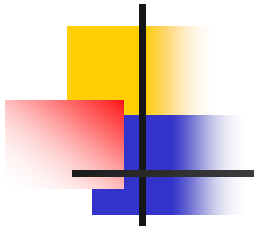
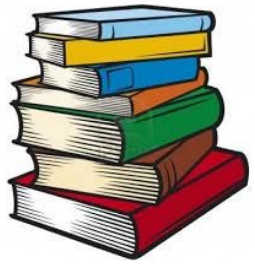
Stacks

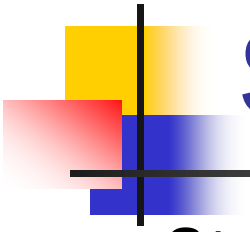


- Stack Data Container
- STL Overview
- STL Stack Class
- Some Stack Applications
- Stack Class Implementations
  - Static Array
  - Dynamic Linked List

Stack Data Container

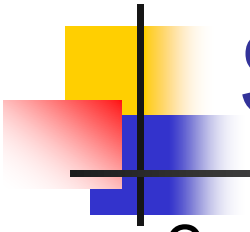


- Aka pushdown store
  - Push → add entry to stack
  - Pop → remove entry from stack
- Ordered collection of entries accessible at only one end → the top
- Entries removed in reverse order
  - Last-In/First-Out (LIFO)



STL Overview

- Standard Template Library (STL) contains generic templates for implementing container objects, algorithms, and iterators
  - HP released demo version in 1994
  - adopted as part of C++ Standard Library and released by Silicon Graphics, Inc. (SGI) in 1996
    - STL Programmer's Guide
      - www.sgi.com/tech/stl/
    - Other STL reference
      - www.cplusplus.com/reference/stl/
      - en.cppreference.com/w/cpp/container



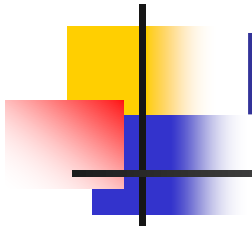
STL Overview

- Containers are ADTs
  - store other objects (elements), and
  - have methods for accessing its elements
- STL has container classes categorized according to:
  - element ordering, and
  - types of operations to access data

STL Container Class Categories

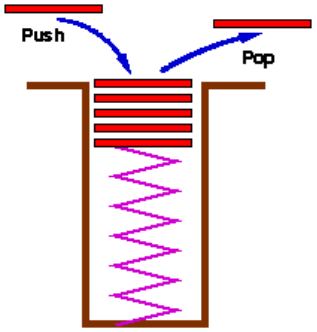
- Sequence
  - stores data by position in linear order
    - vector, deque, list
- Adapter
  - has *another* container as its underlying data structure
  - restricted set of operations on underlying data structure
    - stack, queue, priority_queue
- Associative
  - stores data by key
  - bears no relationship to element location in container
    - set, multiset, map, multimap

STL Containers and Header Files



- Sequence
  - vector — <vector>
  - deque — <deque>
  - list — <list>
- Adapters
  - stack — <stack>
  - queue — <queue>
  - priority_queue — <queue>
- Associative
  - set — <set>
  - multiset — <set>
  - map — <map>
  - multimap — <map>

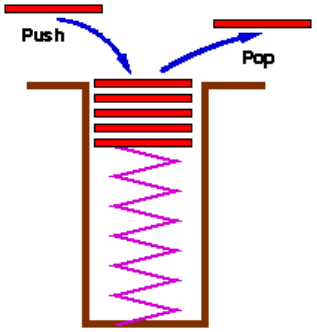
STL Stack Class

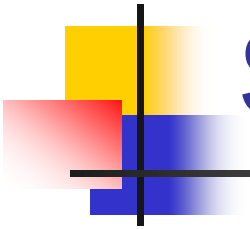


- Template-based class that stores elements of *same* data type
  - allows insertions into, and deletions from, one end
- Online API
  - http://www.sgi.com/tech/stl/stack.html
  - http://cppreference.com/cppstack/index.html
- *Can* be implemented using vectors, lists or deques as the underlying data structure. *By default, it uses deque as the base.*
  - `stack<type> name;`
  - `stack<type, vector<type> > name;`
  - `stack<type, list<type> > name;`

STL Stack Class

- Required header file
  - `#include <stack>`
- Stack Methods
  - pop → void function that *removes* top item of stack
  - push → void function that *adds* item to top of stack
  - empty → bool function that determines if stack is empty
  - size → returns number of items in stack
  - top → returns reference to top stack item without removing it
- Stack Errors
  - Underflow → popping an item from an empty stack
  - Overflow → pushing an item to a full stack





