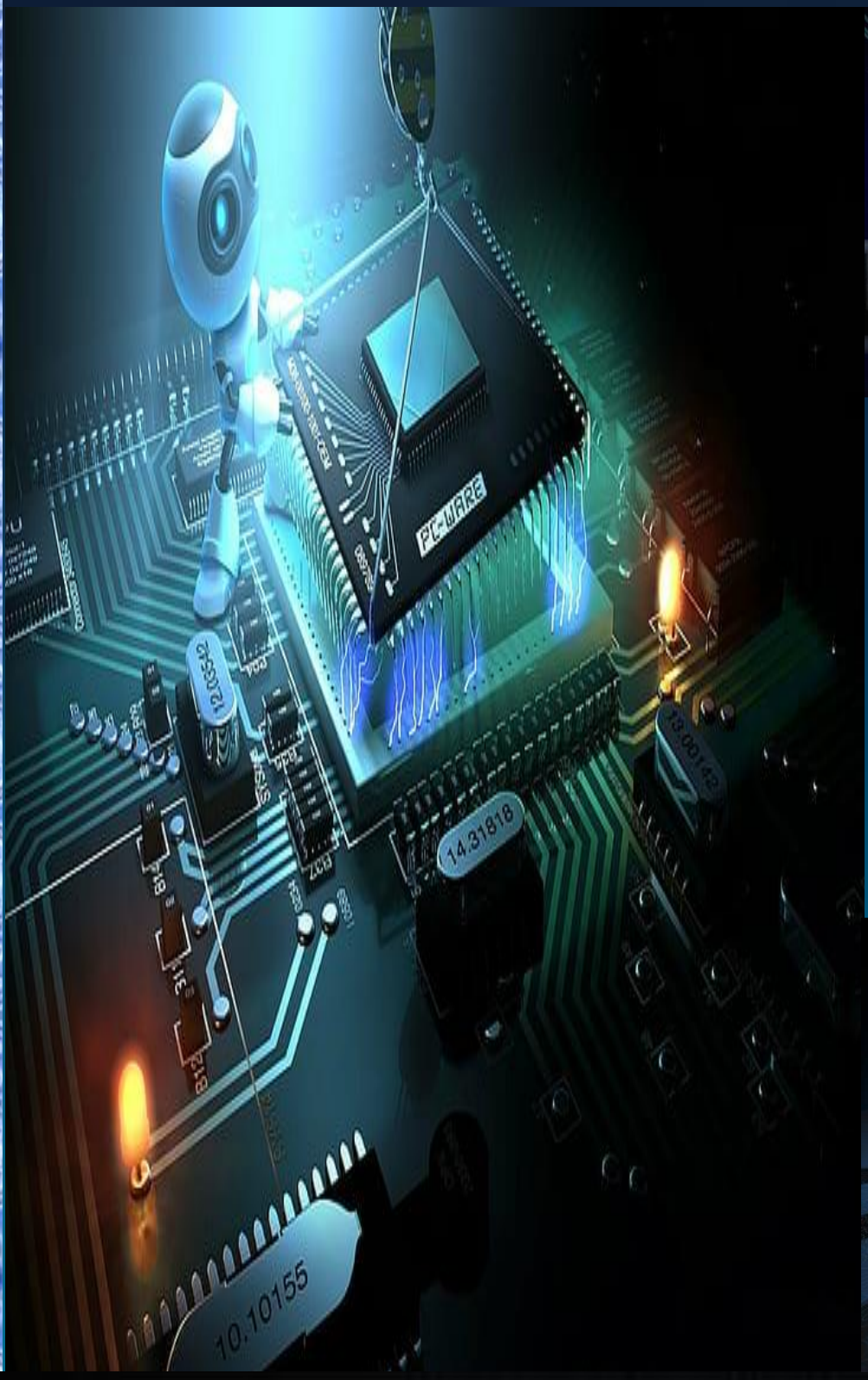
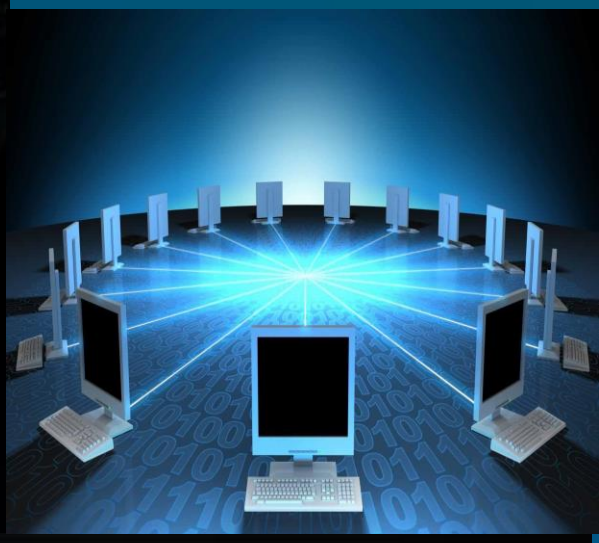


DSA



Data Structure and Algorithm

Presented:

Dr. Salman Naseer

Presented by:

Abdullah	BIT21213
Abdullah Afzal	BIT21237
Muhammad Bilal	BIT21248
Muhammad Usama	BIT21252
Syed Muneeb Abbas Shah	BIT21254

ABSTRACT

The aim of this project is to show the comparison of Quick sort and Merge sort techniques. The program is menu driven to fit to the convenience of the user, depending on how many words he wants to enter, and sort them through the sorting technique of his/her choice. In the end I will be showing which sorting is better, through the respective worst and best case time complexity.

Acknowledgement

I would like to express my special thanks of gratitude to my teacher **Dr. Salman Naseer** who gave me the golden opportunity to do this wonderful project on the topic (Types of websites), which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.
Secondly i would also like to thank my parents and friends who helped me a lot in finalizing this project within the limited time frame.

Table of Contents

List of Figures

List of Tables

List of abbreviation

What is Data Structure?

- A data structure is a storage used to organize, process, retrieve, and store data. It is a way of organizing data on a computer to be accessed and updated efficiently.
- Data Structure is divided into two types: Linear Data Structures and Non-Linear Data Structures.
- For Example, Arrays, Stack, and Queues are examples of Linear Data Structure, and Linked List, Trees are examples of Non-Linear Data Structure.

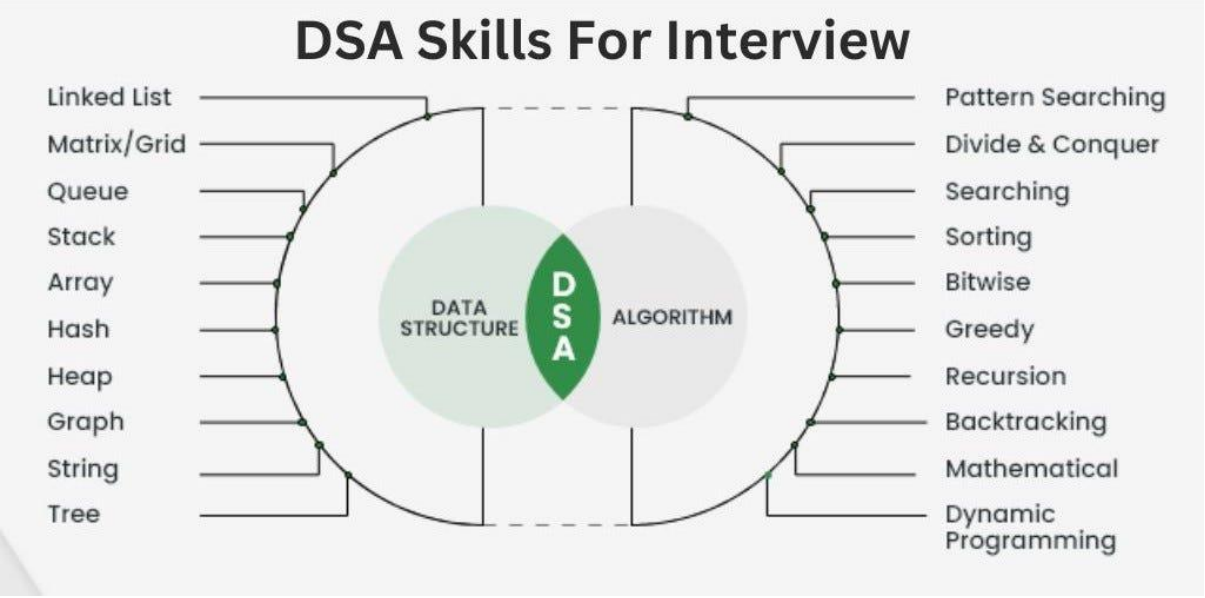
What is an Algorithm?

- An algorithm is a technique for solving a problem or performing an analysis.
- Algorithms act as an accurate list of instructions that execute specified actions step by step in either hardware or software-based practices.
- Algorithms are widely used throughout all regions of IT.

What Are the Benefits of Conducting DSA Projects?

We have two benefits while conducting DSA Project,

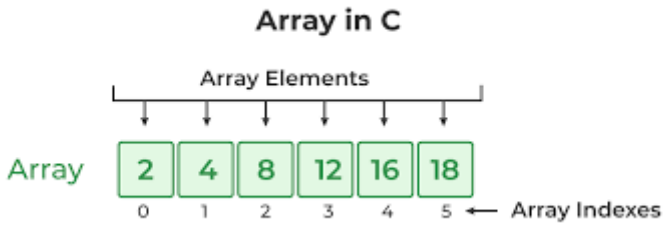
- The first one is we will get to know the real-world use of Data structure and Algorithms.
- And the second benefit is, It will help in placements.
- When your resume has some projects related to Data Structure and Algorithms, the interviewer will get to know that you are aware of some of the Algorithms.
- And also, most of the questions in the interview will be related to that algorithm.



Library Management System

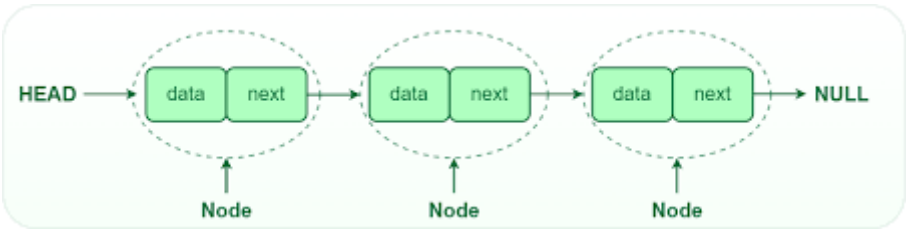
What is Array?

An array is a linear data structure that collects elements of the same data type and stores them in contiguous and adjacent memory locations. Arrays work on an index system starting from 0 to (n-1), where n is the size of the array.



What is link list?

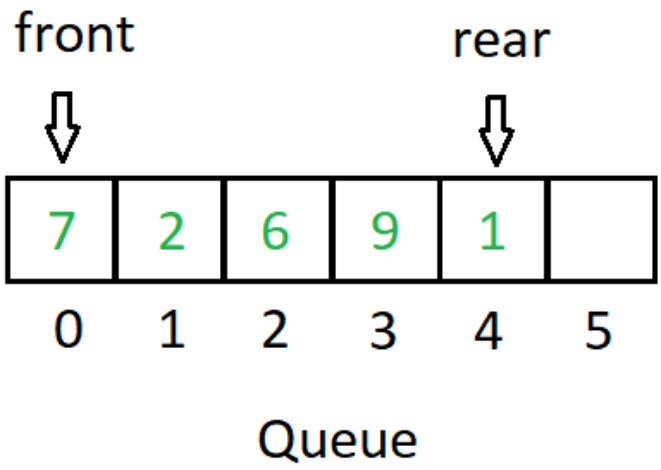
A linked list is the most sought-after data structure when it comes to handling dynamic data elements. A linked list consists of a data element known as a node. And each node consists of two fields: one field has data, and in the second field, the node has an address that keeps a reference to the next node.



What is queue?

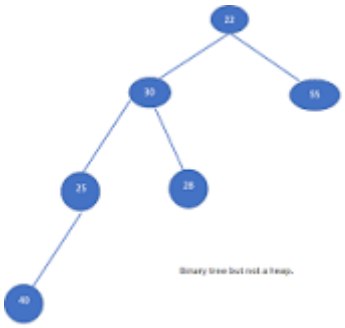
A queue is an object (an abstract data structure - ADT) that allows the following operations:

- Enqueue: Add an element to the end of the queue.
- Dequeue: Remove an element from the front of the queue.
- IsEmpty: Check if the queue is empty.



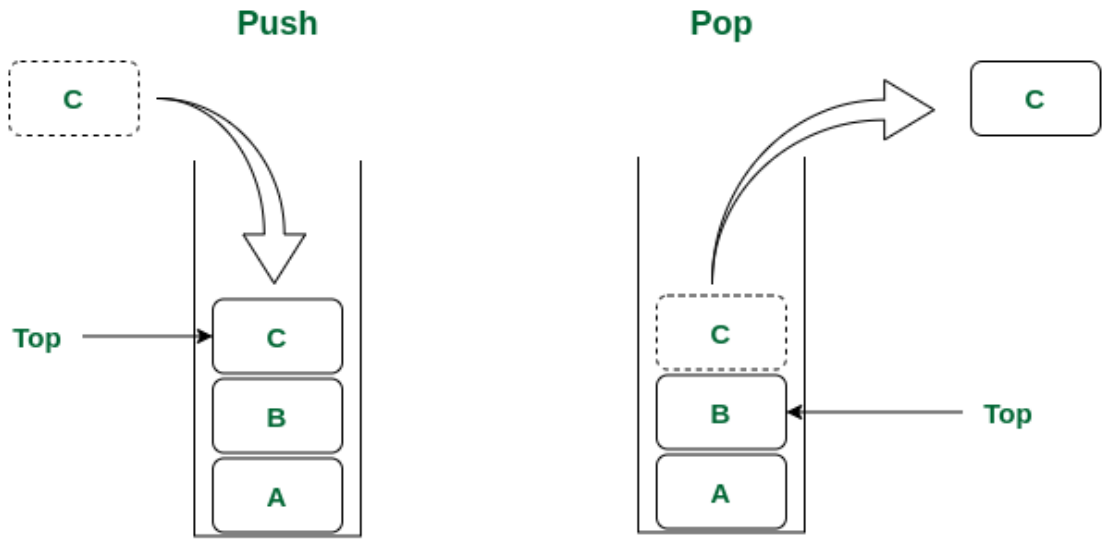
What is heap?

A heap is a special type of tree data structure; a heap tree is typically either a min-heap tree, in which each parent node is smaller than its children; or a max-heap tree, in which each parent node is larger than its children.



What is stack?

