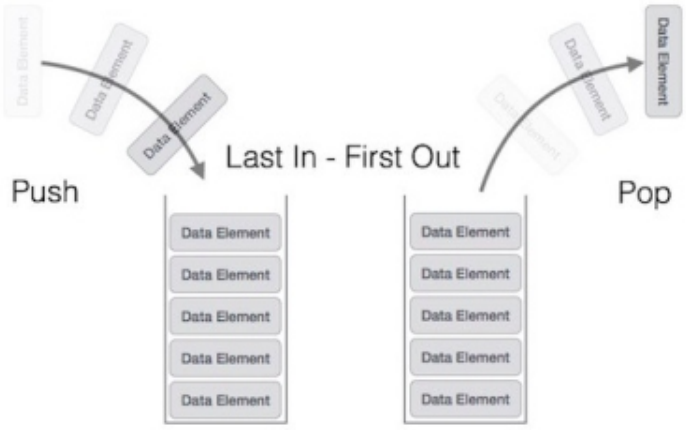
Chapter 4

Stack:
A stack is an Abstract Data Type (ADT), commonly used in most programming languages. It is named stack as it behaves like a real-world stack, for example – a deck of cards or a pile of plates, etc.

A real-world stack allows operations at one end only. For example, we can place or remove a card or plate from the top of the stack only. Likewise, Stack ADT allows all data operations at one end only. At any given time, we can only access the top element of a stack. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and removal operation is called POP operation.

Stack Presentation: The following diagram depicts a stack and its operations –



A stack can b nked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing. Here, we are going to implement stack using arrays, which makes it a fixed size stack implementation.

Basic Operations:
Stack operations may involve initializing the stack, using it and then de-initializing it. Apart from these basic stuffs, a stack is used for the following two primary operations –

push() – Pushing (storing) an element on the stack.

pop() – Removing (accessing) an element from the stack.

To use a stack efficiently, we need to check the status of stack as well. For the same purpose, the following functionality is added to stacks –

peek() – get the top data element of the stack, without removing it.

isFull() – check if stack is full.

isEmpty() – check if stack is empty.

At all times, we maintain a pointer to the last Pushed data on the stack. As this pointer always represents the top of the stack, hence named TOP. The TOP pointer provides top value of the stack without actually removing it.

Procedures to support stack functions –

peek():

Algorithm of peek() function –

```
begin procedure peek

  return stack[top]

end procedure
```

Implementation of peek() function in C programming language –

```
int peek() {
  return stack[top];
}
```

isfull():

Algorithm of isfull() function –

```
begin procedure isfull

  if top equals to MAXSIZE
    return true
  else
    return false
  endif

end procedure
```

Implementation of isfull() function in C programming language –

```
bool isfull() {
  if(top == MAXSIZE)
    return true;
```

```
  else
    return false;
}
```

isempty():
Algorithm of isempty() function –

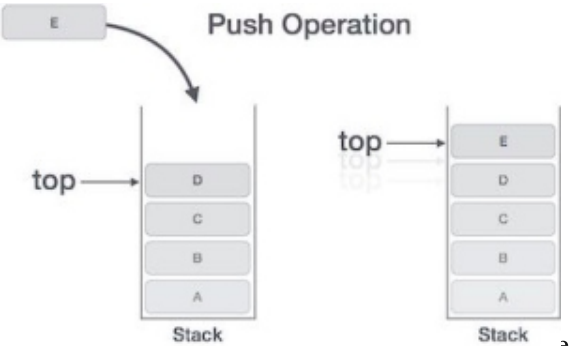
```
begin procedure isempty

  if top less than 1
    return true
  else
    return false
  endif

end procedure
```

Implementation of isempty() function in C programming language is slightly different. We initialize top at -1, as the index in array starts from 0. So we check if the top is below zero or -1 to determine if the stack is empty. Here's the code –

```
bool isempty() {
  if(top == -1)
    return true;
  else
    return false;
}
```

Push Operation: The process of putting a new data element onto stack is known as a Push Operation. Push operation involves a series of steps –

Step 1 – Checks if the stack is full.
Step 2 – If the stack is full, produces an error and exit.
Step 3 – If the stack is not full, increments top to point next empty space.
Step 4 – Adds data element to the stack location, where top is pointing.
Step 5 – Returns success.



