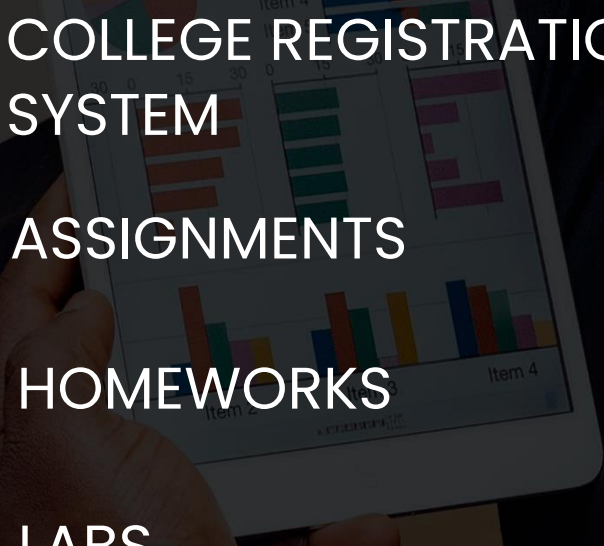


Year
2023

PROJECT REPORT

DATA STRUCTURES AND ALGORITHMS

- 
- 01** COLLEGE REGISTRATION
SYSTEM
 - 02** ASSIGNMENTS
 - 03** HOMEWORKS
 - 04** LABS

PRESENTED TO:

Dr. SALMAN NASEER

DEPARTMENT OF INFORMATION TECHNOLOGY
UNIVERSITY OF THE PUNJAB
GUJRANWALA CAMPUS



PREPARED BY:

<i>Name</i>	<i>Roll no.</i>
<i>Hammad Bin Tariq</i>	<i>BIT21203</i>
<i>Rimsha Javaid</i>	<i>BIT21204</i>
<i>Farhan Ahmad</i>	<i>BIT21218</i>
<i>Kinza Zafar</i>	<i>BIT21247</i>
<i>Muhammad Zubair</i>	<i>BIT21250</i>
<i>Mubeen Fatima</i>	<i>BIT21259</i>

SUBMITTED TO:

Prof. Salman Naseer

Acknowledgment:

*In preparation for this project, we had to take the help and the guidance of a few respected persons, who deserve our deepest gratitude. As the completion of this project gave us much pleasure, we would like to show our indebtedness to **Prof. Salman Naseer**, Course instructor, at Punjab University for giving us good guidelines for project throughout numerous consultations. We would also like to expand our gratitude to all those who have directly and indirectly guided us in writing this project.*

We would also like to thank our classmates, who have made valuable comment suggestions on our paper, which inspired us to improve the quality of our work.

Abstract:

*The **College Registration System** stands as an integrated platform designed to streamline and optimize the course registration process within the academic institution. This comprehensive system facilitates student enrollment, course management, and administrative tasks for faculty and staff. Through a user-centric approach, the system offers a robust student portal and an intuitive admin dashboard, catering to the diverse needs of students, professors, and administrators.*

Key functionalities include a secure user authentication system, efficient course enrollment workflows, real-time course availability checks, and a dynamic waitlist management mechanism. The system employs a multi-tier architecture, utilizing C++ for backend functionalities and console-based interfaces for user interactions.

Rigorous testing phases, including unit testing, integration testing, and user acceptance testing, ensured the system's reliability, security, and performance. The implementation phase encompassed a seamless deployment strategy, transitioning the system into the production environment with careful consideration for user training and acceptance.

This report encapsulates the development, testing, and deployment of the College Registration System, emphasizing its efficiency, scalability, and user-friendly design for enhancing the academic experience within the institution.

Table of Contents:

COLLEGE REGISTRATION SYSTEM	1
1- Phase 1- Introduction	1
1.1 Background	1
1.2 Objectives	1
1.3 Scope and Limitations of the Study	2
2- Phase 2 - System Analysis	8
2.1 Requirement and Collection Ananlysis.....	8
2.2 Analysis	8
3- Phase 3 - System Design	10
3.1 Registration System Design	10
3.2 Database Requirements	11
3.3 Functional Requirements	11
3.4 Database Design	12
3.5 Logical Design	12
4- Phase 4 Coding	13
4.1 Script Writing.....	14
5- Phase 5 Testing	35
5.1 What to Test	36
6- Phase 6 Implementation.....	37
Project Conclusion	39
Project Appendix.....	40
Appendix A	40
Appendix B	40

Appendix C	40
ASSIGNMENTS	41
Assignment 1	42
Assignment 2	45
Assignment 3	61
Assignment 4	77
Assignment 5	83
Assignment 6	87
Assignment 7	92
Assignment 8	98
HOMEWORKS	108
HW 1	109
HW 2	113
HW 3	139
HW 4	144
HW 5	151
HW 6	159
HW 7	168
LABS	184
LAB 1	185
LAB 2	191
LAB 3	194
LAB 4	198
LAB 5	202
LAB 6	209

List of abbreviations:

➤ API	Application Programming Interface
➤ ERD	Entity-Relationship Diagram
➤ UML	Unified Modeling Language
➤ CI/CD	Continuous Integration/ Deployment
➤ DBMS	Database Management System
➤ UAT	User Acceptance Testing
➤ UI	User Interface
➤ UX	User Experience
➤ CPU	Central Processing Unit
➤ SSL	Secure Sockets Layer
➤ HTTP	Hypertext Transfer Protocol
➤ HTTPS	Hypertext Transfer Protocol Secure
➤ QA	Quality Assurance
➤ IDE	Integrated Development Environment
➤ CSV	Comma-Separated Values
➤ DNS	Domain Name System
➤ HTML	Hypertext Markup Language
➤ CSS	Cascading Style Sheets
➤ JS	JavaScript



COLLEGE REGISTRATION SYSTEM

(CRS)



- PHASE 1 INTRODUCTION
- PHASE 2 SYSTEM ANALYSIS
- PHASE 3 SYSTEM DESIGN
- PHASE 4 CODING
- PHASE 5 TESTING
- PHASE 6 IMPLEMENTATION
- CONCLUSION
- APPENDIX



PHASE 1

INTRODUCTION

1.1 Background

1. Existing Registration Processes:

Manual Processes: Historically, college registration involved manual paperwork, long queues, and administrative hassles for both students and staff.

Legacy Systems: In some cases, colleges may have employed outdated or inefficient registration systems that lacked integration or user-friendly interfaces.

