Module Code: CSE401

Session 11: Stacks & Queues

Session Speaker:

Prema Monish
premamonish.tlll@msruas.ac.in

Session Objective

- At the end of this lecture, student will be able to
  - explain the idea of a data structure
  - use the concepts of primitive and derived data structures
  - use the structure and operations of a ***queue*** data structure
  - use the structure and operations of a ***stack*** data structure

Session Topic

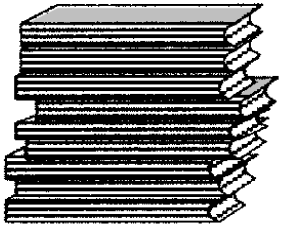
- Stack
- Stack operation
- Application of stack
- Queue operation

Data Structure

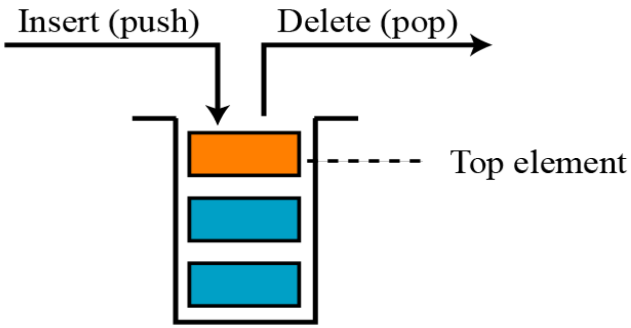
- Data structure is a method of storing data in a computer so that it can be used efficiently.
- Data structure is further classified into –
  - Linear-Arrays,Stacks,Queues,Linked lists
  - Non-linear –Trees,Graphs

STACKS

- A stack is a special type of data structure where elements are inserted from one end and elements are deleted from the same end.
- The position where elements are inserted and deleted is called ***top of stack***
- Hence, stack is also called ***Last In First Out (LIFO)***



Stack of books



Computer stack

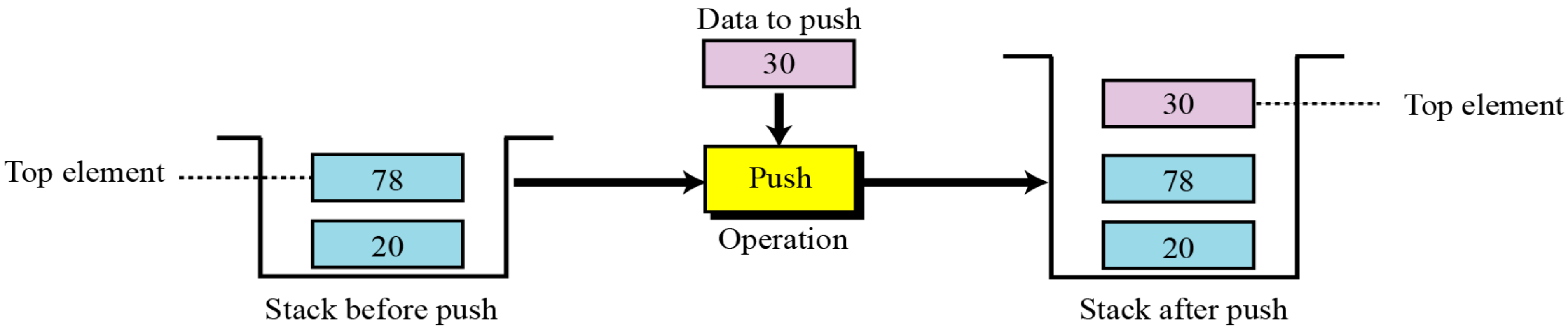
representations of stacks

Operations on stacks

- There are three basic operations:-
  1. Push Operation
  2. Pop Operation
  3. Display Operation

Push Operation

- The ***push operation*** inserts an item at the top of the stack. The following shows the format.
- Trying to insert a item when stack is full results in ***stack overflow***

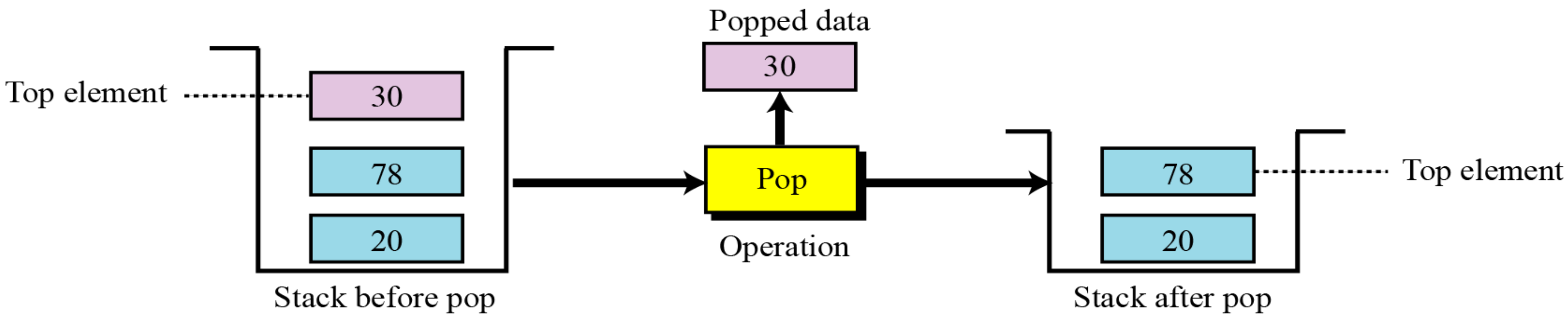


Push Operation

```
void push()//using global variables
{
       if(top==STACK_SIZE-1)
       {
             printf(“stack overflow\n”);
             return;
       }
top=top+1;
s[top]=item;
}
```

pop operation

The pop operation deletes the item at the top of the stack. The following shows the format.



