COURSE TITLE : DATA STRUCTURES AND ALGORITHMS
INSTRUCTOR : MR. MARK KENNETH LIMJOCO

1. Array Data Structure

- **Key Concept**

An **array** is a fundamental data structure that stores a collection of elements of the same data type in contiguous memory locations. Each element in an array is identified by an index, and you can access elements by their index.

- **Learning Tasks**
  - **Merging four arrays into one.**

ViñasJudahPaulo_BSIT2F.cpp

```cpp
// MERGING ARRAYS
#include <iostream>
using namespace std;

int main() {
    int arr1[] = {1, 2, 3};
    int arr2[] = {4, 5, 6};
    int arr3[] = {7, 8, 9};
    int arr4[] = {10, 11, 12};
    int merged[12]; // Merged array (size - 12)

    // Copy elements from arr1 to merged
    for (int i = 0; i < 3; i++) {
        merged[i] = arr1[i];
    }

    // Copy elements from arr2 to merged
    for (int i = 0; i < 3; i++) {
        merged[i + 3] = arr2[i];
    }

    // Copy elements from arr3 to merged
    for (int i = 0; i < 3; i++) {
        merged[i + 6] = arr3[i];
    }

    // Copy elements from arr4 to merged
    for (int i = 0; i < 3; i++) {
        merged[i + 9] = arr4[i];
    }

    // Print the merged array
    cout << "MERGED : ";
    for (int i = 0; i < 12; i++) {
        cout << merged[i] << " ";
    }
    return 0;
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaulo_B...

```
MERGED : 1 2 3 4 5 6 7 8 9 10 11 12
--------------------------------
Process exited after 0.1344 seconds with return value 0
Press any key to continue . . .
```

  - **Delete element in an array.**

ViñasJudahPaulo_BSIT2F.cpp

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

int main() {
    int i, j, n = 3, elem;
    int LA[3] = {1, 3, 5};
    bool x=true;

    cout << "The original array elements are :" << endl;
    for (i = 0; i < n; i++) {
        cout << "LA[" << i << "] = " << LA[i] << endl;
    }
    cout << "Enter an element to delete : ";
    cin >> elem;

    for (i = 0; i < n; i++) {
        if (LA[i] == elem) {
            for (j = i; j < (n - 1); j++) {
                LA[j] = LA[j + 1];
            }
            n--; // Reduce the size of the array
            i--; // Recheck the current index in case the next element is also to be deleted
            x = false;
        }
    }

    if (x) {
        cout << "-- ELEMENT NOT FOUND! --";
    } else {
        cout << "The array elements after deletion :" << endl;
        for (i = 0; i < n; i++) {
            cout << "LA[" << i << "] = " << LA[i] << endl;
        }
    }
    return 0;
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJ...

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
Enter an element to delete : 3
The array elements after deletion :
LA[0] = 1
LA[1] = 5

--------------------------------
Process exited after 14.77 seconds with return value 0
Press any key to continue . . .
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJ...

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
Enter an element to delete : 12
-- ELEMENT NOT FOUND! --
--------------------------------
Process exited after 8.02 seconds with return value 0
Press any key to continue . . .
```

- **Search element in an array.**

ViñasJudahPaulo_BSIT2F.cpp

```cpp
//SEARCHING
#include <iostream>
using namespace std;
int main(){
    int LA[] = {1,3,5,7,8};
    int item, n = 5;
    int i = 0;

    cout << "The original array elements are : " <<endl;
    for(i = 0; i<n; i++) {
        cout << "LA[" << i << "] = " << LA[i] << endl;
    }
    // Input elements to find
    cout << "Enter element to search : ";
    cin >> item;

    //Display the position of
    for(i = 0; i<n; i++) {
        if( LA[i] == item ) {
            cout << "Found element " << item << " at position " << i+1 <<endl;
            break;
        }
    }

    if (LA[i] != item) {
        cout << "ELEMENT NOT FOUND!";
    }
    return 0;
}
```

D:\SCHOOL\Data Structure and Algorithms...