If the linked list is used to implement the stack, then in step 3, we need to allocate space dynamically.

Algorithm for Push Operation:

```
begin procedure push: stack, data
  if stack is full
    return null
  endif

  top ←top + 1

  stack[top] ←data

end procedure
```

Implementation of algorithm in C –

```
void push(int data) {
  if(!isFull()) {
    top = top + 1;
    stack[top] = data;
  }else {
    printf("Could not insert data, Stack is full.\n");
  }
}
```

Pop Operation: Accessing the content while removing it from the stack, is known as a Pop Operation. In an array implementation of pop() operation, the data element is not actually removed, instead top is decremented to a lower position in the stack to point to the next value. But in linked-list implementation, pop() actually removes data element and deallocates memory space.

A Pop operation may involve the following steps –

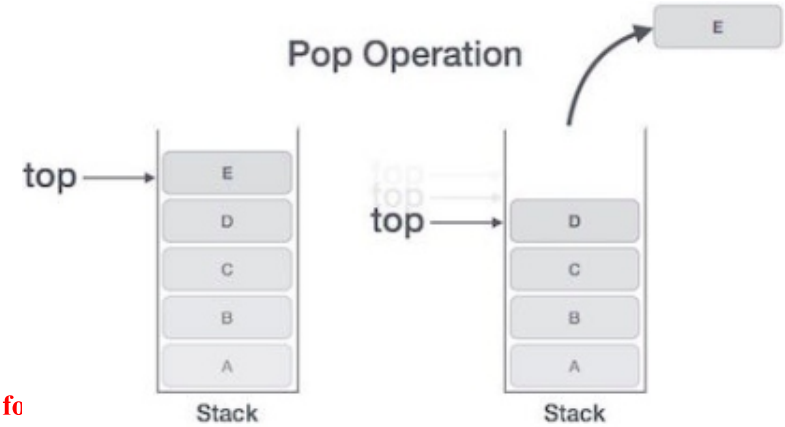
Step 1 – Checks if the stack is empty.

Step 2 – If the stack is empty, produces an error and exit.

Step 3 – If the stack is not empty, accesses the data element at which top is pointing.

Step 4 – Decreases the value of top by 1.

Step 5 – Returns success.



Algorithm fo

```
begin procedure pop: stack

   if stack is empty
      return null
   endif

   data ←stack[top]

   top ←top - 1

   return data

end procedure
```

Implementation of algorithm in C –

```
int pop(int data) {

   if(!isempty()) {
      data = stack[top];
      top = top - 1;
      return data;
   }else {
      printf("Could not retrieve data, Stack is empty.\n");
   }
}
```

Application of Stack: Stack can be used for following purpose-
(a) Arithmetic expression evaluation
(b) Backtracking
(c) Memory Management

(a) Arithmetic expression evaluation: The way to write arithmetic expression is known as a notation. An arithmetic expression can be written in three different but equivalent notations, i.e., without changing the essence or output of an expression. These notations are –

Infix Notation
Prefix (Polish) Notation
Postfix (Reverse-Polish) Notation

These notations are named as how they use operator in expression.

Infix Notation
We write expression in infix notation, e.g. a - b + c, where operators are used in-between operands. It is easy for us humans to read, write, and speak in infix notation but the same does not go well with computing devices. An algorithm to process infix notation could be difficult and costly in terms of time and space consumption.

Prefix Notation
In this notation, operator is prefixed to operands, i.e. operator is written ahead of operands. For example, +ab. This is equivalent to its infix notation a + b. Prefix notation is also known as Polish Notation.

Postfix Notation
This notation style is known as Reversed Polish Notation. In this notation style, the operator is postfixed to the operands i.e., the operator is written after the operands. For example, ab+. This is equivalent to its infix notation a + b.

So the stack is used for conversion an expression from one notation to another notation.

(b) Backtracking: Backtracking is used in algorithms in which there are steps along some path (state) from some starting point to some goal.
Find your way through a maze.
Find a path from one point in a graph (roadmap) to another point.

