Stacks

Objectives

Upon completion you will be able to

- Explain the design, use, and operation of a stack
- Implement a stack using a linked list structure
- Understand the operation of the stack ADT

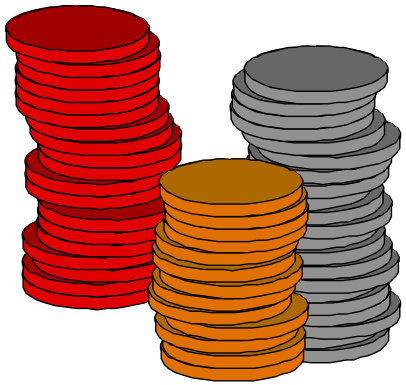
Stacks

- Stack: what is it?
- ADT
- Applications
- Implementation(s)

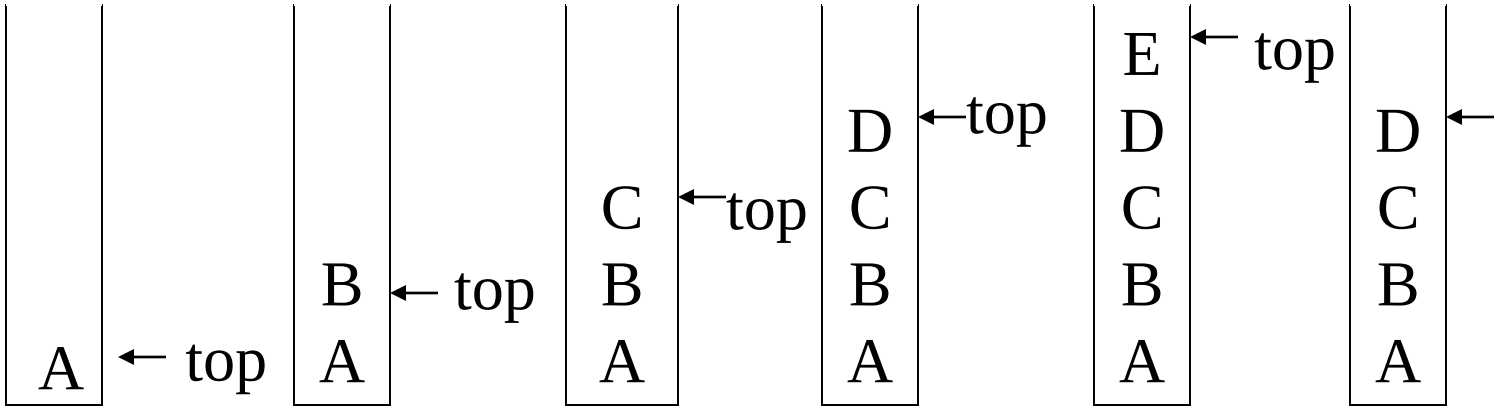
What is a stack?

- Stores a set of elements in a particular order
- Stack principle: LAST IN FIRST OUT
- = LIFO
- It means: the last element inserted is the first one to be removed
- Example

- Which is the first element to pick up?



Last In First Out



Stack Applications

- Real life
  - Pile of books
  - Plate trays
- More applications related to computer science
  - Program execution stack (read more from your text)
  - Evaluating expressions

Stack ADT

OBJECTS: A FINITE ORDERED LIST WITH ZERO OR MORE ELEMENTS.

METHODS:

FOR ALL STACK ∈ STACK, ITEM ∈ ELEMENT, MAX_STACK_SIZE ∈ POSITIVE INTEGER

STACK CREATES(MAX_STACK_SIZE) ::=
CREATE AN EMPTY STACK WHOSE MAXIMUM SIZE IS MAX_STACK_SIZE

BOOLEAN ISFULL(STACK, MAX_STACK_SIZE) ::=
IF (NUMBER OF ELEMENTS IN STACK == MAX_STACK_SIZE)
RETURN TRUE
ELSE RETURN FALSE

