REVIEW WEEK: Linked list, stack, queue

Team 4

- Chea Ilong
- Prom sereyreaksa
- Kim Solida
- Ly Dara
- Yan Kimhong
- Phorn Leangchheng
- Son Nevaly
- Meas Sophanith
- Um Lyrithyreach
- Chetha Navid
- Pich sokreaksa
- Try Bunheng

LINKED LIST

1. What is a linked list data structure?

   A linked list is a data structure that stores a sequence of elements, called nodes, in which each node contains a reference to the next node. The first node is known as the head, and the last node is known as the tail.

2. How do you describe the differences between Linked list and Array?

   An array is a grouping of data elements of equivalent data type. A linked list is a group of entities called a node.

3. Explain the concepts/data structures for creating linked list.

A **linked list** is a dynamic data structure where each element(node) contains:

1. **Data**: Store the value
2. **Pointer**: References the next node in the list.

Types:

- **Singly Linked Lis**t: Nodes point forward
- **Doubly Linked List:** Nodes point forward and backward.
- **Circular Linked List:** Last node points back to the first.

Common operation:

- **Insert:** Add a node at the start, end, or middle.
- **Delete:** Remove a node by adjusting pointers.
- **Traverse:** Visit nodes sequentially.

Advantages: Dynamic resizing, easy insertion/deletion.

Disadvantages: Sequential access, pointer overhead.

4. List down the ADT operations for the linked list data structure.

Answer List down the ADT operations for the linked list data structure:

- insert begin
- insert end
- insert at
- delete begin
- delete end
- delete at
- display all

- Search
- check if empty
- check if full

LINKED LIST

Specification for the ADT operations of Linked list data structure below.

ADT operations	Syntax	Description	Precondition	Example	Time Complexity
Insert begin	void insertBegin(int value)	Inserts a value at the beginning of the list.	None	MyList=[2, 3, 4] insertBegin(1); -> [1, 2, 3, 4]	O(1)

Insert end	void insertEnd(int value)	Inserts a value at the end of the list.	None	MyList=[1, 2, 3] insertEnd(4); -> [1, 2, 3, 4]	O(1) for linked list, O(n) for dynamic array
Insert at	void insertAt(int index, int value)	Insert a value at the given index of the list	Index < 0 || index > length	Mylist =[10, 20, 1, 12, 7] Mylist.insertAt(2, 11) => mylist is now[10, 20, 11, 1, 12, 7]	O(n)
Display all	void displayAll()	Display all elements in linked list	None	myLinkedlist=[1,2,3,4] displayAll(); → Result : [1,2,3,4]	O(n)

Search	int search(int value)	Searches for a value and returns its index.	None	MyList=[5, 8, 10] search(8); -> 1	O(n)
Delete begin	void deleteBegin()	Delete the first element of the list	Size != 0	myList = [5, 2, 14, 8] mylist->deleteBegin(); -> Result: [2, 14, 8]	O(1)
Delete end	Void deleteEnd()	Delete the last element of the list	Size!=0	MyList=[2,5,7,9] MyList->deleteEnd(); >Result:[2,5,7]	O(1)
Delete at	void deleteAt(int index)	Deletes an element at a specified index.	index >= 0 && index < length	MyList=[3, 5, 7] deleteAt(1); -> [3, 7]	O(n)
Check is empty	bool isEmpty()	Returns true if the list is empty, false otherwise.	None	MyList=[] isEmpty(); -> true	O(1)

IMPLEMENTATION OF STACK

a) What is the constructor used for?

In class-based, object-oriented programming, a constructor is a special type of function called to create an object.

```cpp
#include<iostream>
using namespace std;

struct Box{
    int data;
    Box *next;  //linker / connection to other box
};

class LinkedList{
public:
    Box *head, *tail;
    int size;

    //Construcor
    LinkedList(){
        head = NULL;
        tail = NULL;
        size = 0;
    }
    void insertBegin(int newData){
        Box *b;
        b = new Box; //Memory allocation
        b->data = newData;

        //Add more code
    }
    void displayList(){
        Box *t;
        t = head;

        //Add more codes
    }
    void deleteBegin(){

    }
};
```

b) What are the ADT operations that we have here?

From the code provided, the following ADT operation are partially implemented:

- InsertBegin(int newData): Adds new element to the beginning of the list.
- DisplayList(): Traverse and displays all elements in the list
- DeleteBegin(): Should remove an element from the beginning of the list.

c) Are there any ADT operations that we can create?

The ADT operation that we can create:

. Stack ADT

. Queue ADT

d) How do you implement each ADT operation?

- **Push**: Add an item to the top of the stack.
- **Pop**: Remove and return the top item.
- **Peek**: Return the top item without removing it.
- **IsEmpty**: Check if the stack is empty.

e) What are the variables **size head**, and **tail** used for ?

- **Size:** Tracks the number of elements in the queue.
- **Rear:** Points to the last node in the queue (where new elements are added).
- **Front:** Points to the first node in the queue (where elements are removed).

STACK

- Stack is a data structure that implements the LIFO principle. It ……
- **LIFO: Last In First Out** means that Items inserted last will be the first to be removed (retrieve)
- ADT operations:
  - **push()**
  - **pop()**
  - **isempty()**
  - **peek()**
  - **display() or print() or getstring()**
  - **search()**

STACK

Specification for the ADT operations of stack data structure when

Specification	Push(value)	Peek()	Pop()	Display()	Search(value)	IsEmpty()
Syntax	Void push(int value)	Int peek()	Void pop()	String display()	Int search(int value)	Bool IsEmpty()
Description	Add an element to the top of the stack	Return the top element without removing it	Removes the top element from the stack.	Return a string representation of the	Searches for a value and returns its index.	Checks if the stack is empty.