Pop operation

pop operation

```
int pop()//using global variables
{
   int item_deleted ;
   if(top==-1)
   {
      printf(“stack is empty\n”);
      return 0;//stack underflow
      }
      item_deleted=s[top--];
      return item_deleted;
}
```

display operation

- Displays elements in the stack

```
void display()
{
   int i;
   if(top==-1)
   {
      printf(“stack is empty”);
   }
   printf(“contents of the stack”);
   for(i=0;i<=top;i++)
      printf(“%d\n”,s[i]);
}
```

Stack applications

- Stack applications can be classified into four broad categories:
  - Conversion of expression
  - Evaluation of expression
  - Recursion
  - Reverse string
- Representation of expressions –
  - Infix expression – e.g. a+b
  - Postfix expression-e.g. ab+
  - Prefix expression-e.g. +ab

Precedence & Associativity

Operators are ordered based on priority
Always Braces () as highest priority

Operators	Associativity
*,/,%	Left to Right
+,-	Left to Right

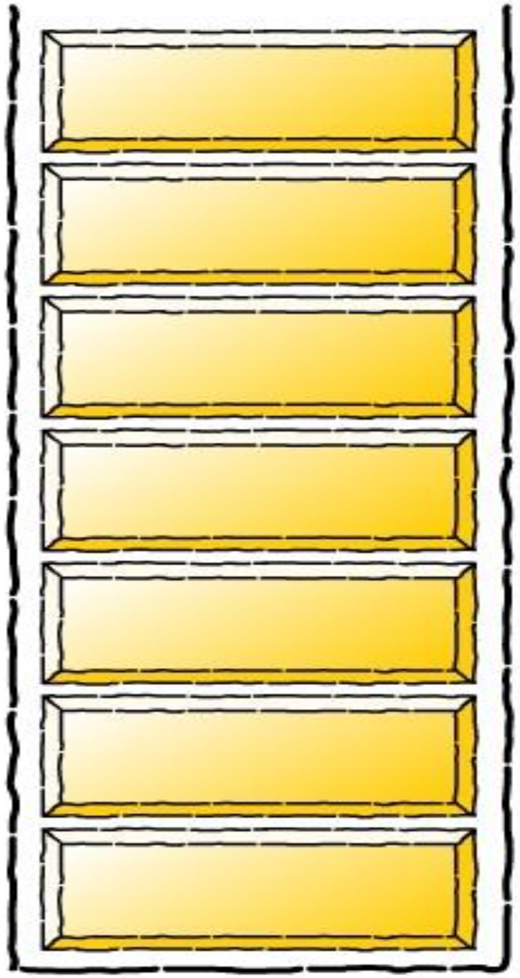
Infix to postfix conversion

To convert Infix Expression into Postfix Expression using a stack data structure, We can use the following steps...

- Read all the symbols one by one from left to right in the given Infix Expression.
- If the reading symbol is operand, then directly print it to the result (Output).
- If the reading symbol is left parenthesis '(', then Push it on to the Stack.
- If the reading symbol is right parenthesis ')', then Pop all the contents of stack until respective left parenthesis is popped and print each popped symbol to the result.
- If the reading symbol is operator (+ , - , * , / etc.,), then Push it on to the Stack. However, first pop the operators which are already on the stack that have higher or equal precedence than current operator and print them to the result.

Infix to postfix conversion

stack



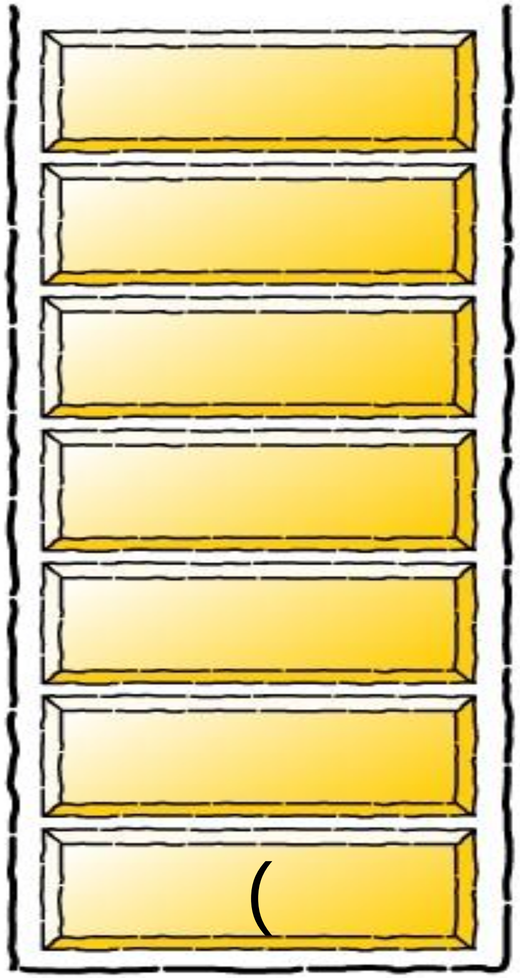
infix Expression

(a + b - c) * d – (e + f)