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Enter element to search : 5
Found element 5 at position 3

--------------------------------
Process exited after 40.62 seconds with return value
0
Press any key to continue . . .
```

D:\SCHOOL\Data Structure and Algorithm...

```
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Enter element to search : 12
ELEMENT NOT FOUND!
--------------------------------
Process exited after 2.413 seconds with return value
0
Press any key to continue . . .
```

- **Takeaways**
  - Arrays are collections of elements stored in contiguous memory locations, accessible by index.
  - Merging arrays involves copying elements from multiple arrays into a single array.
  - Deleting an element from an array may require shifting elements to fill the gap.
  - Searching for an element in an array involves iterating through the array and comparing each element until a match is found or the end is reached.

2. LinkList Data Structure

- **Key Concept**

  A **linked list** is a linear data structure in which elements are stored in nodes, and each node points to the next node in the sequence. Linked lists consist of a head (the first node) and a tail (the last node). They are dynamic in size and efficient for insertions and deletions.

- **Learning Tasks**

  ➢ **Merging of three LinkList into one**.

ViñasJudahPaulo_BSIT2F.cpp

```cpp
// MERGING 3 LINKLIST
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

class Node {
public:
    int value;
    Node* next;
};

// Function to merge two linked lists
void merge(Node* &head1, Node* &head2) {
    if (head1 == NULL) {
        head1 = head2;
        return;
    }
    Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }
    temp->next = head2;
    head2 = NULL; // Set head2 to NULL to avoid dangling pointers.
}

int main() {
    Node* head1 = NULL;
    Node* head2 = NULL;
    Node* head3 = NULL;

    // Create the first linked list
    Node* one = new Node();
    Node* two = new Node();
    Node* three = new Node();

    one->value = 10;
    two->value = 20;
    three->value = 30;
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPa...

```
1st LinkList : 10 20 30
2nd LinkList : 40 50 60
3rd LinkList : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90

--------------------------------
Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
```

ViñasJudahPaulo_BSIT2F.cpp

```cpp
    one->next = two;
    two->next = three;
    three->next = NULL;

    cout << "1st LinkList : ";
    cout << one->value << " " << two->value << " " << three->value << endl;

    // Create the second linked list
    Node* four = new Node();
    Node* five = new Node();
    Node* six = new Node();

    four->value = 40;
    five->value = 50;
    six->value = 60;

    four->next = five;
    five->next = six;
    six->next = NULL;

    cout << "2nd LinkList : ";
    cout << four->value << " " << five->value << " " << six->value << endl;

    // Create the third linked list
    Node* seven = new Node();
    Node* eight = new Node();
    Node* nine = new Node();

    seven->value = 70;
    eight->value = 80;
    nine->value = 90;

    seven->next = eight;
    eight->next = nine;
    nine->next = NULL;

    cout << "3rd LinkList : ";
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPa...

```
1st LinkList : 10 20 30
2nd LinkList : 40 50 60
3rd LinkList : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90

--------------------------------
Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
```

[*] ViñasJudahPaulo_BSIT2F.cpp

```cpp
    cout << "3rd LinkList : ";
    cout << seven->value << " " << eight->value << " " << nine->value << endl;

    // Merge all three linked lists
    merge(head1, one);
    merge(head1, four);
    merge(head1, seven);

    // Print the merged linked list
    Node* current = head1;
    cout << "----- MERGED LINKLIST -----" << endl;
    while (current != NULL) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;

    return 0;
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPa...

```
1st LinkList : 10 20 30
2nd LinkList : 40 50 60
3rd LinkList : 70 80 90
----- MERGED LINKLIST -----
10 20 30 40 50 60 70 80 90

--------------------------------
Process exited after 0.1484 seconds with return value 0
Press any key to continue . . .
```