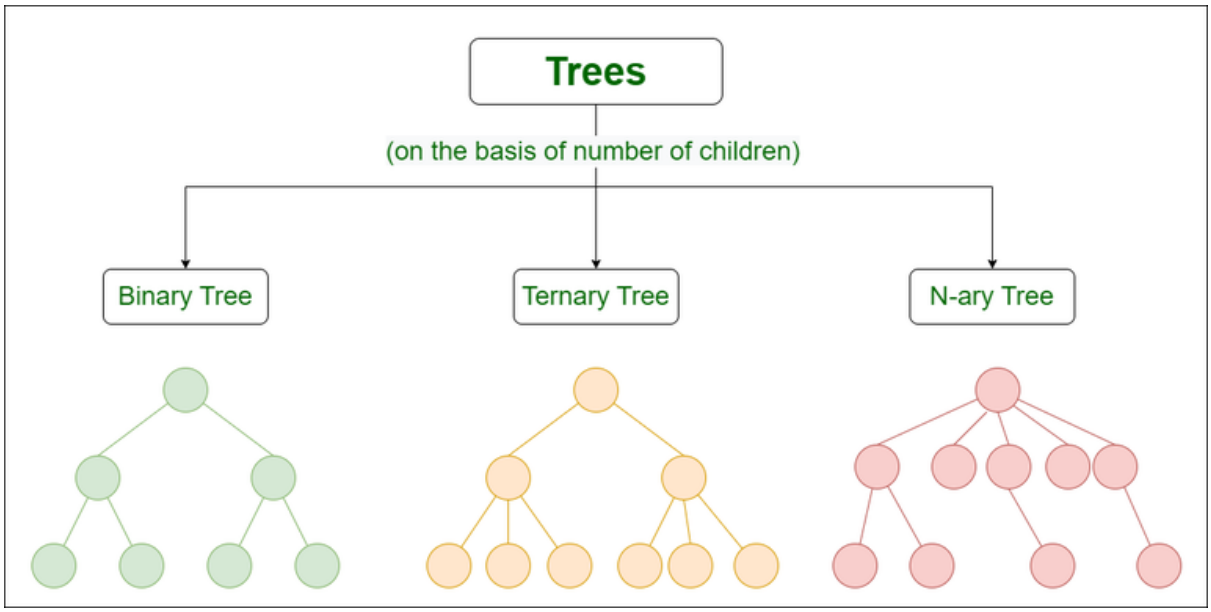
A stack is a linear data structure that follows the principle of Last In First Out (LIFO). This means the last element inserted inside the stack is removed first. You can think of the stack data structure as the pile of plates on top of another. Stack representation similar to a pile of plate.



Stack Data Structure

What is tree?

A tree is a hierarchical data structure defined as a collection of nodes. Nodes represent value and nodes are connected by edges. A tree has the following properties: The tree has one node called root. The tree originates from this, and hence it does not have any parent.



Project code:

```
#include <iostream>
#include <vector>
#include <string>
#include <stack>
#include <queue>
#include <algorithm>

struct Book {
    std::string title;
    std::string author;
    std::string ISBN;
    bool isAvailable;
};

struct Patron {
    std::string name;
    int ID;
    int booksBorrowed;
};

class Library {
private:
    std::vector<Book> books;
    std::vector<Patron> patrons;
    std::stack<Book> borrowedBooks;
    std::queue<Book> bookRequests;

public:
```

```
    // Function declarations
    void addBook(const std::string& title, const std::string& author, const std::string& ISBN);
    void removeBook(const std::string& ISBN);
    void displayBooks();
    void issueBook(int patronID, const std::string& ISBN);
    void returnBook(int patronID, const std::string& ISBN);
    void requestBook(int patronID, const std::string& ISBN);
    void processBookRequests();

    // Example constructor
    Library() {
        // Initialize your library with some default books and patrons
        addBook("The Catcher in the Rye", "J.D. Salinger", "978-0-316-76948-0");
        addBook("To Kill a Mockingbird", "Harper Lee", "978-0-06-112008-4");
        addBook("1984", "George Orwell", "978-0-452-28423-4");

        addPatron("John Doe", 1);
        addPatron("Jane Smith", 2);
    }

private:
    void addPatron(const std::string& name, int ID) {
        patrons.push_back({name, ID, 0});
    }
};

// Function definitions
void Library::addBook(const std::string& title, const std::string& author, const std::string&
ISBN) {
    books.push_back({title, author, ISBN, true});
}
```

```
void Library::removeBook(const std::string& ISBN) {
    auto it = std::find_if(books.begin(), books.end(), [&ISBN](const Book& book) {
        return book.ISBN == ISBN;
    });

    if (it != books.end()) {
        books.erase(it);
        std::cout << "Book with ISBN " << ISBN << " removed from the library.\n";
    } else {
        std::cout << "Book with ISBN " << ISBN << " not found in the library.\n";
    }
}

void Library::displayBooks() {
    std::cout << "\n*** Library Catalog ***\n";
    std::cout << "----------------------------------------\n";
    std::cout << "| ISBN           | Title                |\n";
    std::cout << "----------------------------------------\n";
    for (const auto& book : books) {
        std::cout << "| " << book.ISBN << " | " << book.title;
        // Add spaces to align book titles
        for (size_t i = 0; i < 24 - book.title.length(); ++i) {
            std::cout << " ";
        }
        std::cout << "|\n";
    }
    std::cout << "----------------------------------------\n";
}
```

```
void Library::issueBook(int patronID, const std::string& ISBN) {
    auto it = std::find_if(books.begin(), books.end(), [&ISBN](const Book& book) {
        return book.ISBN == ISBN && book.isAvailable;
    });

    if (it != books.end()) {
        borrowedBooks.push(*it);
        it->isAvailable = false;
        for (auto& patron : patrons) {
            if (patron.ID == patronID) {
                patron.booksBorrowed++;
                break;
            }
        }
        std::cout << "Book with ISBN " << ISBN << " issued to patron ID " << patronID <<
".\n";
    } else {
        std::cout << "Book with ISBN " << ISBN << " not available for issue.\n";
    }
}

void Library::returnBook(int patronID, const std::string& ISBN) {
    if (!borrowedBooks.empty() && borrowedBooks.top().ISBN == ISBN) {
        borrowedBooks.top().isAvailable = true;
        borrowedBooks.pop();
        for (auto& patron : patrons) {
            if (patron.ID == patronID && patron.booksBorrowed > 0) {
                patron.booksBorrowed--;
                std::cout << "Book with ISBN " << ISBN << " returned by patron ID " <<
patronID << ".\n";
                return;
```

```
            }
        }
        std::cout << "Error: Patron ID " << patronID << " has not borrowed this book.\n";
    } else {
        std::cout << "Error: Book with ISBN " << ISBN << " is not at the top of the borrowed books stack.\n";
    }
}

void Library::requestBook(int patronID, const std::string& ISBN) {
    auto it = std::find_if(books.begin(), books.end(), [&ISBN](const Book& book) {
        return book.ISBN == ISBN && !book.isAvailable;
    });

    if (it != books.end()) {
        bookRequests.push(*it);
        std::cout << "Book with ISBN " << ISBN << " requested by patron ID " << patronID << ".\n";
    } else {
        std::cout << "Book with ISBN " << ISBN << " is either available or not found in the library.\n";
    }
}

void Library::processBookRequests() {
    while (!bookRequests.empty() && borrowedBooks.size() < books.size()) {
        Book requestedBook = bookRequests.front();
        bookRequests.pop();
        requestedBook.isAvailable = true;
        borrowedBooks.push(requestedBook);
        std::cout << "Book with ISBN " << requestedBook.ISBN << " is now available. Fulfilling request.\n";
```

```
    }
}

int main() {
    Library library;

    int choice;
    do {
        std::cout << "\nLibrary Management System\n";
        std::cout << "1. Display Books\n";
        std::cout << "2. Add Book\n";
        std::cout << "3. Remove Book\n";
        std::cout << "4. Issue Book\n";
        std::cout << "5. Return Book\n";
        std::cout << "6. Request Book\n";
        std::cout << "0. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                library.displayBooks();
                break;
            case 2: {
                std::string title, author, ISBN;
                std::cout << "Enter book details:\n";
                std::cout << "Title: ";
                std::cin.ignore();
                std::getline(std::cin, title);
                std::cout << "Author: ";
```

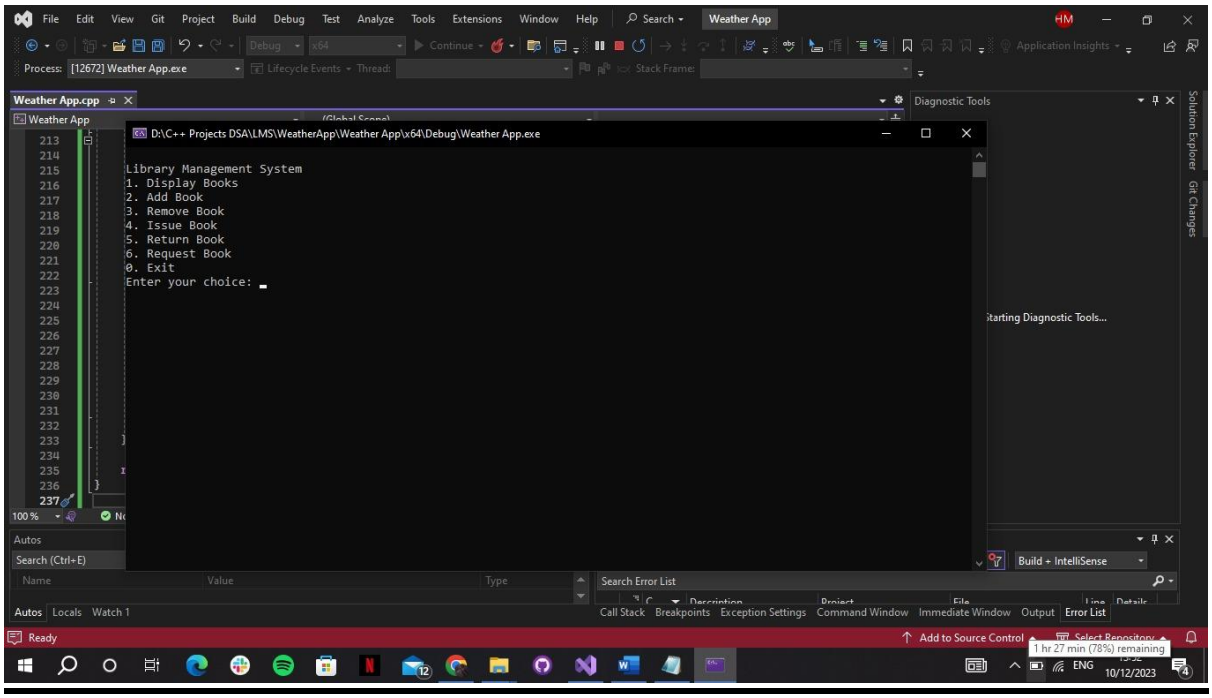
```
        std::getline(std::cin, author);
        std::cout << "ISBN: ";
        std::getline(std::cin, ISBN);
        library.addBook(title, author, ISBN);
        break;
    }
    case 3: {
        std::string ISBN;
        std::cout << "Enter ISBN of the book to remove: ";
        std::cin >> ISBN;
        library.removeBook(ISBN);
        break;
    }
    case 4: {
        int patronID;
        std::string ISBN;
        std::cout << "Enter patron ID: ";
        std::cin >> patronID;
        std::cout << "Enter ISBN of the book to issue: ";
        std::cin >> ISBN;
        library.issueBook(patronID, ISBN);
        break;
    }
    case 5: {
        int patronID;
        std::string ISBN;
        std::cout << "Enter patron ID: ";
        std::cin >> patronID;
        std::cout << "Enter ISBN of the book to return: ";
        std::cin >> ISBN;
```

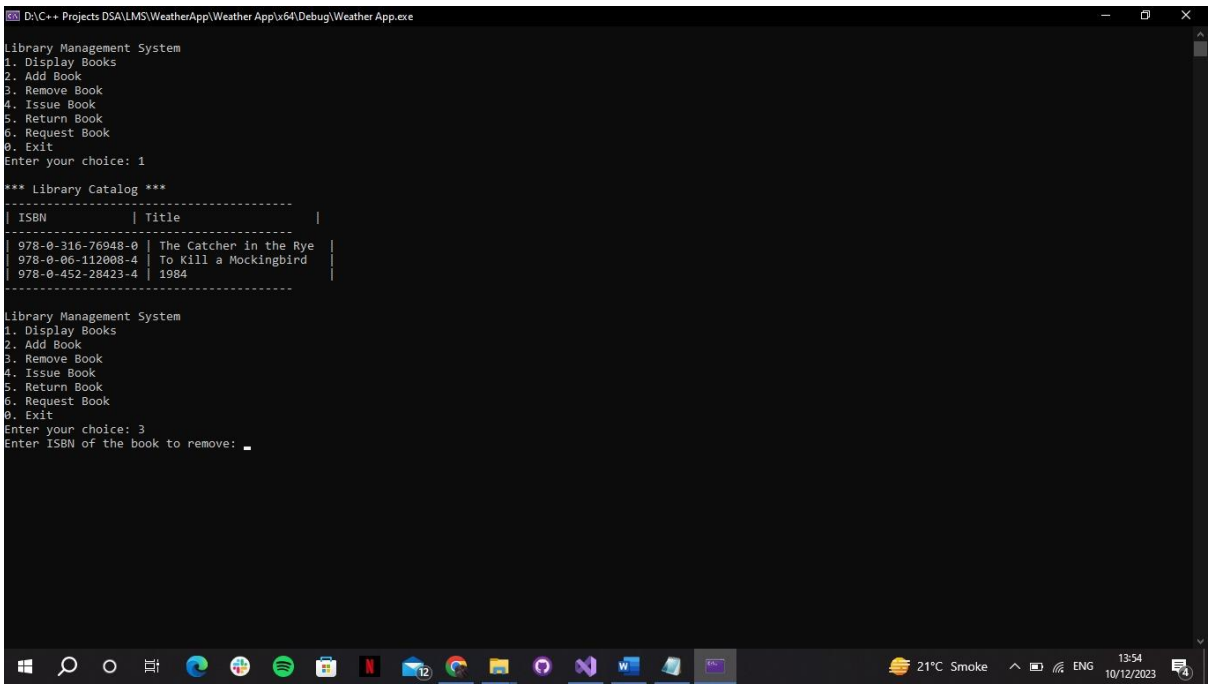
```
            library.returnBook(patronID, ISBN);
            break;
        }
        case 6: {
            int patronID;
            std::string ISBN;
            std::cout << "Enter patron ID: ";
            std::cin >> patronID;
            std::cout << "Enter ISBN of the requested book: ";
            std::cin >> ISBN;
            library.requestBook(patronID, ISBN);
            break;
        }
        case 7:
            library.processBookRequests();
            break;
        case 0:
            std::cout << "Program tw War Gaya.\n";
            break;
        default:
            std::cout << "Invalid choice. Please enter a valid option.\n";
      }

   } while (choice != 0);

   return 0;
}
```

Output:





Algorithms use this project are:

1. Add Book (addBook function):

Algorithm:

Takes parameters for book details (title, author, ISBN).

Creates a new Book object with the provided details and sets it as available.

Adds the new book to the books vector.

2. Remove Book (removeBook function):

Algorithm:

Searches for a book with the given ISBN in the books vector using std::find_if.

If the book is found, it is removed from the vector.

3. Display Books (displayBooks function):

Algorithm:

Iterates through the books vector.

Prints the ISBN and title of each book in a formatted manner.

4. Issue Book (issueBook function):

Algorithm:

Searches for a book with the given ISBN in the books vector that is available.

If the book is found:

Pushes the book onto the borrowedBooks stack.

Marks the book as unavailable.