2. Issues and Challenges:

Inefficiency: Manual processes are time-consuming and prone to errors, impacting the overall efficiency of the registration process.

Data Discrepancies: Maintaining accurate and up-to-date student records, course availability, and enrollment statuses can be challenging without a centralized system.

Limited Accessibility: Lack of online access and real-time updates hampers students' ability to register for courses promptly and efficiently.

3. Stakeholder Feedback and Needs:

Student Perspective: Feedback from students might highlight issues such as difficulty in finding courses, long waitlists, and unclear prerequisites.

Faculty/Administration Perspective: Administrative staff and faculty members may express concerns about manual paperwork, data management, and delays in approval processes.

1.2 Objectives

1. Efficiency Enhancement:

Streamlined Processes: The system aims to streamline registration processes, reducing paperwork and improving efficiency for both students and administrative staff.

Automation: Automation of registration tasks, approval workflows, and notifications can significantly enhance the overall speed of operations.

2. Improved Data Management:

Centralized Database: A centralized system allows for better management of student records, course catalog updates, and real-time enrollment statuses.

Accuracy and Timeliness: Ensuring accurate and up-to-date data accessible to both students and faculty enhances decision-making and transparency.

3. Accessibility and User Experience:

Online Accessibility: Providing an online platform with user-friendly interfaces enables students to easily search for courses, check prerequisites, and register from anywhere.

Real-time Updates: Timely updates on course availability, waitlists, and enrollment statuses empower students to make informed decisions.

4. Scalability and Adaptability:

Accommodating Growth: A robust system should be scalable to handle increased student enrollment and adaptable to changes in course structures or administrative policies.

Future-Proofing: Anticipating future needs and technological advancements ensures the system's relevance and effectiveness in the long term.

1.3 Scope and Limitations of the System

A. Overview of the College Registration System

1. System Purpose and Functionality:

Description: Begin by providing a detailed overview of the College Registration System. Explain its primary purpose, which revolves around managing the enrollment process for students within the college or university.

Functionality: Describe the core functionalities offered by the system, such as student registration, course selection, enrollment management, waitlist handling, and administrative controls.

Scope: Outline the system's scope, covering the specific tasks it facilitates, such as managing student records, course catalog updates, enrollment validation, etc.

2. Stakeholders Involved:

Users: Identify the different user groups involved, including students, faculty members, administrative staff, and possibly external entities like advisors or registrars.

Roles and Interactions: Define the roles of each stakeholder within the system. For instance, students register for courses, faculty members approve enrollments, administrators manage system configurations, etc.

Interactions: Explain the interactions between different stakeholders and how they utilize the system's features to accomplish their respective tasks.

3. System Benefits and Significance:

Efficiency Enhancement: Elaborate on how the system streamlines the registration process, minimizing manual paperwork, reducing errors, and enhancing efficiency.

Accuracy and Reliability: Stress the importance of accurate and reliable data management within the system, ensuring students' academic records, course availability, and enrollment statuses are up-to-date and accessible.

Transparency and Accessibility: Discuss how the system enhances transparency by providing easy access to course information, schedules, prerequisites, and enrollment statuses for both students and faculty.

B. Importance of Data Structures and Algorithms in Registration Systems

1. Explanation of Data Structures:

Relevance: Expound on why the selection of appropriate data structures is crucial for efficient data storage and retrieval. Discuss how different data structures (like Arrays, Linked Lists, Trees, Graphs, etc.) are utilized to manage student data, course catalogs, and enrollment records.

Efficiency Impact: Explain how the choice of data structures directly affects the system's performance, memory utilization, and scalability. Highlight advantages and trade-offs associated with different data structures.

2. Role of Algorithms:

Functionality: Emphasize the fundamental role algorithms play in managing various aspects of the registration system, such as enrollment validation, waitlist management, course prerequisite checks, and system optimizations.

Efficiency and Scalability: Highlight how well-designed and optimized algorithms contribute to the system's responsiveness, scalability, and overall performance. Discuss the impact of algorithmic efficiency on system speed and resource utilization.

3. System Optimization Through Data Structures and Algorithms:

Performance Enhancement: Discuss how utilizing efficient data structures and algorithms can lead to faster processing times, reduced computational complexity, and improved user experiences.

Adaptability and Future-Proofing: Address how the system's adaptability to changes, such as increased user loads or modifications in course structures, relies on the robustness of the implemented data structures and algorithms.

This comprehensive introduction lays the groundwork for understanding the objectives, functionalities, and critical components of the College Registration System, emphasizing the crucial role of data structures and algorithms in ensuring its efficiency and effectiveness.

II. Data Structures in the Registration System

A. Student Data Management

Storage Mechanisms:

For managing student information, a combination of various data structures can be employed based on the specific needs of the system:

Arrays: Useful for fixed-size data where quick access to elements is essential (e.g., Student IDs).

Linked Lists: Suitable for dynamically managing student records with varying lengths, allowing easy insertion and deletion (e.g., Courses Enrolled).

Databases (such as SQL or NoSQL): Provide structured and efficient storage for comprehensive student information, ensuring data integrity, query capabilities, and scalability.

Data Fields:

Attributes stored for each student may include:

Name: First name, last name.

ID: Unique identifier for each student.

Courses Enrolled: List of courses the student is currently registered for.

Grades: Records of grades achieved in completed courses.

Contact Information: Email, phone number, etc.

B. Course Catalog Management

Data Representation:

Utilizing appropriate data structures like Trees or Graphs aids in managing the complexity of course prerequisites, offerings, and organization within the system:

Trees: Represent hierarchical relationships between courses and their prerequisites, allowing easy traversal to check prerequisites.

Graphs: Capture complex relationships between courses, facilitating the management of course offerings and elective paths.

Attributes:

Essential course data fields might encompass:

Course Code: Unique identifier for each course.

Title: Title or name of the course.

Credits: Number of credit hours associated with the course.

Prerequisites: Information detailing course prerequisites needed for enrollment.

C. Registration Records

Implementation Approach:

Data structures like Queues and Stacks can be employed to manage the flow of enrollment requests efficiently:

Queue: Used to handle incoming enrollment requests in a first-come, first-served manner.

Stack: May be utilized for handling approval or rejection processes based on last-in, first-out principles.

Attributes and Processing:

Defining the data fields for registration records and outlining the process flow ensures efficient management of registration requests:

Student ID: Identifier linking the student to the registration request.

Course Code: Identifies the course for which enrollment is requested.

