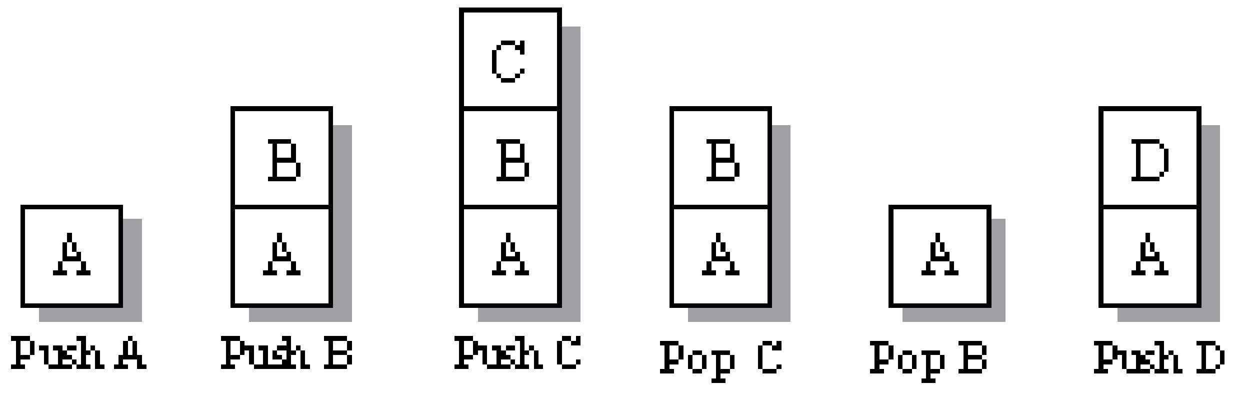
Stack

- Stack
  - New nodes can be added and removed only at the top
  - Similar to a pile of dishes
  - Last-in, first-out (LIFO)
  - Bottom of stack indicated by a link member to **`NULL`**
  - Constrained version of a linked list
- **`push`**
  - Adds a new node to the top of the stack
- **`pop`**
  - Removes a node from the top
  - Stores the popped value
  - Returns **`true`** if **`pop`** was successful

Pushing/Popping a Stack

- Because a pop removes the item last added to the stack, we say that a stack has LIFO (last-in/first-out) ordering.



Applications of Stacks

- Direct applications
  - Page-visited history in a Web browser
  - Undo sequence in a text editor
  - Saving local variables when one function calls another, and this one calls another, and so on.
- Indirect applications
  - Auxiliary data structure for algorithms
  - Component of other data structures

PUSH Pseudo Code:

```
Procedure PUSH (item,  Stack)
Begin
  if (Top=N-1)
      Stack is Full;
Else {
  Top=Top+1;
  Stack[Top]=item;
 }
end
```