STACK PUSH(STACK, ITEM) ::=
IF (ISFULL(STACK)) STACK_FULL
ELSE INSERT ITEM INTO TOP OF STACK AND **RETURN**

Stack ADT (cont'd)

```
Boolean isEmpty(stack) ::=
                    IF(stack == CreateS(max_stack_size))
                    RETURN TRUE
                    ELSE RETURN FALSE
Element pop(stack) ::=
                    IF(IsEmpty(stack)) RETURN
                    ELSE REMOVE AND RETURN THE ITEM ON THE TOP
                         OF THE STACK.
```

Array-based Stack Implementation

- Allocate an array of some size (pre-defined)
  - Maximum N elements in stack
- Bottom stack element stored at element 0
- last index in the array is the *top*
- Increment *top* when one element is pushed, decrement after pop

Stack Implementation: CreateS, isEmpty, isFull

```
Stack createS(max_stack_size) ::=
  #define MAX_STACK_SIZE 100 /* maximum stack size */
  typedef struct {
            int key;
            /* other fields */
            } element;
  element stack[MAX_STACK_SIZE];
  int top = -1;

  Boolean isEmpty(Stack) ::= top< 0;

  Boolean isFull(Stack) ::= top >= MAX_STACK_SIZE-1;
```

Push

```
VOID PUSH(INT *TOP, ELEMENT ITEM)
{
 /* ADD AN ITEM TO THE GLOBAL STACK */
     IF (*TOP >= MAX_STACK_SIZE-1)  {
          STACK_FULL( );
          RETURN;
     }
     STACK[++*TOP] = ITEM;
}
```

Pop

```
ELEMENT POP(INT *TOP)
{
 /* RETURN THE TOP ELEMENT FROM THE STACK */
     IF (*TOP == -1)
          RETURN STACK_EMPTY( );  /* RETURNS AND ERROR KEY */
     RETURN STACK[(*TOP)--];
 }
```

List-based Stack Implementation: Push

```
VOID PUSH(PNODE TOP, ELEMENT ITEM)
{
   /* ADD AN ELEMENT TO THE TOP OF THE STACK */
   PNODE TEMP =
                        (PNODE) MALLOC (SIZEOF (NODE));
   IF (IS_FULL(TEMP)) {
      FPRINTF(STDERR, " THE MEMORY IS FULL\N");
      EXIT(1);
      }
      TEMP->ITEM = ITEM;
      TEMP->NEXT= TOP;
      TOP= TEMP;
}
```

Pop

```
element pop(pnode top) {
/* delete an element from the stack */
    pnode temp = top;
    element item;
    if (is_empty(temp))  {
       fprintf(stderr,  “the stack is empty\n”);
       exit(1);
    }
    item = temp->item;
    top = temp->next;
     free(temp);
     return item;
}
```

Algorithm Analysis

- push O(?)
- pop O(?)
- isEmpty O(?)
- isFull O(?)

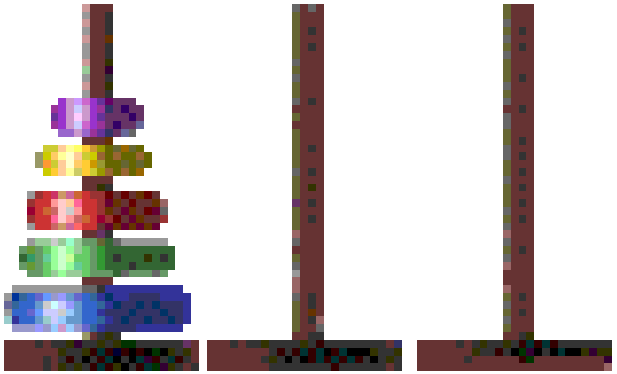
- What if *top* is stored at the beginning of the array?