postfix Expression

Infix to postfix conversion

stack



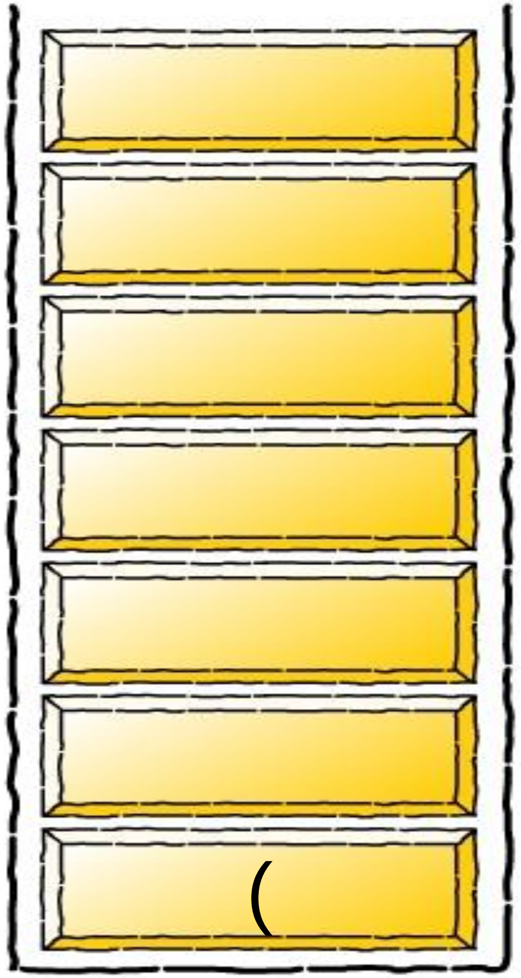
infix Expression

a + b - c) * d – (e + f)

postfix Expression

Infix to postfix conversion

stack



infix Expression

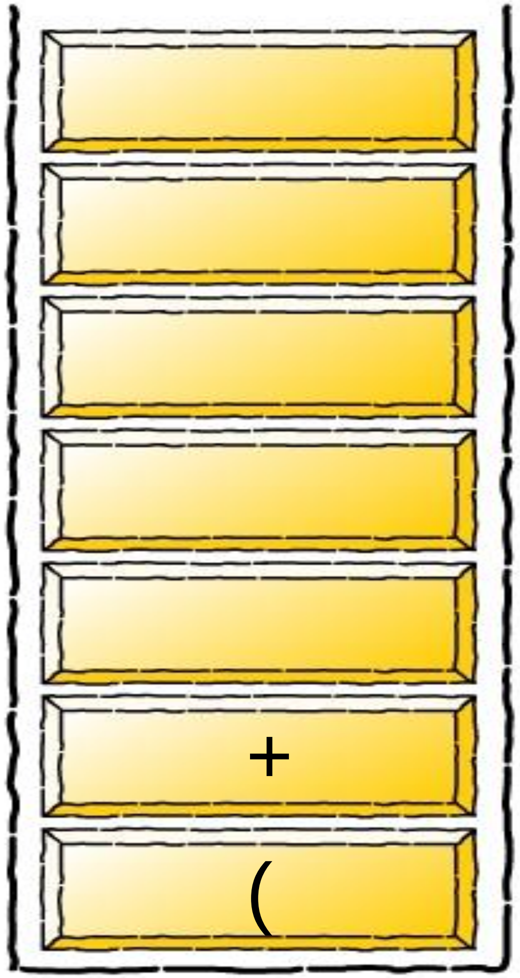
+ b - c) * d – (e + f)

postfix Expression

a

Infix to postfix conversion

stack



infix Expression

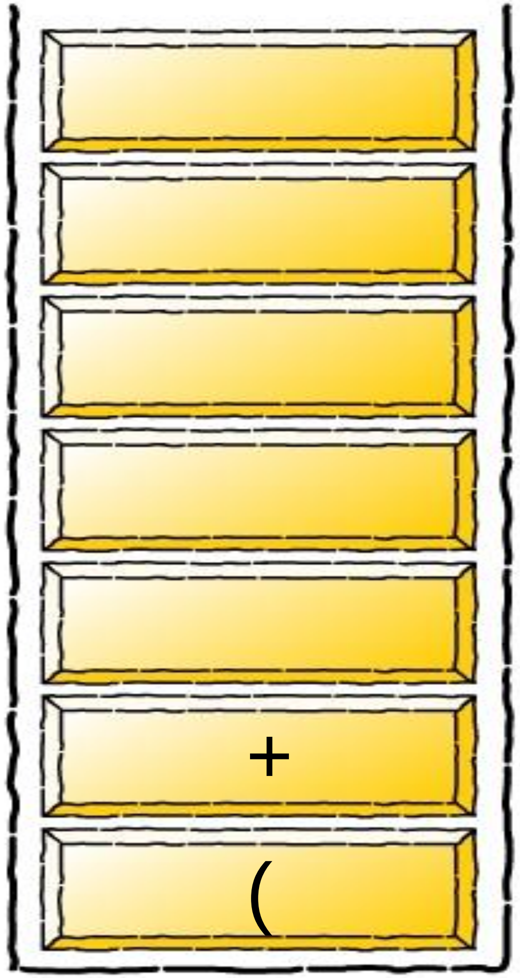
b - c) * d – (e + f)