Updates the number of books borrowed by the patron.

5. Return Book (returnBook function):

Algorithm:

Checks if the borrowedBooks stack is not empty.

Compares the ISBN of the book at the top of the stack with the provided ISBN.

If they match, marks the book as available, pops it from the stack, and updates the patron's book count.

6. Request Book (requestBook function):

Algorithm:

Searches for a book with the given ISBN in the books vector that is not available.

If the book is found, adds it to the bookRequests queue.

7. Process Book Requests (processBookRequests function):

Algorithm:

Iterates through the bookRequests queue.

Marks each requested book as available and adds it to the borrowedBooks stack.

Prints a message indicating that the book is now available.

8. Main Menu (main function):

Algorithm:

Displays a menu with various options.

Takes user input to select an option.

Executes the corresponding function based on the user's choice.

Continues until the user chooses to exit.

Stack use in project:

- Borrowing a Book (issueBook function):

```
borrowedBooks.push(*it);
                        // Push the borrowed book onto the stack
it->isAvailable = false;
                        // Mark the book as unavailable
```

- Returning a Book (returnBook function):

```
borrowedBooks.top().isAvailable = true;
                                    // Mark the book as available
borrowedBooks.pop();                // Pop the book from the stack
```

Heap use in project:

- Structs and Classes:

- Both Book and Patron structures are created on the stack when added to the vectors in the Library class. There are no instances of these structures created using new or malloc, indicating that there is no dynamic memory allocation for these structures.
- Vectors, Stack, and Queue:

- The vectors (books and patrons), stack (borrowedBooks), and queue (bookRequests) are using automatic storage duration, and their memory is managed by the system.
- Library Initialization:

- The default books and patrons are added to the library in the constructor using the addBook and addPatron functions. These objects are created on the stack

Queue use in project:

- **Book Requests Queue (bookRequests):**

```
std::queue<Book> bookRequests;
```

- **Request Book Function (requestBook):**

```
void Library::requestBook(int patronID, const std::string& ISBN) {
    auto it = std::find_if(books.begin(), books.end(), [&ISBN](const Book& book) {
        return book.ISBN == ISBN && !book.isAvailable;
    });

    if (it != books.end()) {
        bookRequests.push(*it);
        std::cout << "Book with ISBN " << ISBN << " requested by patron ID " << patronID << ".\n";
    } else {
        std::cout << "Book with ISBN " << ISBN << " is either available or not found in the library.\n";
    }
```

```
}
```

- **Process Book Requests Function (processBookRequests):**

```
void Library::processBookRequests() {
    while (!bookRequests.empty() && borrowedBooks.size() < books.size()) {
        Book requestedBook = bookRequests.front();
        bookRequests.pop();
        requestedBook.isAvailable = true;
        borrowedBooks.push(requestedBook);
        std::cout << "Book with ISBN " << requestedBook.ISBN << " is now available. Fulfilling request.\n";
    }
}
```

Homework

Question: Design a C++ program to implement a Date class that can perform basic operations such as setting and getting the date and checking for leap years.

```
#include<iostream>

#include<string.h>

using namespace std;

int limit[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};

string
name[13]={"Null","January","February","March","April","May","June","July","August","Se
ptember","October","November","December"};

class date

{

        private:

                int day;

                int month;

                int year;

        public:

                date();

                date(int ,int ,int );

                void setDate(int ,int ,int);

                void setDay(int);

                void setMonth(int);

                void setYear(int);

                int getDay();

                int getMonth();

                int getYear();

                void IsLeap(int);

                void IsLeap();

                void printDate();

                void printAge();

                void operator -=(date i)

      {
```

```
        if(day>i.day)
        day=day-i.day;
        else
        day=i.day-day;
                        if(month>i.month)
                        month=month-i.month;
                        else
                        month=i.month-month;
                        year=i.year-year;
                }
};
date::date()
{
        day=0;
        month=0;
        year=0;
}
date::date(int d,int m,int y)
{
        this->setDate(d,m,y);
}
void date::setDate(int d,int m,int y)
{
        this->setDay(d);
        this->setMonth(m);
        this->setYear(y);
}
void date::setDay(int d){
        day=((d>=1&&d<=limit[this->getMonth()])?d:1);
        if(this->getMonth()==2&&this->getYear()%4==0);
```

```
	day=((d>=1&&d<=29)?d:1);
}
void date::setMonth(int m){
		month=((m>=1&&m<=12)?m:1);
}
void date::setYear(int y){
		year=((y>=1990&&y<=2023)?y:1);
}
int date::getDay()
{
	return day;
}
int date::getMonth()
{
	return month;
}
int date::getYear()
{
	return year;
}
void date::IsLeap()
{
	if(year%4==0)
	cout<<"\tLeap Year"<<endl;
	else
	cout<<"\tNO Leap Year"<<endl;
}
void date::IsLeap(int y)
{
	if(y%4==0)
```

```
        cout<<"\tLeap Year"<<endl;
        else
        cout<<"\tNO Leap Year"<<endl;
}
void date::printDate()
{
        cout<<"Date-of-Birth : "<<this->getDay()<<"-"<<name[this->month]<<"-"<<this-
>getYear();
}
void date::printAge()
{
        cout<<"Current   Age : "<<this->getYear()<<"-"<<this->month<<"-"<<this-
>getDay();
}

int main()
{
        int a,b,c,d,e,f,y;
        cout<<"Enter Date of Birth:"<<endl;
        cin>>a>>b>>c;
        date i;
        i.setDate(a,b,c);
        cout<<endl;
        cout<<"Enter current date: "<<endl;
        cin>>d>>e>>f;
        date j(d,e,f);
        i.printDate();
        i.IsLeap();
        i-=j;
        i.printAge();
        cout<<"Enter year that u want to check is leap or not: ";
```

```
        cin>>y;
        i.IsLeap(y);
}
```

Question: 1. Implement a singly linked list in C++ with basic operations such as insertion, deletion, and traversal. Design a program that prompts the user to perform these operations interactively. 2. Implement a singly circular linked list in C++ with basic operations such as insertion, deletion, and traversal. Design a program that prompts the user to perform these operations interactively. 3. Implement a doubly linked list in C++ with basic operations such as insertion, deletion, and traversal. Design a program that prompts the user to perform these operations interactively. 4. Implement a doubly circular linked list in C++ with basic operations such as insertion, deletion, and traversal. Design a program that prompts the user to perform these operations interactively.

1-

```
#include<iostream>
using namespace std;
class node
{
        private:
                int data;
                node *next;
        public:
                node(int);
                void setData(int);
                int getData();
                void setNext(node*);
                node* getnext();
                void showData();
};
node::node(int d)
{
        setData(d);
        next=NULL;
}
```

```
void node::setData(int d)
{
	data=d;
}
void node::showData()
{
	cout<<" "<<data;
}
int node::getData()
{
	return data;
}
void node::setNext(node *ptr)
{
	next=ptr;
}
node *node::getnext()
{
	return next;
}

class list
{
	private:
		int size;
		node *head;
		node *current;
	public:
		list();
		void addTail(int);
```

```
                void addHead(int);
                void showList();
                int TotalElement();
                int removeTail();
                int removeHead();
                void forward();
                void back();
};
list::list()
{
        size=0;
        head=NULL;
        current=NULL;
};
void list::addHead(int d)
{
        node *ptr=new node(d);
        ptr->setNext(head);
        current->setNext(ptr);
        head=ptr;
        //current=ptr;
        size++;
}
void list::addTail(int d)
{
        node *ptr=new node(d);
        if(size==0)
        {
                head=ptr;
                current=ptr;
```

```
                current->setNext(head);
        }
        else
        {
                ptr->setNext(current->getnext());
                current->setNext(ptr);
                current=ptr;
        }
        size++;
}
void list::showList()
{
        if(size==0)
        {
                cout<<"List is Empty"<<endl;
        }
        else
        {
        node *ptr=head;
        do
        {
                ptr->showData();
                ptr=ptr->getnext();
        }while(ptr!=head);
        }
}
int list::TotalElement()
{
        return size;
}
```

```
int list::removeTail()
{
	if(current==head)
	{
		head=head->getnext();
	}
	node *ptr=head;
	node *tem=current;
	int x=tem->getData();
	do
	{
		if(ptr->getnext()==current)
		{
			break;
		}
		ptr=ptr->getnext();
	}while(ptr!=head);
	ptr->setNext(current->getnext());
	current=current->getnext();
	delete tem;
	size--;
	return x;
}
void list::forward()
{
	if(current!=NULL)
	current=current->getnext();
}
void list::back()
{
```

```
        node *ptr=head;
        do
        {
                if(ptr->getnext()==current)
                {
                        break;
                }
                ptr=ptr->getnext();
        }while(ptr!=head);
}
int main()
{
        cout<<"Hello World"<<endl;
        list singly;
        singly.showList();
        cout<<"\nTotal Elements = "<<singly.TotalElement()<<endl;
        singly.addTail(5);
        cout<<"\nTotal Elements = "<<singly.TotalElement()<<endl;
        cout<<"List: ";
        singly.showList();
        singly.addTail(10);
        singly.addTail(15);
        cout<<"\nTotal Elements = "<<singly.TotalElement()<<endl;
        cout<<"List: ";
        singly.showList();
        cout<<"\ndelete Elemented = "<<singly.removeTail()<<endl;
        cout<<"Total Elements = "<<singly.TotalElement()<<endl;
        cout<<"List: ";
        singly.showList();
        cout<<"\ndelete Elemented = "<<singly.removeTail()<<endl;
```

```
	cout<<"Total Elements = "<<singly.TotalElement()<<endl;
	cout<<"List: ";
	singly.showList();
	singly.addHead(100);
	singly.addHead(200);
	cout<<"\nTotal Elements = "<<singly.TotalElement()<<endl;
	cout<<"List: ";
	singly.showList();
}
```

2-

```
#include<iostream>
using namespace std;
class node
{
	private:
			int data;
			node *next;
	public:
			node(int );
			void setData(int);
			int getData();
			void setNext(node *);
			node* getNext();
			void showData();
};
node::node(int d)
{
		setData(d);
		next = NULL;
}
```

```
void  node::setData(int d)
{
		data = d;
}
int node::getData()
{
		return data;
}
void node::setNext(node* ptr)
{
		next = ptr;
}
node* node::getNext()
{
	return next;
}
void node::showData()
{
		cout<<data<<"    ";
}
class list
{
		private:
				node* head;
				node* current;
				int size;
		public:
				list();
				void add(int );
				void showList();
```

```
                    void remove();
                    void inserthead(int);
                    void forward();
                    void back();
                    void start();
};
list::list()
{
        head = NULL;
        current = NULL;
        size =0;
}
void list::add(int d)
{
        node *ptr = new node(d);
        if(size==0)
        {
                head = ptr;
                current = ptr;
        }
        else
        {
                ptr->setNext(current->getNext());
                current->setNext(ptr);
                current=ptr;
        }
        size++;
}
void list::inserthead(int d)
{
```

```
        node *ptr = new node(d);
        ptr->setNext(head);
        head=ptr;
        current=ptr;
        size++;
}
void list::showList()
{
                node* ptr = head;
                while(ptr!=NULL)
                {
                                ptr->showData();
                                ptr = ptr->getNext();
                }
}
void list::remove()
{
        if(size==0){
        cout<<"List is Empty";
        }
        if(head==current)
        {
                head=head->getNext();
        }
        node *temp=current;
        node *ptr=head;
        while(ptr!=NULL)
        {
                if(ptr->getNext()==current)
                {
```

```
                        break;
                }
                ptr=ptr->getNext();
        }
        ptr->setNext(current->getNext());
        current=current->getNext();
        delete temp;
        size--;
}
/****/

void list::forward()
{
        if(current!=NULL)
        current=current->getNext();
}
void list::back()
{
        node *ptr=head;
        do
        {
                if(ptr->getNext()==current)
                {
                        break;
                }
                ptr=ptr->getNext();
        }while(ptr!=NULL);
}
void list::start()
{
```

```
        current=head;
}



int main()
{
                cout<<"hello world"<<endl;
                list lst;
                lst.add(20);
                lst.add(60);
                lst.add(80);
                lst.add(90);
                lst.add(70);
                lst.add(40);
                lst.showList();
                cout<<endl;
//              lst.remove();
//              lst.showList();
//              cout<<endl;
//              lst.remove();
//              lst.showList();
                cout<<endl;
                lst.add(10);
                lst.showList();
                cout<<endl;
//              lst.remove();
//              lst.showList();
                list lst1;
                lst1.add(20);
                lst1.add(60);
```

```
        lst1.add(80);
        lst1.add(90);
        lst1.add(70);
        lst1.add(10);
        lst1.showList();
        cout<<endl;
        lst1.inserthead(17);
        lst1.showList();
        cout<<endl;
        lst1.inserthead(27);
        lst1.showList();
        cout<<endl;
        lst1.inserthead(37);
        lst1.showList();
        cout<<endl;
        lst1.remove();
        lst1.showList();
}
```

3-

```
#include<iostream>
using namespace std;
class node
{
    private:
        int data;
        node *next;
        node *prev;
    public:
        node(int);
        void setData(int);
```

```
                int getData();
                void setNext(node*);
                node* getnext();
                void setPrev(node *);
                node* getPrev();
                void showData();
};
node::node(int d)
{
        setData(d);
        next=NULL;
        prev=NULL;
}
void node::setData(int d)
{
        data=d;
}
void node::showData()
{
        cout<<" "<<data;
}
int node::getData()
{
        return data;
}
void node::setNext(node *ptr)
{
        next=ptr;
}
node *node::getnext()
```

```
{
        return next;
}
void node::setPrev(node *ptr)
{
                prev = ptr;
}
node* node::getPrev()
{
                return prev;
}
class list
{
        private:
                int size;
                node *head;
                node *current;
        public:
                list();
                void addTail(int);
                void add(int);
                void addHead(int);
                void showList();
                int TotalElement();
                int removeTail();
                int removeHead();
                void forward();
                void back();
                void start();
                void tail();
```

```
};
list::list()
{
	size=0;
	head=NULL;
	current=NULL;
};
void list::addHead(int d)
{
	node *ptr=new node(d);
	ptr->setNext(head);
	ptr->setPrev(head->getPrev());
	head->getPrev()->setNext(ptr);
	head=ptr;
	current=ptr;
	size++;
}
void list::add(int d)
{
	node *ptr=new node(d);
	if(size==0)
	{
		head=ptr;
		current=ptr;
		current->setNext(head);
		current->setPrev(head);
	}
	else
	{
		ptr->setNext(current->getnext());
```

```
                ptr->setPrev(current);
                current->getnext()->setPrev(ptr);
                current->setNext(ptr);
                current=ptr;
        }
        size++;
}
void list::addTail(int d)
{
        current=head->getPrev();
        node *ptr=new node(d);
                ptr->setNext(current->getnext());
                ptr->setPrev(current);
                current->getnext()->setPrev(ptr);
                current->setNext(ptr);
                current=ptr;
        size++;
}
void list::showList()
{
        if(size==0)
        {
                cout<<"List is Empty"<<endl;
        }
        else
        {
        node *ptr=head;
        do
        {
                ptr->showData();
```

```
                ptr=ptr->getnext();
        }while(ptr!=head);
        }
}
int list::TotalElement()
{
        return size;
}
int list::removeTail()
{
        if(current==head)
        {
                head=head->getnext();
        }
        node *ptr=head;
        node *tem=current;
        int x=tem->getData();
        do
        {
                if(ptr->getnext()==current)
                {
                        break;
                }
                ptr=ptr->getnext();
        }while(ptr!=head);
        ptr->setNext(current->getnext());
        ptr->getnext()->setPrev(ptr);
        current=current->getnext();
        current->setPrev(ptr);
        delete tem;
```

```
        size--;
        return x;
}
void list::forward()
{
        if(current!=NULL)
        current=current->getnext();
}
void list::back()
{
        node *ptr=head;
        do
        {
                if(ptr->getnext()==current)
                {
                        break;
                }
                ptr=ptr->getnext();
        }while(ptr!=head);
}
void list::start()
{
        current=head;
}
void list::tail()
{
        current=head->getPrev();
}

int main()
```

```
{
	cout<<"Hello World"<<endl;
	list doubly;
	doubly.showList();
	cout<<"\nTotal Elements = "<<doubly.TotalElement()<<endl;
	doubly.add(5);
	cout<<"\nTotal Elements = "<<doubly.TotalElement()<<endl;
	cout<<"List: ";
	doubly.showList();
	doubly.add(10);
	doubly.add(15);
	doubly.add(20);
	doubly.add(25);
	cout<<"\nTotal Elements = "<<doubly.TotalElement()<<endl;
	cout<<"List: ";
	doubly.showList();
	cout<<"\ndelete Elemented = "<<doubly.removeTail()<<endl;
	cout<<"Total Elements = "<<doubly.TotalElement()<<endl;
	cout<<"List: ";
	doubly.showList();
	doubly.tail();
	cout<<"\ndelete Elemented = "<<doubly.removeTail()<<endl;
	cout<<"Total Elements = "<<doubly.TotalElement()<<endl;
	cout<<"List: ";
	doubly.showList();
	doubly.addHead(1000);
	cout<<"\nTotal Elements = "<<doubly.TotalElement()<<endl;
	cout<<"List: ";
	doubly.showList();
	doubly.addTail(2000);
```

```
        cout<<"\nTotal Elements = "<<doubly.TotalElement()<<endl;
        cout<<"List: ";
        doubly.showList();
}
```

