Unit 2.1: The Stack

The Stack

a. concept and definition

- primitive operations
- Stack as an ADT
- Implementing PUSH and POP operation
- Testing for overflow and underflow conditions

b. The infix, postfix and prefix

- Concept and definition
- Evaluating the postfix operation
- Converting from infix to postfix

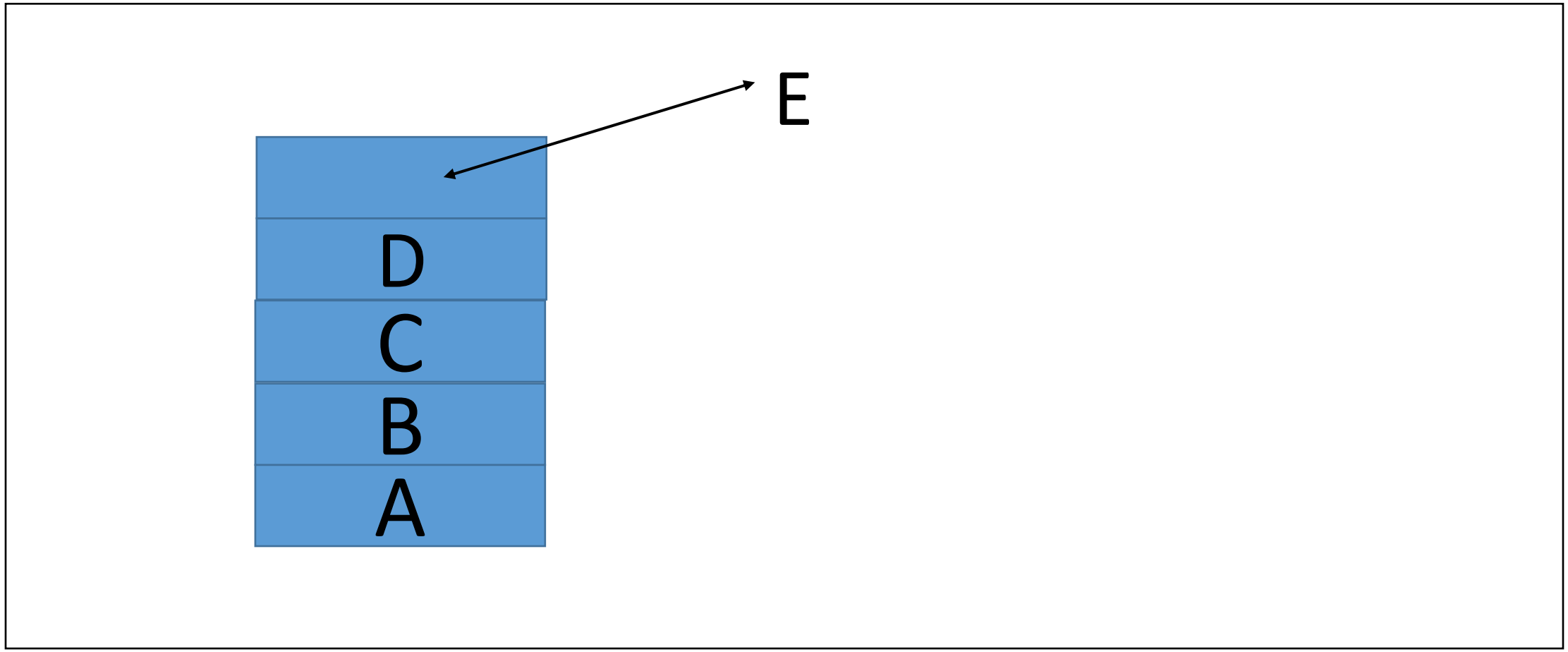
c. Recursion

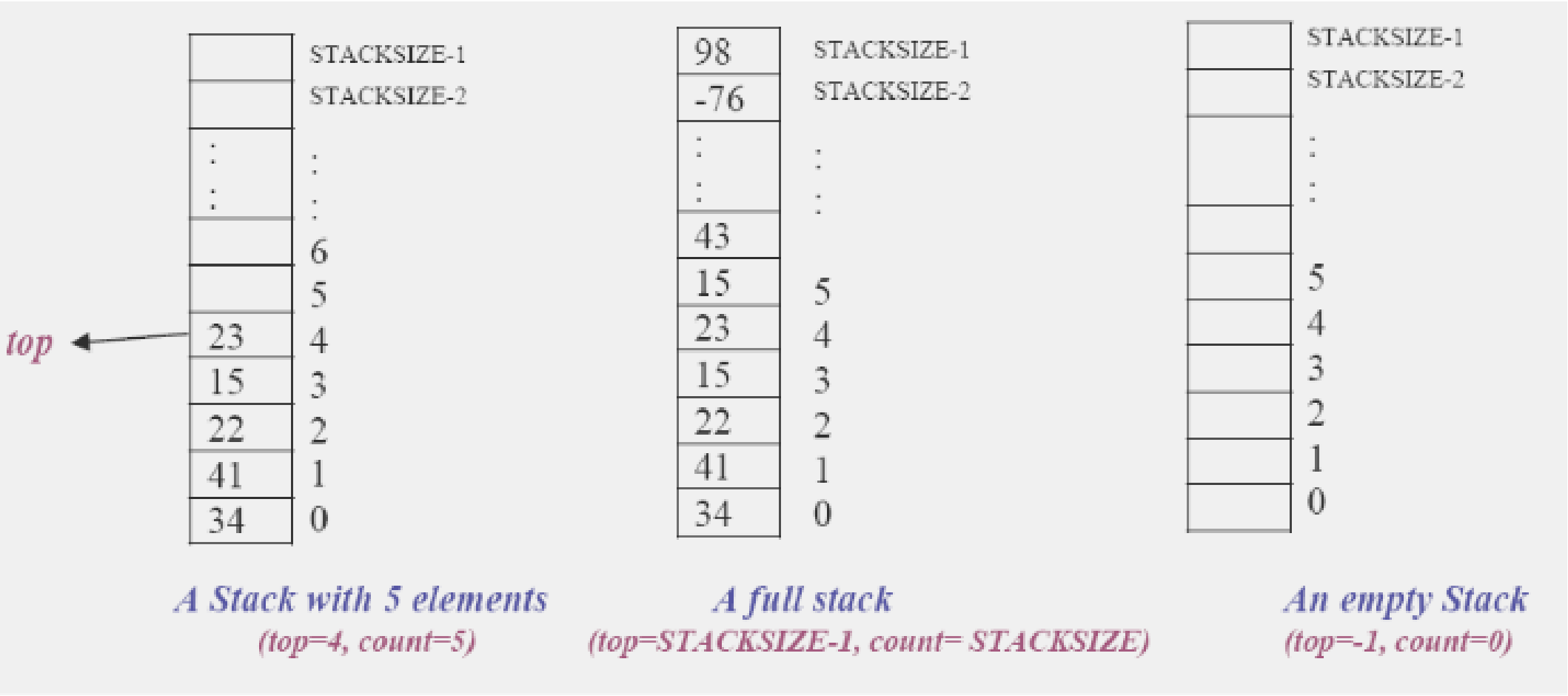
- Concept and definition
- Implementation of:
  - ❑ Multiplication of natural numbers
  - ❑ Factorial
  - ❑ Fibonacci sequences
  - ❑ The tower of Hanoi

Stack: Introduction

- A stack is a **linear data structure** which can be accessed only at one of its ends (called as top) for storing and retrieving data.
  - Eg. Basket pilled in shopping malls.
    - New baskets are put on the top of the stack and taken off the top.
    - The last baskets put on the stack are first baskets to be taken off from the stack.
- Therefore stack is called an LIFO structure: Last in / first out.

LIFO




STACKSIZE-1
STACKSIZE-2
top
23
15
22
41
34
6
5
4
3
2
1
0
A Stack with 5 elements
(top=4, count=5)
98
-76
43
15
23
15
22
41
34
A full stack
(top=STACKSIZE-1, count= STACKSIZE)
An empty Stack
(top=-1, count=0)

Primitive operations

- The PUSH and the POP operations are the basic or primitive operations on a stack.

