

MEMORY RETRIEVE
MANAGE COMPLEXITY
LANGUAGE INFORMATION ORGANIZING **DATA**
INDEX OPERATIONS
EFFICIENT ABSTRACT **STRUCTURE**
STORE TYPES **COMPUTER** APPLICATION PROCEDURES
HASH TABLE DATABASE IMPLEMENTATION
PERFORM
DATA AMOUNTS

DATA STRUCTURE

Stand Alone Application

Vinit K

KODBAO086

INDEX

1. Intrduction

1.1 Data Structure

1.2 Types of DS

1.1.1 Linear Ds

1.1.2 Non-Linear Ds

1.3 Uses of DS

2. SRS

2.1 S/W Requirements

2.2 H/W Requirements

2.3 TQ Requirements

3. Project Implantation

3.1 Array

3.2 Stack

3.3 queue

3.4 Circular Queue

3.5 Linked List

3.6 Doubly Linked List

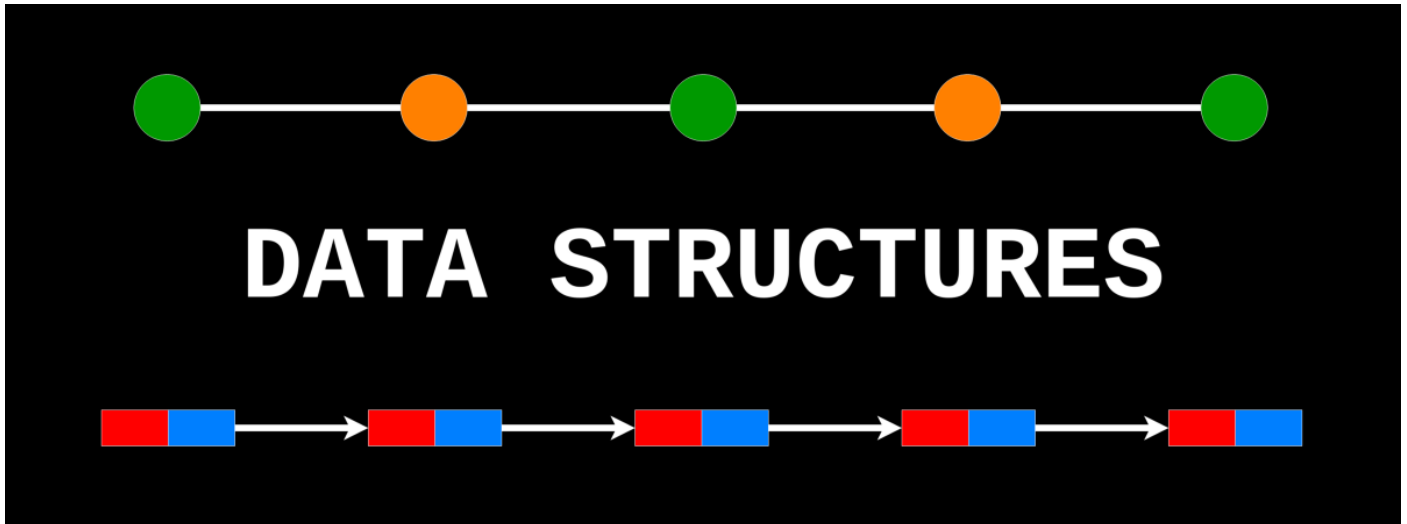
4. Conclusion

5. Future Work

1.Introduction

1.1 Data Structure

Data structure is a way of organizing and storing data so that it can be accessed and used efficiently. It is a collection of data values, the relationships among them, and the functions or operations that can be applied to the data. Data structures provide a means to manage large amounts of data efficiently, such as large databases and internet indexing services. Common data structures include arrays, linked lists, hash tables, trees, and graphs. Data structures can be used to organize the storage and retrieval of information in databases, and they are also used in computer programming to organize the data in the memory and manipulate it efficiently.



1.2 Data Structure Classified in two Types

1.Linear Data Structure

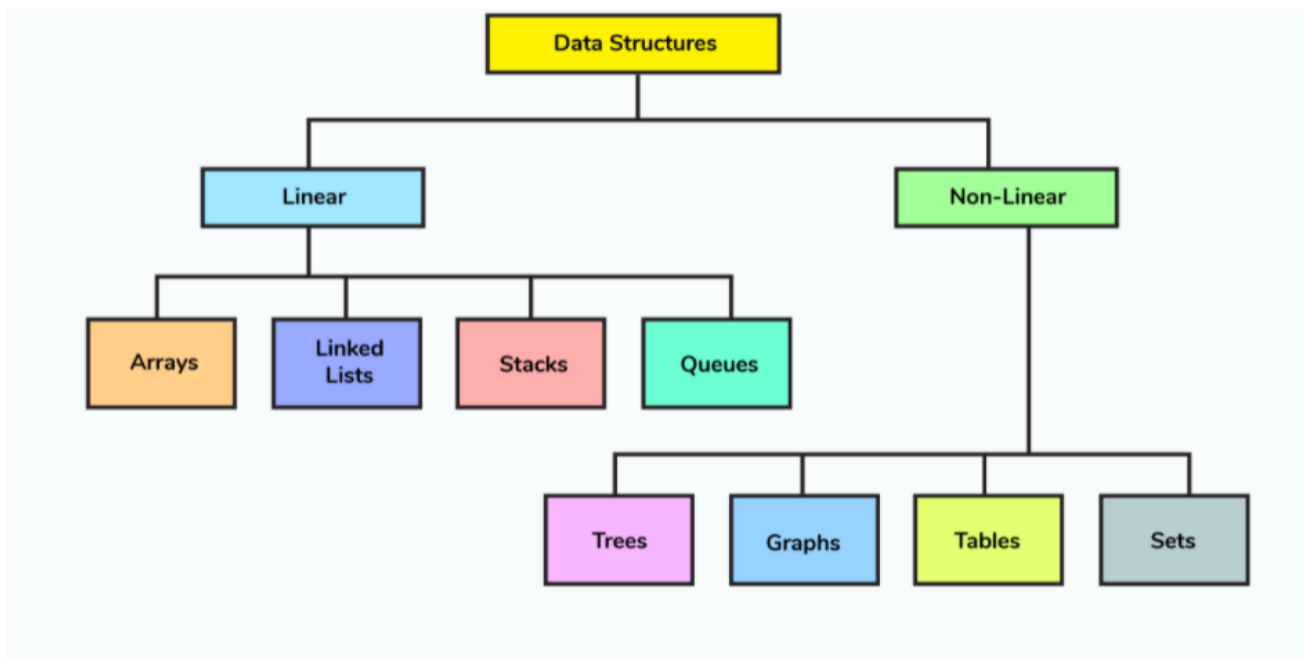
2.Non- Linear Data Structure

1.1.1 Linear Data Structures

A linear data structure is a type of data structure which follows a linear order. This means that any element or data item is organized sequentially and can only be reached or processed in a linear fashion. Examples of linear data structures include arrays, linked lists, stacks, and queues.

1.1.2 Non-Linear Data Structures

A non-linear data structure is a type of data structure in which the data items are organized in a non-linear fashion. This means that the data items are not organized sequentially and can be reached or processed in any order. Examples of non-linear data structures include binary trees, heaps, graphs, and hash tables.



1.4 The uses of data Structure

1. Storing and organizing large amounts of data: Data structures are used to store data in a structural format that allows for efficient retrieval and manipulation.
2. Searching for specific data: Data structures provide efficient means to search for data.
3. Searching for patterns in data: Data structures can help identify patterns in data, such as trends or correlations.
4. Analysing data: Data structures can be used to analyse data to identify patterns and relationships.
5. Designing algorithms: Data structures are essential for designing efficient algorithms.
6. Establishing communication between components: Data structures can be used to establish communication between components or systems.
7. Representing relationships between objects: Data structures can be used to represent relationships between objects, such as family trees or social networks.

2. Software Requirement's Speciation

2.1 Software Requirement's

1. Java Runtime Environment (JRE): Java Runtime Environment (JRE) 8 or higher.
2. IDE: An Integrated Development Environment (IDE) like Eclipse or NetBeans.
3. Libraries and Frameworks: Java Swing, Windows builder

2 Hardware Requirement's

1. Processor: Intel Core i5 or better
2. Memory: 8GB RAM • Hard Disk: 500GB
3. Video Card: Integrated or discrete graphics card
4. Monitor: 17-inch LCD or higher
5. Network Adapter: Ethernet or wireless adapter
6. Operating System: Windows 10 or Mac OS X 10.8 or higher, or Linux.

