Stack Using Arrays

Overview

Stacks –Definition

Stack Operations

Applications of Stacks

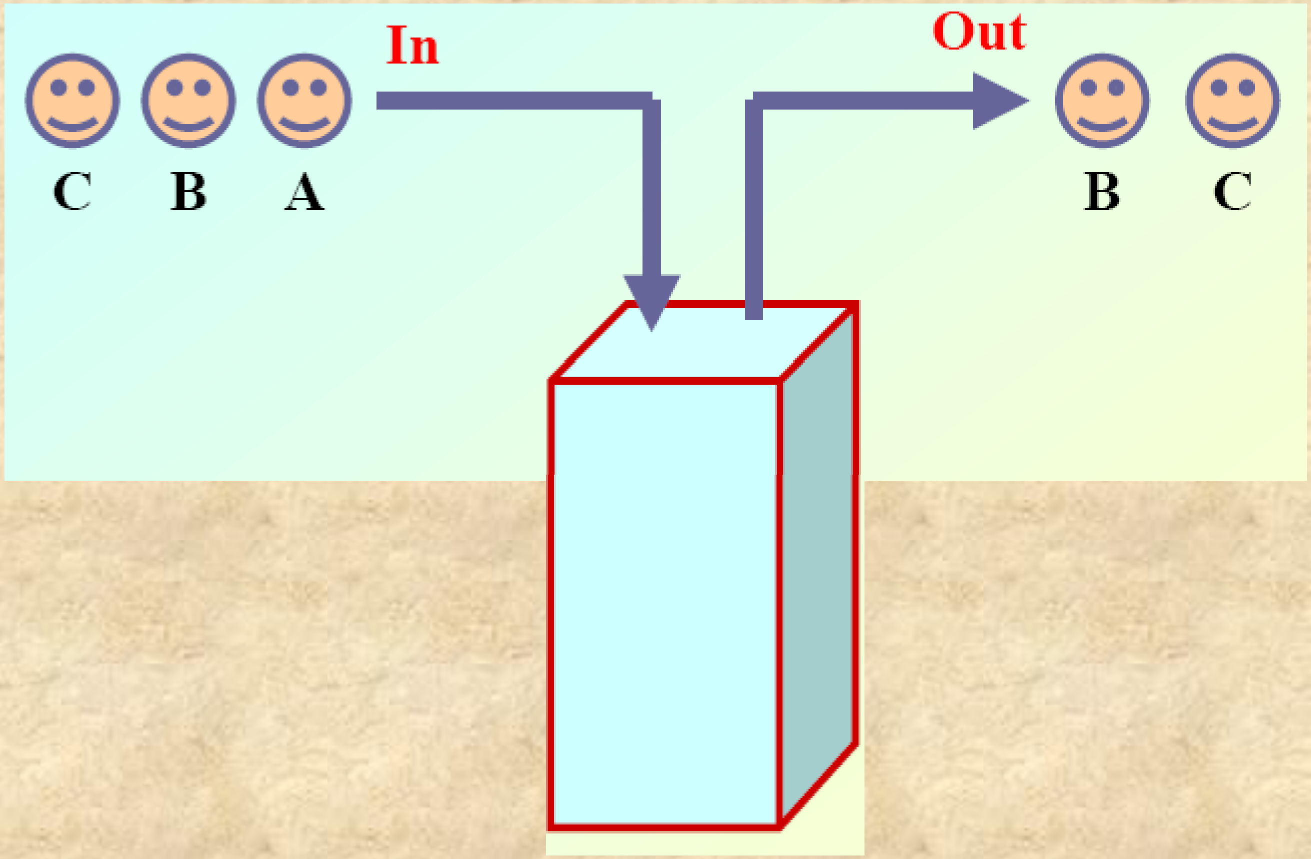
Stacks

- A stack is a type of data structure that allows you access to a single element at any given time: the very last element added to it. This is what is called as a lifo structure (last-in first-out).
- The stack is an ordered list but the insertions and deletions are restricted by certain rules.
- Objects can be inserted at any time, but only the last (the most-recently inserted) object can be removed.
- • Inserting an item is known as "pushing" onto the stack. "Popping" off the stack is synonymous with removing an item.