Some Stack Applications

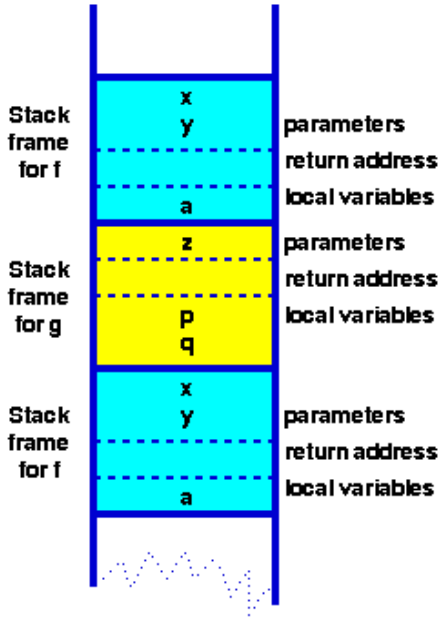
- Solving a Maze
- Memory Management
- Balanced Parentheses
- Multibase Conversion
- Arithmetic Expressions
  - Infix to Postfix Conversion
  - Postfix Evaluation

Solving a Maze

- Try all possible sequences of moves in the maze until either
  - Path of moves that solves maze, or
  - No more moves to try
- Attempting all possible search paths is known as exhaustive search
- Stack keeps track of possible paths
  - Add potential path move(s) to stack until maze solved or unable to move
  - 'Backtrack' by popping stack move and continuing along different path

Memory Management

- Use of frame (or activation record) for working memory
- Each function call has frame consisting of:
  - parameters
  - return address
  - local variables
- Function call adds frame to stack
- Function return removes frame from stack



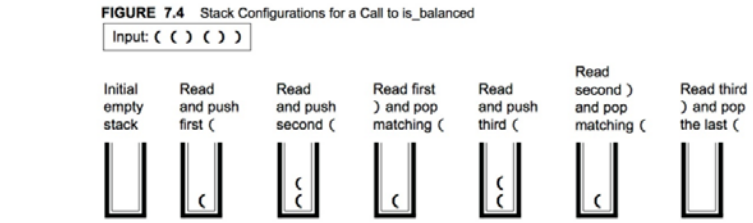

```
int f(int x, int y) {
   int a;
   // other code
   return g(a);
}

int g(int z) {
   int p, q;
   // other code
   return f(p, q);
}

int main() {
   // other code
   return f(1, 2);
}
```

Balanced Parenthesis

- Check expression to see if left and right parentheses match
  - Scan characters from left to right
  - If character is
    - (→ push (on stack
    -) → pop stack
    - anything else → ignore
  - End of characters reached
    - stack empty → matched
    - stack !empty → !matched

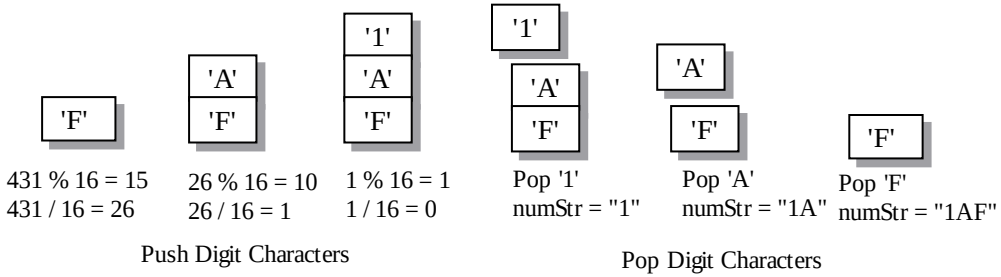



```
bool is_balanced(const string& expression);
```

Multibase Conversion

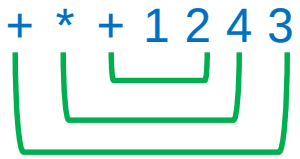
- Given an integer value and base in range from 2 to 16, return converted value
- Algorithm to convert nonnegative decimal integer (N) to base (B) uses repeated division by base
  - Remainder identifies digit (N%B → R)
    - Push R on stack
  - Quotient is next dividend (N/B → N)
    - Terminate when N = 0
  - Pop items off stack to create converted value

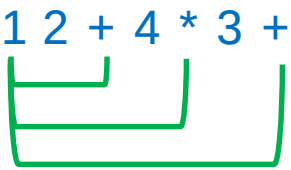
Convert 431 to hex



Arithmetic Expressions

- Infix notation
  - Operators *between* operands
  - ((1 + 2) * 4) + 3
- Prefix notation
  - Aka Polish prefix notation
  - Operators *before* operands
  - + * + 1 2 4 3
- Postfix notation
  - Aka Polish postfix notation or reverse Polish notation
  - Operators *after* operands
  - 1 2 + 4 * 3 +



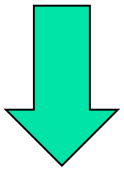


Infix to Postfix Conversion

- Given infix notation (*using parentheses for precedence*)
- Can be written in postfix without parenthesis
- Conversion algorithm
  - For *each* character of infix expression, if character is
    - (→ read character and push on stack
    - Operand → read operand and write to output
    - Operator
      - Repeat get and write operator from stack, while NONE of the following are true
        - stack becomes empty OR
        - stack top == (OR
        - stack top operator has lower precedence
      - read operator and push on stack
    -) → read and discard
      - *repeat* get and write operator from stack, until
      - pop (from stack
  - Get and write remaining operators from stack