Question: Implement a list in C++ with basic operations such as add, fnd, delete and insert etc. Design a program that prompts the user to perform these operations interactively.

```
#include<iostream>
using namespace std;
class list
{
        private:
                int *ptr;
                int length;
                int size;
                int current;
        public:
                list(int);
                void add(int);
                void printList();
                void next();
                void back();
                void start();
                void tail();
                void copy(list );
                void clearList();
                void findPos(int );
                int getLength();
                int getValAtPos(int );
                ~list();
};
```

```
list::list(int l)
{
		if(l>0)
		{
			length = l+1;
			size = 0;
			current = 0;
			ptr = new int[length];
			for(int i=0;i<=length;i++)
				ptr[i]=-1;
		}
}
void list::add(int val)
{
		if(current == length)
		{
			cout<<"list is full"<<endl;

		}
		else
		{

		ptr[++current]=val;
		size++;

		}
}
void list::printList()
{
		for(int i=1;i<=size;i++)
```

```
                cout<<ptr[i]<<"   ";
        cout<<endl;
        if(size==0)
                cout<<"List is empty : "<<endl;
}
void list::next()
{
        if(current == length)
                cout<<"current is alread at end, can't move further"<<endl;
        else
                current++;
}
void list::back()
{
        if(current ==1 )
                cout<<"current is already at start, cant move back more "<<endl;
        else
                current--;
}
void list::start()
{
        current = 1;
}
void list::tail()
{
        current = size;
}
void list::copy(list l)
{
        length = l.length;
```

```
		size =  l.size;
		current = l.current;
		ptr = new int [length];
		for(int i = 1;i<=length;i++)
			ptr[i]=l.ptr[i];
}
void list::clearList()
{
		size=0;
		current = 0;
		for(int i=1;i<=length;i++)
			ptr[i]=-1;
}
void list::findPos(int val)

{
		for(int i=1;i<=size;i++)
		{
			if(ptr[i]== val)
			{
				cout<<"value found at index number :
"<<i<<endl;
				return;
			}
		}
		cout<<"value dose not exist : "<<endl;
}
int list::getLength()
{
		return length;
}
```

```
int list::getValAtPos(int pos)
{
		if(pos<=size&&pos>=1)
			return ptr[pos];
		else
			{
				cout<<"invalid position"<<endl;
				return -1;
			}

}
list::~list()
{
		delete []ptr;
}
int main()
{
		cout<<"hello world"<<endl;
		list l1(10);
		l1.add(2);
		l1.add(6);
		l1.add(8);
		l1.add(7);
		l1.add(1);
		l1.printList();
		list l2(11);
		l2.copy(l1);
		l2.add(8);
		l2.printList();
		l2.clearList();
```

```
        l2.printList();
        l1.findPos(6);

}
```

Question: Design a C++ program to implement a stack using an array. Include functionalities to push, pop, and check the top element of the stack. Ensure that the stack can handle a specifed maximum number of elements to prevent overfoo. Implement error handling for stack underfoo conditions. Design a C++ program to implement a stack using a linked list. Defne a ode structure for the linked list and include functionalities to push, pop, and check the top element of the stack. Implement error handling for stack underfoo conditions.

```
#include<iostream>
using namespace std;
/*********************************
    stack with array
*********************************/
class Stack
{
    private:
        int *arr;
        int top;
        int length;
    public:
        Stack(int);
        void push(int);
        int pop();
        int getTop();
        bool IsFull();
        bool IsEmpty();
};
Stack::Stack(int l)
{
```

```
        length=l;
        arr=new int[length];
        top=-1;
}
void Stack::push(int d)
{
        if(top==length-1)
        {
                cout<<"Stack is full"<<endl;
        }
        else
        {
        arr[++top]=d;
        }
}
int Stack::pop()
{
        if(top==-1)
        {
                return -1;
        }
        else
        {
        return arr[top--];
        }
}
int Stack::getTop()
{
        if(top==-1)
        return -1;
```

```
        else
        return arr[top];
}
bool Stack::IsEmpty()
{
        if(top==-1)
        {
                return true;
        }
        else
        return false;
}
bool Stack::IsFull()
{
        if(top==length-1)
        {
                return true;
        }
        else
        return false;
}

/*********************************
        stack with Linked list
*********************************/
class node
{
        private:
                int data;
                node *next;
```

```
        public:
                node(int);
                void setData(int);
                int getData();
                void showData();
                void setNext(node*);
                node* getNext();
};
node::node(int d)
{
        setData(d);
        next=NULL;
}
void node::setData(int d)
{
        data=d;
}
int node::getData()
{
        return data;
}
void node::showData()
{
        cout<<data<<"  ";
}
void node::setNext(node *ptr)
{
        next=ptr;
}
node* node::getNext()
```

```
{
        return next;
}
class stack
{
        private:
                node *top;
        public:
                stack();
                void push(int);
                int pop();
                int getTop();
                bool IsEmpty();
};
stack::stack()
{
        top=NULL;
}
void stack::push(int d)
{
        node *ptr=new node(d);
        ptr->setNext(top);
        top=ptr;
}
int stack::pop()
{
        int x=top->getData();
        node *temp=top;
        top=top->getNext();
        delete temp;
```

```
	return x;
}
int stack::getTop()
{
	return top->getData();
}
bool stack::IsEmpty()
{
	if(top==NULL)
	{
		return true;
	}
	else
	return false;
}


int main()
{
	cout<<"Hello World! "<<endl;

cout<<"**********************************\n"<<endl;
	cout<<"\tstack with array\n"<<endl;
cout<<"**********************************\n"<<endl;
	Stack s(5);
//*****check stack is full or not and Empty or not*****
	cout<<"Is Empty= "<<s.IsEmpty()<<endl;
	cout<<"Is FULL = "<<s.IsFull()<<endl;
	//*****Push data into stack*****
	for(int i=1;i<=5;i++)
```

```
	{
		int a;
		cout<<"Enter number: ";
		cin>>a;
		s.push(a);
	}
	s.push(20);
	//*****After Pushing*****
//*****check stack is full or not and Empty or not*****
	cout<<"Is Empty= "<<s.IsEmpty()<<endl;
	cout<<"Is FULL = "<<s.IsFull()<<endl;
	//*****Get the value At the top of stack(only get vale at the top of stack not remove)*****
	cout<<"Top = "<<s.getTop()<<endl;
	//*****Pop data From stack*****
	for(int i=1;i<=5;i++)
	{
		cout<<"Pop = "<<s.pop()<<endl;
	}
	//*****After Poping*****
//*****check stack is full or not and Empty or not*****
//*****Answer will be empty(0) and full (0)*****
	cout<<"Is Empty= "<<s.IsEmpty()<<endl;
	cout<<"Is FULL = "<<s.IsFull()<<endl;

cout<<"**********************************\n"<<endl;
	cout<<"\tstack with Linked list\n"<<endl;
cout<<"**********************************\n"<<endl;
stack s1;
	cout<<"Is Empty = "<<s1.IsEmpty()<<endl;
	s1.push(1);
```

```
    cout<<"Top = "<<s1.getTop()<<endl;
    s1.push(2);
    cout<<"Top = "<<s1.getTop()<<endl;
    s1.push(3);
    cout<<"Is Empty = "<<s1.IsEmpty()<<endl;
    cout<<"Top = "<<s1.getTop()<<endl;
    cout<<"Pop = "<<s1.pop()<<endl;
    cout<<"Top = "<<s1.getTop()<<endl;
    cout<<"Pop = "<<s1.pop()<<endl;
    cout<<"Is Empty = "<<s1.IsEmpty()<<endl;
    cout<<"Top = "<<s1.getTop()<<endl;
    cout<<"Pop = "<<s1.pop()<<endl;
    cout<<"Is Empty = "<<s1.IsEmpty()<<endl;

}
```

Question: Design a C++ program to implement a queue using an array. Include functionalities to enqueue, dequeue, and check the front element of the queue. Ensure that the queue can handle a specifed maaimum number of elements to prevent overfoo. Implement error handling for queue underfoo conditions. Design a C++ program to implement a queue using a linked list. Defne a Node structure for the linked list and include functionalities to enqueue, dequeue, and check the front element of the queue. Implement error handling for queue underfoo conditions.

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

/************************************
                Queue with Array
************************************/
template <class t>
```

```
class queue
{
	private:
		t *data;
		int size;
		int front;
		int rear;
		int ele;
	public:
		queue(int);
		void enqueue(t);
		t dequeue();
		t getFront();
		int Elements();
		bool Isempty();
		bool IsFull();
};
template <class t>
queue<t>::queue(int s)
{
	size=s;
	front=1;
	rear=0;
	data=new t[size];
}
template <class t>
void queue<t>::enqueue(t x)
{
	rear=(rear+1)%size;
	data[rear]=x;
```

```
        ele++;
}
template <class t>
t queue<t>::dequeue()
{
        t x=data[front];
        front=(front+1)%size;
   ele--;
        return x;
}
template <class t>
t queue<t>::getFront()
{
        return data[front];
}
template <class t>
int queue<t>::Elements()
{
        return ele;
}
template <class t>
bool queue<t>::Isempty()
{
        if(front==0)
        return true;
        else
        return false;
}
template <class t>
bool queue<t>::IsFull()
```

```
{
        if(rear==size)
        return true;
        else
        return false;
}

/************************************
                Queue with Linked List
************************************/
template <class t>
class node
{
        private:
                t data;
                node<t> *next;
        public:
                node(t);
                void setData(t);
                t getData();
                void showData();
                void setNext(node<t>*);
                node<t>* getNext();
};
template <class t>
node<t>::node(t d)
{
        setData(d);
        next=NULL;
}
```

```
template <class t>
void node<t>::setData(t d)
{
	data=d;
}
template <class t>
t node<t>::getData()
{
	return data;
}
template <class t>
void node<t>::showData()
{
	cout<<data<<"  ";
}
template <class t>
void node<t>::setNext(node<t> *ptr)
{
	next=ptr;
}
template <class t>
node<t>* node<t>::getNext()
{
	return next;
}
template<class T>
class Queue
{
	private:
		node<T> *front;
```

```
        node<T> *rare;
    public:
        Queue();
        int enqueuq(T);
        T dequeue();
        T frontE();
        bool IsEmpty();
};
template<class T>
Queue<T>::Queue()
{
    front=NULL;
    rare=NULL;
}
template<class T>
int Queue<T>::enqueuq(T d)
{
    node<T> *ptr=new node<T>(d);
    if(front==NULL)
    {
        front=ptr;
        rare=ptr;
    }
    else
    //ptr->setNext(NULL);
    rare->setNext(ptr);
    rare=ptr;
    return 1;
}
template<class T>
```

```
T Queue<T>::dequeue()
{
	T x=front->getData();
	node<T>* ptr=front;
	front=front->getNext();
	delete ptr;
	return x;
}
template<class T>
bool Queue<T>::IsEmpty()
{
	return (front==NULL);

}
template<class T>
T Queue<T>::frontE()
{
	return front->getData();
}
int main()
{
	cout<<"Hello world!"<<endl;

cout<<"************************************\n"<<endl;
	cout<<"\tQueue with Array \n"<<endl;
cout<<"************************************\n"<<endl;
	queue<int> i(5);
	i.enqueue(1);
	i.enqueue(2);
	i.enqueue(3);
```

```
        i.enqueue(4);
        i.enqueue(5);
        cout<<"Front   Element = "<<i.getFront()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Is  Empty Queue = "<<i.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i.IsFull()<<endl;
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        i.enqueue(10);
        i.enqueue(20);
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        cout<<"Front   Element = "<<i.getFront()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        cout<<"Front   Element = "<<i.getFront()<<endl;
        cout<<"Is  Empty Queue = "<<i.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i.IsFull()<<endl;


cout<<"************************************\n"<<endl;
        cout<<"\tQueue with Linked List\n"<<endl;
cout<<"************************************\n"<<endl;

        Queue<int> q;
        cout<<"Empty = "<<q.IsEmpty()<<endl;
        cout<<"Enque data = "<<q.enqueuq(1)<<endl;
```

```
        cout<<"Empty = "<<q.IsEmpty();
        cout<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Enqueuue data = "<<q.enqueuq(2)<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Empty = "<<q.IsEmpty();
        cout<<endl;
        cout<<"Enque data = "<<q.enqueuq(10)<<endl;
        cout<<"Enque data = "<<q.enqueuq(100)<<endl;
        cout<<"Enque data = "<<q.enqueuq(1000)<<endl;
        cout<<"Enque data = "<<q.enqueuq(10000)<<endl;
        cout<<"Enque data = "<<q.enqueuq(100000)<<endl;
        cout<<"Enque data = "<<q.enqueuq(1000000)<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Front Element = "<<q.frontE()<<endl;
        cout<<"Dequeue = "<<q.dequeue()<<endl;
        cout<<"Empty = "<<q.IsEmpty();
    cout<<"\n\n************************\n************************\n\n"<<endl;
        Queue<string> q1;
```

```
cout<<"Empty = "<<q1.IsEmpty()<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21218")<<endl;
cout<<"Empty = "<<q1.IsEmpty();
cout<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Enqueuue data = "<<q1.enqueuq("FARHAN AHMAD")<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Empty = "<<q1.IsEmpty();
cout<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21219")<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21220")<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21221")<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21222")<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21223")<<endl;
cout<<"Enque data = "<<q1.enqueuq("BIT21224")<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Front Element = "<<q1.frontE()<<endl;
cout<<"Dequeue = "<<q1.dequeue()<<endl;
cout<<"Empty = "<<q1.IsEmpty();
```

}

Question: Design a C++ program to implement a binary tree. Defne a Node structure that includes data, a pointer to the left child, and a pointer to the right child. Implement functions to perform the operations. Demonstrate the usage of your program by inserting nodes into the tree and then calling these recursive functions. Demonstrate the usage of your program by inserting nodes into the tree and then calling these non-recursive functions. Demonstrate the usage of your program by inserting nodes into the tree and then performing the level-order traversal.