Status: Indicates the current status of the enrollment request (pending, approved, waitlisted, rejected).

Timestamp: Records the time and date of the enrollment request.

Process Flow:

Submission of Enrollment Request: Student submits a request to enroll in a course.

Queue/Stack Handling: Requests are added to a queue/stack based on the system's logic.

Enrollment Validation: The system checks prerequisites, availability, and other criteria.

Status Update: The status of the request is updated based on validation results.

Enrollment Confirmation: Approved requests are processed, and students are enrolled in the respective courses.

This robust structuring of data and utilization of appropriate data structures enable efficient management of student information, course catalogs, and registration records within the College Registration System.

III. Algorithms in the Registration System

A. Student Enrollment Algorithm

Course Availability Check:

Algorithm Elaboration:

Input: Student's course selection.

Process:

The system checks the course capacity or available seats.

If the available seats are not full, the course is marked as available.

Output: Boolean value (True/False) indicating course availability.

Enrollment Validation:

Algorithm Explanation:

Input: Student's enrollment request with chosen courses.

Process:

For each course in the request:

Verify if the course is available (using the Course Availability Check algorithm).

Check if the student meets prerequisites for the chosen courses.

Validate the student's schedule against course timings and potential conflicts.

Output: Confirmation or rejection of the enrollment request based on validation results.

B. Waitlist Management Algorithm

Queue Processing:

Algorithm Details:

Input: Students in the waitlist queue.

Process:

Continuously monitor the waitlist queue.

As seats become available in courses due to withdrawals or additional slots, process students in the waitlist based on their queue position.

Enroll the next student in the queue if a seat becomes available.

Output: Automatic enrollment of waitlisted students based on availability.

Automated Enrollment from Waitlist:

Description of the Process:

Trigger: When a seat becomes available due to a dropped course or increased capacity.

System Action:

Automatically identifies the next student in the waitlist queue for the specific course.

Sends notification of enrollment to the student.

Updates the enrollment status for the student and the course.

C. Course Prerequisite Checking Algorithm

Prerequisite Verification:

Algorithm Explanation:

Input: Chosen course for enrollment.

Process:

Retrieve the prerequisites for the selected course from the course catalog.

Check if the student has completed the required prerequisites.

Output: Boolean value indicating whether the student meets the prerequisites for enrollment.

Error Handling:

Handling Missing Prerequisites:

Process:

If the student does not meet the prerequisites:

Generate an error message indicating missing prerequisites.

Offer alternative courses or suggest steps to fulfill prerequisites.

Provide guidance on how the student can fulfill missing requirements or seek permission to bypass prerequisites if applicable.

These algorithms ensure the efficient management of student enrollments, waitlist handling, and validation of prerequisites, while also incorporating error handling mechanisms for exceptional cases within the College Registration System.

Step-by-Step Process:

Sequential Registration Steps for Students:

Course Search: Students browse and select courses from the catalog based on their academic requirements.

Enrollment Request Submission: Students submit their preferred course selections through the system.

Enrollment Validation: The system validates course availability, prerequisites, and schedule conflicts.

Approval/Waitlist Placement: Approved courses are added to the student's schedule, while others are placed on waitlists if necessary.

Confirmation and Payment: Students receive confirmation of enrolled courses and make necessary payments if applicable.

Approval Mechanism:

Admin/Faculty Course Registration Approval Process:

Upon receiving student enrollment requests, admins/faculty:

Review enrollment requests based on availability, prerequisites, and other criteria.

Approve eligible enrollments or reject requests that don't meet requirements.

Update the status of enrollment requests, informing students of their course status.

ERD Design:

Entity-Relationship Diagram Illustration:

Illustrate the relationships and attributes of entities like:

Student: Attributes such as StudentID, Name, ContactInfo.

Course: Attributes like CourseID, Title, Credits, Prerequisites.

Registration: Attributes including StudentID, CourseID, Status, Timestamp.

Display connections between entities representing how they relate to each other (e.g., a student enrolls in multiple courses).

DBMS Selection:

Discussion on Database Management System Choice:

Considerations: Discuss factors such as data volume, scalability, security, and query requirements.

Selection Justification: Choose a DBMS (e.g., MySQL, PostgreSQL, MongoDB) based on the system's needs and justify the choice.

Scalability and Performance: Explain how the chosen DBMS meets scalability needs and provides optimal performance for handling concurrent user operations.

This comprehensive breakdown covers the vital system components including user interfaces, registration workflows, and database structures, ensuring a holistic understanding of the College Registration System's architecture and functionality.

PHASE 2

SYSTEM ANALYSIS

2.1 Requirement Collection and Analysis

I. System Overview

A. System Description

Provide an overview of the current College Registration System, its purpose, and functionalities.

Identify the stakeholders involved: students, faculty, administrators, etc.

B. Technology Stack

List the technologies, frameworks, programming languages, and tools used in the system.

II. Current System Evaluation

A. Functionalities Assessment

Strengths:

Highlight the system's successful functionalities and robust features.

Weaknesses:

Identify areas where the system lacks efficiency, usability, or functionality.

B. User Experience

Evaluate the user interface and experience for students, faculty, and administrators.

Analyze usability, navigation, and accessibility aspects.

C. Performance Analysis

Evaluate the system's performance metrics:

Response Time: How quickly the system responds to user requests.

Scalability: Ability to handle increased user loads.

Reliability: System uptime and error handling.

D. Data Management and Security

Data Integrity:

Assess the accuracy and consistency of stored data.

Security:

Evaluate measures for data protection, access controls, and encryption protocols.

III. Areas for Improvement

A. Usability Enhancements

Propose improvements to the user interface, navigation, or accessibility.

B. Performance Optimization

Suggestions for improving system responsiveness, reducing loading times, and enhancing scalability.

C. Data Management and Security Upgrades

Recommendations for enhancing data integrity, backups, and fortifying security measures against potential vulnerabilities.

IV. Stakeholder Feedback and User Surveys

A. Gather Feedback

Conduct surveys or interviews with stakeholders to gather their experiences and suggestions.

B. User Stories and Pain Points

Analyze user stories to understand their needs and identify pain points in the current system.

V. Future Roadmap and Recommendations

A. Future System Enhancements

Outline a roadmap for implementing suggested improvements.

Prioritize enhancements based on criticality and feasibility.

B. Technological Upgrades

Suggest updates to the technology stack or frameworks to enhance system capabilities.

