Stacks

- Linear list.
- One end is called top.
- Other end is called bottom.
- Additions to and removals from the top end only.
Stack Of Cups

- Add a cup to the stack.
- Remove a cup from new stack.
- A stack is a LIFO list.
The Abstract Class stack

template<class T>
class stack
{
  public:
    virtual ~stack() {}
    virtual bool empty() const = 0;
    virtual int size() const = 0;
    virtual T& top() = 0;
    virtual void pop() = 0;
    virtual void push(const T& theElement) = 0;
};
Parentheses Matching

• \(((a+b)*(c+d-e))/(f+g)-(h+j)*(k-l))/(m-n)\)
  – Output pairs \((u,v)\) such that the left parenthesis at position \(u\) is matched with the right parenthesis at \(v\).
    • (2,6) (1,13) (15,19) (21,25) (27,31) (0,32) (34,38)

• \((a+b))*)((c+d)\)
  – (0,4)
  – Right parenthesis at 5 has no matching left parenthesis
  – (8,12)
  – Left parenthesis at 7 has no matching right parenthesis
Parentheses Matching

- scan expression from left to right
- when a left parenthesis is encountered, add its position to the stack
- when a right parenthesis is encountered, remove matching position from stack
Example

- \(((a+b)\cdot c+d-e)/(f+g)-(h+j)\cdot (k-l))/(m-n)\)
Example

\[ (((a+b) \cdot c+d-e)/(f+g)-(h+j) \cdot (k-l))/(m-n) \]
Example

• \[((a+b)*c+d-e)/(f+g)-(h+j)*(k-l))/(m-n)\]
Example

• \[((a+b)\times c+d-e)/(f+g)-(h+j)\times (k-l))/(m-n)\]
Example

- \(((a+b)\times c+d-e)/(f+g)-(h+j)\times (k-l))/(m-n)\)

- (2,6) (1,13) (15,19) (21,25)(27,31) (0,32)

- and so on
Towers Of Hanoi/Brahma

- 64 gold disks to be moved from tower A to tower C
- each tower operates as a stack
- cannot place big disk on top of a smaller one
Towers Of Hanoi/Brahma

- 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

• 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

• 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

- 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

- 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

- 3-disk Towers Of Hanoi/Brahma
Towers Of Hanoi/Brahma

- **3-disk Towers Of Hanoi/Brahma**
Towers Of Hanoi/Brahma

- 3-disk Towers Of Hanoi/Brahma
- 7 disk moves
Recursive Solution

- \( n > 0 \) gold disks to be moved from A to C using B
- move top \( n-1 \) disks from A to B using C
Recursive Solution

- move top disk from A to C
Recursive Solution

• move top $n-1$ disks from B to C using A
Recursive Solution

• \( \text{moves}(n) = 0 \) when \( n = 0 \)

• \( \text{moves}(n) = 2 \times \text{moves}(n-1) + 1 = 2^n - 1 \) when \( n > 0 \)
Towers Of Hanoi/Brahma

- moves(64) = $1.8 \times 10^{19}$ (approximately)
- Performing $10^9$ moves/second, a computer would take about 570 years to complete.
- At 1 disk move/min, the monks will take about $3.4 \times 10^{13}$ years.
Chess Story

- 1 grain of rice on the first square, 2 for next, 4 for next, 8 for next, and so on.
- Surface area needed exceeds surface area of earth.
• 1 penny for the first square, 2 for next, 4 for next, 8 for next, and so on.

• $3.6 * 10^{17}$ (federal budget ~ $2 * 10^{12}$). 
Switch Box Routing

Routing region
Routing for pins 1-3 and 18-40 is confined to lower left region.

Routing for pins 5 through 16 is confined to upper right region.
(u, v), u < v is a 2-pin net.

u is start pin.

v is end pin.

Examine pins in clock-wise order beginning with pin 1.
Routing A 2-pin Net

Start pin => push onto stack.

End pin => start pin must be at top of stack.
public void a()
{ ...; b(); ...}

public void b()
{ ...; c(); ...}

public void c()
{ ...; d(); ...}

public void d()
{ ...; e(); ...}

public void e()
{ ...; c(); ...}

return address in d()
return address in c()
return address in e()
return address in d()
return address in c()
return address in b()
return address in a()
Try-Throw-Catch

- When you enter a try block, push the address of this block on a stack.
- When an exception is thrown, pop the try block that is at the top of the stack (if the stack is empty, terminate).
- If the popped try block has no matching catch block, go back to the preceding step.
- If the popped try block has a matching catch block, execute the matching catch block.
Rat In A Maze
Rat In A Maze

- Move order is: right, down, left, up
- Block positions to avoid revisit.
Rat In A Maze

- Move order is: right, down, left, up
- Block positions to avoid revisit.
Rat In A Maze

- Move backward until we reach a square from which a forward move is possible.
• Move down.
Rat In A Maze

- Move left.
Rat In A Maze

- Move down.
• Move backward until we reach a square from which a forward move is possible.
Rat In A Maze

- Move backward until we reach a square from which a forward move is possible.
- Move downward.
• Move right.
• Backtrack.
• Move downward.
• Move right.
• Move one down and then right.
• Move one up and then right.
• Move down to exit and eat cheese.
• Path from maze entry to current position operates as a stack.