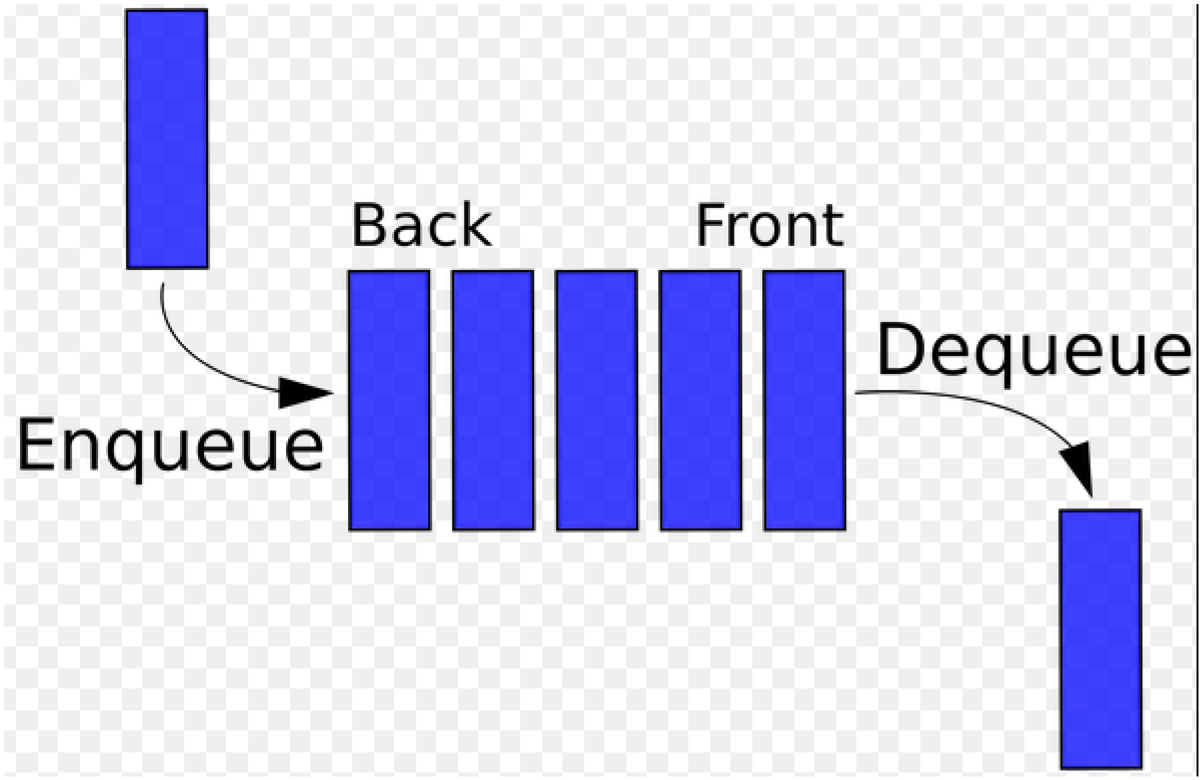
POP Pseudo Code:

```
Procedure POP (item,  Stack)
Begin
  if (Top=-1)
      Stack is Empty;
Else {
  item=Stack[Top];
  Top=Top-1;
 }
end
```

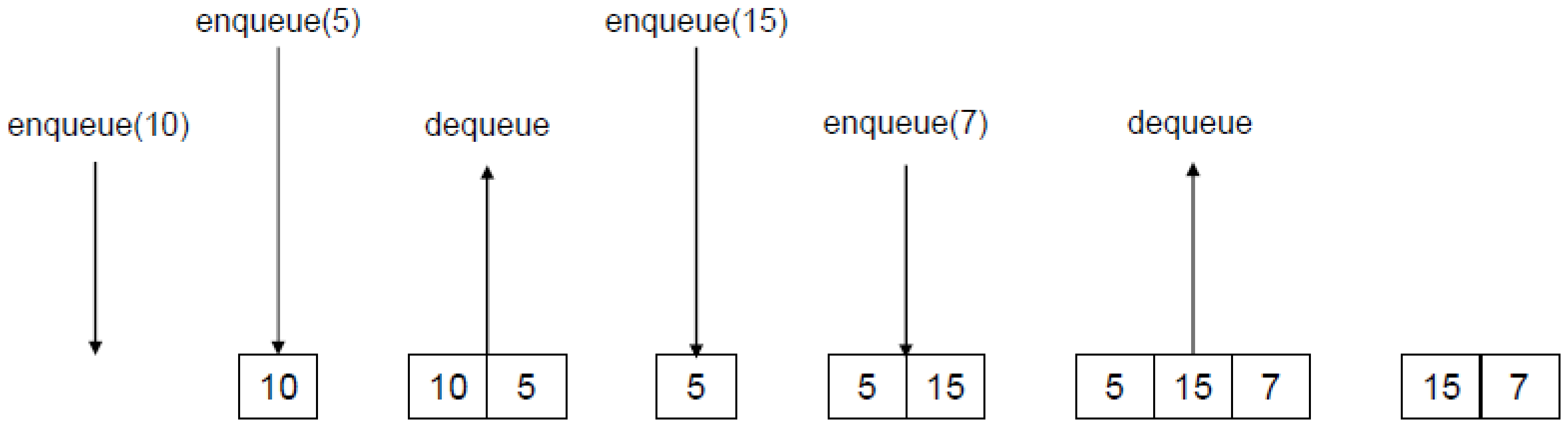
Queue

- Queue
  - Similar to a supermarket checkout line
  - First-in, first-out (FIFO)
  - Nodes are removed only from the head
  - Nodes are inserted only at the tail
- Insert and remove operations
  - Enqueue (insert) and dequeue (remove)