In all of these cases, there are choices to be made among a number of options. We need some way to remember these decision points in case we want/need to come back and try the other alternative

Consider the maze. At a point where a choice is made, we may discover that the choice leads to a dead-end. We want to retrace back to that decision point and then try the other (next) alternative.

Again, stacks can be used as part of the solution. Recursion is another, typically more favored, solution, which is actually implemented by a stack.

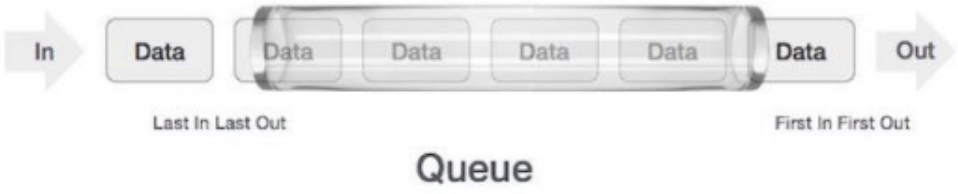
(c) Memory Management: Any modern computer environment uses a stack as the primary memory management model for a running program. Whether it's native code (x86, Sun, VAX) or JVM, a stack is at the center of the run-time environment for Java, C++, Ada, FORTRAN, etc.

Queue:
Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

A r s

first, exits first. More real-world examples can be seen as queues at the ticket windows of bus-stops and others.

Queue presentation:
As we now understand that in queue, we access both ends for different reasons. The following diagram given below tries to explain queue representation as data structure −



As in stacks, a queue can also be implemented using Arrays, Linked lists, Pointers and Structures. For the sake of simplicity, we shall implement queues using one-dimensional array.

Basic Operations:
Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with
queues −

enqueue() − add (store) an item to the queue.

dequeue() − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are −

peek() − Gets the element at the front of the queue without removing it.

isfull() − Checks if the queue is full.

isempty() − Checks if the queue is empty.

supportive functions of a queue −

peek()
The algorithm of peek() function is as follows −

```
begin procedure peek

  return queue[front]

end procedure
```

Implementation of peek() function in C programming language –

Example

```
int peek() {
  return queue[front];
}
```

isfull():

As we are using single dimension array to implement queue, we just check for the rear pointer to reach at MAXSIZE to determine that the queue is full. In case we maintain the queue in a circular linked-list, the algorithm will differ. Algorithm of isfull() function –

```
begin procedure isfull

  if rear equals to MAXSIZE
    return true
  else
    return false
  endif

end procedure
```

Implementation of isfull() function in C programming language –

```
bool isfull() {
  if(rear == MAXSIZE - 1)
    return true;
  else
    return false;
}
```

isempty():

Algorithm of isempty() function –

```
begin procedure isempty

  if front is less than MIN  or front is greater than rear
    return true
  else
    return false
  endif
```

end procedure

If the value of front is less than MIN or 0, it tells that the queue is not yet initialized, hence empty.

Here's the C programming code –

```
bool isempty() {
  if(front < 0 || front > rear)
    return true;
  else
    return false;
```

Enqueue Operation:

Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.

The following steps should be taken to enqueue (insert) data into a queue –

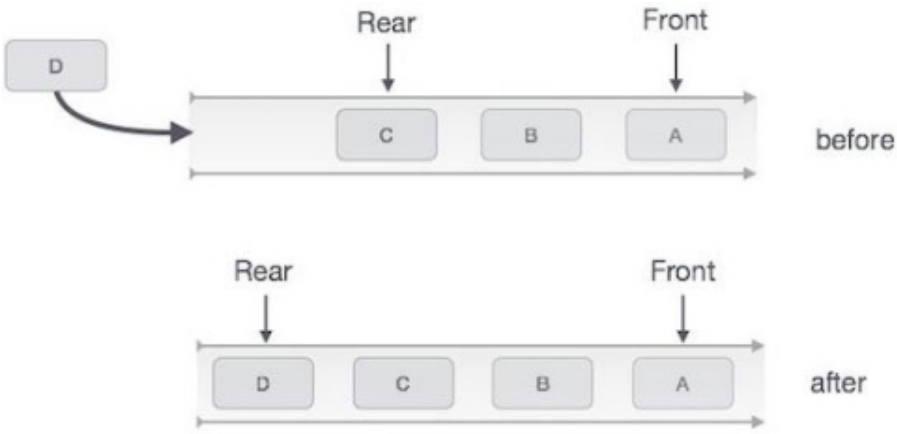
Step 1 – Check if the queue is full.