```
#include<iostream>

using namespace std;

class node

{

        private:

                int data;

                node *left;

                node *right;

        public:

                node(int);

                void setData(int);

                int getData();

                void showData();

                void setLeft(node*);

                node* getLeft();

                void setRight(node*);

                node* getRight();

                bool IsLeaf();

};

node::node(int d)

{

        setData(d);
```

```
        left=NULL;
        right=NULL;
}
void node::setData(int d)
{
        data=d;
}
int node::getData()
{
        return data;
}
void node::showData()
{
        cout<<data<<"  "<<endl;
}
void node::setLeft(node *ptr)
{
        left=ptr;
}
node* node::getLeft()
{
        return left;
}
void node::setRight(node *ptr)
{
        right=ptr;
}
node* node::getRight()
{
        return right;
```

```
}
bool node::IsLeaf()
{
        if(left==NULL&&right==NULL)
        return true;
        else
        return false;
}

void insert(node *root,int val)
{
        node *ptr=new node(val);
        node *p;
        node *q;
        p=root;
        q=root;
        while(q!=NULL&&p->getData()!=val)
        {
                p=q;
                if(val<p->getData())
                q=p->getLeft();
                else
                q=p->getRight();
        }
        if(p->getData()==val)
        cout<<"Attempt to insert duplicate value "<<val<<endl;
        else if(val<p->getData())
        {
                p->setLeft(ptr);
                cout<<val<<endl;
```

```
        }
        else
        {
                p->setRight(ptr);
                cout<<val<<endl;
        }
}
int main()
{
        cout<<"Hello World!"<<endl;
        int arr[]={14,15,4,9,7,18,3,5,16,4,20,17,9,14,5,-1};
        node *root=new node(arr[0]);
        for(int i=1;arr[i]>0;i++)
        {
                insert(root,arr[i]);
        }
}
```

LABS

LAB:1

```
#include <iostream>
using namespace std;
class shape
{
public:
   virtual void displayarea() = 0;
};

class triangle : public shape
{
private:
   int length;
   int base;

public:
   triangle(int, int);
   void setlength(int);
   void setbase(int);
   void displayarea();
   int getlength();
   int getbase();
};

triangle::triangle(int len, int bas)
{
   setlength(len);
```

```
    setbase(bas);
}

void triangle::setlength(int len)
{
    length = len;
}

int triangle::getlength()
{
    return length;
}

void triangle::setbase(int bas)
{
    base = bas;
}

int triangle::getbase()
{
    return base;
}

void triangle::displayarea()
{
    int area = (length * base) / 2;
    std::cout << "Area of triangle is " << area << std::endl;
}

class square : public shape {
```

```
private:
    int length1;
    int length2;

public:
    square(int, int);
    void setlength1(int);
    void setlength2(int);
    int getlength1();
    int getlength2();
    void displayarea();
};

square::square(int l1, int l2) {
    setlength1(l1);
    setlength2(l2);
}

void square::setlength1(int l1) {
    length1 = l1;
}

int square::getlength1() {
    return length1;
}

void square::setlength2(int l2) {
    length2 = l2;
}
```

```
int square::getlength2() {
    return length2;
}

void square::displayarea() {
    int area = length1 * length2;
    std::cout << "Area of square is " << area << std::endl;
}



class rectangle : public shape
{
private:
    int length;
    int width;

public:
    rectangle(int, int);
    void setlength(int);
    void setwidth(int);
    void displayarea();
    int getlength();
    int getwidth();
};

rectangle::rectangle(int len, int wid)
{
    setlength(len);
    setwidth(wid);
}
```

```
void rectangle::setlength(int len)
{
   length = len;
}

int rectangle::getlength()
{
   return length;
}

void rectangle::setwidth(int wid)
{
   width = wid;
}

int rectangle::getwidth()
{
   return width;
}

void rectangle::displayarea()
{
   int area = length * width;
   std::cout << "Area of rectangle is " << area << std::endl;
}

int main()
{
   shape *s = new triangle(5, 5);
```

```
    s->displayarea();
    delete s;

    s = new rectangle(5, 6);
    s->displayarea();
    delete s;

    s = new square(5, 7);
    s->displayarea();
    delete s;

    return 0;
}
```

Lab:2

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

class Stack {
private:
    int size;
    int noOfElements;
    int *data;

public:
 Stack(int);
 ~Stack();
 bool push (int ); // add element to stack.
 int pop ();     // remove the top most element from stack and return true if successful otherwise false..
 int top();
```

```
 int totalElements();
 void display();

 };
Stack::Stack(int stackSize) {
      size = stackSize;
      noOfElements = 0;
      data = new int[size];
   }

   Stack::~Stack() {
      delete[] data;
   }

   bool Stack:: push(int element) {
      if (noOfElements < size) {
         data[noOfElements] = element;
         noOfElements++;
         return true;
      } else {
         return false; // Stack is full
      }
   }

   int Stack:: top() {
      if (noOfElements > 0) {
         return data[noOfElements - 1];
      } else {
         return -1; // Stack is empty
      }
```

```
    }

    int Stack:: pop() {
        if (noOfElements > 0) {
            int poppedElement = data[noOfElements - 1];
            noOfElements--;
            return poppedElement;
        } else {
            return -1; // Stack is empty
        }
    }

    int Stack:: totalElements() {
        return noOfElements;
    }

    void Stack:: display() {
        cout<<"stack:"<<"  ";
        for (int i = 0; i < noOfElements; i++) {
            cout << data[i] << "  , ";
        }
        cout << endl;
    }


int main() {
    Stack s(4);
    cout << s.push(5) << endl;    // 1 (true)
    cout << s.push(10) << endl;   // 1 (true)
    cout << s.push(15) << endl;   // 1 (true)
```

```
    cout << s.push(20) << endl;   // 1 (true)
    cout << s.push(25) << endl;   // 0 (false), stack is full
    cout << s.pop() << endl;      // 20
    cout << s.pop() << endl;      // 15
    cout << s.push(30) << endl;   // 1 (true)
    cout << s.top() << endl;      // 30
    cout << s.totalElements() << endl; // 3
    s.display(); // Displays: 5 10 30

    return 0;
}
```

Lab:3

```
#include <iostream>
using namespace std;
class Stack
{
private:
    int size;
    int noOfElements;
    int *data;

public:
    Stack(int);
    ~Stack(); // delete the allocated memory from heap
    bool push(int);

    int top(); // return the top of the stack without removing it from stack
    int pop(); // return the top of the stack and also remove it from stack.
```

```
    int totalElements(); // return the total number of elements from the stack
    void display();      // display all the elements of the stack from bottom to top
};
Stack::Stack(int stacksize)
{
    size = stacksize;
    noOfElements = 0;     // initialize the count variable as zero initially;
    data = new int[size]; // dynamically allocate space for array in heap ;
}

    bool Stack:: push(int element) {
        if (noOfElements >= size) {
            // If stack is full, allocate new memory with increased size
            int newSize = size + 5;
            int *newData = new int[newSize];

            // Copy the elements from the old array to the new array
            for (int i = 0; i < size; i++) {
                newData[i] = data[i];
            }

            // Deallocate the old memory
            delete[] data;

            // Update the data pointer and size
            data = newData;
            size = newSize;
        }

        data[noOfElements] = element;
```

```
        noOfElements++;
        return true;
    }
int Stack::top()
{
    if (noOfElements > 0)
    {
        return data[noOfElements -1];
    }
    else
    {
        return -1;
    }
}
int Stack::pop()
{
    if (noOfElements > 0)
    {
        int poped = data[noOfElements - 1];
        noOfElements--;
        return poped;
    }
    else
    {
        return -1;
    }
}
int Stack ::totalElements()
{
    return noOfElements;
```

```
}
void Stack::display()
{
   cout << "stack:  ";
   for (int i = 0; i < noOfElements; i++)
   {

      cout << data[i] << "   ";
   }
}
int main()
{

   Stack *s = new Stack(3);
   cout << s->push(5) << endl;
   cout << s->push(10) << endl;
   cout << s->push(15) << endl;
   cout << s->push(20) << endl;
   cout << s->push(25) << endl;
   cout << s->pop() << endl;
   cout << s->pop() << endl;
   cout << s->push(30) << endl;
   cout << s->top() << endl;
   cout << s->totalElements() << endl;
   s->display();
   return 0;
}
```

Lab:4

```
#include <iostream>
```

```
using namespace std;

class node
{
private:
    int value;
    node *next;

public:
    node(int);
    void setvalue(int);
    int getvalue();
    void setnext(node *);
    node *getnext();
    void showdata();
};

node::node(int val)
{
    setvalue(val);
    next = NULL;
}

void node::setvalue(int val)
{
    value = val;
}

int node::getvalue()
{
```

```
    return value;
}

void node::setnext(node *ptr)
{
    next = ptr;
}

node *node::getnext()
{
    return next;
}

void node::showdata()
{
    cout << value << "    ";
}

class list
{
private:
    node *head;

public:
    list();
    void insertAtTail(int);
    int deleteFromTail();
    void insertAtHead(int);
    int deleteFromHead();
    void display();
```

```
    int totalElements();
};

list::list()
{
    head = NULL;
}

void list::insertAtTail(int elem)
{
    node *newNode = new node(elem);
    if (!head)
    {
        head = newNode;
    }
    else
    {
        node *current = head;
        while (current->getnext() != NULL)
        {
            current = current->getnext();
        }
        current->setnext(newNode);
    }
}

void list::insertAtHead(int elem)
{
    node *ptr = new node(elem);
    ptr->setnext(head);
```

```
    head = ptr;
}

int list::deleteFromHead()
{
    if (!head)
    {
        return -1;
    }
    else
    {
        int value = head->getvalue();
        node *ptr = head;
        head = head->getnext();
        delete ptr;
        return value;
    }
}

int list::deleteFromTail()
{
    if (!head)
    {
        return -1;
    }
    else if (head->getnext() == NULL)
    {
        int value = head->getvalue();
        delete head;
        head = NULL;
```

```
        return value;
    }
    else
    {
        node *current = head;
        while (current->getnext()->getnext() != NULL)
        {
            current = current->getnext();
        }
        int value = current->getnext()->getvalue();
        delete current->getnext();
        current->setnext(NULL);
        return value;
    }
}

int list::totalElements()
{
    node *ptr = head;
    int count = 0;
    while (ptr != NULL)
    {
        count++;
        ptr = ptr->getnext();
    }
    return count;
}

void list::display()
{
```

```
    node *current = head;
    while (current != NULL)
    {
        current->showdata();
        current = current->getnext();
    }
}

int main()
{
    list l;
    l.insertAtTail(40);
    l.insertAtTail(50);
    l.insertAtTail(60);
    l.insertAtHead(10);
    l.insertAtHead(20);
    l.insertAtHead(30);

    cout << "Total Elements: " << l.totalElements() << endl;
    l.display();
    cout << l.deleteFromTail() << endl;
    cout << l.deleteFromHead() << endl;
    cout << "Total Elements: " << l.totalElements() << endl;
    l.display();

    return 0;
}
```

Lab:5

```
#include <iostream>
```

```
using namespace std;

class node
{
private:
    int value;
    node *next;

public:
    node(int);
    void setvalue(int);
    int getvalue();
    void setnext(node *);
    node *getnext();
    void showdata();
};

node::node(int val)
{
    setvalue(val);
    next = NULL;
}

void node::setvalue(int val)
{
    value = val;
}

int node::getvalue()
{
```

```
    return value;
}

void node::setnext(node *ptr)
{
    next = ptr;
}

node *node::getnext()
{
    return next;
}

void node::showdata()
{
    cout << value << "    ";
}

class list
{
private:
    node *head;

public:
    list();
    void insertathead(int);
    int delfromhead();
    void insertattail(int);
    int delfromtail();
    int totalelem();
```

```
    void display();
};

list::list()
{
    head = NULL;
}

void list::insertathead(int d)
{
    node *ptr = new node(d);
    if (!head)
    {
        head = ptr;
        ptr->setnext(head);
    }
    else
    {
        node *current = head;
        while (current->getnext() != head)
        {
            current = current->getnext();
        }

        ptr->setnext(head);
        current->setnext(ptr);
        head = ptr;
    }
}
```

```
int list::delfromhead()
{
   if (!head)
   {
      return -1;
   }
   else
   {
      int value = head->getvalue();
      if (head->getnext() == head)
      {
         delete head;
         head = NULL;
      }
      else
      {
         node *current = head;
         while (current->getnext() != head)
         {
            current = current->getnext();
         }
         node *temp = head;
         head = head->getnext();
         current->setnext(head);
         delete temp;
      }
      return value;
   }
}
void list::insertattail(int d)
```

```
{
    node *ptr = new node(d);
    if (!head)
    {
        head = ptr;
        ptr->setnext(head);
    }
    else
    {
        node *current = head;
        while (current->getnext() != head)
        {
            current = current->getnext();
        }
        current->setnext(ptr);
        ptr->setnext(head);
    }
}
int list::delfromtail()
{
    if (!head)
    {
        cout << "list is empty";
    }
    else
    {
        if (head->getnext() == head)
        {
            delete head;
            head = NULL;
```

```
        }
        else
        {
            node *current;
            node *temp;
            while (temp->getnext() != head)
            {
                temp = temp->getnext();
            };
            int value = temp->getvalue();
            while (current->getnext() != temp)
            {
                current = current->getnext();
            }
            delete temp;
            current->setnext(head);
            return value;

        }
    }
}
int list::totalelem()
{
    int count = 1;
    node *current = head;
    while (current->getnext() != head)
    {
        count++;
        current = current->getnext();
    }
```

```
    return count;
}
void list::display()
{
    if (!head)
    {
        cout << "Circular Linked List is empty." << endl;
        return;
    }

    node *current = head;
    cout << "list : ";
    do
    {
        cout <<current->getvalue() << " ";
        current = current->getnext();
    } while (current != head);
    cout << endl;
}

int main()
{
    list l;
 l.insertattail(40);
 l.insertattail(50);
 l.insertattail(60);
 l.insertathead(10);
 l.insertathead(20);
 l.insertathead(30);
 l.display();
```

```
 cout << l.delfromtail() <<endl;
 cout << l.delfromhead() <<endl;
 cout << "Total Elements: " << l.totalelem() << endl;
 l.display();

    return 0;
}
```