C. Training and Support

Recommend training sessions or documentation updates to assist users in utilizing system improvements effectively.

VI. Conclusion

A. Summary

Summarize the system's current state, strengths, weaknesses, and potential for improvement.

B. Recommendations Summary

Recapitulate key recommendations and their significance for system enhancement.

This comprehensive system analysis aims to provide an in-depth evaluation of the current College Registration System, highlighting its successes, shortcomings, and areas for improvement to enhance functionality, user experience, and overall system efficiency.

PHASE 3

SYSTEM DESIGN

Designing a College Registration System involves planning the architecture, modules, and interactions to meet the system requirements effectively. Here's an outline for the system design:

I. System Architecture

A. High-Level Overview

Describe the system's high-level architecture, such as client-server, three-tier architecture, microservices, etc.

B. Components and Interactions

Identify major components/modules (e.g., user interface, backend logic, database) and their interactions.

Define communication protocols between different system components.

II. System Modules

A. User Interfaces

Detail interfaces for students, faculty, and administrators with their functionalities and design. Sketch wireframes or mockups for visual representation.

B. Backend Logic

Design the backend logic:

Enrollment Logic: Algorithms for enrollment validation, course availability checks, and waitlist management.

User Authentication and Authorization: Handling user login, role-based access control, and session management.

C. Database Structure

Design the database schema:

Entity Relationship Diagram (ERD): Illustrate relationships between student, course, registration, etc.

Define tables, fields, relationships, and constraints.

D. System Integration

Outline mechanisms for integrating external systems or APIs, if applicable, for additional functionalities or data exchange.

III. System Workflow

A. Student Registration Workflow

Define the step-by-step process a student follows to register for courses.

Illustrate the flow from course selection to enrollment confirmation.

B. Admin Approval Workflow

Detail the approval process for administrators/faculty:

How enrollment requests are reviewed.

Actions taken upon approval or rejection.

IV. Technology Stack

A. Backend Technologies

Choose programming languages, frameworks, and tools for backend development.

Determine the server architecture (e.g., Node.js, Django, Flask, etc.).

B. Database Management System (DBMS)

Select a suitable DBMS (e.g., MySQL, PostgreSQL, MongoDB) based on system requirements and scalability needs.

C. Frontend Technologies

Determine the technologies for building user interfaces (e.g., React.js, Angular, Vue.js).

V. Scalability and Security

A. Scalability Plan

Outline strategies for scaling the system to handle increased user loads and future growth.

B. Security Measures

Define security protocols, encryption standards, and measures to safeguard user data and system integrity.

VI. Error Handling and Logging

A. Error Handling

Design mechanisms to handle exceptions, errors, and edge cases in the system.

B. Logging and Monitoring

Define logging mechanisms for tracking system activities, user actions, and system performance.

VII. Testing Plan

A. Testing Strategies

Define testing methodologies (unit testing, integration testing, etc.) for each system component.

B. Quality Assurance (QA)

Plan QA processes to ensure system functionality, usability, and reliability.

VIII. Documentation and Training

A. System Documentation

Prepare comprehensive documentation covering system architecture, modules, functionalities, and usage guidelines.

B. User Training

Develop training materials or sessions to assist users in navigating and utilizing the system effectively.

IX. Deployment Plan

A. Deployment Strategy

Define the deployment environment (development, staging, production).

Plan deployment steps and procedures.

B. Continuous Improvement

Establish protocols for system updates, bug fixes, and incorporating user feedback for ongoing enhancements.

This system design outline forms a comprehensive framework for designing a College Registration System, encompassing various aspects from architecture to deployment, ensuring a robust and efficient system meeting user needs.

PHASE 4

CODING

I. Setting Up Development Environment

A. Development Tools

Choose a suitable C++ IDE (Integrated Development Environment) such as Visual Studio, Code Blocks, or CLion.

Set up the project and configure the build environment.

II. Design and Structure

A. Class Design

Identify essential classes (e.g., Student, Course, Registration System) based on the system design.

Create class diagrams or UML diagrams to visualize relationships and attributes.

B. Data Structures

Define appropriate data structures (arrays, linked lists, trees) to manage student information, course catalogs, and registration records.

III. Backend Development

A. User Authentication and Authorization

Implement authentication logic using user credentials and authorization checks for different user roles.

B. Database Handling

Simulate database functionality using file handling or implement a basic database structure in C++ (not a traditional database, but file-based storage).

C. Backend Logic Implementation

Implement backend logic for enrollment validation, course availability checks, waitlist management, etc., using C++ functions and algorithms.

Code algorithms for prerequisites verification, enrollment workflow, and error handling.

IV. Frontend Development

A. Console-Based User Interface

Develop a console-based UI for student interactions (course selection, enrollment, view schedule) and admin functionalities (course management, approval/rejection).

B. User Interactions

Implement user interaction flows using console input/output and menu-driven functionalities for different user roles.

V. Integration and Testing

A. Backend-Frontend Interaction

Integrate frontend interactions with backend functionalities, ensuring data retrieval and processing.

Perform unit testing to verify the correctness of functions and system components.

VI. Security Measures and Error Handling

A. Security Implementation

Apply basic security measures such as input validation to prevent vulnerabilities like buffer overflows.

Handle user authentication securely to protect user data.

B. Error Handling

Implement error handling mechanisms to manage exceptions and edge cases gracefully.

VII. Documentation

A. Code Documentation

Document code functions, classes, and significant functionalities for future reference.

VIII. Deployment (Simulated)

A. Local Deployment

Deploy the system locally for testing purposes, considering it might not involve actual server deployment.

IX. Continuous Improvement

A. Refactoring and Enhancement

Refactor code for efficiency, readability, and maintainability.

Plan for iterative updates and enhancements based on user feedback and system performance.

X. Version Control and Collaboration

A. Version Control

Utilize version control (Git, SVN) to manage code changes, collaborate, and maintain code versions.

This outline provides a structured approach to building a simplified version of the College Registration System using C++, focusing on implementing core functionalities, simulating database handling, console-based user interaction, security, and ongoing improvement aspects.