- **Delete node at the beginning and end.**

ViñasJudahPaulo_BSIT2F.cpp

```cpp
//DELETING NODE
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

// Creating a node
class Node {
    public:
    int value;
    Node* next;
};
int main() {
    Node* head;
    Node* one = NULL;
    Node* two = NULL;
    Node* three = NULL;
    Node* four = NULL;
    Node* five = NULL;
    // allocate 3 nodes in the heap
    one = new Node();
    two = new Node();
    three = new Node();
    four = new Node();
    five = new Node();
    // Assign value values
    one->value = 1;
    two->value = 2;
    three->value = 3;
    four->value = 45;
    five->value = 55;
    // Connect nodes
    one->next = two;
    two->next = three;
    three->next = four;
    four->next = five;
    five->next = NULL;

    // display before deleting
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJud...

```
----- Before Deleting -----
1
2
3
45
55
----- After Deleting -----
2
3
45

--------------------------------
Process exited after 0.1106 seconds with return value 0
Press any key to continue . . .
```

[*] ViñasJudahPaulo_BSIT2F.cpp

```cpp
    // display before deleting
    Node* val = one;
    cout << "----- Before Deleting -----" << endl;
    while (val != NULL) {
        cout << val->value << endl;
        val = val->next;
    }

    cout << "----- After Deleting -----" << endl;
    head = one;
    head = head->next;     //delete form start

    // print the linked list value
    while (head->next != NULL) {   //delete form end
        cout << head->value<<endl;
        head = head->next;
    }
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJud...

```
----- Before Deleting -----
1
2
3
45
55
----- After Deleting -----
2
3
45

--------------------------------
Process exited after 0.1106 seconds with return value 0
Press any key to continue . . .
```

- **Insert node at the beginning, middle and end.**

ViñasJudahPaulo_BSIT2F.cpp

```cpp
// INSERTING NODE
#include <bits/stdc++.h>
#include <iostream>
using namespace std;

// Creating a node
class Node {
    public:
    int value;
    Node* next;
};
int main() {
    Node* head;
    Node* one = NULL;
    Node* two = NULL;
    Node* three = NULL;
    Node* four = NULL;
    Node* five = NULL;
    // allocate nodes in the heap
    one = new Node();
    two = new Node();
    three = new Node();
    four = new Node();
    five = new Node();
    // Assign value values
    one->value = 10;
    two->value = 20;
    three->value = 30;
    four->value = 40;
    five->value = 50;
    // Connect nodes
    one->next = two;
    two->next = three;
    three->next = four;
    four->next = five;
    five->next = NULL;

    cout << "----- Before Inserting -----" << endl;
```

Select D:\SCHOOL\Data Structure and Algorithms\...

```
----- Before Inserting -----
10
20
30
40
50
----- After Inserting -----
5
10
20
30
35
40
50
55

--------------------------------
Process exited after 0.05602 seconds with return value 0
Press any key to continue . . .
```

ViñasJudahPaulo_BSIT2F.cpp

```cpp
    cout << "----- Before Inserting -----" << endl;
    Node* val = one;
    while (val != NULL) {
        cout << val->value << endl;
        val = val->next;
    }

    //inserting on the start
    Node* start = NULL;
    start = new Node();
    start->value = 5;
    start->next = one;
    //inserting on the middle
    Node* mid = NULL;
    mid = new Node();
    mid->value = 35;
    three->next = mid;
    mid->next = four;
    //inserting to the last
    Node* last = NULL;
    last = new Node();
    last->value = 55;
    five->next = last;

    cout << "----- After Inserting -----" << endl;
    // print the linked list value
    head = start;
    while (head != NULL) {
        cout << head->value<<endl;
        head = head->next;
    }
}
```

Select D:\SCHOOL\Data Structure and Algorithms\...

```
----- Before Inserting -----
10
20
30
40
50
----- After Inserting -----
5
10
20
30
35
40
50
55

--------------------------------
Process exited after 0.05602 seconds with return value 0
Press any key to continue . . .
```

- **Takeaways**
  - Linked lists are dynamic data structures where elements are stored in nodes, and each node has a pointer to the next node.
  - Merging linked lists involves updating the next pointers appropriately.
  - Deleting and inserting nodes at different positions in a linked list require updating pointers accordingly.
  - Linked lists are useful when you need dynamic size and efficient insertions and deletions compared to arrays.