Lab:6

```
#include <iostream>

class Stack {
private:
    int* array;
    int topIndex;
    int maxSize;

public:
    // Constructor to initialize the stack with a maximum size
    Stack(int size) : maxSize(size) {
        array = new int[maxSize];
        topIndex = -1;
    }

    // Destructor to free the allocated memory
    ~Stack() {
        delete[] array;
    }

    // Add an element to the top of the stack if it's not full
```

```
bool push(int element) {
    if (topIndex < maxSize - 1) {
        array[++topIndex] = element;
        return true;
    }
    return false; // Stack is full
}

// Return the top element without removing it from the stack
int top() {
    if (topIndex >= 0) {
        return array[topIndex];
    }
    return -1; // Stack is empty
}

// Remove and return the top element from the stack
int pop() {
    if (topIndex >= 0) {
        return array[topIndex--];
    }
    return -1; // Stack is empty
}

// Return the total number of elements in the stack
int totalElements() {
    return topIndex + 1;
}

// Display all elements of the stack from bottom to top
```

```
    void display() {
        for (int i = 0; i <= topIndex; ++i) {
            std::cout << array[i] << " ";
        }
        std::cout << std::endl;
    }
};

int main() {
    Stack* s = new Stack(5);
    std::cout << s->push(5) << std::endl;
    std::cout << s->push(10) << std::endl;
    std::cout << s->push(15) << std::endl;
    std::cout << s->push(20) << std::endl;
    std::cout << s->push(25) << std::endl;
    std::cout << s->pop() << std::endl;
    std::cout << s->pop() << std::endl;
    std::cout << s->push(30) << std::endl;
    std::cout << s->top() << std::endl;
    std::cout << s->totalElements() << std::endl;
    s->display();

    delete s; // Don't forget to free the allocated memory

    return 0;
}
```


Assignment

Question: Design a C++ program to calculate the age of a person given their birthdate. Follow these steps: Allow the user to input their birthdate (day, month, and year). Implement a function that calculates the age based on the current date. Display the calculated age. Ensure that your program handles diferent date formats, accounts for leap years, and validates input for reasonable birthdates. Demonstrate the usage of your program by allowing the user to input their birthdate and then calculating and displaying their age.

```
#include<iostream>

#include<string.h>

using namespace std;

int limit[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};

string
name[13]={"Null","January","February","March","April","May","June","July","August","Se
ptember","October","November","December"};

class date

{

        private:

                int day;

                int month;

                int year;

        public:

                date(int ,int ,int );

                void setDate(int ,int ,int);

                void setDay(int);

                void setMonth(int);

                void setYear(int);

                int getDay();

                int getMonth();

                int getYear();

                void printDate();

                void printAge();

                void operator -=(date i)

    {
```

```
        if(day>i.day)
        day=day-i.day;
        else
        day=i.day-day;
                        if(month>i.month)
                        month=month-i.month;
                        else
                        month=i.month-month;
                        year=i.year-year;
                }
};
date::date(int d,int m,int y)
{
        this->setDate(d,m,y);
}
void date::setDate(int d,int m,int y)
{
        this->setDay(d);
        this->setMonth(m);
        this->setYear(y);
}
void date::setDay(int d){
        day=((d>=1&&d<=limit[this->getMonth()])?d:1);
        if(this->getMonth()==2&&this->getYear()%4==0);
        day=((d>=1&&d<=29)?d:1);
}
void date::setMonth(int m){
                month=((m>=1&&m<=12)?m:1);
}
void date::setYear(int y){
```

```
                year=((y>=1990&&y<=2023)?y:1);
}
int date::getDay()
{
        return day;
}
int date::getMonth()
{
        return month;
}
int date::getYear()
{
        return year;
}

void date::printDate()
{
        cout<<"Date-of-Birth : "<<this->getDay()<<"-"<<name[this->month]<<"-"<<this-
>getYear()<<endl;
}
void date::printAge()
{
        cout<<"Current   Age : "<<this->getYear()<<"-"<<this->month<<"-"<<this-
>getDay()<<endl;
}

int main()
{
        int a,b,c,d,e,f;
        cout<<"Enter Date of Birth:"<<endl;
        cin>>a>>b>>c;
```

```
	cout<<"Enter Current Date :"<<endl;
	cin>>d>>e>>f;
	date dob(a,b,c);
	date cd(d,e,f);
	cout<<endl;
	dob.printDate();
	dob-=cd;
	dob.printAge();
}
```

Question: Design a C++ program to make the result card of student having class date for calculation of age, class data for course code, course name, marks and GPA, then make a class list for record of all students and then make class student for result card.

```
#include<iostream>
#include<string.h>
#include<iomanip>
using namespace std;
int limit[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
string
name[13]={"Null","January","February","March","April","May","June","July","August","Se
ptember","October","November","December"};
class date
{
	private:
		int day;
		int month;
		int year;
	public:
		date(int ,int ,int );
		void setDate(int ,int ,int);
		void setDay(int);
		void setMonth(int);
```

```
                void setYear(int);
                int getDay();
                int getMonth();
                int getYear();
                void printDate();
                void printAge();
                void operator -=(date i)
        {
           if(day>i.day)
           day=day-i.day;
           else
           day=i.day-day;
                        if(month>i.month)
                        month=month-i.month;
                        else
                        month=i.month-month;
                        year=i.year-year;
                }
};
date::date(int d,int m,int y)
{
        this->setDate(d,m,y);
}
void date::setDate(int d,int m,int y)
{
        this->setDay(d);
        this->setMonth(m);
        this->setYear(y);
}
void date::setDay(int d){
```

```
        day=((d>=1&&d<=limit[this->getMonth()])?d:1);
        if(this->getMonth()==2&&this->getYear()%4==0);
        day=((d>=1&&d<=29)?d:1);
}
void date::setMonth(int m){
                month=((m>=1&&m<=12)?m:1);
}
void date::setYear(int y){
                year=((y>=1990&&y<=2023)?y:1);
}
int date::getDay()
{
        return day;
}
int date::getMonth()
{
        return month;
}
int date::getYear()
{
        return year;
}

void date::printDate()
{
        cout<<"Date-of-Birth : "<<this->getDay()<<"-"<<name[this->month]<<"-"<<this-
>getYear()<<endl;
}
void date::printAge()
{
```

```
        cout<<"Current   Age : "<<this->getYear()<<"-"<<this->month<<"-"<<this-
>getDay()<<endl;
}
/*********************************************
                        class cource data
*********************************************/
class data
{
        private:
                string cCode;
                string cName;
                int marks;
                int crdhr;
                float gpa;
        public:
                data();
                data(string,string,int,int);
                setData(string,string,int,int);
                void setCode(string);
                void setName(string);
                void setMarks(int);
                void setCrdHr(int);
      string getCode();
                string getName();
                int getMarks();
                int getCrdHr();
                float getGradepoint();
                float getGPA();
                void printinformation();
};
data::data()
```

```
{
        cCode="00";
            cName="00";
            marks=0;
            crdhr=0;
            gpa=0.0;
}
data::data(string c,string n,int m,int cr)
{
            cCode=c;
            cName=n;
            marks=m;
            crdhr=cr;
            gpa=0.0;
}
data::setData(string c,string n,int m,int cr)
{
            setCode(c);
            setName(n);
            setMarks(m);
            setCrdHr(m);
}
void data::setCode(string c)
{
    cCode=c;
}
void data::setName(string n)
{
    cName=n;
}
```

```
void data::setMarks(int m)
{
  marks=m;
}
float data::getGradepoint()
{
        if(marks>=85&&marks<=100)
        return 4.00*crdhr;
        else if(marks>=80&&marks<=84)
        return 3.70*crdhr;
        else if(marks>=75&&marks<=79)
        return 3.30*crdhr;
        else if(marks>=70&&marks<=74)
        return 3.00*crdhr;
        else if(marks>=65&&marks<=69)
        return 2.70*crdhr;
        else if(marks>=61&&marks<=64)
        return 2.30*crdhr;
        else if(marks>=58&&marks<=60)
        return 2.00*crdhr;
        else if(marks>=55&&marks<=57)
        return 1.70*crdhr;
        else if(marks>=50&&marks<=54)
        return 1.00*crdhr;
        else
        return 0.00*crdhr;
}
void data::setCrdHr(int cr)
{
        crdhr=cr;
```

```
}
string data::getCode()
{
	return cCode;
}
string data::getName()
{
	return cName;
}
int data::getMarks()
{
	return marks;
}
int data::getCrdHr()
{
	return crdhr;
}
void data::printinformation()
{
	cout<<this->getCode()<<setw(40)<<setfill(' ')<<this->cName<<setw(10)<<setfill('
')<<this->getMarks()<<setw(10)<<setfill(' ')<<this->getCrdHr()<<setw(10)<<setfill('
')<<this->getGPA()<<setw(10)<<setfill(' ')<<this->getGradepoint();
	cout<<endl;
}
float data::getGPA()
{
	if(marks>=85&&marks<=100)
	gpa= 4.00;
	else if(marks>=80&&marks<=84)
	gpa= 3.70;
	else if(marks>=75&&marks<=79)
```

```
        gpa= 3.30;
        else if(marks>=70&&marks<=74)
        gpa= 3.00;
        else if(marks>=65&&marks<=69)
        gpa= 2.70;
        else if(marks>=61&&marks<=64)
        gpa= 2.30;
        else if(marks>=58&&marks<=60)
        gpa= 2.00;
        else if(marks>=55&&marks<=57)
        gpa= 1.70;
        else if(marks>=50&&marks<=54)
        gpa= 1.00;
        else
        gpa= 0.00;
        return gpa;
}

/*********************************************
                    class course list
*********************************************/

class list
{
        private:
                data *ptr;
                int current;
                int size;
                int length;
        public:
```

```
                list();
                list(int);
                void add(data);
                float getCGP();
                void printList();
                void copyList(list);
                void next();
                void back();
                void start();
                void tail();
                void clearList();
                int getLength();
                ~list();
};
list::list()
{
                length=0;
                size=0;
                current=0;
}
list::list(int l)
{
        if(l>0)
        {
                length=l+1;
                size=0;
                current=0;
                ptr=new data[length];

        }
```

```
}
void list::add(data val)
{
	if(current==length)
	{
		cout<<"list is full"<<endl;
	}
	else
	{
		ptr[++current]=val;
		size++;
	}
}
float list::getCGP()
{
	float sum=0.0;
	int cr=0;
	for(int i=1;i<=size;i++)
	{
		sum=sum+ptr[i].getGradepoint();
		cr=cr+ptr[i].getCrdHr();
	}
	  return sum/cr;
}
void list::printList()
{
	if(size==0)
	{
		cout<<"List is empty :"<<endl;
	}
```

```
cout<<"**********************************************************************
**********************\n";
        cout<<"Cource code \t\t Subject \t\t\t Marks \t Cr.hr \t G_P\t T_G_P"<<endl;
        cout<<"**********************************************************************
*****************************\n";


        for(int i=1;i<=size;i++)
        {
        ptr[i].printinformation();
        }
        cout<<endl;
        cout<<"\tGPA = "<<this->getCGP()<<endl;
        }
void list::next()
{
        if(current==size)
        {
                cout<<"current is alread at end, cant move further"<<endl;
        }
        else
        current++;
}
void list::back()
{
        if(current==1)
        {
                cout<<"current is alread at start, cant move further"<<endl;
        }
        else
        current--;
```

```
}
void list::start()
{
	current==1;
}
void list::tail()
{
	current==size;
}
void list::clearList()
{
	current=0;
	size=0;
}
void list::copyList(list a)
{
	length=a.length;
	current=a.current;
	size=a.size;
	ptr=new data[length];
	for(int i=1;i<=length;i++)
		{
			ptr[i]=a.ptr[i];
		}
}
int list::getLength()
{
	return length;
}
list::~list()
```

```
{
        delete []ptr;
}

/*********************************************
                        class Student
*********************************************/

class student
{
        private:
                string id;
                string name;
                string address;
                char gender;
                list *ptr;
                int current;
                int size;
                int length;
        public:
                student(int);
                void add(list);
                void setData(string,string,string,char);
                void showData();
                void setId(string);
                void setName(string);
                void setAddress(string);
                void setGender(char);
                string getId();
                string getName();
```

```
                string getAddress();
                char getGender();
};
student::student(int l)
{
        if(l>0)
        {
                length=l+1;
                size=0;
                current=0;
                ptr=new list[length];
        }
}
void student::setData(string Id,string Name,string Address,char Gender)
{
        id=Id;
        name=Name;
        address=Address;
        gender=Gender;
}
void student::setId(string Id)
{
        id=Id;
}
void student::setName(string Name)
{
        name=Name;
}
void student::setAddress(string Address)
{
```

```
	address=Address;
}
void student::setGender(char Gender)
{
	gender=Gender;
}
string student::getId()
{
	return id;
}
string student::getName()
{
	return name;
}
string student::getAddress()
{
	return address;
}
char student::getGender()
{
	return gender;
}
void student::add(list val)
{
	if(current==length)
	{
		cout<<"list is full"<<endl;
	}
	else
	{
```

```
                ptr[++current]=val;
                size++;
        }
}
void student::showData()
{
        cout<<"\tStudent ID : "<<id<<"\t\tName : "<<name<<"\t\tGender : "<<gender<<endl;
        cout<<"\n\tAddress : "<<address<<"\t\tProgram : BSIT"<<endl;
        if(size==0)
        {
                cout<<"List is empty :"<<endl;
        }
        else
        {
        float sum;
        for(int i=1;i<=size;i++)
        {
        ptr[i].printList();
        cout<<"  ";
        sum=sum+ptr[i].getCGP();
        }
        cout<<endl;
        cout<<"CGPA= "<<sum/size;
        }
}
int main()
{
        int d,m,y,cd,cm,cy;
        char gender;
        string roll,name,address;
```

```
        cout<<"Enter Student ID: ";
        getline(cin,roll);
        cout<<"Enter Student Name: ";
        getline(cin,name);
        cout<<"Enter Student Address: ";
        getline(cin,address);
        cout<<"Enter Student Gender: ";
        cin>>gender;
        cout<<"Enter Date of Birth: ";
        cin>>d>>m>>y;
        cout<<"Enter Current Date: ";
        cin>>cd>>cm>>cy;

//******************Semester 1**************************
        data s1d1,s1d2,s1d3,s1d4,s1d5,s1d6;
        s1d1.setData("GE-161","Intro. to infor. & Comm. Technology",75,3);
        s1d2.setData("MS-152 ","Probability     and     Statistics",68,3);
        s1d3.setData("GE-162","English composition & Comprehension",84,3);
        s1d4.setData("CC-111 ","Discrete              Structure",88,3);
        s1d5.setData("MS-151 ","Applied                 Physics",52,3);
   s1d6.setData("HQ-001 ","Quranic             Translation",82,0);

                list lS1(6);
                lS1.add(s1d1);
                lS1.add(s1d2);
                lS1.add(s1d3);
                lS1.add(s1d4);
                lS1.add(s1d5);
                lS1.add(s1d6);
                //******************Semester 2**************************
```

```
        data s2d1,s2d2,s2d3,s2d4,s2d5,s2d6;
s2d1.setData("CC-212  ","Programming          Fundamentls",88,4);
        s2d2.setData("CC-215  ","Database               System",77,4);
        s2d3.setData("GE-165  ","Pakistan               Studies",80,3);
        s2d4.setData("GE-164  ","Communication &Presentation skill",75,3);
        s2d5.setData("UE-171  ","Introduction    to    Economics",80,3);
        s2d6.setData("QT-002  ","Quran              Translation",89,1);
        list lS2(6);
        lS2.add(s2d1);
        lS2.add(s2d2);
        lS2.add(s2d3);
        lS2.add(s2d4);
        lS2.add(s2d5);
        lS2.add(s2d6);
        //******************Semester 3**************************
        data s3d1,s3d2,s3d3,s3d4,s3d5,s3d6;
        s3d1.setData("CC-211 ","  Object   Oriented   Programming",75,4);
        s3d2.setData("UE-273 ","  Introduction   to    Sociology",68,3);
        s3d3.setData("MS-152 ","  Calculus and Analytical Geometry",84,3);
        s3d4.setData("EI-231 ","Computer Organization and Assembly",88,3);
        s3d5.setData("CC-214 ","  Computer              Networks",52,3);
s3d6.setData("HQ-003 ","  Quran              Translation",80,2);
        list lS3(6);
        lS3.add(s3d1);
        lS3.add(s3d2);
        lS3.add(s3d3);
        lS3.add(s3d4);
        lS3.add(s3d5);
        lS3.add(s3d6);
        //**********************************************
```

```
        student s(3);
        s.add(lS1);
        s.add(lS2);
        s.add(lS3);
        s.setData(roll,name,address,gender);
        date i(d,m,y);
        date j(cd,cm,cy);
        cout<<"\n\n\t";
        i.printDate();
        i-=j;
        cout<<"\t\t";
        i.printAge();
        cout<<endl;
        s.showData();
}
```