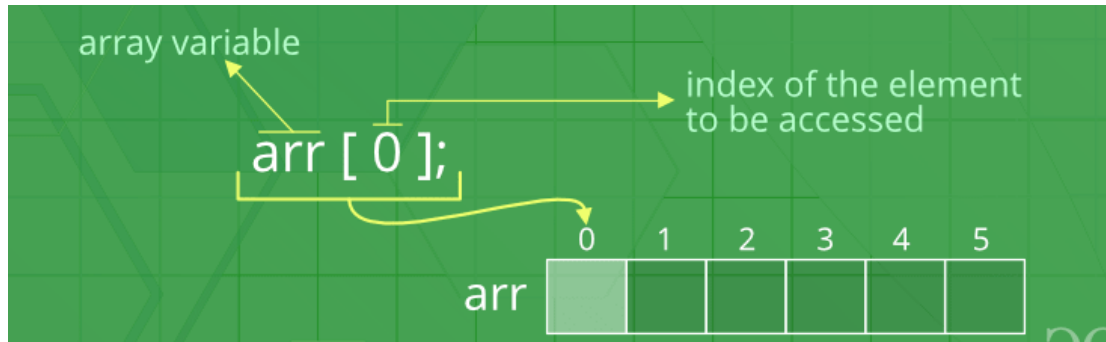
2.3 Technical Requirement's

1. Java.
2. An understanding of Object-Oriented Programming (OOP).
3. An understanding of Model-View-Controller (MVC) design pattern.
4. An understanding of the Java Swing APIs.

3. Project Implantation

3.1 Arrays

An array is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. After creation, its length is fixed. Each item in an array is called an element and is accessed by its numerical index.



Advantages of the array

1. Arrays provide faster access to data than other data structures such as linked lists.
2. Arrays are easier to use and understand than other data structures such as linked lists or trees.
3. Arrays use less memory than other data structures as they store data in contiguous memory locations.
4. Arrays are easy to traverse and can be used to store and retrieve data quickly and efficiently.
5. Arrays allow for dynamic memory allocation which means that memory can be allocated and deallocated as needed.

Disadvantages of arrays

1. Arrays use up more memory than other data structures such as linked lists.
2. Arrays are static, which means that once an array is created, its size cannot be changed.
3. Inserting or deleting an element from an array can be difficult and time consuming.
4. Accessing an element from an array is slow compared to other data structures such as linked lists.
5. Arrays are not suitable for frequent insertion and deletion operations.

Arrays Operations

1. Insertion
2. Deletion
3. Display

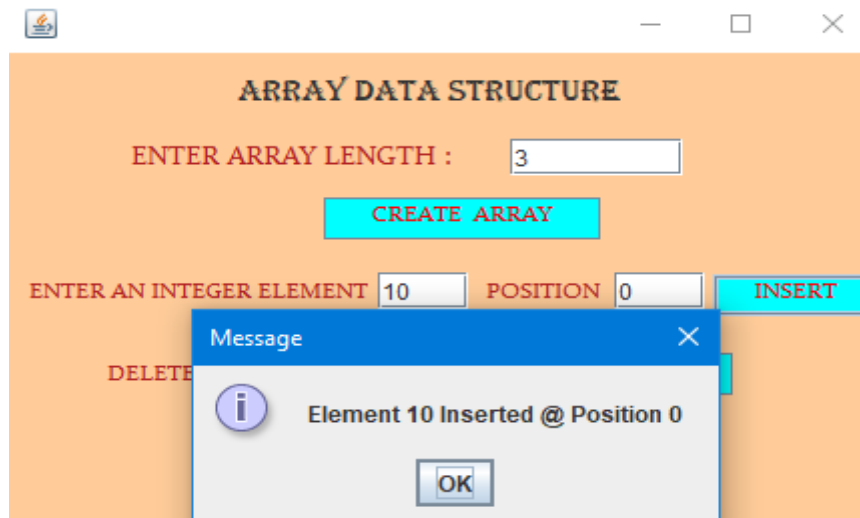
The screenshot shows a window titled "ARRAY DATA STRUCTURE" with a blue header bar. The main area has an orange background. It contains the following elements:

- Label: "ENTER ARRAY LENGTH :" followed by an empty text input field.
- Button: "CREATE ARRAY" (cyan).
- Label: "ENTER AN INTEGER ELEMENT" followed by an empty text input field.
- Label: "POSITION" followed by an empty text input field.
- Button: "INSERT" (cyan).
- Label: "DELETE POSITON" (note the typo) followed by an empty text input field.
- Button: "DELETE" (cyan).
- Button: "DISPLAY" (cyan).
- A large empty text area at the bottom for displaying the array.

This screenshot shows the same application window, but with the "ENTER ARRAY LENGTH" field containing the value "3". The "CREATE ARRAY" button is highlighted. A "Message" dialog box is overlaid on the window, displaying the message "Array of length 3 created" with an information icon and an "OK" button.

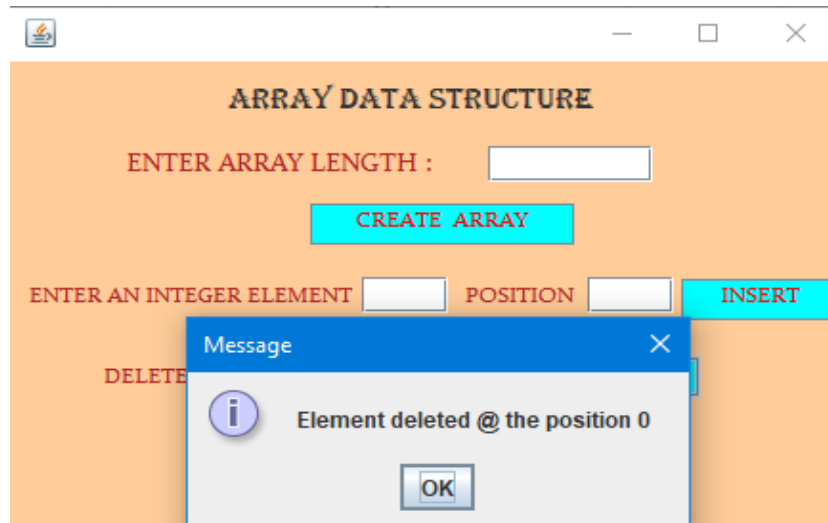
Insert Operation code

```
int ele=Integer.valueOf(elem.getText());
int pos=Integer.valueOf(ipos.getText());
arr[pos]=ele;
String message="Element "+ele+" Inserted @ Position "+pos;
JOptionPane.showMessageDialog(contentPane, message);
elem.setText("");
ipos.setText("");
```



Deletion Operation Code

```
int pos=Integer.valueOf(dpos.getText());
arr[pos]=0;
String message="Element deleted @ the position "+pos;
JOptionPane.showMessageDialog(contentPane, message);
dpos.setText("");
```



Display Operation Code

```
String msg="";  
for(int i=0;i<=arr.length-1;i++)  
{  
    msg=msg+" "+arr[i];  
}
```

ARRAY DATA STRUCTURE

ENTER ARRAY LENGTH : CREATE ARRAY

ENTER AN INTEGER ELEMENT POSITION INSERT

DELETE POSITON DELETE

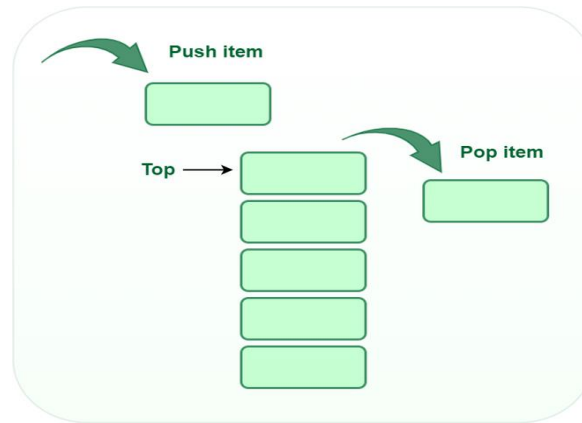
DISPLAY

Real Time Application of an Arrays

- Arrays are used in real-time applications in a variety of ways. One of the most common applications of arrays is in data storage and retrieval. For example, databases use arrays to store and organize data, allowing for efficient and fast retrieval of information.
- Arrays are also used to store and manipulate images, such as in digital photo editing software.
- Arrays are also used in computer graphics and animation, allowing for faster and more efficient rendering of complex scenes.
- Arrays are also used in many scientific computing applications, such as weather prediction, climate modeling, and astronomical simulations.

3.2 Stack

A stack is a basic data structure used for storing and manipulating data. It is a linear data structure, which means that data is stored and accessed in a single, sequential order. A stack is a Last In, First Out (LIFO) structure, meaning that the most recently added item is the first one to be removed. The two basic operations performed on a stack are push and pop. Push adds an item to the top of the stack, while pop removes an item from the top of the stack.