3. Stack Data Structure

- **Key Concept**

A **stack** is a linear data structure that follows the Last-In-First-Out (LIFO) principle. In a stack, elements are added and removed from the same end, known as the top. It is often used for tasks that involve keeping track of the order of elements, such as function call execution and expression evaluation.

- **Learning Tasks**
  - **Stack Array**

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp | [*] StackData.cpp

```cpp
// S T A C K    A R R A Y
#include <iostream>
#define MAX 4
using namespace std;

int stack_arr[MAX];
int top = -1;

int isFull(){
    if(top == MAX-1) return 1;
    else return 0;
}

void push(int data){
    if(isFull()){
        cout << "\n----------------------------\n";
        cout<<"Stack overflow!\n"<<endl;
        cout << "----------------------------\n";
        return;
    }
    top = top + 1;
    stack_arr[top]=data;
}

int isEmpty(){
    if(top == -1) return 1;
    else return 0;
}

int pop(){
    int value;
    if(isEmpty()){
        cout << "\n----------------------------\n";
        cout<<"Stack underflow!" <<endl; //exit(1);
        cout << "----------------------------\n";
```

D:\SCHOOL\Data Structure and Algorithms\Vi±as...

```
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1

Enter value to push in a stack: 1

Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1

Enter value to push in a stack: 2

Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 1
```

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp | [*] StackData.cpp

```cpp
    }
    value=stack_arr[top];
    top = top - 1;
    return value;
}

int peek(){
    if(isEmpty()){
        cout << "\n----------------------------\n";
        cout<<"Stack underflow!\n" <<endl; //exit(1);
        cout << "----------------------------\n";
    }
    return stack_arr[top];
}

void display(){

    if(isEmpty()){
        cout << "\n----------------------------\n";
        cout<<"Stack underflow!\n"<<endl; //exit(1);
        cout << "----------------------------\n";
    }
    cout << "\n----------------------------";
    cout<<"\nThe value of stack: "<<endl;
    for(int i = top; i >=0; i--){
      cout<<stack_arr[i]<<endl;
    }
    cout << "----------------------------\n";
}

int main() {
//cout<<"top value is "<<peek();
    while(1){
        int choice, data;
        cout<<"\nChoices: \n";
```

D:\SCHOOL\Data Structure and Algorithms\Vi±as...

```
Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 3

----------------------------
Top value is 2
----------------------------

Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 4

----------------------------
The value of stack:
2
1
----------------------------

Choices:
1. push element into stack
2. pop element from stack
```

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp | [*] StackData.cpp

```cpp
int main() {
//cout<<"top value is "<<peek();
    while(1){
        int choice, data;
        cout<<"\nChoices: \n";
        cout<<"1. push element into stack \n";
        cout<<"2. pop element from stack \n";
        cout<<"3. peek top value of stack \n";
        cout<<"4. display elements of stack \n";
        cout<<"5. exit \n";
        cout<<"Enter your choice: ";
        cin>>choice;
        switch(choice){
            case 1:
                cout<<"\nEnter value to push in a stack: ";
                cin>>data;
                push(data);
                break;
            case 2:
                pop();
                break;
            case 3:
                cout << "\n----------------------------";
                cout<< "\nTop value is " <<peek() << endl;
                cout << "----------------------------\n";
                break;
            case 4:
                display();
                break;
            case 5:
                exit(1);
        }
    }
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±as...