Question: Design a C++ program to make the result card of student having class date for calculation of age, class data for course code, course name, marks and GPA, then make a class list for record of all students and then make class student for result card using linked list.

```
#include<iostream>
#include<iomanip>
#include<string.h>
using namespace std;
int limit[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
string
name[13]={"Null","January","February","March","April","May","June","July","August","September","October","November","December"};

/***********************************************
                    class date
```

```
**************************************************/
class date
{
        private:
                int day;
                int month;
                int year;
        public:
                date(int ,int ,int );
                void setDate(int ,int ,int);
                void setDay(int);
                void setMonth(int);
                void setYear(int);
                int getDay();
                int getMonth();
                int getYear();
                void printDate();
                void printAge();
                void operator -=(date i)
      {
        if(day>i.day)
        day=day-i.day;
        else
        day=i.day-day;
                        if(month>i.month)
                        month=month-i.month;
                        else
                        month=i.month-month;
                        year=i.year-year;
                }
```

```
};
date::date(int d,int m,int y)
{
	this->setDate(d,m,y);
}
void date::setDate(int d,int m,int y)
{
	this->setDay(d);
	this->setMonth(m);
	this->setYear(y);
}
void date::setDay(int d){
	day=((d>=1&&d<=limit[this->getMonth()])?d:1);
	if(this->getMonth()==2&&this->getYear()%4==0);
	day=((d>=1&&d<=29)?d:1);
}
void date::setMonth(int m){
		month=((m>=1&&m<=12)?m:1);
}
void date::setYear(int y){
		year=((y>=1990&&y<=2023)?y:1);
}
int date::getDay()
{
	return day;
}
int date::getMonth()
{
	return month;
}
```

```
int date::getYear()
{
	return year;
}

void date::printDate()
{
	cout<<"Date-of-Birth : "<<this->getDay()<<"-"<<name[this->month]<<"-"<<this->getYear();
}
void date::printAge()
{
	cout<<"Current   Age : "<<this->getYear()<<"-"<<this->month<<"-"<<this->getDay();
}
/************************************************
                                        class data
************************************************/
class data
{
	private:
		string cCode;
		string cName;
		float marks;
		int crdhr;
		float gpa;
		data *next;
	public:
		data(string,string,float,int);
		void setCode(string);
		void setName(string);
```

```
                void setMarks(float);
                void setCrdHr(int);
                void setNext(data *);
                data *getNext();
        string getCode();
                string getName();
                float getMarks();
                int getCrdHr();
                float getGradepoint();
                float getGPA();
                void printinformation();
};
data::data(string c,string n,float m,int cr)
{
                cCode=c;
                cName=n;
                marks=m;
                setMarks(m);
                crdhr=cr;
                gpa=0.0;
}
void data::setCode(string c)
{
        cCode=c;
}
void data::setName(string n)
{
        cName=n;
}
void data::setMarks(float m)
```

```
{
  marks=m;
}
void data::setNext(data *ptr)
{
        next=ptr;
}
data* data::getNext()
{
        return next;
}
float data::getGradepoint()
{
        if(marks>=85&&marks<=100)
        return 4.00*crdhr;
        else if(marks>=80&&marks<=84)
        return 3.70*crdhr;
        else if(marks>=75&&marks<=79)
        return 3.30*crdhr;
        else if(marks>=70&&marks<=74)
        return 3.00*crdhr;
        else if(marks>=65&&marks<=69)
        return 2.70*crdhr;
        else if(marks>=61&&marks<=64)
        return 2.30*crdhr;
        else if(marks>=58&&marks<=60)
        return 2.00*crdhr;
        else if(marks>=55&&marks<=57)
        return 1.70*crdhr;
        else if(marks>=50&&marks<=54)
```

```
        return 1.00*crdhr;
        else
        return 0.00*crdhr;
}
void data::setCrdHr(int cr)
{
        crdhr=cr;
}
string data::getCode()
{
        return cCode;
}
string data::getName()
{
        return cName;
}
float data::getMarks()
{
        return marks;
}
int data::getCrdHr()
{
        return crdhr;
}
void data::printinformation()
{
cout<<this->getCode()<<setw(40)<<setfill(' ')<<this->cName<<setw(15)<<setfill(' ')<<this-
>getMarks()<<setw(10)<<setfill(' ')<<this->getCrdHr()<<setw(10)<<setfill(' ')<<this-
>getGPA()<<setw(10)<<setfill(' ')<<this->getGradepoint();
                cout<<endl;
}
```

```
float data::getGPA()
{
        if(marks>=85&&marks<=100)
        gpa= 4.00;
        else if(marks>=80&&marks<=84)
        gpa= 3.70;
        else if(marks>=75&&marks<=79)
        gpa= 3.30;
        else if(marks>=70&&marks<=74)
        gpa= 3.00;
        else if(marks>=65&&marks<=69)
        gpa= 2.70;
        else if(marks>=61&&marks<=64)
        gpa= 2.30;
        else if(marks>=58&&marks<=60)
        gpa= 2.00;
        else if(marks>=55&&marks<=57)
        gpa= 1.70;
        else if(marks>=50&&marks<=54)
        gpa= 1.00;
        else
        gpa= 0.00;
        return gpa;
}

/************************************************
                        class list
************************************************/

class list
```

```
{
        private:
                data *head;
                data *current;
                int size;
                list *next;
        public:
                list();
                void add(string,string,float,int);
                void showList();
                float getCGP();
                void setNext(list *);
                list *getNext();
};
list::list()
{
        size=0;
        head=NULL;
        current=NULL;
}
void list::setNext(list *ptr)
{
        next=ptr;
}
list* list::getNext()
{
        return next;
}
void list::add(string c,string n,float m,int cr)
{
```

```
	data* ptr=new data(c,n,m,cr);
	if(size==0)
	{
		head=ptr;
		current=ptr;
	}
	else
	{
		ptr->setNext(current->getNext());
		current->setNext(ptr);
		current=ptr;
	}
	size++;
}
void list::showList()
{
	data* ptr=head;
	cout<<"**********************************************************************
**********************************\n";
	cout<<"Cource code \t\t Subject \t\t\t Marks \t   Cr.hr\tG_P\tT_G_P"<<endl;
	cout<<"**********************************************************************
**********************************\n";
	while(ptr!=NULL)
	{
		ptr->printinformation();
		ptr=ptr->getNext();
	}
	cout<<endl;
	cout<<"GPA = "<<this->getCGP()<<endl;
	cout<<endl;
}
```

```
float list::getCGP()
{
        data* ptr=head;
        float sum=0.0;
        int cr=0;
        while(ptr!=NULL)
        {
                sum=sum+ptr->getGradepoint();
                cr=cr+ptr->getCrdHr();
                ptr=ptr->getNext();
        }
   return sum/cr;
}

/************************************************
                                        class student
************************************************/
class student
{
        private:
                string id;
                string name;
                string address;
                char gender;
                list *head;
                list *current;
                int Size;
        public:
                student();
                void add(list);
```

```
                void setData(string,string,string,char);
                void showStudent();
                void setId(string);
                void setName(string);
                void setAddress(string);
                void setGender(char);
                string getId();
                string getName();
                string getAddress();
                char getGender();
};
student::student()
{
        Size=0;
        head=NULL;
        current=NULL;
        id="NULL";
        name="NULL";
        address="NULL";
}
void student::setData(string Id,string Name,string Address,char Gender)
{
        id=Id;
        name=Name;
        address=Address;
        gender=Gender;
}
void student::setId(string Id)
{
        id=Id;
```

```
}
void student::setName(string Name)
{
	name=Name;
}
void student::setAddress(string Address)
{
	address=Address;
}
void student::setGender(char Gender)
{
	gender=Gender;
}
string student::getId()
{
	return id;
}
string student::getName()
{
	return name;
}
string student::getAddress()
{
	return address;
}
char student::getGender()
{
	return gender;
}
void student::add(list d)
```

```
{
	list* ptr=new list(d);
	if(Size==0)
	{
		head=ptr;
		current=ptr;
	}
	else
	{
		ptr->setNext(current->getNext());
		current->setNext(ptr);
		current=ptr;
	}
	Size++;
}
void student::showStudent()
{
	cout<<"\tStudent ID : "<<id<<"\t\\ttName : "<<name<<"\t\tGender :
"<<gender<<endl;
	cout<<"\n\tAddress : "<<address<<"\t\tProgram : BSIT"<<endl;
	if(Size==0)
	{
		cout<<"List is empty :"<<endl;
	}
	else
	{
	list* ptr=head;
	while(ptr!=NULL)
	{
		ptr->showList();
		ptr=ptr->getNext();
```

```
        }
        }
}
int main()
{
        string S1code[7]={"Null","GE-161","MS-152","GE-162","CC-111","MS-151","HQ-001"};
        string S1name[7]={"Null","Intro.to infor. & Comm. Technology","Probability and Statistics","English composition & Comprehension","Discrete Structure","Applied Physics","Quranic Translation"};
        int S1chr[7]={0,3,3,3,3,3,0};
        string S2code[7]={"Null","CC-212","CC-215","GE-165","GE-164","UE-171","QT-002"};
        string S2name[7]={"Null","Programming Fundamentls","Database System","Pakistan Studies","Communication &Presentation skill","Introduction to Economics","Quran Translation"};
        int S2chr[7]={0,4,4,3,3,3,1};
        string S3code[7]={"Null","CC-211","UE-273","MS-152","EI-231","CC-214","HQ-003"};
        string S3name[7]={"Null","Object Oriented Programming","Introduction to Sociology","Calculus and Analytical Geometry","Computer Organization and Assembly","Computer Networks","Quran Translation"};
        int S3chr[7]={0,4,3,3,4,4,0};
        char gender;
        int d,m,y,cd,cm,cy;
        string roll,name,address;
        cout<<"Enter Student ID: ";
        getline(cin,roll);
        cout<<"Enter Student Name: ";
        getline(cin,name);
        cout<<"Enter Student Address: ";
        getline(cin,address);
        cout<<"Enter Student Gender: ";
        cin>>gender;
```

```
        cout<<"Enter Date of Birth: ";
        cin>>d>>m>>y;
        cout<<"Enter Current Date: ";
        cin>>cd>>cm>>cy;
        cout<<endl;
        cout<<"******************Semester 1**************************"<<endl;
        cout<<"\n--------------Enter Data of 1st Semester--------------\n"<<endl;
        list lst1;
        for(int i=1;i<=6;i++){
                int marks;
                cout<<"Enter "<<S1name[i]<<"  Marks : ";
                cin>>marks;
                lst1.add(S1code[i],S1name[i],marks,S1chr[i]);
        }
        cout<<"\n******************Semester
2**************************\n"<<endl;
        cout<<"\n--------------Enter Data of 2nd Semester--------------\n"<<endl;
        list lst2;
        for(int i=1;i<=6;i++){
                int marks;
                cout<<"Enter "<<S2name[i]<<"  Marks : ";
                cin>>marks;
                lst2.add(S2code[i],S2name[i],marks,S2chr[i]);
        }
        cout<<"\n******************Semester
3**************************\n"<<endl;
        cout<<"\n--------------Enter Data of 3rd Semester--------------\n"<<endl;
        list lst3;
        for(int i=1;i<=6;i++){
                int marks;
                cout<<"Enter "<<S3name[i]<<"  Marks : ";
```

```
            cin>>marks;
            lst3.add(S3code[i],S3name[i],marks,S3chr[i]);
        }
        student s;
        s.add(lst1);
        s.add(lst2);
        s.add(lst3);
        s.setData(roll,name,address,gender);
        date i(d,m,y);
        date j(cd,cm,cy);
        cout<<"\n\n\t";
        i.printDate();
        i-=j;
        cout<<"\t\t";
        cout<<"Final CGPA = "<<(lst1.getCGP()+lst2.getCGP()+lst3.getCGP())/3;
        cout<<"   ";
        i.printAge();
        cout<<endl;
        s.showStudent();
}
```