A Legend
The Towers of Hanoi

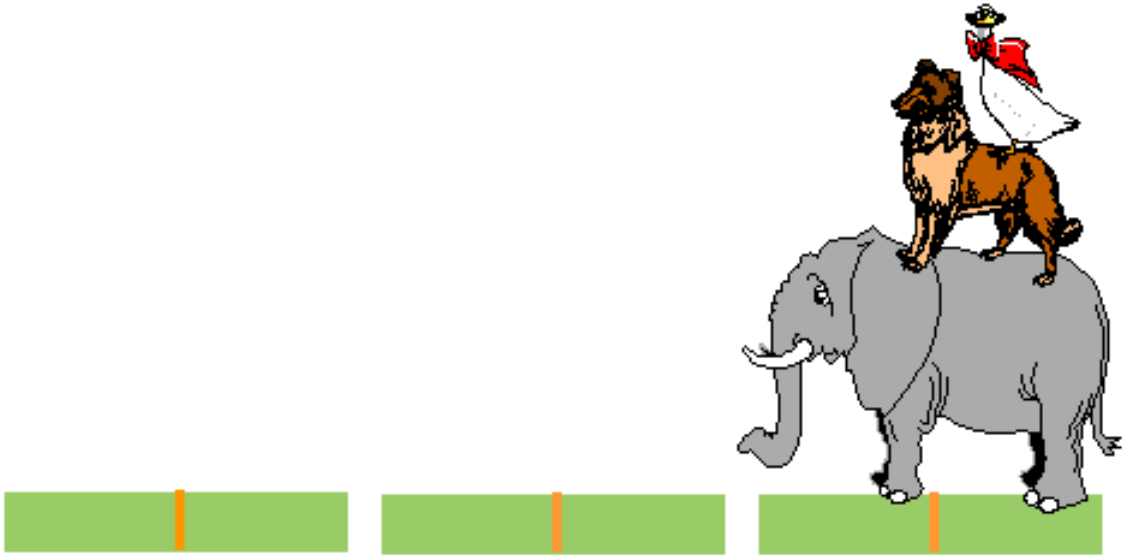
- *In the great temple of Brahma in Benares, on a brass plate under the dome that marks the center of the world, there are 64 disks of pure gold that the priests carry one at a time between these diamond needles according to Brahma's immutable law: No disk may be placed on a smaller disk. In the begging of the world all 64 disks formed the Tower of Brahma on one needle. Now, however, the process of transfer of the tower from one needle to another is in mid course. When the last disk is finally in place, once again forming the Tower of Brahma but on a different needle, then will come the end of the world and all will turn to dust.*

The Towers of Hanoi
A Stack-based Application

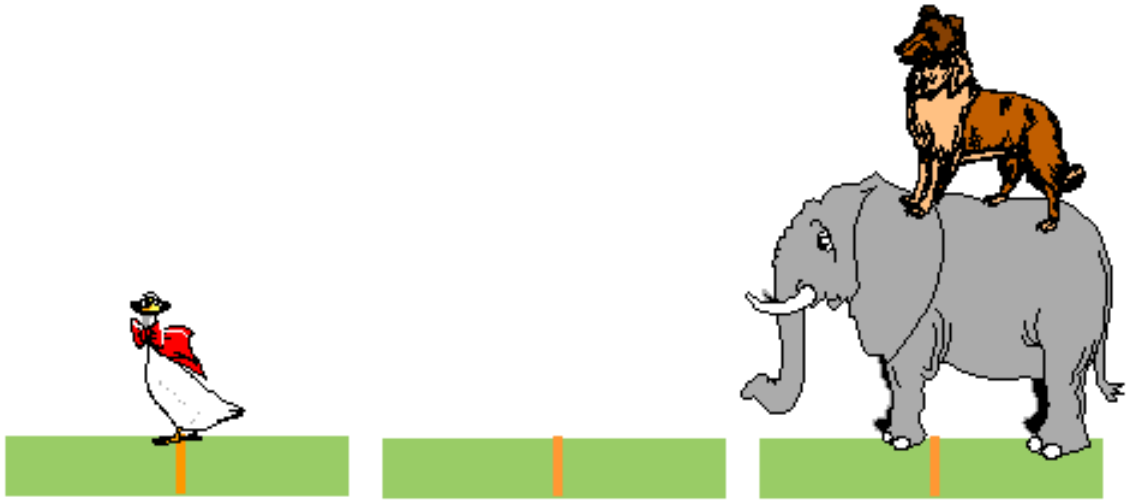
- GIVEN: three poles
- a set of discs on the first pole, discs of different sizes, the smallest discs at the top
- GOAL: move all the discs from the left pole to the right one.
- CONDITIONS: only one disc may be moved at a time.
- A disc can be placed either on an empty pole or on top of a larger disc.