```
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice: 4

----------------------------
The value of stack:
2
1
----------------------------

Choices:
1. push element into stack
2. pop element from stack
3. peek top value of stack
4. display elements of stack
5. exit
Enter your choice:
```

- **Stack LinkList**

ViñasJudahPaulo_BSIT2F.cpp | [*] StackData.cpp | QueueArray.cpp

```cpp
// S T A C K     L I N K L I S T
#include <iostream>
using namespace std;

struct node{
    int value;
    struct node* link;
} *top = NULL;

void push(int value){
    struct node* newNode;
    newNode = new node();
    //newNode = malloc(sizeof(newNode));
    if(newNode==NULL){
        cout<<"Stack Overflow";
        exit(1);
    }
    newNode->value=value;
    newNode->link=NULL;
    newNode->link=top;
    top=newNode;
}
int isEmpty(){
    if(top == NULL) return 1;
    else return 0;
}
int pop(){
    if(isEmpty()){
        cout<<"Stack Underflow!";
        exit(1);
    }
    int val = top->value;
    top=top->link;
    //free(top);
    //top=NULL;
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaul...

```
The topmost element of the stack is 1
The last deleted: 112

The stack ELements are:
1

--------------------------------
Process exited after 0.1446 seconds with return value 0
Press any key to continue . . .
```

ViñasJudahPaulo_BSIT2F.cpp | [*] StackData.cpp | QueueArray.cpp

```cpp
    //free(top);
    //top=NULL;
    return val;
}
int peek(){
    if(isEmpty()){
        cout<<"Stack Underflow!";
        exit(1);
    }return top->value;
}

int main() {

    int val;
    push(1);
    push(112);
    push(23);
    val = pop();
    val = pop(); //pop();pop();pop();pop();
    cout<<"The topmost element of the stack is " <<peek(
    cout << "The last deleted: " << val << endl <<endl;
    cout<<"The stack ELements are: \n";
    while(top){
        cout<<top->value<<endl;
        top=top->link;
    }
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaul...

```
The topmost element of the stack is 1
The last deleted: 112

The stack ELements are:
1

--------------------------------
Process exited after 0.1446 seconds with return value 0
Press any key to continue . . .
```

- **Takeaways**
  - A stack is a data structure that follows the LIFO (Last-In-First-Out) principle.
  - Elements are added and removed from the top of the stack.
  - The ‘stack’ container in C++ Standard Library simplifies stack operations.
  - Stacks are commonly used in scenarios where the order of operations matters, such as function call execution, undo functionality, and expression evaluation.

4. Queue Data Structure

- **Key Concept**

A **queue** is a linear data structure that follows the First-In-First-Out (FIFO) principle. In a queue, elements are added at the rear (enqueue) and removed from the front (dequeue). It is often used in scenarios where elements are processed in the order they are added, like task scheduling, print queue, etc.

- **Learning Tasks**
  - **Queue Array**

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp

```cpp
// Q U E U E    A R R A Y
#include <iostream>
#define MAX 4
using namespace std;

int q_arr[MAX];
int rear = -1;
int front=0;
int count=0;

int isFull(){
    return count == MAX;
}

void enqueue(int data){
    if(!isFull()){
        if(rear == MAX-1){
            rear =-1;
        }
        q_arr[++rear] = data;
        count++;
    }else
    cout<<"Queue Overflow"<<endl;
}
int isEmpty(){
    return count==0;
}

int dequeue(){
    if(isEmpty()){
        cout<<"Queue underflow!"<<endl; exit(1);
    }
    int value=q_arr[front++];
    if(front==MAX){
        front=0;
    }
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaul...

```
Queue:  233 244
--------------------------------
Process exited after 0.1198 seconds with return value 0
Press any key to continue . . .
```

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp

```cpp
    if(front==MAX){
        front=0;
    }
    count--;
    return value;
}

int peek(){
    if(isEmpty()){
        cout<<"Queue underflow!"<<endl; exit(1);
    }
    return q_arr[front];
}


int main() {
    enqueue(22);
    enqueue(2);
    enqueue(233);
    enqueue(244); //enqueue(4355);
    dequeue();dequeue();//dequeue();dequeue();dequeue();
    //cout<<"the first element is "<<peek()<<endl;
    cout<<"Queue: ";
    while(!isEmpty()){
        int val = dequeue();
        cout<<" "<<val;
    }
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaul...