Abstract Data Type

- What is NOT an Abstract Data Type (in C)?
  - Int - int is a built-in type in the C language
  - Double- double is a built-in type in the C language
- There are certainly many others we can list
- These data types are already built into the language.
- It is a data type that is NOT built into the language
  - It is a data type that we will "build".
    - We will specify what it is, how it is used, etc.
  - It is often **defined in terms of its behavior** rather than its implemented representation
- Nice definition from Wikipedia:
  - An abstract data type is defined indirectly, only by the operations that may be performed on it (i.e. behavior)

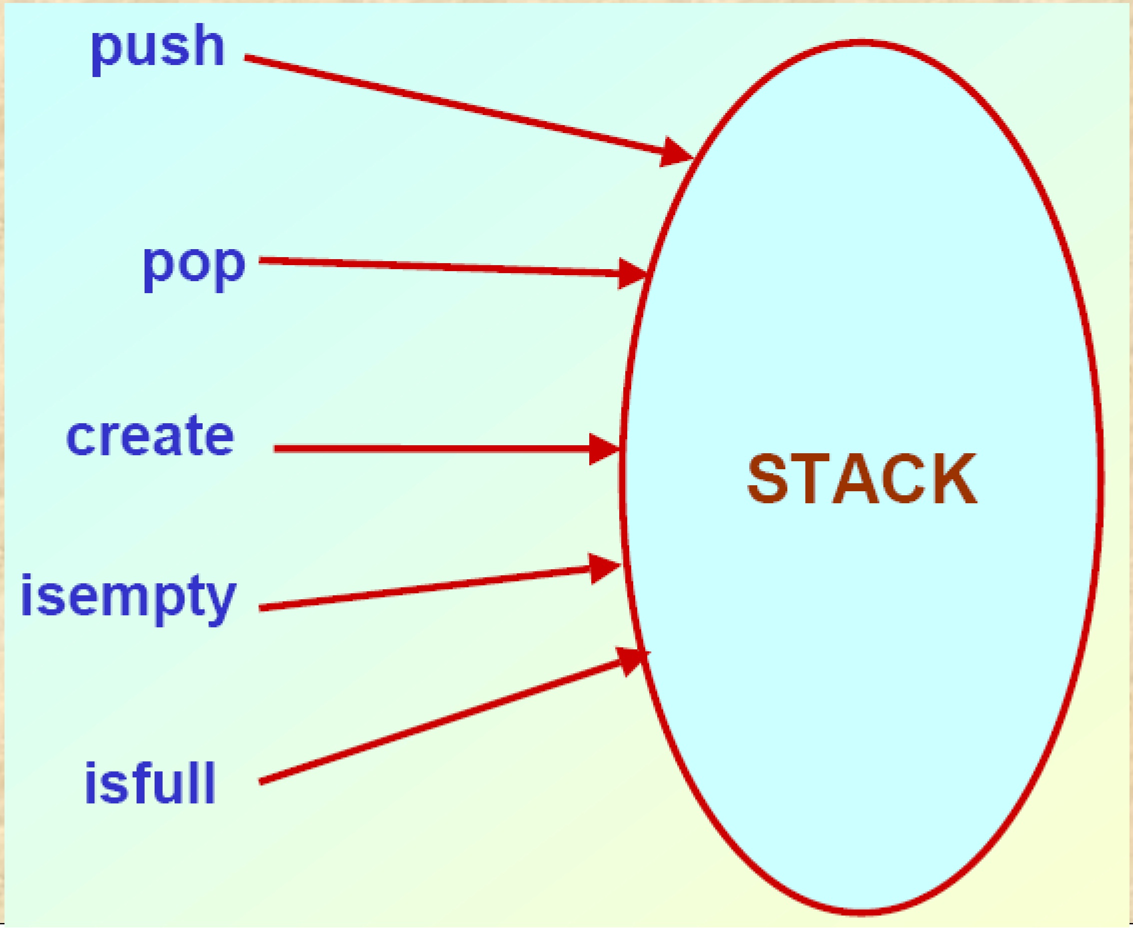
Stacks cont.,



Introduction to Stacks

- Using stack we can perform the following operations:
  - Add data, one item at a time, **Push**
  - Access the data item on the top (other data items are hidden from view), **Top**
  - Remove and discard the data item on the top, thus exposing the next item which now becomes the top item, **Pop**.
- The order in which data in added ("pushed"), accessed and removed ("popped") is LIFO (last in, first out).
- This is different from a queue, in which it's FIFO (first in, first out).
- Stacks can be implemented based on arrays or linked lists.

Operations on Stack



Functions for stack operations

- **Assume:: stack contains integer elements**

```
void push (stack *s, int element);
/* Insert an element in the stack */