Advantages of the Stack

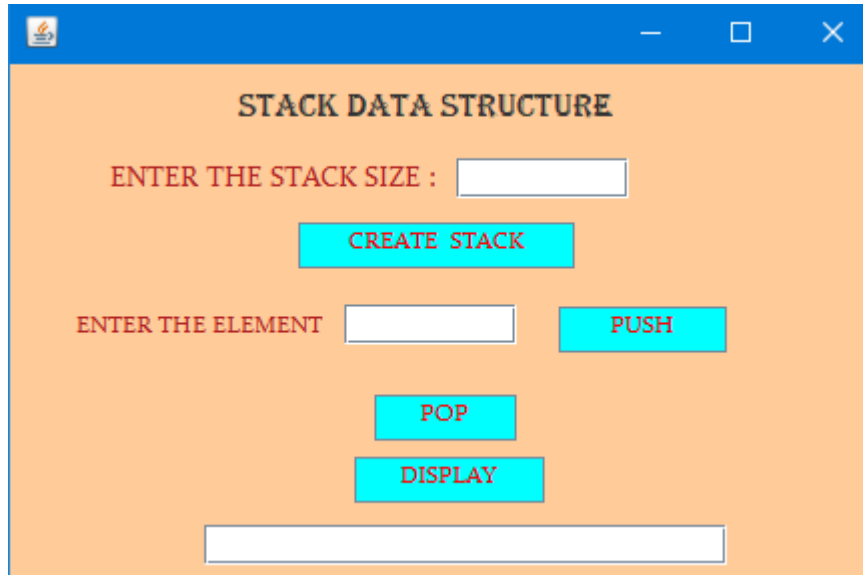
1. Low cost: Setting up a stack is relatively inexpensive. The cost of the hardware and software is minimal compared to other solutions.
2. Versatility: Stacks are highly customizable and can be tailored to the specific needs of a business.
3. Scalability: Stacks are designed to be highly scalable, allowing users to easily add or remove components as needed.
4. Security: Stack platforms are generally more secure than other solutions due to the use of secure protocols and authentication techniques.

Disadvantages of the Stack

1. High Initial Cost: Setting up a stack can be expensive, as it requires purchasing the required hardware, software and hiring personnel to set it up and maintain it.
2. Complexity: Stacks can be complex to manage, as each layer of the stack must be configured and maintained to ensure optimal performance.
3. Limited Flexibility: Stacks are less flexible than other types of architectures as they are often designed to be used for specific workloads.
4. Lack of Scalability: Stacks can be difficult to scale up or down as the entire stack must often be reconfigured.

Stack Operation

1. Push
2. Pop
3. Display



STACK DATA STRUCTURE

ENTER THE STACK SIZE :

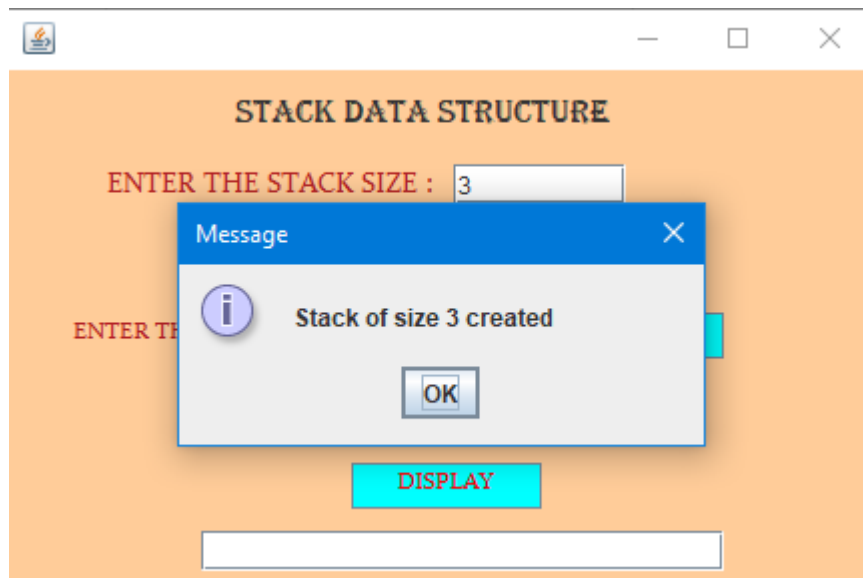
CREATE STACK

ENTER THE ELEMENT

PUSH

POP

DISPLAY



STACK DATA STRUCTURE

ENTER THE STACK SIZE : 3

ENTER THE ELEMENT

DISPLAY

Message

Stack of size 3 created

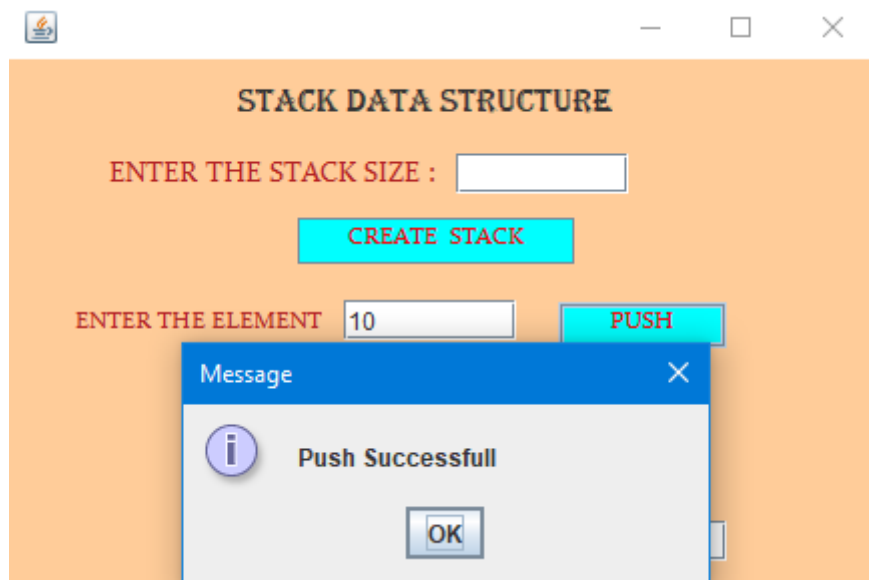
OK

PUSH Operation Code

```

int elem;
if (top == size - 1)
{
    JOptionPane.showMessageDialog(contentPane, "Push is not Possible ");
}
else
{
    elem=Integer.valueOf(element.getText());
    ++top;
    s[top] = elem;
    JOptionPane.showMessageDialog(contentPane, "Push Successfull");
    element.setText("");
}

```



POP Operation Code

```

if (top == -1)
{
    JOptionPane.showMessageDialog(contentPane, "pop is not Possible ");
}
else
{
    String message="Element deleted is "+s[top];
    JOptionPane.showMessageDialog(contentPane, message);
    --top;
}

```

STACK DATA STRUCTURE

ENTER THE STACK SIZE :

ENTER THE ELEMENT :

Message

Element deleted is 30

OK

DISPLAY

STACK DATA STRUCTURE

ENTER THE STACK SIZE :

ENTER THE ELEMENT :

Message

Element deleted is 20

OK

DISPLAY

STACK DATA STRUCTURE

ENTER THE STACK SIZE :

CREATE STACK

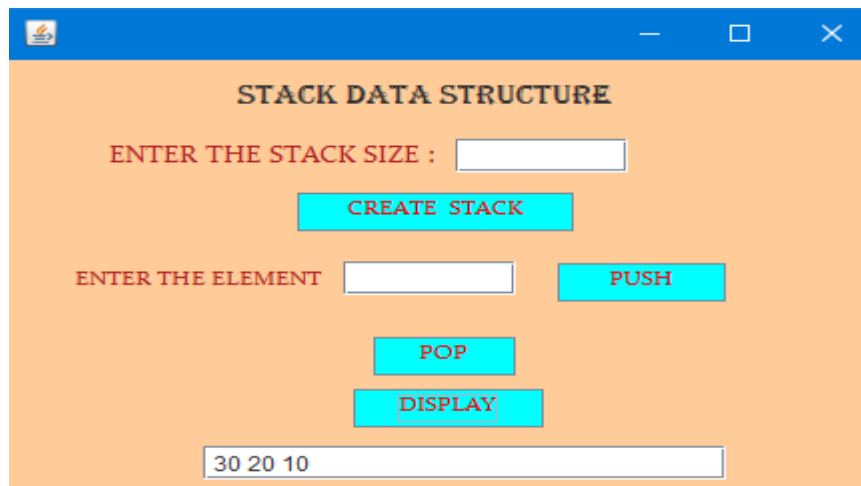
ENTER THE ELEMENT PUSH

POP

DISPLAY

Display Operation Code

```
String msg="";
if (top == -1)
{
    JOptionPane.showMessageDialog(contentPane, "Display is not Possible");
}
else
{
    for (int i = top; i >= 0; i--)
    {
        msg=msg+" "+s[i];
    }
    tx3.setText(msg);
}
```

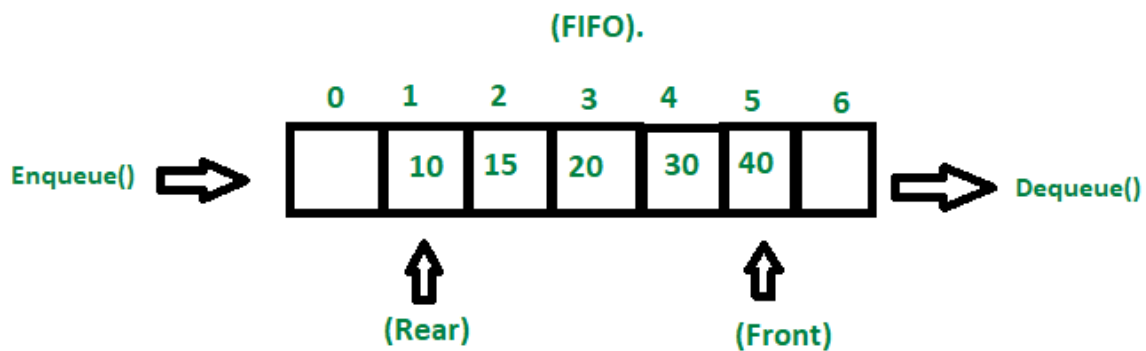


Real Time Application of Stack

1. The most common real-time application of a stack is in computing. Stacks are used in computer memory management, program execution, and handling interrupts. For instance, when a program is running, it uses the stack to keep track of where it is in the program, store data, and manage function calls.
2. When an interrupt is received, the stack is used to store the processor state and resume the interrupted program when the interrupt has been processed.
3. Another common use for stacks is in the evaluation of mathematical expressions.