postfix Expression

a

Infix to postfix conversion

stack



infix Expression

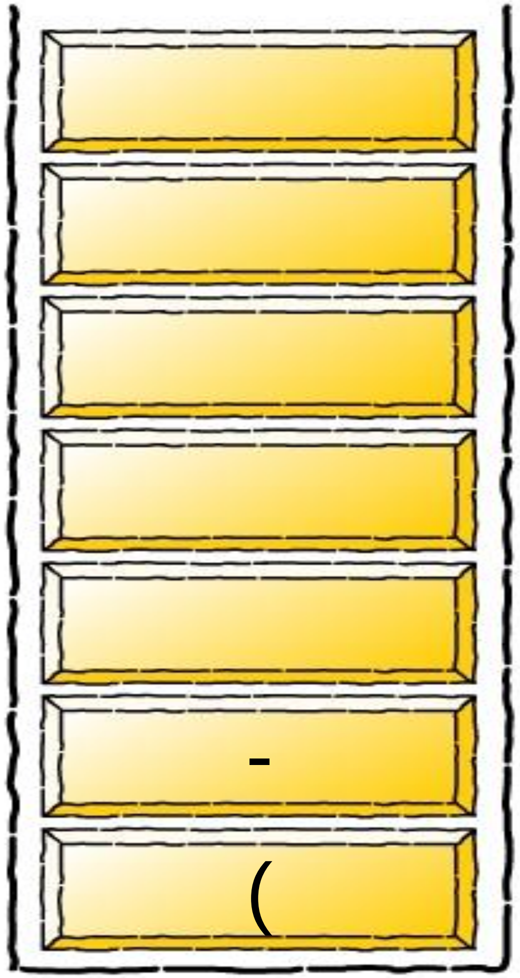
- c) * d – (e + f)

postfix Expression

a b

Infix to postfix conversion

stack



infix Expression

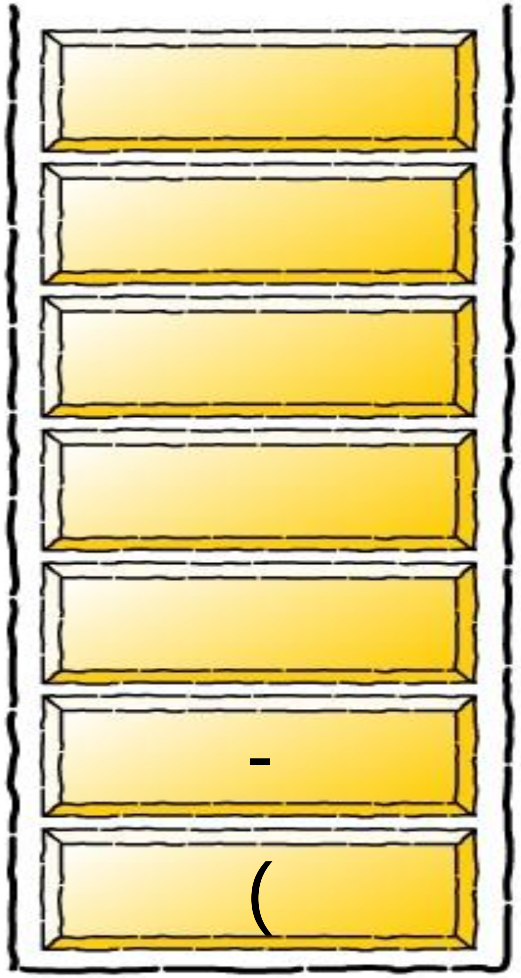
c) * d – (e + f)

postfix Expression

a b +

Infix to postfix conversion

stack



-

(

infix Expression

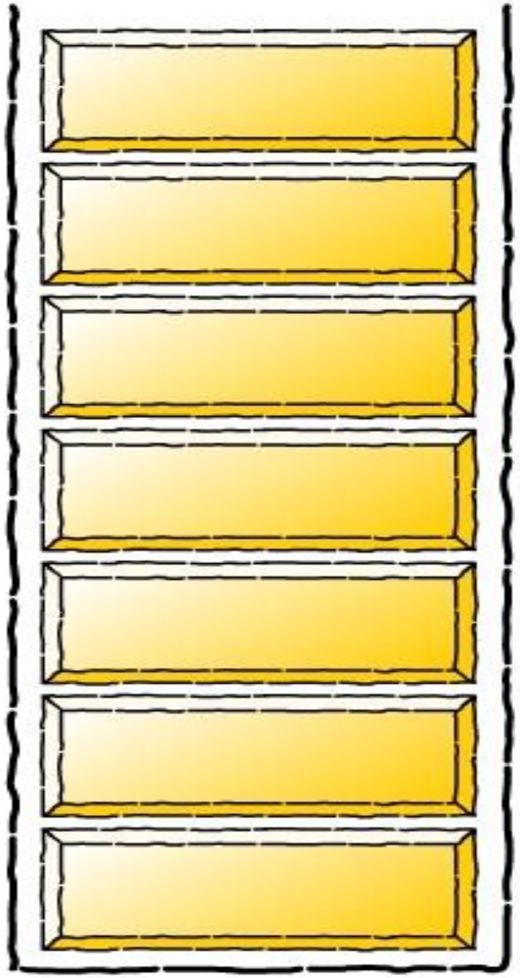
) * d – (e + f)