```
Queue:  233 244
--------------------------------
Process exited after 0.1198 seconds with return value 0
Press any key to continue . . .
```

- **Queue LinkList**

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp

```cpp
// Q U E U E   L I N K L I S T
#include <iostream>
using namespace std;

struct node{
    int value;
    struct node* link;
}
*front = NULL,
*rear = NULL;

int isEmpty(){
    if(front == NULL) return 1;
    else return 0;}

void push(int data){
    node* newNode = new node();
    if (isEmpty()) {
        front = newNode;
        rear = newNode;
    }
    newNode->value=data;
    rear->link = newNode;
    rear = newNode;
}


int pop(){
    if(isEmpty()){
        cout<<"Stack Underflow!";
        exit(1);
    }
    int val = front->value;
    front=front->link;

    return val;
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPaul...

```
The stack Elements are:
63 63 63 63 63 63 63
--------------------------------
Process exited after 0.1318 seconds with return value 0
Press any key to continue . . .
```

ViñasJudahPaulo_BSIT2F.cpp | QueueArray.cpp

```cpp
        exit(1);
    }
    int val = front->value;
    front=front->link;

    return val;

}
int peek(){
    if(isEmpty()){
        cout<<"Queue Underflow!";
        exit(1);
    }return front->value;
}

int main() {

    push(112);
    push(23);//pop();pop();pop();pop();
    push(53);
    push(63);
    pop();pop();pop();
    push(63);push(63);push(63);push(63);push(63);push(63)
    //cout<<"The topmost element of he stack is "<<peek()

    cout<<"The stack Elements are: \n";
    while(front){
        cout<<front->value<< " ";
        front=front->link;
    }
}
```

D:\SCHOOL\Data Structure and Algorithms\Vi±asJudahPa...

```
The stack Elements are:
63 63 63 63 63 63 63
--------------------------------
Process exited after 0.1318 seconds with return value 0
Press any key to continue . . .
```

- **Takeaways**
  - A queue is a data structure that follows the FIFO (First-In-First-Out) principle.
  - Elements are added to the rear (enqueue) and removed from the front (dequeue) of the queue.
  - The ‘queue’ container in C++ Standard Library simplifies queue operations.
  - Queues are commonly used in scenarios where the order of processing matters, such as task scheduling and managing resources.

Overall Reflection:

Course Impact:

The data structures course, which covered arrays, linked lists, queues, and stacks, greatly shaped my general understanding and perspective on data structures and algorithms. It has given me a firm understanding of the fundamental concepts, implementation details, and real-world applications of various data structures.

I obtained practical insights into how these data structures function and how they may be utilized for solving various computational problems through hands-on examples and code implementations. This knowledge has improved not just my programming skills but also my problem-solving ability.

The course also emphasized the importance of choosing the right data structure for individual applications, taking into account considerations such as efficiency, memory utilization, and ease of manipulation. It has increased my understanding of the importance of data structures in software development and their function in optimizing algorithmic solutions.

Recommendations:

- **Algorithm Analysis**: Include a section on algorithm analysis for a better understanding of time and space complexity in relation to data structures.
- **Problem-Solving Challenges**: Integrate coding challenges and problem-solving exercises that apply data structure concepts to real-world scenarios for practical experience.
- **Visualizations**: Incorporate visual aids and animations to make abstract data structure concepts more accessible and engaging.