3.3 Queue

A Queue is a linear data structure which follows a particular order in which the operations are performed. The order is First In First Out (FIFO). A queue is a collection of elements in which an element can be inserted from one end called the rear and can be deleted from the other end called the front.



Advantages of the Queue

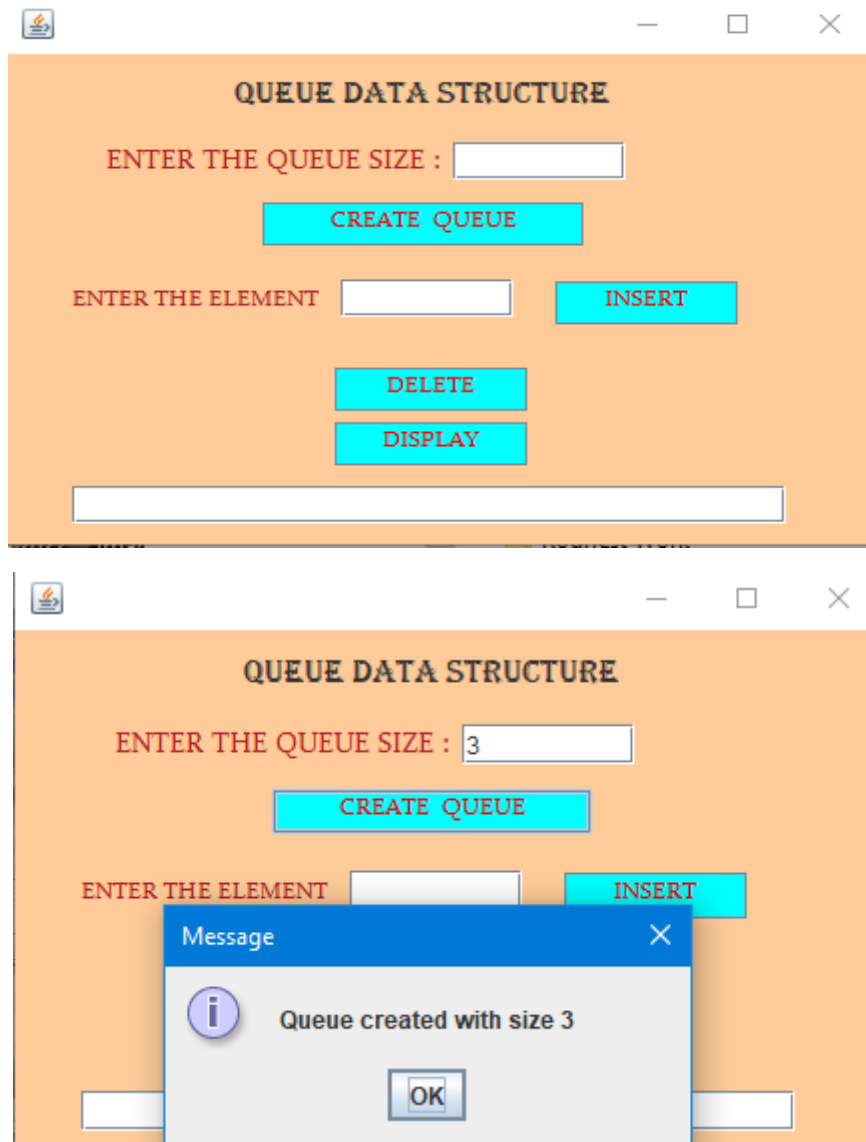
1. Queues provide an efficient way to manage the flow of tasks and data.
2. Queues can be used to implement task scheduling, event handling, and other processes.
3. Queues can be used to store and manage data that is to be processed by multiple threads or processes.
4. Queues can be used to buffer data in order to prevent overloads.

Disadvantages of the Queue

1. Queues have limited data storage, meaning that once the queue is full, no more data can be added until some of the existing data is removed.
2. Queues are not suitable for storing large amounts of data since it can quickly become overwhelmed with data and slow down the entire system.
3. Since queues are first-in-first-out data structures, the performance can be unpredictable. If there is a sudden influx of data, it can cause the queue to become backlogged and slow down the system.
4. Queues are not easily scalable and can become a bottleneck if they are under too much load.

Queue Operation

1. Insertion
2. Deletion
3. Display



Queue Insertion Operation

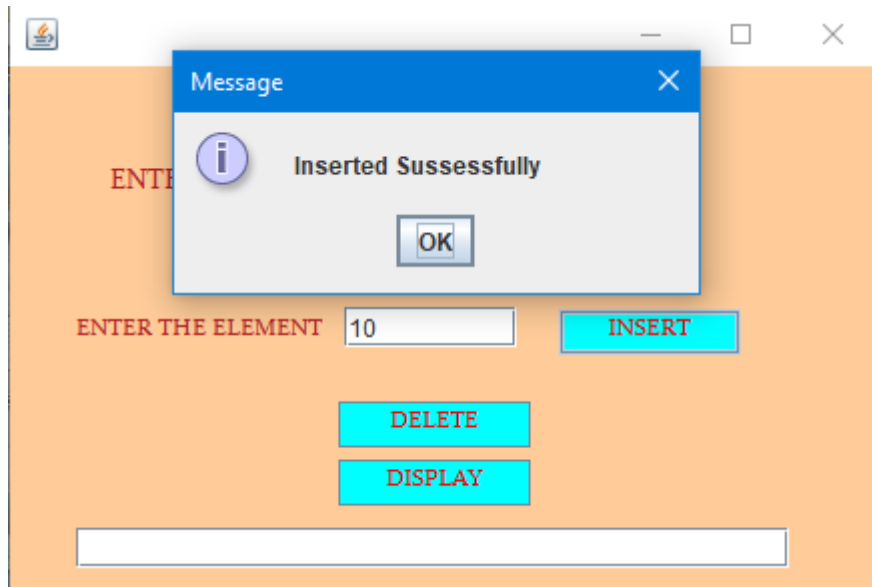
```
int elem;
if(r==size-1)
{
    JOptionPane.showMessageDialog(contentPane, "Insertion is not
    possible");
}
else
{
```



```

elem=Integer.valueOf(element.getText());
++r;
q[r]=elem;
JOptionPane.showMessageDialog(contentPane, " Inserted
Sussessfully");
element.setText("");
}

```

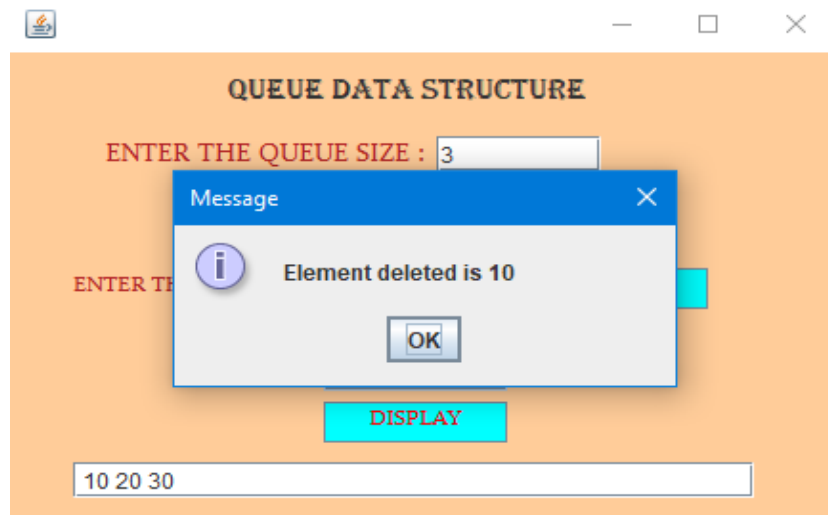


Queue Deletion Operation Code

```

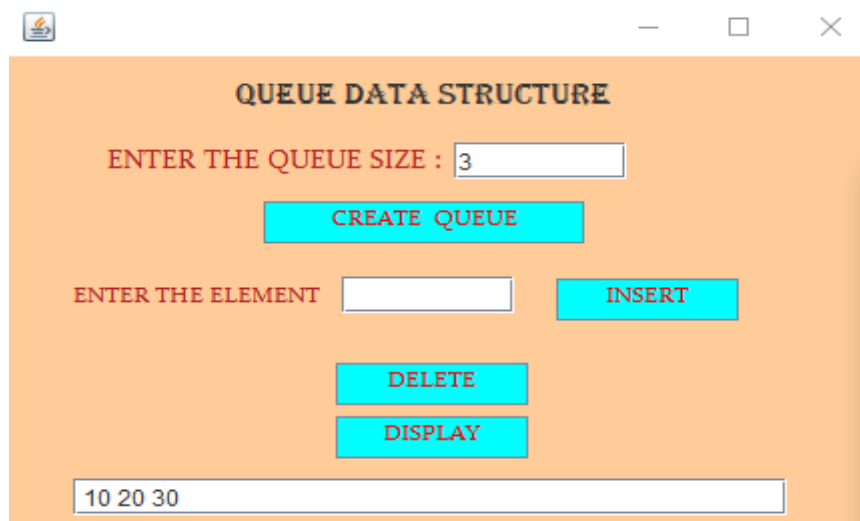
if(r== -1 || f > r)
{
    JOptionPane.showMessageDialog(contentPane, "Deletion is not
    possible");
}
else
{
    String message="Element deleted is "+q[f];
    JOptionPane.showMessageDialog(contentPane, message);
    f++;
}