postfix Expression

a b + c

Infix to postfix conversion

stack



infix Expression

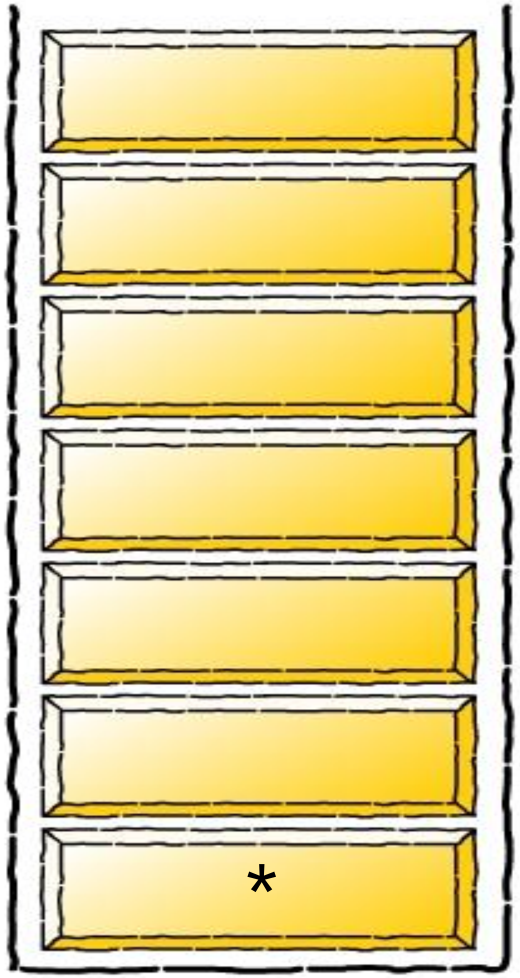
* d – (e + f)

postfix Expression

a b + c -

Infix to postfix conversion

stack



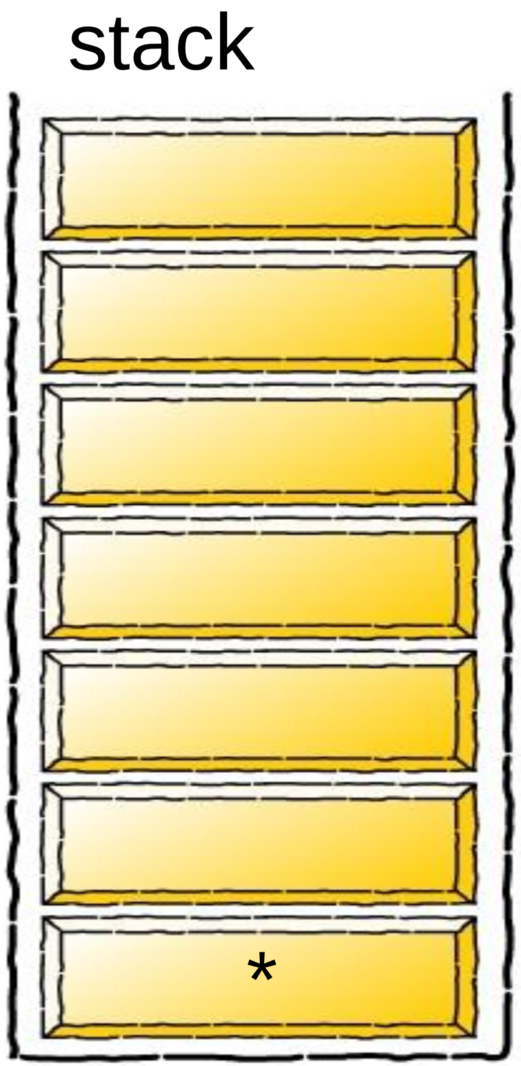
infix Expression

d – (e + f)

postfix Expression

a b + c -

Infix to postfix conversion



infix Expression

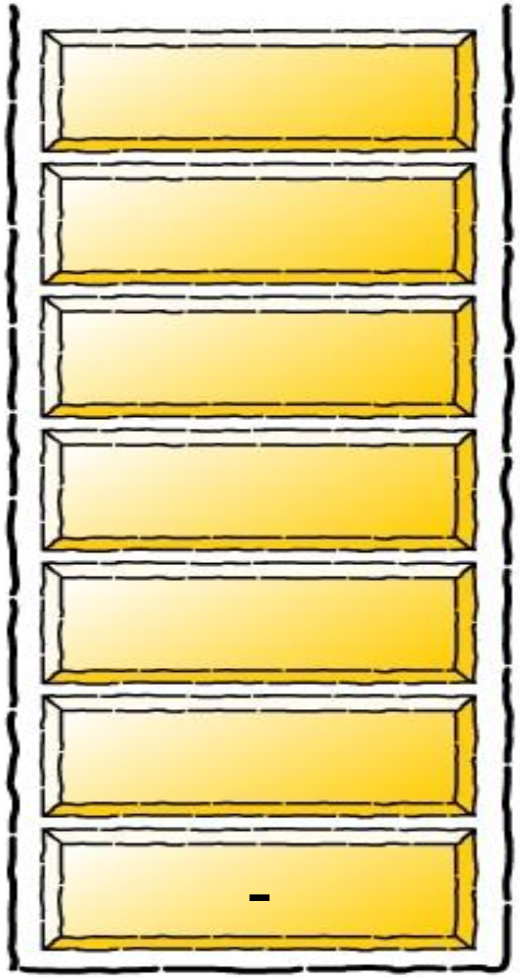
– (e + f)