Application of Queue:
- Print job


Back
Front
Dequeue
Enqueue

QUEUES



enqueue(10)
enqueue(5)
10
dequeue
10
5
enqueue(15)
5
enqueue(7)
5
15
dequeue
5
15
7
15
7

Add Pseudo Code:

```
Procedure Add (item,  Queue)
Begin
  if (Rear=N-1)
      Queue is Full;
Else {
  Rear=Rear+1;
  Queue[Rear]=item;
 }
end
```

Delete Pseudo Code:

```
Procedure Delete (item,  Queue)
Begin
  if (Front =Rear)
      Queue is Empty;
Else {
  Front=Front+1;
  item=Queue[Front];
 }
end
```

C implementation in Stack & Queue

```
//PUSH & POP OPERATION
#include<stdio.h>
void push(int [100]);
void pop(int [100]);
void display(int [100]);
int top=-1;
main()
{
int a[100],ch;
do
{
printf("\n\n Enter the choice that you need \n");
printf("\t1.PUSH\n\t2.POP\n\t3.DISPLAY\n\t4.EXIT\n\n");
scanf("%d",&ch);
switch(ch)
{
case 1:push(a);break;
case 2:pop(a);break;
case 3:display(a);break;
default:printf("\n\n\t Thank You!!!\n\n");break;
}
}while(ch<4);
}
```

```
void push(int a[100])
{
int item;
if(top>=100)
printf("\n |STACK OVERFLOW| \n");
else
{
printf("\n Enter the element: ");
scanf("%d",&item);
top++;
a[top]=item;
}
}
void pop(int a[100])
{
if(top==-1)
printf("\n |STACK UNDERFLOW| \n");
else
{
printf("\n The deleted element is %d \n",a[top]);
top--;
}
}
```

```
void display(int a[100])
{
int i;
if(top==-1)
printf("\n Stack is Empty \n ");
else
{
for(i=top;i>=0;i--)
printf("\n\t%d",a[i]);
printf("\n\n");
}
}
```

Queue

```
#include<stdio.h>
void enque(int [100]);
void deque(int [100]);
void display(int [100]);
int front=-1,rear=-1;
main()
{
int a[100],ch;
do
{
printf("\n\n Enter the choice that you need \n");
printf("\t1.ENQUE(insertion)\n\t2.DEQUE(deletion)\n\t3.DISPLAY\n\t4.EXIT\n\n");
scanf("%d",&ch);
switch(ch)
{
case 1:enque(a);break;
case 2:deque(a);break;
case 3:display(a);break;
default:printf("\n\n\t Thank You!!!\n\n");break;
}
}while(ch<4);
}
```

```
void enque(int a[100])
{
int item;
if(front>100)
printf("\n\n |QUEUE OVERFLOW| \n");
else
{
printf("\n Enter element into the Queue: ");
scanf("%d",&item);
if(rear==-1)
front=0;
rear++;
a[rear]=item;
}
}
void deque(int a[100])
{
if(front==rear+1)
printf("\n |QUEUE UNDERFLOW| \n");
else
{
printf("\n Deleted element is %d \n",a[front]);
front++;
}
}
```

```
void display(int a[100])
{
int i;
if(rear==-1)
printf("\n\n Queue is Empty \n\n");
else
{
for(i=front;i<=rear;i++)
printf("\n\t%d",a[i]);
printf("\n\n");
}
}
```