```



Queue Display Operation Code

```
String msg="";
if(r== -1 || f>r)
{
    JOptionPane.showMessageDialog(contentPane, "Display is not posible");
}
else
{
    for(int i=f;i<=r;i++)
    {
        msg=msg+" "+q[i];
    }
    res.setText(msg);
}
```

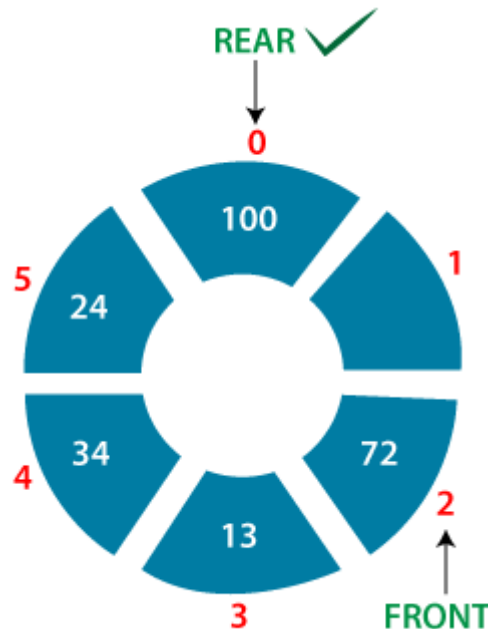


Real Time Application of a Queue

1. **Web Server Requests:** A queue is often used to handle requests coming from web servers. When a user visits a website, a request is sent to the web server. This request is added to a queue and is processed in the order in which it was received.
2. **Printing Jobs:** In large offices, queues are often used to manage printing jobs. All the documents that need to be printed are placed in a queue and are printed in the order in which they are received.
3. **Operating Systems:** Queues are used by operating systems to handle multiple tasks. When an application is opened, the operating system adds it to a queue and processes the tasks in the order they were received.
4. **Messaging Apps:** Messaging apps like WhatsApp use queues to manage messages sent between users. When a message is sent, it is added to a queue and is delivered in the order in which it was received.

3.3 Circular Queue

A circular queue is a type of data structure used for storing data in a queue. It is similar to a regular queue, except that the last item in the queue is connected to the first item, creating a loop. This allows the queue to be managed efficiently, as items can be added and removed from the queue without having to shift all the other items in the queue.



Circular Queue Advantages

1. Circular queues are a very efficient way of storing and accessing data.
2. Insertion and deletion operations are easier to implement in a circular queue when compared to a linear queue.
3. It eliminates the need for shifting existing elements in memory when one is inserted or deleted, as the queue wraps around.
4. It does not have the problem of overflow or underflow because of its circular nature.
5. It has a constant-time performance for enqueue and dequeue operations.
6. It can be easily implemented using an array.

Circular Queue Disadvantages

1. Wasting memory: A circular queue requires additional memory space to store the addresses of the front and rear positions. This extra memory space is wasted if the queue is not completely filled.

2. More Complex Algorithms: Circular queue requires more complex algorithms to manipulate the data structure.
3. Overwriting Data: In a circular queue, the rear pointer can overwrite the data of the front element if the queue is not managed properly.

Circular Queue Operation

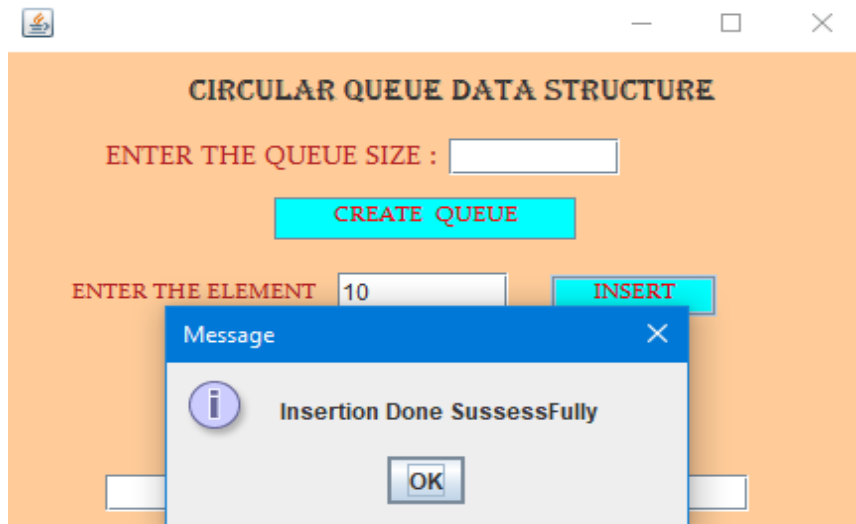
1. Insertion
2. Deletion
3. Display

Circular Queue Insertion Operation Code

```

int elem;
if (count == size)
{
    JOptionPane.showMessageDialog(contentPane, "Insertion not
    Possible");
}
else
{
    elem = Integer.valueOf(element.getText());
    r = (r + 1) % size;
    cq[r]=elem;
    count++;
    JOptionPane.showMessageDialog(contentPane, "Insertion Done
    SussessFully");
}

```

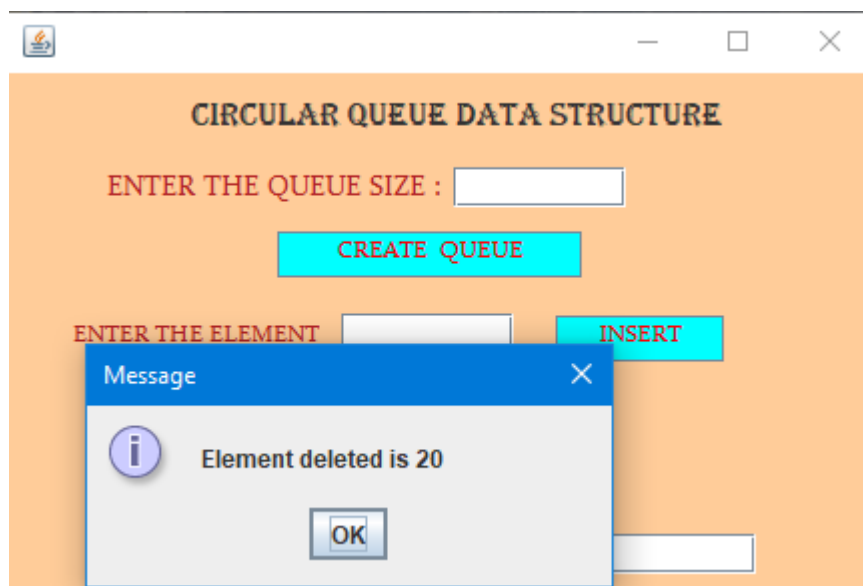


Circular Queue Deletion Operation Code

```

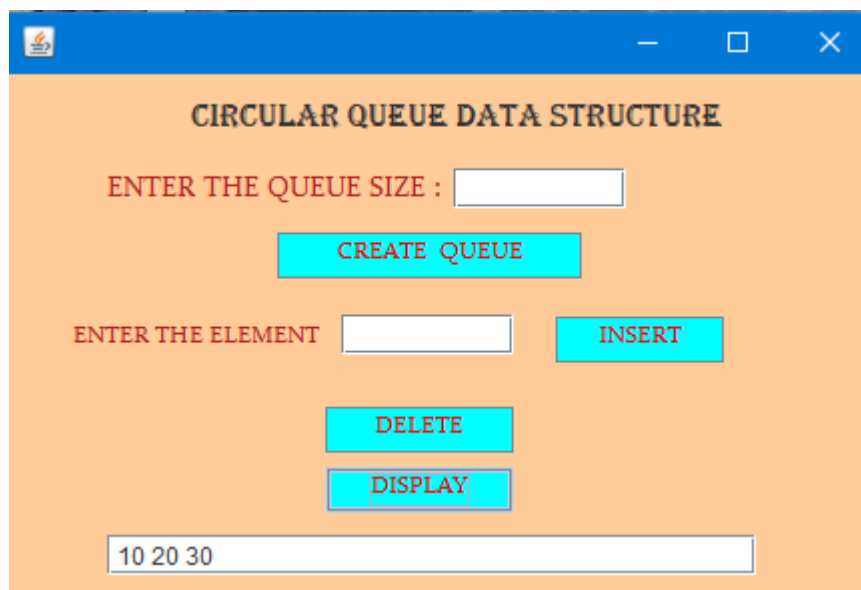
if (count == 0)
{
    JOptionPane.showMessageDialog(contentPane, "Delete not possible");
}
else
{
    String message="Element deleted is " + cq[f];
    JOptionPane.showMessageDialog(contentPane, message);
    f = (f + 1) % size;
    --count;
}