postfix Expression

a b + c - d

Infix to postfix conversion

stack



infix Expression

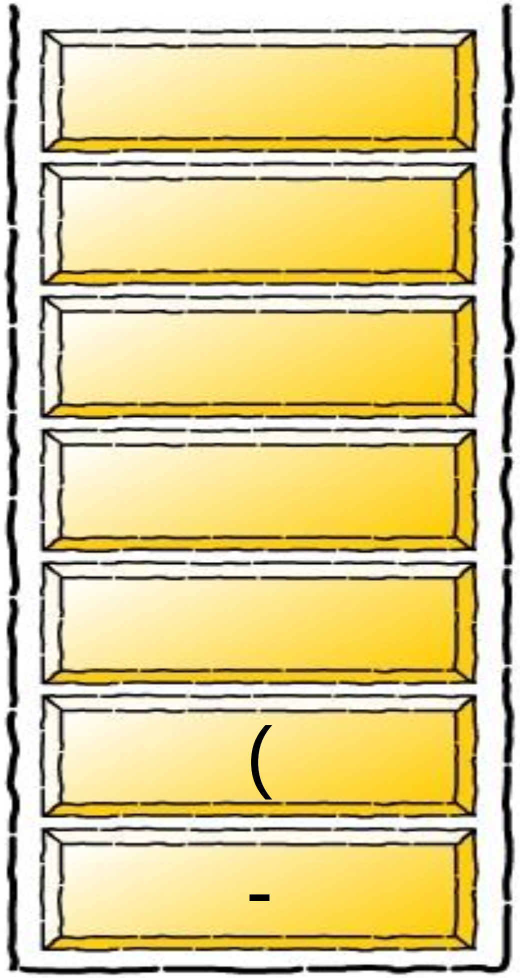
(e + f)

postfix Expression

a b + c – d *

Infix to postfix conversion

stack



infix Expression

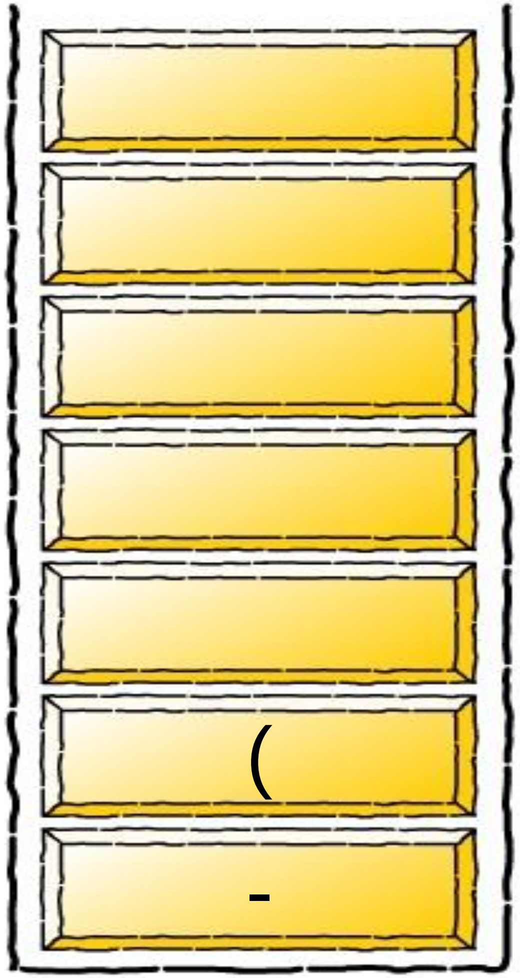
e + f)

postfix Expression

a b + c – d *

Infix to postfix conversion

stack



infix Expression

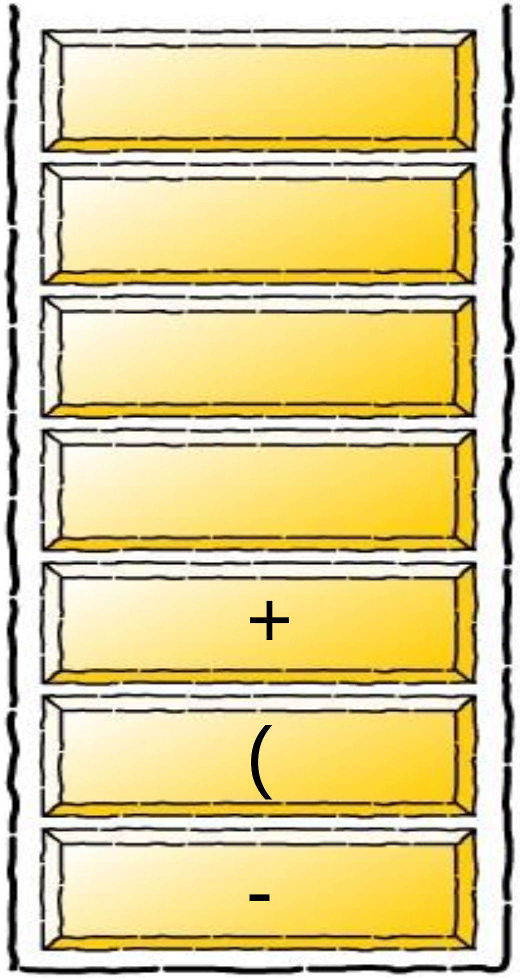
+ f)