SOURCE CODE:

```
--***** CLASS AGE *****
```

```
#INCLUDE<IOSTREAM>
USING NAMESPACE STD;
CLASS AGE
{
    PRIVATE:
        INT DAY;
```

```

    INT MONTH;
    INT YEAR;
PUBLIC:
    AGE();
    VOID SETDATE();
    VOID SETDAY(INT);
    VOID SETMONTH(INT);
    VOID SETYEAR(INT);
    VOID PRINTDATE();
    VOID FPRINTDATE();
    INT GETDAY();
    INT GETMONTH();
    INT GETYEAR();
    VOID OPERATOR -=(AGE 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;
    }
};
AGE::AGE(){
    DAY=0;
    MONTH=0;
    YEAR=0;
}
VOID AGE::SETDATE()
{
    INT D,Y,M;
    COUT<<"ENTER DAY: ";

```

```

CIN>>D;
COUT<<"ENTER MONTH: ";
CIN>>M;
COUT<<"ENTER DAY: ";
CIN>>Y;
THIS->SETDAY(D);
THIS->SETMONTH(M);
THIS->SETYEAR(Y);
}
VOID AGE::SETDAY(INT D)
{
    DAY=((D>=1&&D<=31)?D:1);
}
VOID AGE::SETMONTH(INT M)
{
    MONTH=((M>=1&&M<=12)?M:1);
}
VOID AGE::SETYEAR(INT Y)
{
    YEAR=((Y>=1990&&Y<=2023)?Y:1990);
}
VOID AGE::PRINTDATE()
{
    COUT<<DAY<<"-"<<MONTH<<"-"<<YEAR<<ENDL;
}
VOID AGE::FPRINTDATE()
{
    COUT<<YEAR<<"-"<<MONTH<<"-"<<DAY<<ENDL;
}
INT AGE::GETDAY()
{
    RETURN DAY;
}
INT AGE::GETMONTH()

```



```

{
    RETURN MONTH;
}
INT AGE::GETYEAR()
{
    RETURN YEAR;
}

```

--*** CLASS MODULE *******

```

#include <Iostream>
#include <fstream>
#include <cstring>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
char modulesrecord[] = "MODRECORD.TXT";
class module{    ///TWO
public:
    //*****1ST FUNCTION IS STARTED*****
    int displayanymoduleinfo()
    {
        int moduleno;
        cout<<"\n# FIND MODULE DETAILS #\n";
        cout<<"\nENTER MODULE NO :";
        cin>>moduleno;
        int flag = -1;
        ifstream file(modulesrecord);
        string str;
        while (getline(file, str))
        {
            if(str.length()>0){
                char temp[1000];
                memset(temp,1000,sizeof(temp));
            }
        }
    }
}

```

```

STRCPY(TEMP, STR.C_STR());
CHAR * PTR;
PTR = STRTOK(TEMP, "|");
INT TEMPNO = ATOI(PTR);
COUT<<"\\NMODULE NO IS : "<<TEMP;
IF(MODULENO == TEMPNO )
{
COUT<<"\\N"<<STR;
FLAG = 1;
BREAK;
}
}
}
IF(!FLAG)
COUT<<"\\NNO MODULE FOUND : "<<MODULENO;
FILE.CLOSE();
GETCHAR();GETCHAR();
}
//*****2ND FUNCTION IS STARTED*****

INT DISPLAYALLMODULESINFO()
{
COUT<<"\\N# LIST OF ALL MODULES #\\N";
//READ ALL MODULES FROM RECORD
IFSTREAM FILE(MODULESRECORD);
STRING STR;
COUT<<"\\NMODULE NO | TITLE | LEVEL";
WHILE (GETLINE(FILE, STR))
{
IF(STR.LENGTH()>0)
COUT<<"\\N"<<STR;
}
FILE.CLOSE();
GETCHAR();GETCHAR();
}

```

```

//ADD NEW MODULE TO DB
INT ADDMODULE(CHAR *MODULERECORD)
{
    INT MODULENO;
    CHAR TITLE[50];
    CHAR LEVEL[50];
    OFSTREAM OUT(MODULERECORD, IOS_BASE::APP);
    COUT<<"\N# ADD NEW MODULE #";
    COUT<<"\NENTER MODULE NO : ";
    CIN>>MODULENO;
    OUT<<ENDL<<MODULENO<<" | ";
    COUT<<"\NENTER TITLE OF MODULE (50 CHAR) :";
    CIN>>TITLE;
    OUT<<TITLE<<" | ";
    COUT<<"\NENTER LEVEL UNDERGRAD/GRAD.:";
    CIN>>LEVEL;
    OUT<<LEVEL<<" | ";
    COUT<<"\NMODULE ADDED !!";
    GETCHAR();
    OUT.CLOSE();
}
};

```

--***** CLASS QUEUE *****

```

#include<Iostream>
using namespace std;
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 ISEMPY();
    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;
}

```

--*** CLASS STUDENT *******

```

#include <Iostream>
#include <fstream>
#include <cstring>
#include <stdio.h>
#include <stdlib.h>
#include "time.cpp"
#include "age.cpp"
#include "queue.cpp"

```

```

USING NAMESPACE STD;
CHAR STDSREC[] = "STDTRECORD.TXT";
CHAR TEMPFILE[] = "TEMPFILE.TXT";
CLASS STD
{
    //ONE
PUBLIC:
//ADD ONE STD'S RECORD TO DB
//*****1ST FUNCTION IS STARTED*****
INT ADDSTDRECORD(CHAR *STDSREC)
{
    INT STDNO;
    CHAR NAME[50];
    AGE D,D1;
    TIME T;
    OFSTREAM OUT(STDSREC, IOS_BASE::APP);
    COUT<<"\NENTER STD NO : ";
    CIN>>STDNO;
    OUT<<ENDL<<STDNO<<" | ";
    COUT<<"\NENTER NAME (50 CHAR):";
    CIN>>NAME;
    OUT<<NAME<<" | ";
    COUT<<"\NENTER DATE OF BIRTH:\N";
    D.SETDATE();
    COUT<<"\NENTER DATE DATE:\N";
    D1.SETDATE();
    D-=D1;
    OUT<<D.GETYEAR()<<"-"<<D.GETMONTH()<<"-
"<<D.GETDAY()<<" | ";
    COUT<<"ENTER CURRENT TIME OF REGISTRATION:\N";
    T.SETTIME();
    OUT<<T.GETHOURS()<<":"<<T.GETMINUTES()<<":"<<T.GETSECO
NDS()<<"|";
    INT FLAG=-1;
    COUT<<"\N0 - UNDER GRAD.,1 - POSTGRAD";
    COUT<<"\NENTER :";
    CIN>>FLAG;
    INT MODULECOUNT =-1;
    IF(FLAG==0)

