArrayStack(int cap = DEF CAPACITY);
  int size() const;
  bool empty() const;
  const E& top() const;
  void push(const E& e);
  void pop();
private:
  E* S;
  int capacity;
  int t;
};
```

- ArrayStack(int cap)
  - S = new E(cap)
  - t=-1
- size()
  - return t+1
- empty()
  - return t<0
- top
  - return [t]
- pop
  - t=t-1
- push(e)
  - t=t+1
  - S[t] = e

Limitations

- The maximum size of the stack must be defined a priori and cannot be changed
- Trying to push a new element into a full stack causes an implementation-specific exception

Example use in C++

```
ArrayStack<int> A;                      // A = [ ], size = 0
A.push(7);                              // A = [7*], size = 1
A.push(13);                             // A = [7, 13*], size = 2
cout << A.top() << endl; A.pop();       // A = [7*], outputs: 13
A.push(9);                              // A = [7, 9*], size = 2
cout << A.top() << endl;                // A = [7, 9*], outputs: 9
cout << A.top() << endl; A.pop();       // A = [7*], outputs: 9
ArrayStack<string> B(10);               // B = [ ], size = 0
B.push("Bob");                          // B = [Bob*], size = 1
B.push("Alice");                        // B = [Bob, Alice*], size = 2
cout << B.top() << endl; B.pop();       // B = [Bob*], outputs: Alice
B.push("Eve");                          // B = [Bob, Eve*], size = 2
```

Write efficient method to reverse the words of a sentence!

Limitations of Array-Based Implementation

- The maximum size of the stack must be defined a priori and cannot be changed
- Trying to push a new element into a full stack causes an implementation-specific exception

Linked List-Based Stack

```
template <typename E> class LinkedStack {
  public:
  LinkedStack();
  int size() const;
  bool empty() const;
  const E& top() const;
  void push(const E& e);
  void pop();
  private:
    SLinkedList<E> S;
    int t;
};
```

Main Operations

- size()
  - return n
- empty
  - return n==1
- top
  - return ref to front
- pop
  - remove from front
  - n--
- push
  - add to front
  - n++

Some Stack Applications

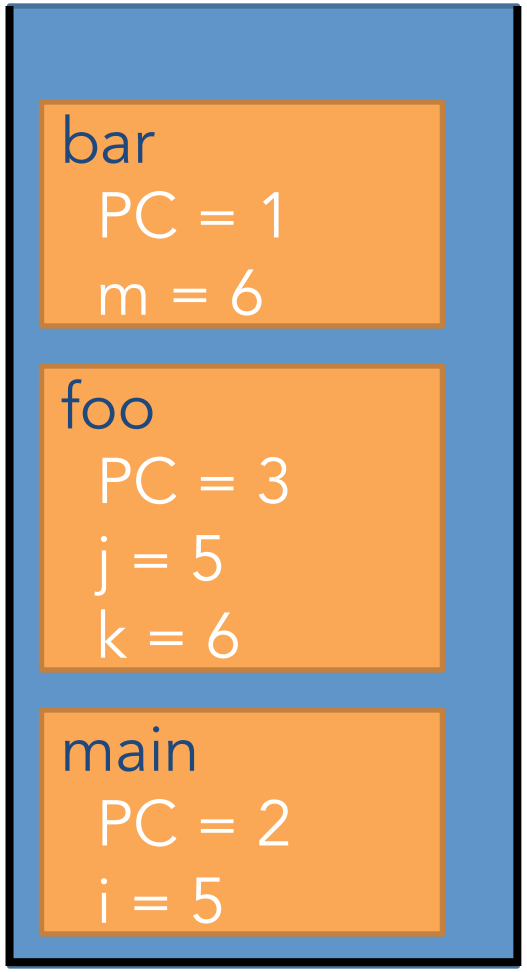
- C++ Run-Time Stack
- Arithmetic expression evaluation
- Checking C++/HTML syntax

C++ Run-Time Stack

```
main() {
   int i = 5;
   foo(i);
   }

foo(int j) {
   int k;
   k = j+1;
   bar(k);
   }

bar(int m) {
   …
   }
```



Arithmetic expression evaluation

5 * (6 + 2) – 12/4

=37

Infix, Prefix and Postfix Expressions

Infix ➔ 5 * (6 + 2) – 12/4

Postfix ➔ 5 6 2 + * 12 4 / –

Prefix ➔ –* 5 + 6 2 / 12 4

Evaluating Postfix Expression

```
while (scan left to right, till not end of P)
      if operand
            push it onto the stack
      end-if
      if operator (op)
            pop the stack and call it A
            pop the stack and call it B
            evaluate B op A
            push the resulting value onto the stack
      end-if
end-while
pop the stack (this is the final value)
```

1. convert infix to postfix expression
2. evaluate post-fix expression

Infix to Postfix Conversion

5 * (6 + 2) – 12/4 ➔ 5 6 2 + * 12 4 / –

- operator precedence
- () > *,/ > - +

C++ Group Symbol Matching Algorithm

- ()
- {}
- []

```
Algorithm ParenMatch(X,n):
for i=0 to n-1 do
    if X[i] is an opening grouping symbol then
     S.push(X[i])
    else if X[i] is a closing grouping symbol then
     if S.empty() then
          return false //nothing to match with
     if S.top() does not match the type of X[i] then
          return false //wrong type
    else S.pop()
if S.empty() then
    return true //every symbol matched
else return false //some symbols were never matched
```