postfix Expression

a b + c – d * e

Infix to postfix conversion

stack



infix Expression

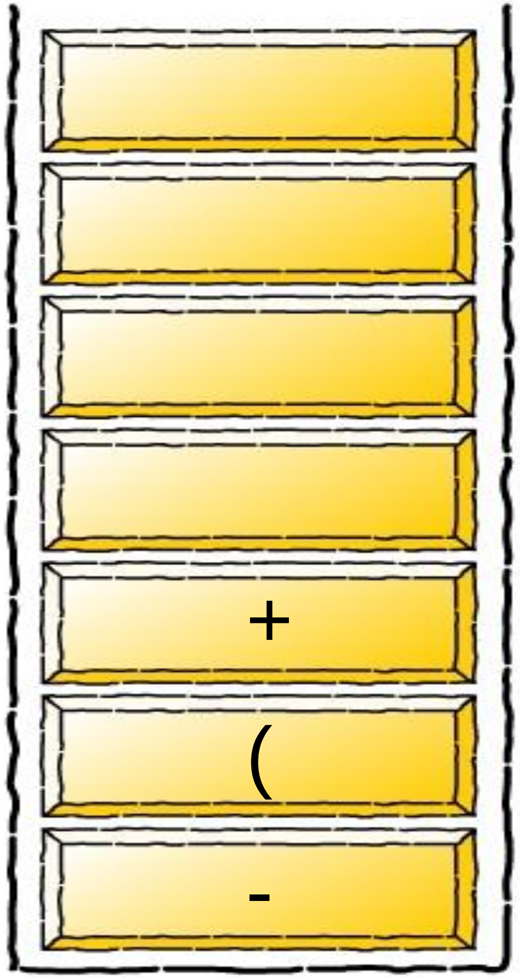
f)

postfix Expression

a b + c – d * e

Infix to postfix conversion

stack



infix Expression

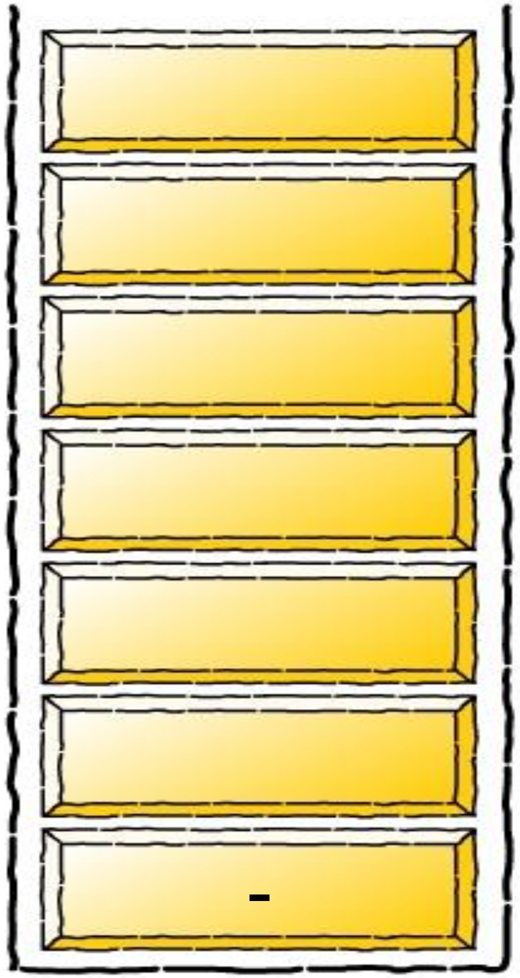
)

postfix Expression

a b + c – d * e f

Infix to postfix conversion

stack



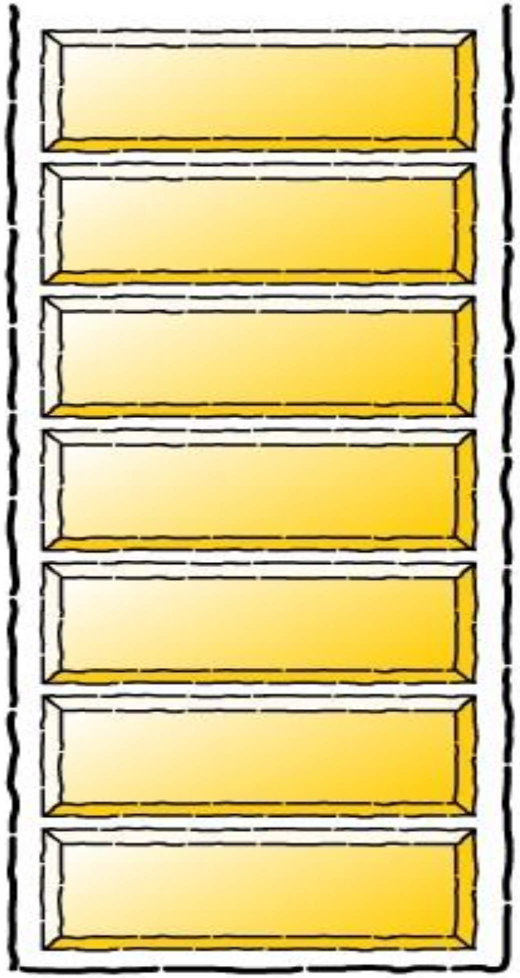
infix Expression

postfix Expression

a b + c – d * e f +

Infix to postfix conversion

stack



infix Expression

postfix Expression

a b + c – d * e f + -

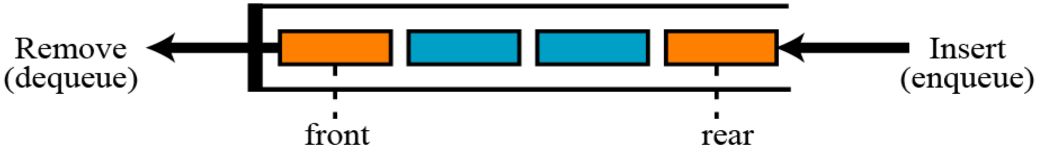
postfix to Infix conversion

- Read all the symbols one by one from left to right in the given Postfix Expression
- If the reading symbol is operand, then push it on to the Stack.
- If the reading symbol is operator (+ , - , * , / etc.,), then perform TWO pop operations and store the two popped operands in two different variables (operand1 and operand2). Then perform reading symbol operation using operand1 and operand2 and push result back on to the Stack.
- Finally! perform a pop operation and display the popped value as final result.