```

```

MODULECOUNT = 6;
ELSE IF(FLAG ==1)
MODULECOUNT = 4;
ELSE{
COUT<<"\\NERR: PL. ENTER EITHER oOR1 ";
RETURN -1;
}
QUEUE<INT> MOD(MODULECOUNT);
FOR(INT I=0; I<MODULECOUNT; I++){
INT MODULE;
COUT<<"\\NENTER MODULE NO"<<I<<" : ";
CIN>>MODULE;
MOD.ENQUEUE(MODULE);
//OUT<<MODULE<<" | ";
OUT<<MOD.DEQUEUE()<<" | ";
}
CHAR CHOICE;
COUT<<"\\NADD MARKS AGAINST EACH MODULE? Y/N :";
CIN>>CHOICE;
//SEPARATOR TO FIND START OF MARKS
OUT<<" = |";

IF(CHOICE=='Y')
{
FOR(INT I=0; I<MODULECOUNT; I++){
DOUBLE MARKS;
COUT<<"\\NENTER MARKS OF MODULE NO"<<I<<" : ";
CIN>>MARKS;
OUT<<MARKS<<" | ";
}
}
OUT.CLOSE();
}
//*****2ND FUNCTION IS STARTED*****
INT DISPLAYSTDRECORD()
{
INT STDNO;
//DISPLAY IF STD RECORD EXIST

```

```

COUT<<"\N# FIND ANY MODULE DETAILS #\N";
COUT<<"\NENTER STD NO TO BE SEARCHED: ";
CIN>>STDNO;
INT FLAG = -1;
IFSTREAM FILE(STDSDREC);
STRING STR;
WHILE (GETLINE(FILE, STR))
{
    IF(STR.LENGTH()>0){
        CHAR TEMP[1000];
        MEMSET(TEMP,1000,SIZEOF(TEMP));
        STRCPY(TEMP, STR.C_STR());
        CHAR * PTR;
        PTR = STRTOK(TEMP, "|");
        INT TEMPNO = ATOI(PTR);
        IF(STDNO == TEMPNO )
        {
            COUT<<"\N"<<STR;
            FLAG = 1;
            BREAK;
        }
    }
}
IF(FLAG==1)
COUT<<"\NNO STD FOUND WITH NO :"<<STDNO;
FILE.CLOSE();
GETCHAR();GETCHAR();
}
//*****2ND FUNCTION IS COMPLETED*****
//DISPLAY RECORD OF ALL STDS
//*****3RD FUNCTION IS STARTED*****
INT DISPLAYALLSTDRECORD()
{
    IFSTREAM FILE(STDSDREC);
    STRING STR;

    COUT<<"\N# LIST OF ALL STD #";

```



```

IF(!FILE) {
COUT<<"\NNO RECORD FOUND !!";
RETURN -1;
}

WHILE (GETLINE(FILE, STR))
{
IF(STR.LENGTH()>0)
COUT<<"\N\t"<<STR;
}
FILE.CLOSE();
GETCHAR();GETCHAR();
}
//*****3RD FUNCTION IS COMPLETED*****

//*****4TH FUNCTION IS STARTED*****
INT FINDSTDMARKS()
{
COUT<<"\N# FIND MARKS OF ANY STD #\N";
INT STDNO;

COUT<<"\NENTER STD NO : ";
CIN>>STDNO;

INT FLAG = -1;
IFSTREAM FILE(STDsREC);
STRING STR;

WHILE (GETLINE(FILE, STR))
{
IF(STR.LENGTH()>0){

CHAR TEMP[1000];
MEMSET(TEMP,1000, sizeof(TEMP));
STRCPY(TEMP, STR.C_STR());

CHAR * PTR;

```

```

PTR = STRTOK(TEMP, "|");

INT TEMPNO = ATOI(PTR);

IF(STDNO == TEMPNO )
{
    MEMSET(TEMP,1000,SIZEOF(TEMP));
    STRCPY(TEMP, STR.C_STR());
    CHAR *PTR1 = STRTOK(TEMP, "=");
    PTR1 = STRTOK(NULL, "=");
    //COUT<<"\N====> "<<PTR1;
    PTR1 = STRTOK(PTR1, "|");
    DOUBLE SUM = ATOF(PTR1);
    DOUBLE MINNO = SUM;
    //COUT<<"\NMIN NO IS : "<< MINNO;
    INT COUNTSUB = 0;
    WHILE(PTR1 = STRTOK(NULL, "|")){
        //COUT<<"-> "<<PTR1;
        IF(ATOF(PTR1)<MINNO)
            MINNO = ATOF(PTR1);

        SUM += ATOF(PTR1);
        COUNTSUB++;
    }

    //COUT<<"\NSUM IS : "<<SUM;
    //COUT<<"\NCOUNT SUB : "<<COUNTSUB;
    //COUT<<"\NMIN NO IS : "<<MINNO;

    IF(COUNTSUB == 6){
        SUM -= MINNO;
        COUT<<"\NSUM IS : "<<SUM;
        SUM = SUM / 5;
        COUT<<"\NAVG SUM IS TOP 5 SUB: "<<SUM;

        IF(SUM>70) COUT<<"\NGRADE : A";
        ELSE IF(SUM>=60 && SUM<=69)
            COUT<<"\NGRADE IS : B+";
    }
}

```

```

ELSE IF(SUM >=50 && SUM<=59)
COUT<<"\NGRADE IS : B";
ELSE IF(SUM>=41 && SUM<=49)
COUT<<"\NGRADE IS : C";
ELSE IF(SUM>=40)
COUT<<"\NGRADE IS : D";
ELSE IF(SUM<40)
COUT<<"\NGRADE IS: F";

}
ELSE{
SUM = SUM / 4;
COUT<<"\NAVG SUM OF 4 SUB : "<< SUM;

IF(SUM>= 76) COUT<<"\NDISTINCTION";
ELSE IF(SUM>=61 && SUM<=75 )
COUT<<"\NMERIT";
ELSE IF(SUM>=50 && SUM<=60)
COUT<<"\NPASS";
ELSE IF(SUM<50) COUT<<"\NFAIL ";
}

FLAG = 1;
BREAK;
}

}

}

IF(FLAG== -1)
COUT<<"\NNO STD FOUND WITH NO : "<<STDNO;

FILE.CLOSE();
GETCHAR();GETCHAR();
}

//*****4TH FUNCTION IS COMPLETED*****