int pop (stack *s);
/* Remove and return the top element */

void create (stack *s); /* Create a new stack */

int isempty(stack *s); /* Check if stack is empty
  */

int isfull(stack *s); /* Check if stack is full */
```

Implementation of stack

- **In the array implementation, we would:**
  - **Declare an array of fixed size (which determines the maximum size of the stack).**
  - **Keep a variable which always points to the "top" of the stack.**
  - **Contains the array index of the "top" element.**

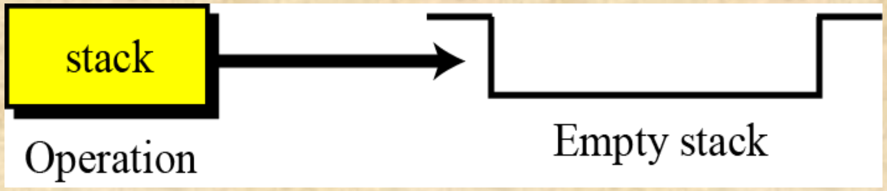
- There are three basic operations on a stack: push onto a stack, pop off an element from a stack, and verify whether the stack is empty or not. Here we will give pseudocode for stack operations.
- a) push operation
    - Algorithm push(newElement)
        - //increment index to the top element;
        - topElementIndex = topElementIndex + 1
        - stack[topElementIndex] = newElement
- b) pop operation
  - Algorithm pop()
    - elementToReturn = stack[topElementIndex]
    - topElementIndex = topElementIndex - 1
    - return elementToReturn
- c) isEmpty operation
  - Algorithm isEmpty()
    - if topElementIndex = -1
      - return true
    - else return false

Declaration

```
#define MAXSIZE 100

       int st[MAXSIZE];
       int top;
```

Stack Creation



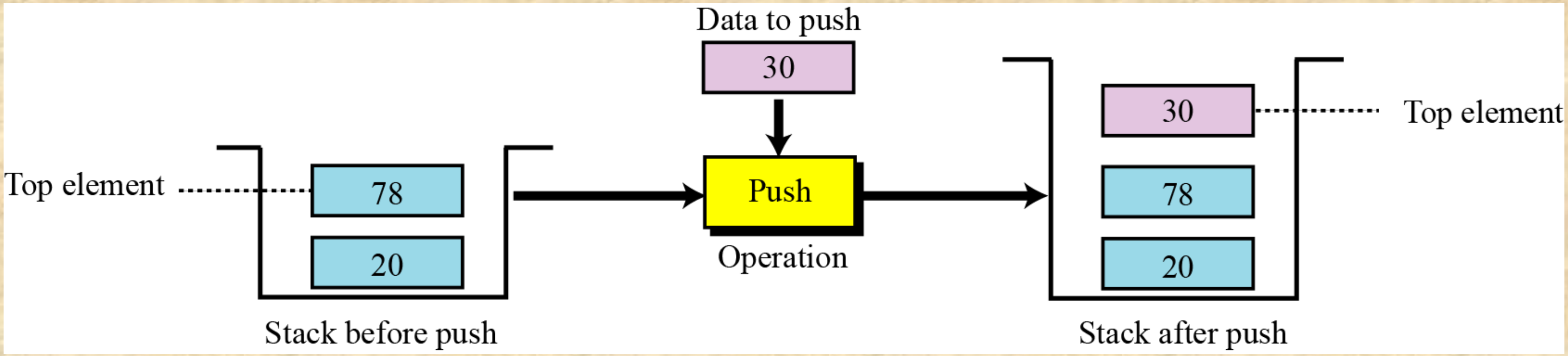
```
void create ()
{
   top = -1;


}
```

Top variable that holds the recently inserted element within it.