QUEUES

- A queue is a linear list in which data can only be inserted at one end, called the *rear*, and deleted from the other end, called the *front*.
- queue is a first in, first out (FIFO) structure.

A queue of people



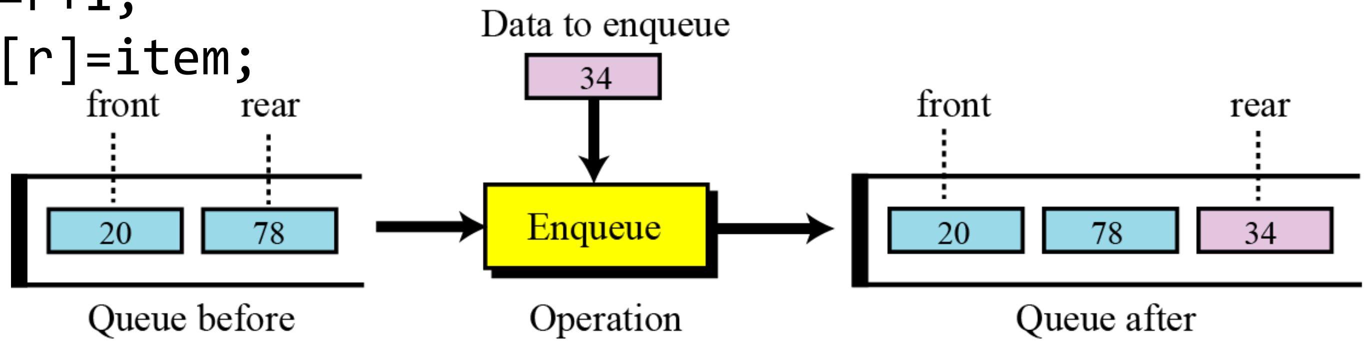
A computer queue

Queues Operation

- There are three basic operations:-
  1. Insert an item into queue from rear end
  2. Delete an item in queue from front end
  3. Display contents of queue

Inserting into queue from rear end

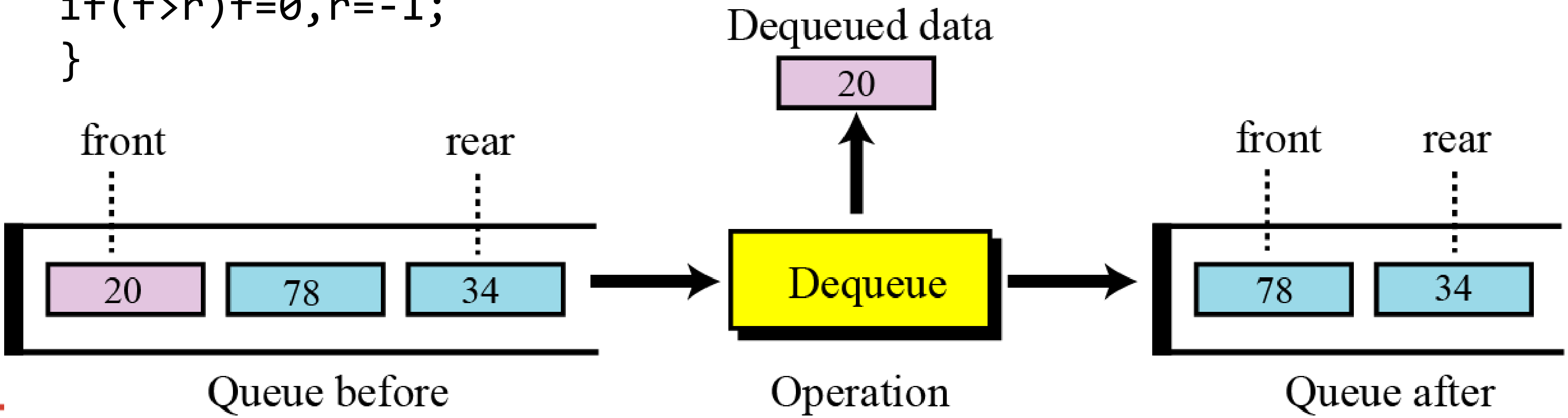
```
void insert()//using global variables
{
        if(r==QUEUE_SIZE-1)
        {
        printf(“queue overflow”);
        return;
        }
r=r+1;
q[r]=item;
}
```



- Inserting element into queue.

Delete an item in queue from front end

```
void delete()//using global variables
{
      if(f>r)
      {
            printf(“queue underflow\n”);
      }
      printf(“element deleted is %d”,q[f++]);
if(f>r)f=0,r=-1;
}
```



- Deleting item from queue.

summary

- Stack is last In first out.
- Stack operations-push,pop,display
- Queue is first in last out.
- Queue operations-insert,delete,display