- Push(el) – Put the element el on the top of the stack
- Pop() – take the topmost element from the stack
- Clear() –Clear the stack.
- isEmpty() – Check to see if the stack is empty
- topEl() – Return the topmost element in the stack without removing it.

Applications of Stack:

- To evaluate the expressions (postfix, prefix)
- To keep the page-visited history in a Web browser
- To perform the undo sequence in a text editor
- Used in recursion
- To pass the parameters between the functions in a C program
- Can be used as an auxiliary data structure for implementing algorithms
- Can be used as a component of other data structures

The Stack ADT:

- A stack of elements of type T is a finite sequence of elements of T together with the operations
  - CreateEmptyStack(S): Create or make stack S be an empty stack
  - Push(S, x): Insert x at one end of the stack, called its top
  - Top(S): If stack S is not empty; then retrieve the element at its top
  - Pop(S): If stack S is not empty; then delete the element at its top
  - IsFull(S): Determine if S is full or not. Return true if S is full stack; return false otherwise
  - IsEmpty(S): Determine if S is empty or not. Return true if S is an empty stack; return false otherwise.

Implementation of Stack:

- Stack can be implemented in two ways:
    1. Array Implementation of stack (or static implementation)
    2. Linked list implementation of stack (or dynamic)

Array Implementation

- Uses one dimensional array to store data
- top is an integer value (an index of an array) that indicates the top position of a stack
- Each time data is added or removed, top is incremented or decremented accordingly
- By convention, in C implementation the empty stack is indicated by setting the value of top to -1 .

  top=-1

Array Implementation

```
#define MAX 10
struct stack
{
        int items[MAX]; //Declaring an array to store items
        int top; //Top of a stack
};
typedef struct stack st;
```

Creating Empty stack :

The value of top=-1 indicates the empty stack in C implementation.

```
/*Function to create an empty stack*/
void create_empty_stack(st *s)
{
        s->top=-1;
}
```

Stack Empty or Underflow:

- This is the situation when the stack contains no element.
- At this point the top of stack is present at the bottom of the stack.
- In array implementation of stack, conventionally top=-1 indicates the empty.

The following function return 1 if the stack is empty, 0 otherwise.

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

Stack Full or Overflow

- This is the situation when the stack becomes full, and no more elements can be pushed onto the stack.
- At this point the stack top is present at the highest location (MAXSIZE-1) of the stack.

The following function returns true (1) if stack is full false (0) otherwise.

```
int isfull(st *s)
{
        if(s->top==MAX-1)
        return 1;
        else
        return 0;
}
```

Algorithm for PUSH operation on Stack

Algorithm for PUSH (inserting an item into the stack) operation:

This algorithm adds or inserts an item at the top of the stack

1.[Check for stack overflow?]

```
        if top=MAXSIZE-1 then
                print “Stack Overflow” and Exit
        else
                Set top=top+1 [Increase top by 1]
        Set Stack[top]:= item [Inserts item in new top position]
```

2. Exit

Algorithm for POP (removing an item from the stack) operation

This algorithm deletes the top element of the stack and assign it to a variable item

1. [Check for the stack Underflow]
    If top<0 then
        Print “Stack Underflow” and Exit
    else
    [Remove the top element]
        Set item=Stack [top]
    [Decrement top by 1]
    Set top=top-1
    Return the deleted item from the stack
2. Exit

The PUSH function

```
void push()
{
        int item;
        if(top == MAXSIZE - 1)                  //Checking stack overflow
                printf("\n The Stack Is Full");
        else
        {
                printf("Enter the element to be inserted");
                scanf("%d", &item);             //reading an item
                top= top+1;                     //increase top by 1
                stack[top] = item;              //storing the item at the top of the stack
        }
}
```

The POP Function

```
void pop()
{
        int item;
        if(top = -1) //Checking Stack Underflow
                printf("The stack is Empty");
        else
        {
                item = stack[top]; //Storing top element to item variable
                top = top-1; //Decrease top by 1
                printf(“The popped item is=%d”,item); //Displaying the deleted item
        }
}
```