Infix to Postfix Conversion

((1 + 2) * 4) + 3



1 2 + 4 * 3 +

Peek	Operation	Stack	Output
(	read and push (	(	
(	read and push (	((	
1	read and output 1	((	1
+	stack top == (read and push +	((+	1
2	read and output 2	((+	12
)	read and discard (get and write + pop (	(	12+
*****	stack top == (push *	(*	12+
4	read and output 4	(*	12+4
)	read and discard (get and write * pop (		12+4*
+	stack becomes empty push +	+	12+4*
3	read and output 3	+	12+4*3
	get and write +		12+4*3+

Postfix Evaluation

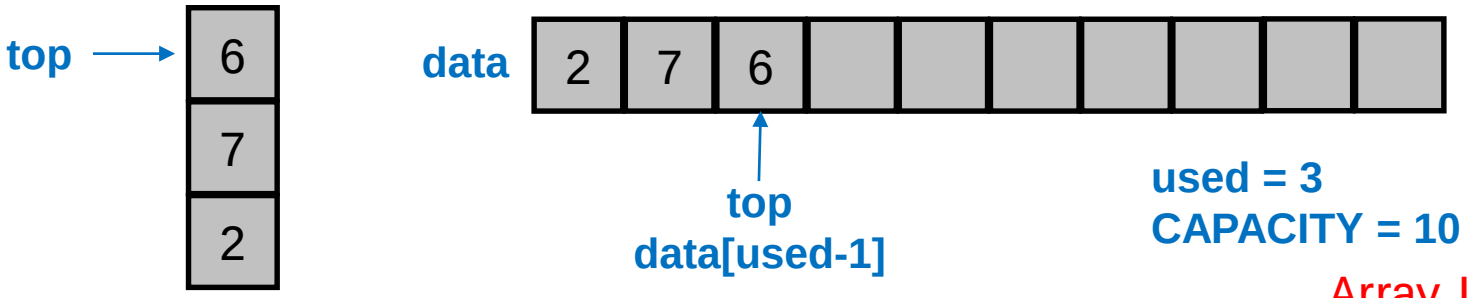
- For *each* character of postfix expression, if character is
  - Operand → read character and push on stack
  - Operator
    - pop two operands from stack
    - evaluate expression
      - second operand is lhs
    - push result on stack

1 2 + 4 * 3 +

Character	Operation	Stack
1	push 1	1
2	push 2	1,2
+	pop 2 pop 1 evaluate 1 + 2 push 3	3
4	push 4	3,4
*****	pop 4 pop 3 evaluate 3 * 4 push 12	12
3	push 3	12,3
+	pop 3 pop 12 evaluate 12 + 3 push 15	15

Stack Class Implementations

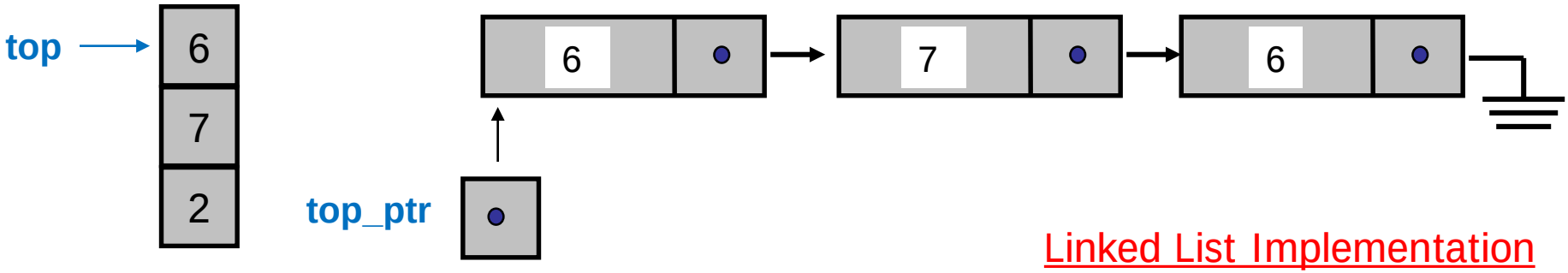
- Static implementation using fixed-sized array
- Rules for implementation
  - Array called **`data`** holds up to **`CAPACITY`** items
  - *Number* of items stored in member variable **`used`**
  - *Stack* items stored in **`data[0]`** to **`data[used-1]`**
    - Top of stack at **`data[used-1]`**



Array Implementation

Stack Class Implementations

- Dynamic implementation using linked list
- Rules for implementation
  - Items stored in linked list
  - Top of stack in list is stored in member variable `top_ptr`



Linked List Implementation