Pushing an element into the stack (Array)

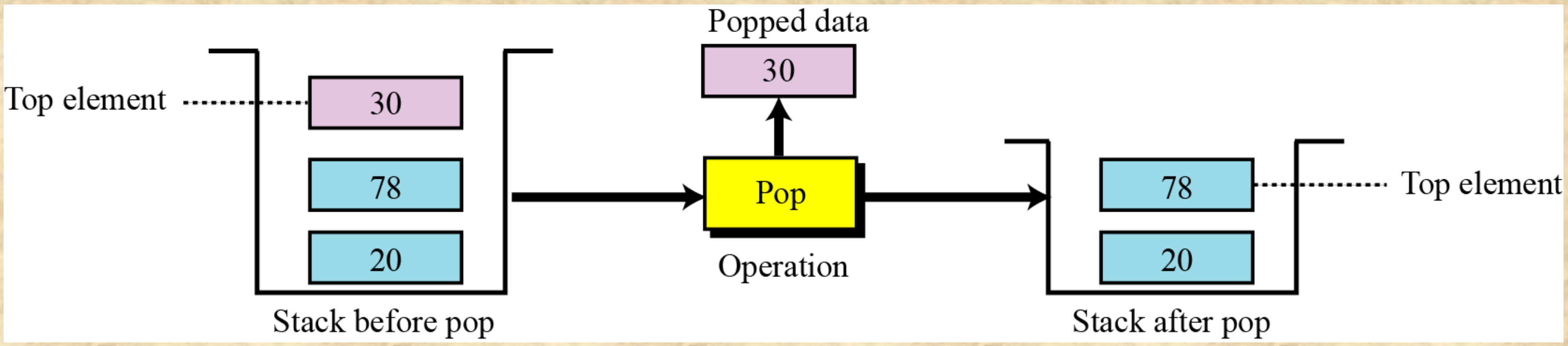
push (dataItem)



Pushing an element into the stack (Array)

```
void push (int element)
{
  if (top == (MAXSIZE-1))
  {
      printf("\n Stack overflow");
      exit(-1);
  }
  else
  {
      top ++;
      st[top] = element;
  }
}
```

Popping an element from the stack (Array)



Popping an element from the stack (Array)

```
int pop ()
{
   if (top == -1)
   {
         printf("\n Stack underflow");
         exit(-1);
   }
   else
   {
         return (st[top--]);
   }
}
```

Checking for stack empty

```
int isempty()
{
      if (top == -1)
            return 1;
      else
            return (0);
}
```

Checking for stack full (Array)

```
int isfull()
{
      if (top == (MAXSIZE-1))
            return 1;
      else
            return (0);
}
```

Applications of stacks

- **Arithmetic expression evaluation:**
  - An important application of stacks is in *parsing*.
  - In high level languages, infix notation cannot be used to evaluate expressions.
  - A common technique is to convert a infix notation into postfix notation, then evaluating it.
- Java bytecode is interpreted on (virtual) stack based processor.

Applications of stacks

- **Reversing Data:** We can use stacks to reverse data. (example: files, strings). Very useful for finding palindromes.
- **Backtracking** : Stacks can be used to backtrack to achieve certain goals.
- **Function calls:** Perhaps the most important application of stacks is to implement function calls. Most compilers implement function calls by using a stack.

Reversing

- Reversing data items requires that a given set of data items be reordered so that the first and last items are exchanged, with all of the positions between the first and last also being relatively exchanged. For example, the list (2, 4, 7, 1, 6, 8) becomes (8, 6, 1, 7, 4, 2).