				Stack (Top -> bottom)		
Precondition	none	Stack not empty	Stack not empty	none	none	none
Example	myStack <- [ ] myStack.push(11) myStack.push(22) Result: [22, 11]	myStack <- [33, 22, 11] result = myStack.peek() Result: result = 33	myStack <- [33, 22, 11] myStack.pop() Result [22, 11]	myStack <- [33, 22, 11] result = myStack.print() Result: result = 33 22 11	myStack <- [33, 22, 11] myStack.search(22) Result: index 1	myStack <- [33, 22, 11] myStack.isEmpty() Result: false
Time complexity	O(1)	O(1)	O(1)	O(n)	O(n)	O(1)

IMPLEMENTATION OF STACK

a) What is the constructor used for?

The constructor in the Stack class initializes the attributes size and top:

- size is set to 0 to indicate an empty stack.
- top is set to nullptr to represent that there are no elements in the stack.

b) What are the ADT operations that we have here?

The stack class includes the following common ADT operation:

```
1  #include<iostream>
2  using namespace std;
3
4  struct Box{
5      int data;
6      Box *next;
7  };
8
9  class Stack {
10 private:
11     int size;
12     Box *top;
13
14 public:
15     Stack() {
16
17     }
18
19     void push(int newData){
20
21     }
22     int pop(){
23
24     }
25     bool isEmpty(){
26
27     }
28     int peek(){
29
30     }
31     void displayStack(){
32
33     }
34 };
```

- **Push:** Adds an element to the top of the stack.
- **Pop:** Removes and returns the top element from the stack.
- **IsEmpty:** Checks if the stack is empty.
- **Peek:** Returns the data at the top of the stack without removing it.
- **DisplayStack:** Displays all elements in the stack.

c) How do you implement each ADT operation?

- **Push:** Create a new Box node with the provided data, link it to the current top, and update top to point to the new node. Increment size.
- **Pop:** Remove the node at top, update top to point to the next node, return the data of the old top node, and decrement size.
- **IsEmpty:** Check if size == 0 or if top == nullptr.
- **Peek:** Access and return the data in the top node.

- **DisplayStack:** Traverse from top to the bottom and print the data in each node.

d) What do the variables **size** and **top** represent?

- **Size**: Tracks the number of elements in the stack.
- **Top:** Point to the top node in the stack (the last added element).

QUEUE

❑Queue Is a data structure that implements the FIFO principle.

It ……

❑**FIFO : First In First Out** mean that …..Items inserted first will be the first to be removed (pop)

ADT operations:

- **enqueue() deq()**
- **isEmpty()**
- **peek()**
- **display() or print() or getString()**
- **search()**

QUEUE

Specification for the ADT operations of queue data structure when implementing using linked list.

Specification	Enqueue(value)	Peek()	Dequeue()	Display()	Search(value)	IsEmpty()
Syntax	Void enqueue(value)	int peek()	void dequeue()	void display()	int search(int value)	bool isEmpty()
Description	Add data to the rear of queue	Returns the front element of the queue without removing it.	Removes the front element from the queue.	Displays all elements in the queue.	Searches for a value in the queue and returns its position.	Returns true if the queue is empty, false otherwise.
Precondition	None	Queue is not empty	Queue is not empty	None	None	None

Example	MyQueue = [1, 2, 3] MyQueue -> enqueue(5); -> Result : [1, 2, 3, 5]	myQueue = [1, 2, 3] peek(); -> 1	myQueue = [1, 2, 3] dequeue(); -> [2, 3]	myQueue = [1, 2, 3] display(); -> [1, 2, 3]	myQueue = [1, 2, 3, 5] search(3); -> 2	myQueue = [] isEmpty(); -> true
Time complexity	O(1)	O(1)	O(1)	O(n)	O(n)	O(1)

IMPLEMENTATION OF QUEUE

a) What is the constructor used for?

Constructor is used to initialize the queue object when it is created. The constructor is a queue class that typically sets up an empty queue (or initializes it with some values, depending on the implementation) and prepares it to store items. It often defines internal variables, such as list or an array, to hold the elements in the queue.

b) What are the ADT operations that we have here?

```
1  #include<iostream>
2  using namespace std;
3  struct Box{
4      int data;
5      Box *next;
6  };
7
8  class Queue{
9   private:
10     //Attributes /Variables
11     int size;
12     Box *rear;
13     Box *front;
14
15  public:
16     //Constructor
17     Queue ()
18     {
19
20     }
21
22     //Method / ADT operations / Functions
23     void enqueue(int newData){
24
25     }
26     void dequeue(){
27
28     }
29
30     bool isEmpty(){
31
32     }
33     int peek(){
34
35     }
36     void displayQueue(){
37
38     }
39 };
```

The ADT operations that we have:

. enqueue

. dequeue

. peek

. isEmpty

. displayQueue

c) How do you implement each ADT operation?

. Enqueue: Add an item to the rear of the queue.

. Dequeue: Remove and return the front item.

. Front: Return the front item without removing it.

. IsEmpty: Check if the queue is empty.

. Size Get the number of elements in the queue.

D. What do the variables **size**, **rear**, and **front** represent?

- **size**:
- **Represents**: The number of elements currently in the queue.
- **Purpose**: It helps track how many items are in the queue, useful for checking if the queue is full (in a bounded queue) or empty.
- **front**:
- **Represents**: The element at the front of the queue, which is the next element to be dequeued.

- **Purpose**: It points to the first item in the queue and is accessed when removing items (dequeue operation).
- **rear**:
- **Represents**: The element at the rear (or end) of the queue, where new elements are added.
- **Purpose**: It points to the last item in the queue and is used when adding items (enqueue operation).