```

```
};
```

```
--***** CLASS TIME *****
```

```
#INCLUDE<Iostream>
using namespace std;
class TIME
```

```
{
    PRIVATE:
        INT HOURS;
        INT MINUTES;
        INT SECONDS;
    PUBLIC:
        TIME();
        VOID SETTIME();
        VOID SETHOURS(INT);
        VOID SETMINUTES(INT);
        VOID SETSECONDS(INT);
        VOID PRINTTIME();
        INT GETHOURS();
        INT GETMINUTES();
        INT GETSECONDS();
};
```

```
};
TIME::TIME(){

    HOURS=0;
    MINUTES=0;
    SECONDS=0;
}
VOID TIME::SETTIME()
{
    INT H,M,S;
    COUT<<"ENTER HOURS:";
    CIN>>H;
    COUT<<"ENTER MINUTES:";
    CIN>>M;
```

```

    COUT<<"ENTER SECONDS:";
    CIN>>S;
    THIS->SETHOURS(H);
    THIS->SETMINUTES(M);
    THIS->SETSECONDS(S);
}
VOID TIME::SETHOURS(INT H)
{
    HOURS=((H>=1&&H<=24)?H:1);
}
VOID TIME::SETMINUTES(INT M)
{
    MINUTES=((M>=1&&M<=60)?M:1);
}
VOID TIME::SETSECONDS(INT S)
{
    SECONDS=((S>=1&&S<=60)?S:1);
}
VOID TIME::PRINTTIME()
{
    COUT<<HOURS<<":"<<MINUTES<<":"<<SECONDS<<ENDL;
}
INT TIME::GETHOURS()
{
    RETURN HOURS;
}
INT TIME::GETMINUTES()
{
    RETURN MINUTES;
}
INT TIME::GETSECONDS()
{
    RETURN SECONDS;
}

```

--***** CLASS UPDATES *****

```

#include <Iostream>
#include <Fstream>
#include <Cstring>
#include<stdio.h>
#include<stdlib.h>
// #include "STUDENT.CPP"
// #include "MODULE.CPP"
using namespace std;
class UPDATES
{
    public:
        //*****1ST FUNCTION IS STARTED*****
        int UPDATEMODULESINFO()
        {
            cout<<"\n# UPDATE MODULE INFO #";
            int MODULENO;
            cout<<"\nENTER MODULE NO TO BE MODIFIED:";
            cin>>MODULENO;
        }
        //*****2ND FUNCTION IS STARTED*****

        int UPDATEMODULE()
        {
            cout<<"\n# UPDATE MODULE RECORD #\n";
            int MODULENO;
            cout<<"\nENTER MODULE NO : ";
            cin>>MODULENO;
            int FLAG = -1;
            ifstream FILE(MODULESRECORD);
            string STR;
            int MODULEFOUND = 0;
            while (getline(FILE, STR))
            {
                if(STR.length()>0){
                    char TEMP[1000];
                    memset(TEMP,1000,sizeof(TEMP));
                    strcpy(TEMP, STR.c_str());
                    char * PTR;

```

```

PTR = STRTOK(TEMP, "|");
INT TEMPNO = ATOI(PTR);
IF(MODULENO == TEMPNO )
{
    MODULE M;
M.ADDMODULE(TEMPFILE);
MODULEFOUND = 1;
}
ELSE{
OFSTREAM FILE(TEMPFILE, IOS_BASE::APP);
FILE<<ENDL<<STR;
FILE.CLOSE();
}
}
}
FILE.CLOSE();
IF(MODULEFOUND){
REMOVE(MODULESRECORD);
RENAME(TEMPFILE, MODULESRECORD);
}
}
//*****3RD FUNCTION IS STARTED*****
INT UPDATESTDRECORD()
{
INT STDNO;
COUT<<"\NENTER STD NUMBER :";
CIN>>STDNO;
//MODIFY IF IT EXIST E
}
//*****4TH FUNCTION IS STARTED*****
INT UPDATESTD()
{
COUT<<"\N# UPDATE STD RECORD #\N";
INT STDNO;

COUT<<"\NENTER STD NO : ";
CIN>>STDNO;

```

```

INT FLAG = -1;
IFSTREAM FILE(STDSDREC);
STRING STR;
INT STDFOUND = 0;
WHILE (GETLINE(FILE, STR))
{
    IF(STR.LENGTH()>0){

        CHAR TEMP[1000];
        MEMSET(TEMP,1000,SIZEOF(TEMP));
        STRCPY(TEMP, STR.C_STR());

        CHAR * PTR;
        PTR = STRTOK(TEMP, "|");

        INT TEMPNO = ATOI(PTR);

        IF(STDNO == TEMPNO )
        {
            STD S;
            S.ADDSTDRECORD(TEMPFILE);
            STDFOUND = 1;
        }
        ELSE{
            OFSTREAM FILE(TEMPFILE, IOS_BASE::APP);
            FILE<<ENDL<<STR;
            FILE.CLOSE();

        }
    }
}

FILE.CLOSE();

IF(STDFOUND){
    REMOVE(STDSDREC);
    RENAME(TEMPFILE, STDSDREC);
}

```



```
}  
};
```

```
--***** CLASS COLLEGE *****
```

```
#INCLUDE <Iostream>  
#INCLUDE <fstream>  
#INCLUDE <cstring>  
#INCLUDE <stdio.h>  
#INCLUDE <stdlib.h>  
#INCLUDE "STUDENT.CPP"  
#INCLUDE "MODULE.CPP"  
#INCLUDE "UPDATES.CPP"  
using namespace std;  
int main(int argc, char** argv) {  
    std::stdobj;  
    module modobject;  
    updates up;  
    int choice = -1;  
    while(choice != 0){  
        system("CLS");  
        cout<<"\n# COLLEGE REGISTRATION SYSTEM #";  
        cout<<"\n 1> ADD RECORD";  
        cout<<"\n 2> UPDATE RECORD";  
        cout<<"\n 3> GET ANY RECORD";  
        cout<<"\n 4> GET ALL RECORD";  
        cout<<"\n 5> ADD MODULE";  
        cout<<"\n 6> GET ANY MODULE DETAILS";  
        cout<<"\n 7> GET ALL MODULES DETAILS";  
        cout<<"\n 8> UPDATE MODULE";  
        cout<<"\n 9> FIND STD MARKS";  
        cout<<"\n 0> EXIT..";  
        cout<<"\n ENTER YOUR CHOICE : ";  
        cin>>choice;  
        switch(choice){  
            case 1:
```

```

STDOBJ.ADDSTDRECORD(STDSREC);
BREAK;
CASE 2:
UP.UPDATESTD();
BREAK;
CASE 3:
STDOBJ.DISPLAYSTDRECORD();
BREAK;
CASE 4:
STDOBJ.DISPLAYALLSTDRECORD();
BREAK;
CASE 5:
MODOBJECT.ADDMODULE(MODULESRECORD);
BREAK;
CASE 6:
MODOBJECT.DISPLAYANYMODULEINFO();
BREAK;
CASE 7:
MODOBJECT.DISPLAYALLMODULESINFO();
BREAK;
CASE 8:
UP.UPDATEMODULE();
BREAK;
CASE 9:
STDOBJ.FINDSTDMARKS();
BREAK;
CASE 0:
COUT<<"\NTHANKYOU !! GOOD BYE\N ";
BREAK;
DEFAULT:
COUT<<"\NINVALID CHOICE..";
GETCHAR();
}
}
RETURN 0;
}