```



Circular Queue Display Operation Code

```
String msg="";
int f1 = f;
if (count == 0)
{
    JOptionPane.showMessageDialog(contentPane, "Display is not
    Possible");
}
else
{
    for (int i = 1; i <= count; i++)
    {
        msg=msg+" "+cq[f1];
        f1 = (f1 + 1) % size;
    }
    res.setText(msg);
}
```

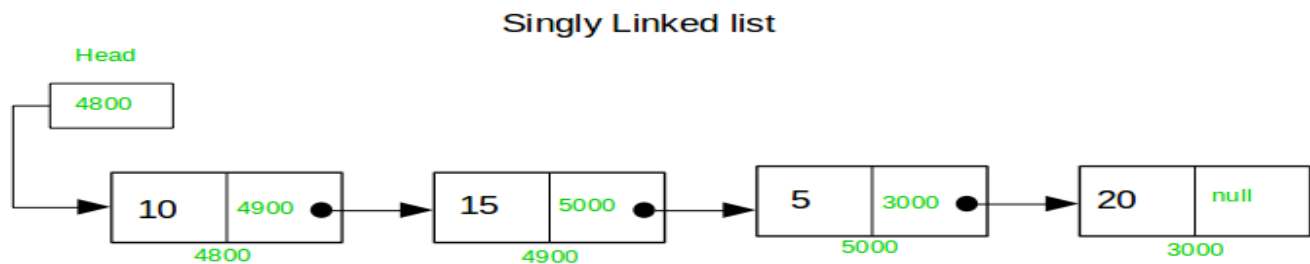


Real Time Application of a Circular Queue

1. Circular queues are commonly used in operating systems to manage processes. In this application, the queue is used to keep track of the processes that are running, and the system can add or remove items from the queue as needed.
2. The circular queue allows the operating system to keep track of the processes in an efficient way, and it can also be used to limit the number of processes that can be running simultaneously.
3. the circular queue can be used to prioritize certain processes over others, ensuring that the most important tasks are handled first.

3.4 Single Linked List

A singly linked list is a linear data structure where each element is a separate object linked to the previous and/or next element. Each element contains a reference to the next element in the list and possibly to the previous element as well. The last element in the list points to null or nothing. It is a simple data structure that allows items to be inserted or removed in constant time. It is the most basic form of a linked list.



Advantages of the Linked List

1. A linked list is a dynamic data structure which can grow and shrink in size during execution of a program.
2. Adding or removing elements from a linked list is fast and efficient as compared to an array.
3. Memory utilization: A linked list uses much less memory compared to an array.
4. Memory wastage is not an issue in a linked list since it allocates memory only when needed.

Disadvantages of the Linked List

1. Unlike an array, elements in a linked list cannot be accessed directly.
2. Extra memory space for a pointer is required with each element of the list.
3. Reverse traversing is difficult in a linked list.
4. Linked List is not cache friendly. Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Single Linked List Operation

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display

SINGLE LINKED LIST DATA STRUCTURE

ENTER THE ELEMENT : INSERT FRONT

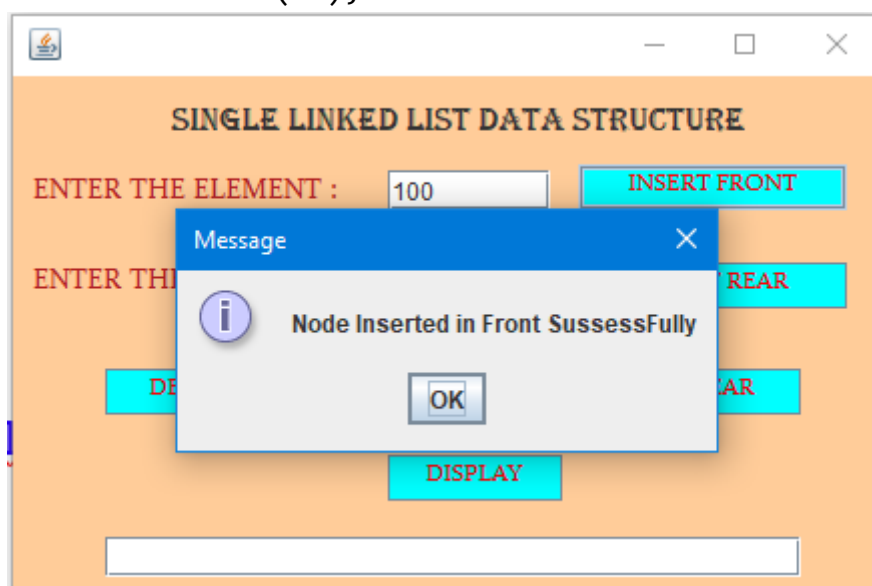
ENTER THE ELEMENT : INSERT REAR

DELETE FRONT DELETE REAR

DISPLAY

Insert Front Operation

```
Node temp;
int elem=Integer.valueOf(insertfornt.getText());
Node newnode = new Node();
newnode.data = elem;
newnode.link = null;
if (first == null)
{
    first = newnode;
}
else
{
    newnode.link = first;
    first = newnode;
}
JOptionPane.showMessageDialog(contentPane, "Node Inserted in Front
SussessFully");
insertfornt.setText("");
```

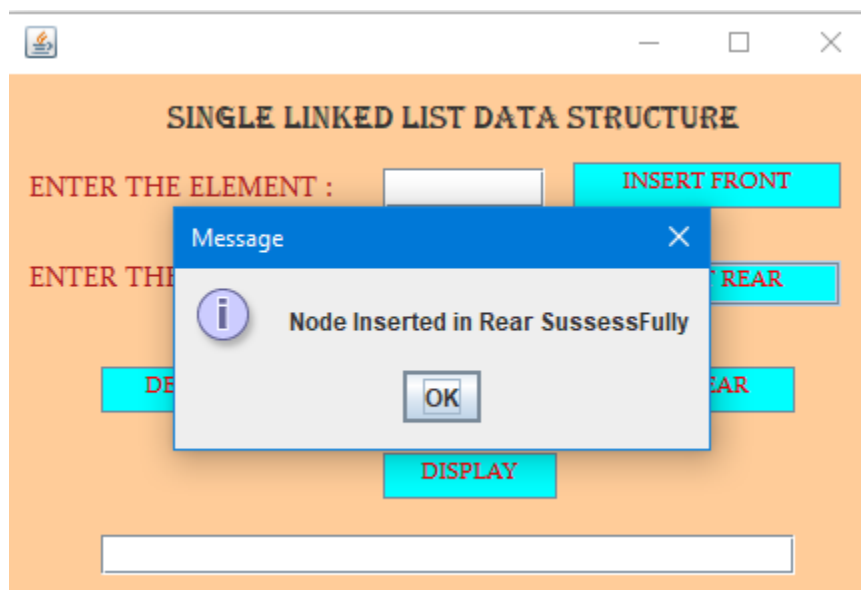


Insert Rear Operation

```

Node temp;
int elem=Integer.valueOf(insertrear.getText());
Node newnode = new Node();
newnode.data = elem;
newnode.link = null;
if (first == null)
{
    first = newnode;
}
else
{
    temp = first;
    while (temp.link != null)
    {
        temp = temp.link;
    }
    temp.link = newnode;
}

```



Display Operation

```
String msg="";
Node temp;
    if (first == null)
    {
        JOptionPane.showMessageDialog(contentPane, "Display is
        not Possible");
    }
    else if (first.link == null)
    {
        msg=msg+first.data;
    }
    else
    {
        temp = first;
        while (temp != null)
        {
            msg=msg+" "+temp.data;
            temp = temp.link;
        }
    }
}
```