Question: Design a C++ program to implement a doubly circular linked list. Defne a Node structure with data, a pointer to the next node, and a pointer to the previous node. Implement functions to perform the operations. Demonstrate the usage of your program by performing various insertions, deletions, traversals, and searches on the doubly circular linked list.

```
#include<iostream>
using namespace std;
class node
{
        private:
            int data;
```

```
                node *next;
                node *prev;
        public:
                node(int);
                void setData(int );
                int getData();
                void setNext(node *);
                node* getNext();
                void setPrev(node *);
                node* getPrev();
                void showData();
};
node::node(int d)
{
        setData(d);
        next=NULL;
        prev=NULL;
}
void node::setData(int d)
{
        data = d;
}
int node::getData()
{
        return data;
}
void node::setNext(node *ptr)
{
        next = ptr;
}
```

```
node* node::getNext()
{
        return next;
}
void node::setPrev(node *ptr)
{
        prev = ptr;
}
node* node::getPrev()
{
        return prev;
}
void node::showData()
{
        cout<<data<<"   ";
}
class list
{
        private:
                node *head;
                node *current;
                int size;
        public:
                list();
                void add(int );
                void showList();
                void remove();
                void insertAtHead(int);
};
list::list()
```

```
{
		head = NULL;
		current = NULL;
		size =0;
}
void list::add(int d)
{
		node *ptr = new node(d);
		if(size == 0){
			head = ptr;
			current = ptr;
			current->setNext(head);
			current->setPrev(head);
		}
		else{
			ptr->setNext(current->getNext());
			ptr->setPrev(current);
			current->getNext()->setPrev(ptr);
			current->setNext(ptr);
			current = ptr;
		}
		size++;
}
void list::insertAtHead(int d)
{
	node *ptr=new node(d);
	ptr->setNext(head);
	ptr->setPrev(current);
	current->setNext(ptr);
	current->getNext()->setPrev(ptr);
```

```
        head=ptr;
        current=ptr;
        size++;
}
void list::showList()
{
        if(size==0)
        {
                cout<<"List is Empty";
        }
                node *ptr=head;
                do
                {
                        ptr->showData();
                        ptr=ptr->getNext();
                }while(ptr!=head);
}
void list::remove()
{
        if(size==0)
        {
                cout<<"List is Empty";
        }
        if(head==current)
        {
                head=head->getNext();
        }
        node *temp=current;
        node *ptr=head;
        do{
```

```
                if(head==current)
                {
                        break;
                }
                ptr=ptr->getNext();
        }
        while(ptr!=head);
        ptr->setNext(current->getNext());
        current=current->getNext();
        current->getNext()->setPrev(ptr);
        delete temp;
        size--;
}
int main()
{
                cout<<"hello world"<<endl;
                list l;
                l.add(40);
                l.showList();
                cout<<endl;
                l.add(90);
                l.showList();
                cout<<endl;
                l.add(91);
                l.showList();
                cout<<endl;
                l.add(30);
                l.showList();
                cout<<endl;
                l.insertAtHead(10);
```

```
l.showList();
cout<<endl;
l.add(20);
l.showList();
l.remove();
cout<<endl;
l.showList();
l.remove();
cout<<endl;
l.showList();
l.remove();
cout<<endl;
l.showList();
l.remove();
cout<<endl;
l.showList();
l.remove();
cout<<endl;
l.showList();
l.remove();
cout<<endl;
l.add(40);
l.showList();
cout<<endl;
l.add(90);
l.showList();
cout<<endl;
l.add(91);
l.showList();
cout<<endl;
```

```
    l.add(30);
    l.showList();
    cout<<endl;
    l.insertAtHead(10);
    l.showList();
    cout<<endl;
    l.add(20);
    l.showList();
    l.remove();
    cout<<endl;
    l.showList();
    l.remove();
    cout<<endl;
    l.showList();
    l.remove();
    cout<<endl;
    l.showList();
    l.remove();
    cout<<endl;
    l.showList();
    l.remove();
    cout<<endl;
    l.showList();
    l.remove();
    cout<<endl;
}
```

Question: Design a C++ program to implement a generic queue using an array with templates. Include functionalities to enqueue, dequeue, and check the front element of the queue. Implement error handling for queue underflow conditions. In addition, use templates to make the queue suitable for storing elements of any data type. Demonstrate

the usage of your generic queue implementation by enqueuing and dequeuing elements of various data types.

```
#include<iostream>
using namespace std;
template <class t>
class queue
{
	private:
		t *data;
		int size;
		int front;
		int rear;
		int ele;
	public:
		queue(int);
		int enqueue(t);
		t dequeue();
		t getFront();
		int Elements();
		bool Isempty();
		bool IsFull();
};
template <class t>
queue<t>::queue(int s)
{
	size=s;
	front=1;
	rear=0;
	data=new t[size];
}
template <class t>
```

```
int queue<t>::enqueue(t x)
{
        rear=(rear+1)%size;
        data[rear]=x;
        ele++;
        return 1;
}
template <class t>
t queue<t>::dequeue()
{
        t x=data[front];
        front=(front+1)%size;
   ele--;
        return x;
}
template <class t>
t queue<t>::getFront()
{
        return data[front];
}
template <class t>
int queue<t>::Elements()
{
        return ele;
}
template <class t>
bool queue<t>::Isempty()
{
        if(ele==0)
        return true;
```

```
        else
        return false;
}
template <class t>
bool queue<t>::IsFull()
{
        if(ele==size)
        return true;
        else
        return false;
}
//********************
int main()
{
        queue<int> i(5);
        cout<<"Enqueue data = "<<i.enqueue(1)<<endl;
        cout<<"Enqueue data = "<<i.enqueue(2)<<endl;
        cout<<"Enqueue data = "<<i.enqueue(3)<<endl;
        cout<<"Enqueue data = "<<i.enqueue(4)<<endl;
        cout<<"Enqueue data = "<<i.enqueue(5)<<endl;
        cout<<"Front   Element = "<<i.getFront()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Is  Empty Queue = "<<i.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i.IsFull()<<endl;
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        cout<<"Enqueue data = "<<i.enqueue(10)<<endl;
        cout<<"Enqueue data = "<<i.enqueue(20)<<endl;
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        cout<<"Front   Element = "<<i.getFront()<<endl;
```

```
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i.dequeue()<<endl;
        cout<<"No.  of Element = "<<i.Elements()<<endl;
        cout<<"Front   Element = "<<i.getFront()<<endl;
        cout<<"Is  Empty Queue = "<<i.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i.IsFull()<<endl;
cout<<"\n**********************\n**********************\n"<<endl;
        queue<string> i1(5);
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Front   Element = "<<i1.getFront()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"Is  Empty Queue = "<<i1.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i1.IsFull()<<endl;
        cout<<"No.  of Element = "<<i1.Elements()<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"Enqueue data = "<<i1.enqueue("BIT21218")<<endl;
        cout<<"No.  of Element = "<<i1.Elements()<<endl;
        cout<<"Front   Element = "<<i1.getFront()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
```

```
        cout<<"Dequeue Element = "<<i1.dequeue()<<endl;
        cout<<"No.  of Element = "<<i1.Elements()<<endl;
        cout<<"Front   Element = "<<i1.getFront()<<endl;
        cout<<"Is  Empty Queue = "<<i1.Isempty()<<endl;
        cout<<"Is  Full  Queue = "<<i1.IsFull()<<endl;
}
```

Question: Design a C++ program to implement a generic queue using a linked list with templates. Define a Node structure for the linked list and include functionalities to enqueue, dequeue, and check the front element of the queue. Implement error handling for queue underflow conditions. Additionally, use templates to make the queue suitable for storing elements of any data type. Demonstrate the usage of your generic queue implementation by enqueuing and dequeuing elements of various data types.

```
#include<iostream>
using namespace std;
template <class t>
class node
{
        private:
                t data;
                node<t> *next;
        public:
                node(t);
                void setData(t);
                t getData();
                void showData();
                void setNext(node<t>*);
                node<t>* getNext();
};
template <class t>
node<t>::node(t d)
{
```

```
        setData(d);
        next=NULL;
}
template <class t>
void node<t>::setData(t d)
{
        data=d;
}
template <class t>
t node<t>::getData()
{
        return data;
}
template <class t>
void node<t>::showData()
{
        cout<<data<<"  ";
}
template <class t>
void node<t>::setNext(node<t> *ptr)
{
        next=ptr;
}
template <class t>
node<t>* node<t>::getNext()
{
        return next;
}
template<class T>
class queue
```

```
{
        private:
                node<T> *front;
                node<T> *rare;
        public:
                queue();
                int enqueuq(T);
                T dequeue();
                T frontE();
                bool IsEmpty();
};
template<class T>
queue<T>::queue()
{
        front=NULL;
        rare=NULL;
}
template<class T>
int queue<T>::enqueuq(T d)
{
        node<T> *ptr=new node<T>(d);
        if(front==NULL)
        {
                front=ptr;
                rare=ptr;
        }
        else
        //ptr->setNext(NULL);
        rare->setNext(ptr);
        rare=ptr;
```

```
        return 1;
}
template<class T>
T queue<T>::dequeue()
{
        T x=front->getData();
        node<T>* ptr=front;
        front=front->getNext();
        delete ptr;
        return x;
}
template<class T>
bool queue<T>::IsEmpty()
{
        return (front==NULL);

}
template<class T>
T queue<T>::frontE()
{
        return front->getData();
}
int main()
{
        cout<<"Hello world!"<<endl;
        queue<int> q;
        cout<<"Empty = "<<q.IsEmpty()<<endl;
        cout<<"Enque data = "<<q.enqueuq(1)<<endl;
        cout<<"Empty = "<<q.IsEmpty();
        cout<<endl;
```

```
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Enqueuue data = "<<q.enqueuq(2)<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Empty = "<<q.IsEmpty();
	cout<<endl;
	cout<<"Enque data = "<<q.enqueuq(10)<<endl;
	cout<<"Enque data = "<<q.enqueuq(100)<<endl;
	cout<<"Enque data = "<<q.enqueuq(1000)<<endl;
	cout<<"Enque data = "<<q.enqueuq(10000)<<endl;
	cout<<"Enque data = "<<q.enqueuq(100000)<<endl;
	cout<<"Enque data = "<<q.enqueuq(1000000)<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Front Element = "<<q.frontE()<<endl;
	cout<<"Dequeue = "<<q.dequeue()<<endl;
	cout<<"Empty = "<<q.IsEmpty();
cout<<"\n\n*************************\n*************************\n\n"<<endl;
	queue<string> q1;
	cout<<"Empty = "<<q1.IsEmpty()<<endl;
	cout<<"Enque data = "<<q1.enqueuq("BIT21218")<<endl;
```

```
        cout<<"Empty = "<<q1.IsEmpty();
        cout<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Enqueuue data = "<<q1.enqueuq("FARHAN AHMAD")<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Empty = "<<q1.IsEmpty();
        cout<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21219")<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21220")<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21221")<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21222")<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21223")<<endl;
        cout<<"Enque data = "<<q1.enqueuq("BIT21224")<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Front Element = "<<q1.frontE()<<endl;
        cout<<"Dequeue = "<<q1.dequeue()<<endl;
        cout<<"Empty = "<<q1.IsEmpty();

}
```

Question: Design a C++ program to implement a tree structure using a dictionary. Use a dictionary data structure to represent a hierarchical structure where each node has a word and a meaning associated with it.

```
#include<iostream>
#include<conio.h>
#include<string.h>
#include<iomanip>
using namespace std;
class node
{
	private:
		string word;
		string meaning;
		node *left;
		node *right;
	public:
		node(string,string);
		void set_data(string,string);
		void show_data();
		void set_word(string);
		string get_word();
		void set_meaning(string);
		string get_meaning();
		void set_left(node *);
		node* get_left();
		void set_right(node*);
		node* get_right();
		bool isLeaf();
};
node::node(string d,string m)
```

```
{
   word = d;
        meaning = m;
        left = NULL;
        right = NULL;
}
void node::set_data(string d,string m)
{
        word = d;
        meaning = m;
}
void node::show_data()
{
        cout<<word<<"\t\t"<<meaning<<endl;
}
void node::set_word(string d)
{
        word = d;
}
string node::get_word()
{
        return word;
}
void node::set_meaning(string m)
{
        meaning = m;
}
string node::get_meaning()
{
        return meaning;
```

```
}
void node::set_left(node *ptr)
{
        left = ptr;
}
node* node::get_left()
{
        return left;
}
void node::set_right(node * ptr)
{
        right = ptr;
}
node * node::get_right()
{
        return right;
}
bool node::isLeaf()
{
                if(left==NULL && right==NULL)
                return true;
                else
                return false;
}
  //function to add new node in binary tree
void insert (node* r,string word,string meaning) ;
        //functions to display the binary tree values in
        //preorder....inorder.....postorder....
void preorder(node *);
void inorder(node *);
```

```
void postorder(node *);


//defintion of insert function
void insert(node *r,string word,string meaning)
{
   node *newnode=new node(word,meaning);
   node *p,*f;
   p=f=r;
   while(strcmp(word,p->get_word())!=0&& f !=NULL)
   {
            p=f;
            if(word>p->get_word())
            f=f->get_right();
            else
            f=f->get_left();
        }
   if(word==p->get_word())
   {
   cout<<"\nDuplicate Value....."<<word<<endl;
        delete newnode;
        return;
        }
        else if(word>p->get_word())
        p->set_right(newnode);
        else
        p->set_left(newnode);
}
//definition the the preorder function
void preorder(node *r)
{
```

```
        if(r!=NULL)
   {
        r->show_data();
        preorder(r->get_left())          ;
        preorder(r->get_right());
        }
}
//definiton of the member function
void inorder(node *r)
{
   if(r!=NULL)
   {
        inorder(r->get_left())  ;
        r->show_data();
        inorder(r->get_right());
        }
}
//definiton of the member function post order
void postorder(node *r)
{
   if(r!=NULL)
   {
        postorder(r->get_left())          ;
        postorder(r->get_right());
        r->show_data();
        }
}
        //****Main body**************
int main()
{
```

```
//        int a[7];
//        int b[7];
          string
a[100]={"about","abstract","almost","animated","backbone","backside","bad","becoming","c
andy","charter","cheesy","chorus"," dune "," dazzling "," domain "," dignity "," excite ","
eradicate "," evil "," ethical "," faith "," firm "," forsake "," foreign "," galvanized ","
geared","garbage","gay","hall","handsome","happily","hard","idler","if","illiberal","immediat
e","jealous","joy","jerkweed","knowingly","lacking","last","leading","legitimate","madness",
"magician","material","manmade","naked","nameless","napkin","necessary","obdurate","obj
ect","obligatory","oblique","painting","particular","passable","pattern","quite","reasonable","
refrain","reliable","religious","remainder","remark","reminiscence","sacrity","satisfied","scar
city","scrumptious","second","select","selection","signal","tailored","temper","terror","testa
ment","ultimate","uncommon","uncooked","undeniable","vacancy","vague","vain","valueles
s","warranty","well-timed","winery","well-
mannered","unhurt","unfortunate","unlawful","unmarried","trustworthy","twister","twosome
","zenith"};
          string
b[100]={"approximate","summary","nearly","lively","spine","behind","poor","fitting","sweet
","constitution","corny","refrain"," mound "," blurring "," field "," grace "," activate ","
remove "," vice "," proper "," belief "," staunch "," desert "," alien "," arouse "," ready
","rubbish","homosexual","corridor","good-
looking","fortunately","tough","loafer","whether","intolerant","instant","envious","detestable
","delight","deliberately","missing","final","main","valid","insanity","conjuror","fabric","arti
fical","bare","anonymous","serviette","essential","stubborn","thing","compulsory","indirect",
"portray","specific","satisfactory","sample","fair","fair","chorus","dependable","devout","rest
","comment","memory","vestry","convinced","shortage","delicious","moment","choose","ch
oice","sign","tailor-
made","mood","terrorism","testimony","final","unusual","raw","indisputable","emptiness","i
ndistinct","useless","worthless","guarantee","timelt","vineyard","polite","unharmed","unluck
y","illegal","single","reliable","tornado","pair","peak",};
          node *root=new node("NULL","NULL");
//        node<int> *root=new node<int>(0,0);
          root->set_data(a[0],b[0]);
//        cout<<"Word\t\tSynonym"<<endl;
          for(int i=1;i<100;i++)
          {
//                cin>>a[i];
//                cout<<"\t\t";
//                cin>>b[i];
```

```
        insert(root,a[i],b[i]);
    }
    cout<<"\nDisplay data in preorder format...\n";
    cout<<"Word\t\tSynonym\n"<<endl;
        preorder(root);
    cout<<"\nDisplay data in inorder format....\n";
    cout<<"Word\t\tSynonym\n"<<endl;
    inorder(root);
    cout<<"\nDisplay data in postorder format....\n";
    cout<<"Word\t\tSynonym\n"<<endl;
        postorder(root);
}
```