Step 2 – If the queue is full, produce overflow error and exit.

Step 3 – If the queue is not full, increment rear pointer to point the next empty space.

Step 4 – Add data element to the queue location, where the rear is pointing.

Step 5 – Return success.



Queue Enqueue

Algo

```
procedure enqueue(data)
  if queue is full
    return overflow
  endif
  rear ←rear + 1
  queue[rear] ←data

  return true

end procedure
```

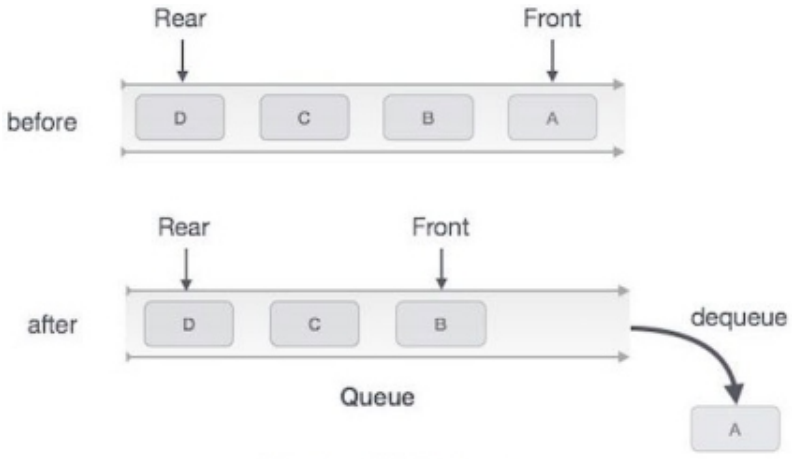
Implementation of enqueue() in C programming language –

```
int enqueue(int data)
  if(isfull())
    return 0;

  rear = rear + 1;
  queue[rear] = data;

  return 1;
end procedure
```

Dequeue Operation:

Accessing data from the queue is a process of two tasks – access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation –

Step 1 – Check if the queue is empty.
Step 2 – If the queue is empty, produce underflow error and exit.
Step 3 – If the queue is not empty, access the data where front is pointing.
Step 3 – Increment front pointer to point to the next available data element.
Step 5 – Return success.



Queue Dequeue

Algorithn

```
procedure dequeue
  if queue is empty
    return underflow
  end if

  data = queue[front]
  front ←front + 1

  return true
end procedure
```

Implementation of dequeue() in C programming language –

```
int dequeue() {

  if(isempty())
    return 0;

  int data = queue[front];
  front = front + 1;

  return data;
}
```

Important Points

- A stack is an Abstract Data Type (ADT), commonly used in most programming languages.
- A stack can be implemented by means of Array, Structure, Pointer, and Linked List. Stack can either be a fixed size one or it may have a sense of dynamic resizing.
- Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends.
- Any modern computer environment uses a stack as the primary memory management model for a running program.

Exercise

Objective type questions.

Q1. Which of the following name does not relate to stacks.

a. FIFO lists
b. LIFO list
c. Pop
d. Push-down lists

Q2. The term "push" and "pop" is related to the

a. array

b. lists
c. stacks
d. all of above

Q3. A data structure where elements can be added or removed at either end but not in the middle.

a. Linked lists
b. Stacks
c. Queues
d. Dequeue

Q4.The data structure required for Breadth First Traversal on a graph is.

a .Stack
b. Array
c. Queue
d. Tree

Q5. A queue is a.

a. FIFO (First In First Out) list
b. LIFO (Last In First Out) list.
c. Ordered array
d. Linear tree

Short answer type questions.

Q1. Define stack ?
Q2. Define Queue ?
Q3. What is Push operation ?
Q4. What is Pop operation ?

Essay type questions.

Q1. What are some of the applications for the stack data structure ?
Q2. Explain stack operations in detail ?
Q3. Explain circular queue in detail ?
Q4. Explain dequeue ?

Answers

Ans1. a Ans2. c Ans3. d
Ans4. c Ans5. a