SINGLE LINKED LIST DATA STRUCTURE

ENTER THE ELEMENT : INSERT FRONT

ENTER THE ELEMENT : INSERT REAR

DELETE FRONT
DELETE REAR

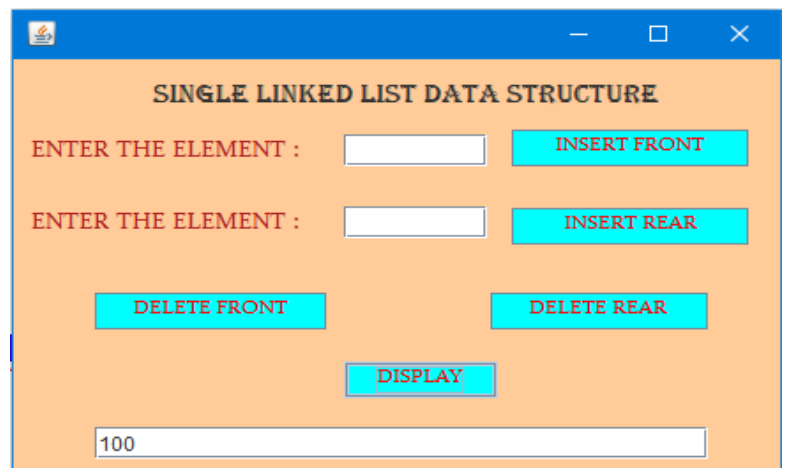
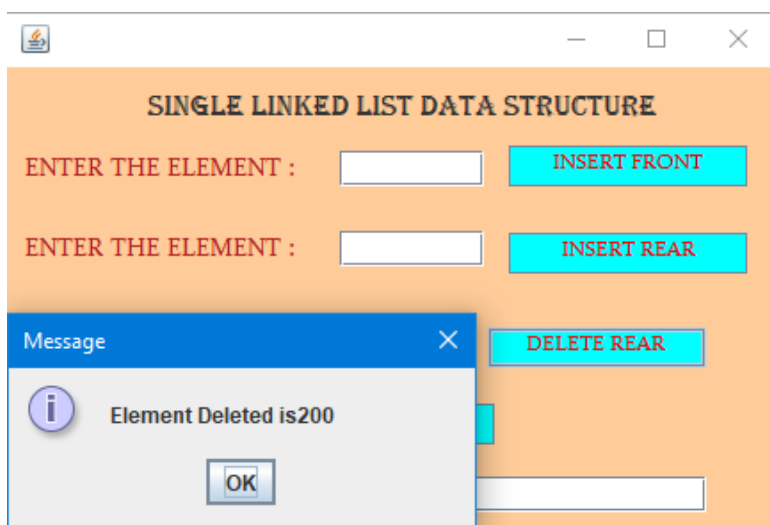
DISPLAY

Deletion Rear Operation

```

Node temp;
if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Deletion is not
    Possible");
}
else if (first.link == null)
{
    String msg="Element Deleted is"+first.data;
    JOptionPane.showMessageDialog(contentPane, msg);
    first = null;
}
else
{
    temp = first;
    while (temp.link.link != null) {
        temp = temp.link;
    }
}

```

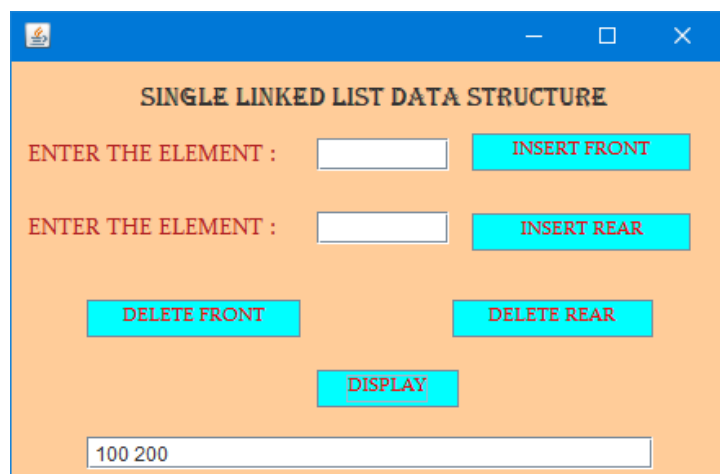
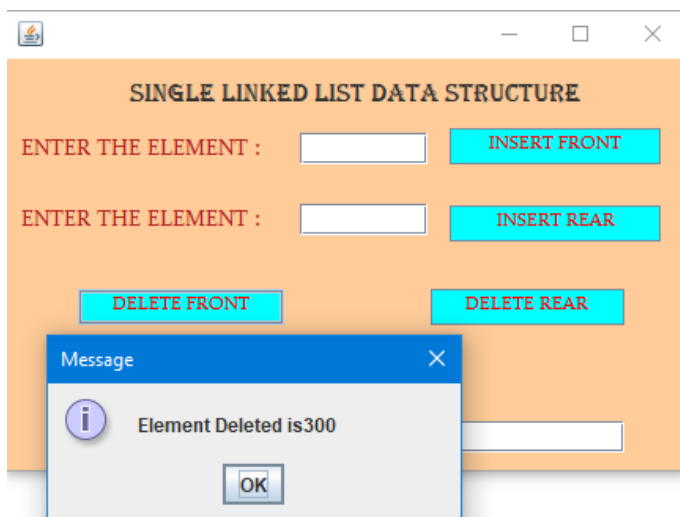


Delete Front Operation

```

if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Deletion is not
    Possible");
}
else if (first.link == null)
{
    String msg="Element Deleted is"+first.data;
    JOptionPane.showMessageDialog(contentPane, msg);
    first = null;
}
else
{
    String msg="Element Deleted is"+first.data;
    JOptionPane.showMessageDialog(contentPane, msg);
    first = first.link;
}

```



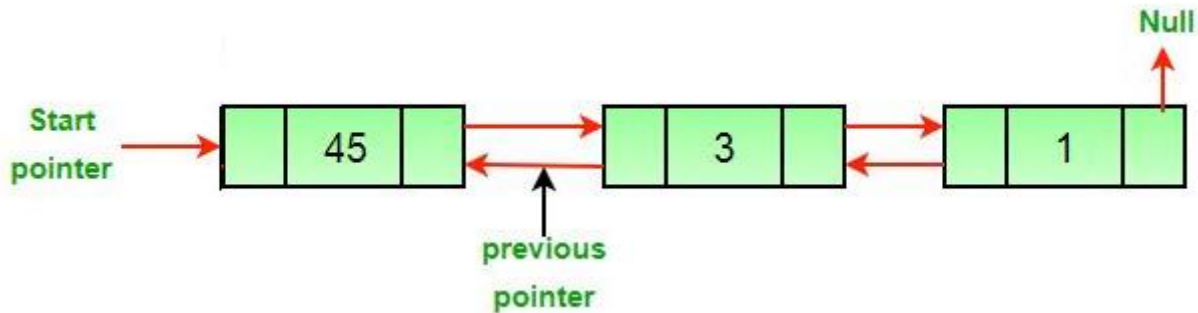
Real Time Application of a Single Linked List

Real-time application of a single linked list can be seen in a number of areas, such as:

1. **Memory Allocation in Operating Systems:** Single linked list is used by the memory manager in operating systems to keep track of the available memory blocks and the allocated memory blocks.
2. **Graph Algorithms:** Single linked list is used to store the adjacency list of the graph.
3. **CPU Scheduling:** Single linked list can be used to store and manage the ready queue of processes in CPU scheduling.
4. **File System:** Single linked list can be used to store and manage the directories and files in a file system.
5. **Network Routing Protocol:** Single linked list can be used to store and manage the routing table for a network routing protocol.
6. **Hash Tables:** Single linked list can be used to store and manage the hash table for an application.

3.6 Doubly Linked

A doubly linked list is a data structure that consists of a set of sequentially linked records called nodes. Each node contains two fields, called links, that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list. It can be used for all sorts of data structures, such as stacks, queues, and associative arrays.



Advantages of doubly linked list

1. A doubly linked list allows for traversal in both directions, from head to tail and from tail to head. This makes it easier to traverse the list in either direction, making it more efficient than singly linked lists.
2. Insertion and deletion operations are easier and faster in doubly linked list compared to singly linked list, as we don't have to traverse the list to find the previous node and adjust the pointers.
3. Doubly linked list can be used to implement stacks and queues.
4. It can be used for more complex applications such as Undo/ Redo operations in text editors.

Disadvantages of doubly linked list

1. More memory is required to store a doubly linked list compared to a singly linked list, since each node requires an extra pointer.
2. A doubly linked list requires more time to traverse than a singly linked list, since pointers have to be followed in both directions.
3. It is more difficult to implement a doubly linked list than a singly linked list, as there are more variables to keep track of.
4. Inserting and deleting an element into a doubly linked list can be more complex than in a singly linked list.