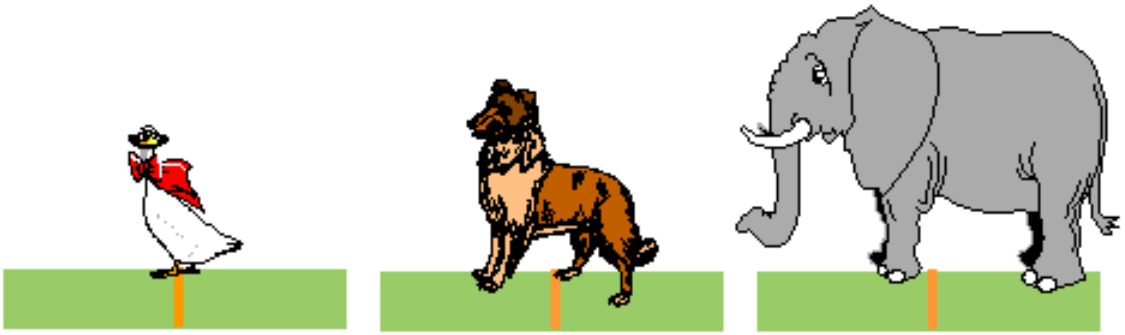
Towers of Hanoi



Towers of Hanoi



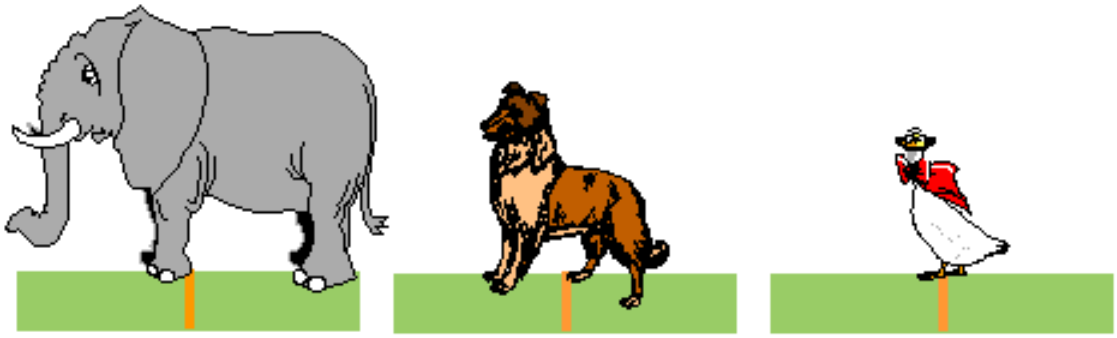
Towers of Hanoi



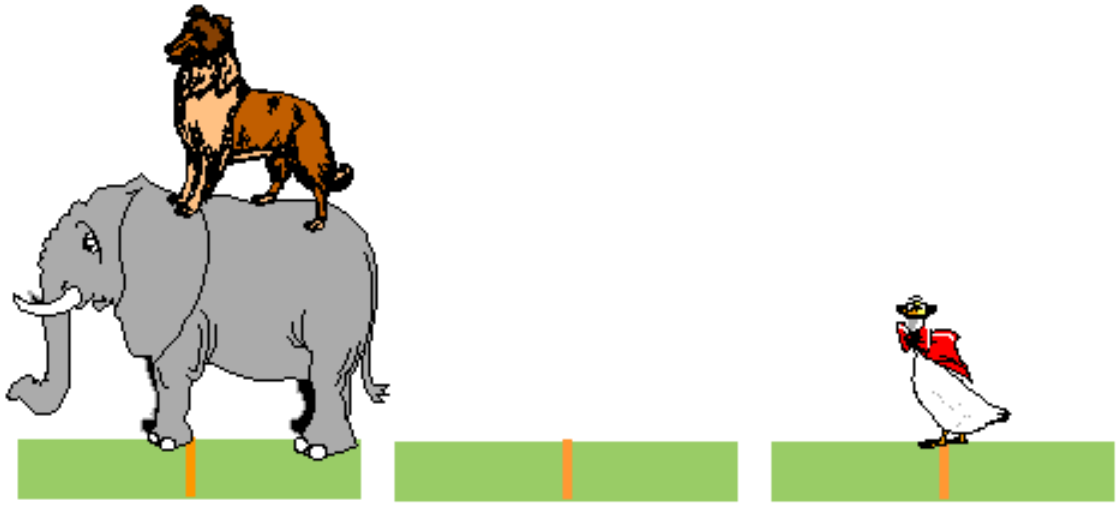
Towers of Hanoi

Towers of Hanoi

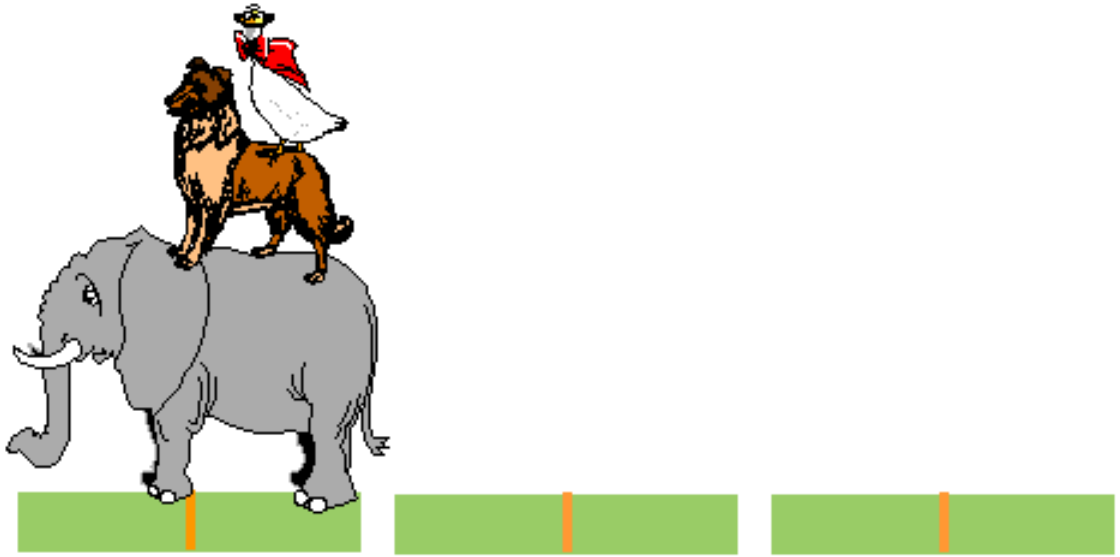
Towers of Hanoi



Towers of Hanoi



Towers of Hanoi



Towers of Hanoi – Recursive Solution

```
    void hanoi (int discs,
                Stack fromPole,
                Stack toPole,
                Stack aux) {
Disc d;
if( discs >= 1) {
        hanoi(discs-1, fromPole, aux, toPole);
        d = fromPole.pop();
        toPole.push(d);
        hanoi(discs-1,aux, toPole, fromPole);
}
```

Is the End of the World Approaching?

- Problem complexity 2^n
- 64 gold discs
- Given 1 move a second

→ 600,000,000,000 years until the end of the world ☺