Doubly Linked List Operation

1. Insert Front
2. Insert Rear
3. Delete Front
4. Delete Rear
5. Display Forward
6. Display Reverse

Insert Front Operation

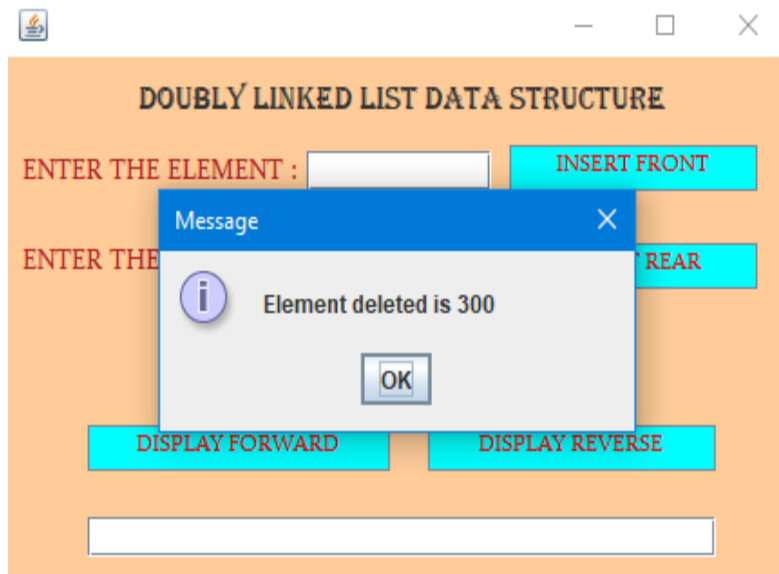
```
int elem =Integer.valueOf(insertfront.getText());
Node newnode = new Node();
newnode.data = elem;
newnode.prelink = null;
newnode.nextlink = null;
if (first == null)
{
    first = newnode;
}
else
{
    newnode.nextlink = first;
    first.prelink = newnode;
    first = newnode;
}
```


Delete Front Operation

```

if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Delete is not
    Possible");
}
else if (first.nextlink == null)
{
    String message="Element deleted is "+first.data;
    JOptionPane.showMessageDialog(contentPane, message);
    first = null;
}
else
{
    String message="Element deleted is "+first.data;
    JOptionPane.showMessageDialog(contentPane, message);
    first = first.nextlink;
    first.prelink=null;
}

```

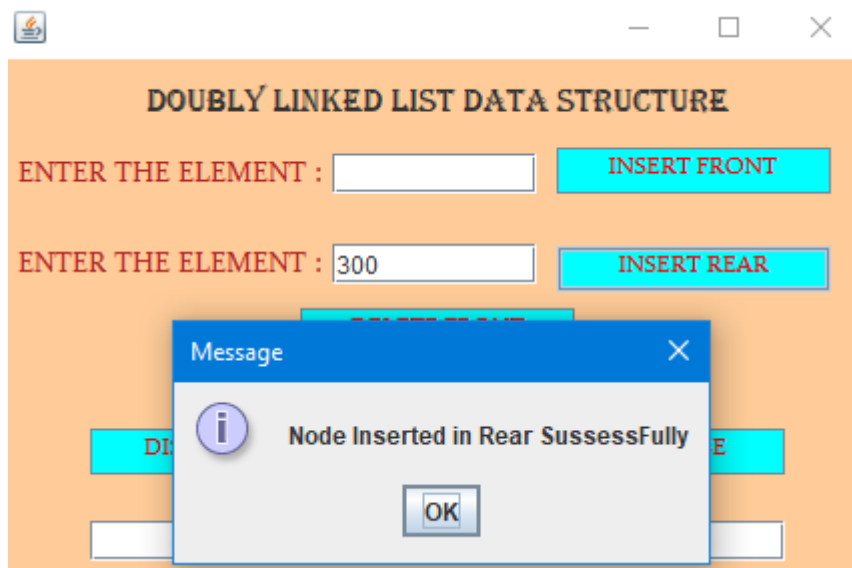


Insert Rear Operation

```

Node temp;
int elem =Integer.valueOf(insertrear.getText());
Node newnode = new Node();
newnode.data = elem;
newnode.prelink = null;
newnode.nextlink = null;
if (first == null)
{
    first = newnode;
}
else
{
    temp = first;
    while (temp.nextlink != null)
    {
        temp = temp.nextlink;
    }
    temp.nextlink = newnode;
    newnode.prelink = temp;
}

```

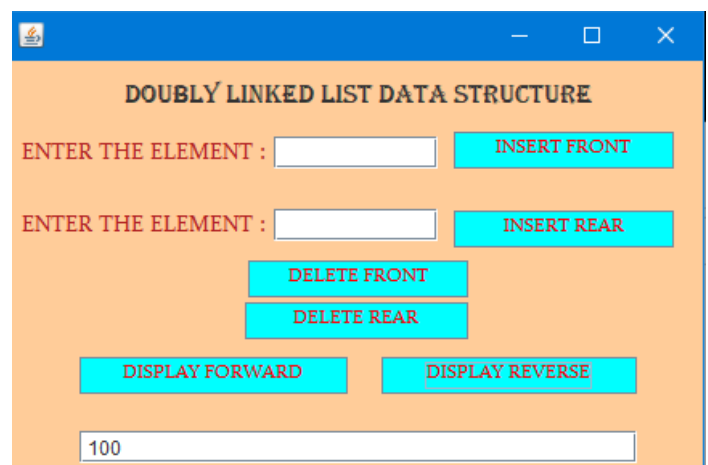
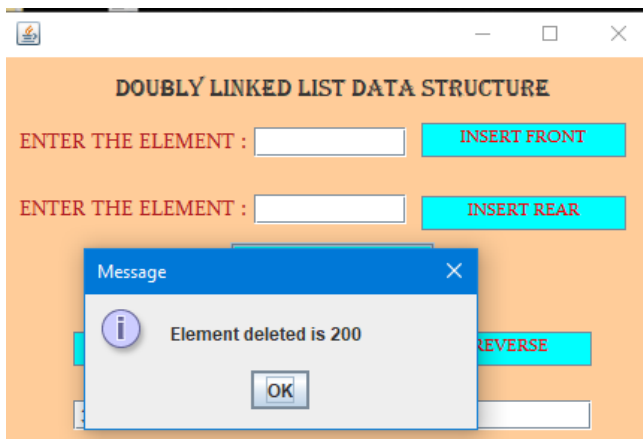


Delete Rear Operation

```

Node temp;
if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Deletion is not
    Possible");
}
else if (first.nextlink == null)
{
    String message="Element deleted is "+first.data;
    JOptionPane.showMessageDialog(contentPane, message);
    first = null;
}
else
{
    temp = first;
    while (temp.nextlink.nextlink != null)
    {
        temp = temp.nextlink;
    }
    String message="Element deleted is "+temp.nextlink.data;
    JOptionPane.showMessageDialog(contentPane, message);
    temp.nextlink = null;
}

```



Display Forward Operation

```
String msg="";
Node temp;
if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Display is not
    Possible");
    System.out.println("Display is not Possible");
}
else if (first.nextlink == null)
{
    msg=" " +first.data;
    res.setText(msg);
}
else
{
    temp = first;
    while (temp != null)
    {
        msg=msg+" "+temp.data;
        temp = temp.nextlink;
    }
}
```

DOUBLY LINKED LIST DATA STRUCTURE

ENTER THE ELEMENT :

ENTER THE ELEMENT :

300 100 200

Display Reverse Operation

```

Node temp;
String msg="";
if (first == null)
{
    JOptionPane.showMessageDialog(contentPane, "Display is not
    Possible");
}
else if (first.nextlink == null)
{
    msg=" "+first.data;
    res.setText(msg);
}
else
{
    temp = first;
    while (temp.nextlink != null)
    {
        temp = temp.nextlink;
    }
    while (temp != null)
    {
        msg=msg+" "+temp.data;
        temp = temp.prelink;
    }
}

```

DOUBLY LINKED LIST DATA STRUCTURE

ENTER THE ELEMENT :

ENTER THE ELEMENT :

Real Time Application of an Doubly Linked List

1. **Browser Caching:** Browser caching is a feature used by web browsers to store a copy of a web page in the local browser cache in order to speed up the loading time of the page when it is revisited. The doubly linked list data structure is often used to handle the memory cache size and eviction policy of a browser cache.
2. **Operating Systems:** Operating systems often use doubly linked list data structures to manage memory, manage processes, and maintain file directories. Memory management algorithms such as the buddy system and memory slab allocation use doubly linked lists to track memory blocks and maintain a list of free and allocated memory blocks.
3. **Network Routing:** Network routing protocols such as OSPF and BGP use doubly linked list data structures to store routing information and compute the shortest paths in a network. The data structure is used to store information about the network topology, such as the cost of each link in the network.

4. Conclusion

In linear data structure, data elements are ordered in a sequential order, with each element connected to the previous and next element.

Arrays, Linked List, Stack, and Queue are the different types of linear data structures.

Array elements store in a contiguous memory location but Linked list elements can be stored anywhere in the memory.

Stack follows Last in First out and Queue follows First in First out.

Both insert and delete operations inside stack and queue take $O(1)$ time.

In linear data structures memory is not efficiently utilized as compared to the non-linear data structures.

5. Future Work

1. Data structures are important because they allow technology professionals to organize and sort information on a digital system. They also allow non-technology professionals an easy-to-access tool for finding key business information.
2. We have concluded only using the linear data structure in this project, non-linear data structure will be implemented in further project.
3. Using singly linked list I have implemented the both functionalities of insert (front & rare) and delete (front & rare) inserting and deleting the element in the middle functionality will be implemented in the future.
4. As of now error message for Negative elements and strings values are not been implemented in this project. These functionalities will be implemented future projects.
5. Exception handling should be implemented