```

PHASE 5

TESTING

During the testing phase of the College Registration System, various tests are conducted to ensure that the system functions as expected, meets requirements, and delivers a satisfactory user experience. Here's an outline for the testing phase:

I. Test Planning

A. Test Strategy Development

Define the overall approach for testing (e.g., functional testing, integration testing, user acceptance testing).

Determine testing objectives and goals.

B. Test Scope and Coverage

Identify the scope of testing, including functionalities, modules, and user interactions to be covered.

Ensure comprehensive coverage of critical system components.

II. Types of Testing

A. Unit Testing

Test individual components and functions in isolation to validate their functionality.

Ensure that each unit operates correctly as per design.

B. Integration Testing

Verify interactions between different modules or components to ensure they work together seamlessly.

Test the integration of backend and frontend components.

C. System Testing

Perform end-to-end testing of the entire system to validate overall system behavior and functionalities.

Test system-wide scenarios such as enrollment, course selection, and registration workflows.

D. User Acceptance Testing (UAT)

Involve users (students, faculty, administrators) to evaluate the system's usability and validate if it meets their requirements.

Gather feedback from real users to assess user satisfaction and system functionality.

III. Testing Process

A. Test Case Preparation

Create detailed test cases for each testing phase, covering various scenarios and expected outcomes.

Include positive and negative test scenarios to validate different use cases.

B. Test Execution

Execute test cases according to the defined test plan.

Record test results and any observed deviations or issues.

IV. Performance and Security Testing

A. Performance Testing

Evaluate system performance under various conditions, such as load testing to assess response times and scalability.

Measure system stability and resource utilization under different user loads.

B. Security Testing

Conduct security assessments to identify vulnerabilities and weaknesses in the system.

Perform penetration testing to ensure robustness against potential security threats.

V. Regression Testing

A. Regression Test Suites

Re-run previously executed tests to ensure that new updates or fixes haven't caused any regression or unintended changes in existing functionalities.

B. Continuous Testing

Incorporate automated testing tools and frameworks for continuous integration and continuous testing (CI/CT) to facilitate ongoing testing during development phases.

VI. Acceptance and Sign-Off

A. User Acceptance Testing

Involve end-users in the final acceptance testing phase.

Obtain user feedback and approval on the system's readiness for deployment.

B. Sign-Off

Obtain sign-off from stakeholders, confirming that the system meets the specified requirements and is ready for deployment.

VII. Documentation

A. Test Reports

Prepare test reports summarizing the testing process, findings, identified issues, and resolution status.

Document any pending issues or risks that need post-deployment attention.

VIII. Retrospective and Continuous Improvement

A. Post-Testing Analysis

Conduct a post-testing analysis to identify lessons learned, areas for improvement, and best practices.

B. Iterative Improvement

Plan for iterative improvements based on test outcomes and user feedback for future system enhancements.

This testing phase outline aims to ensure comprehensive testing coverage, validation of system functionalities, security assessments, and user acceptance before the College Registration System is deemed ready for deployment.

PHASE 6

IMPLEMENTATION

During the implementation phase of the College Registration System, the focus is on deploying the system into the production environment following the completion of development, testing, and verification processes. Here's an outline for the implementation phase:

I. Deployment Preparation

A. Final System Verification

Ensure that all development and testing phases are completed successfully.
Verify that the system meets all specified requirements and quality standards.

B. Release Planning

Plan the deployment schedule and define a deployment strategy.
Determine a rollback plan in case of unexpected issues during deployment.

II. Deployment Process

A. Pre-Deployment Tasks

Prepare the production environment and servers for deployment.
Backup necessary data and configurations before making changes.

B. Configuration Management

Configure the production environment settings, including database connections, server configurations, and security settings.

C. Deployment Execution

Deploy the College Registration System to the production environment following the deployment plan.
Monitor deployment progress and ensure smooth execution.

III. Post-Deployment Activities

A. Verification and Smoke Testing

Perform initial verification tests after deployment to ensure that the system is operational and accessible.
Conduct smoke tests to check critical functionalities.

B. System Monitoring and Performance Tuning

Monitor the system's performance, resource utilization, and user activities in the production environment.
Optimize system performance and address any performance bottlenecks if identified.

IV. User Acceptance and Training

A. User Acceptance Testing (UAT)

Involve users (students, faculty, administrators) in user acceptance testing in the production environment.
Gather feedback and address any user-reported issues or concerns.

B. User Training and Support

Provide training sessions or materials to users on how to use the system effectively.
Offer ongoing support and assistance to users encountering difficulties.

V. Transition to Production

A. Transition Planning

Plan the transition from the development phase to full production usage.
Update documentation and notify stakeholders about the system's availability.

B. System Handover

Hand over the system to the operations team or designated administrators responsible for managing and maintaining the production environment.

VI. Post-Implementation Review

A. Post-Implementation Review

Conduct a post-implementation review meeting to evaluate the deployment process, successes, challenges, and lessons learned.
Document feedback for future improvements and reference.

B. Continuous Improvement Plan

Develop strategies for continuous monitoring, maintenance, and improvement of the system in the production environment.

VII. Communication and Documentation

A. Communication

Communicate the successful deployment and availability of the system to stakeholders, users, and support teams.
Provide contact information for support queries and issues.

B. Documentation

Update system documentation, including deployment procedures, configurations, and post-implementation reports.
The implementation phase outlines the steps for transitioning the College Registration System from development to production, ensuring a smooth deployment process, user acceptance, and readiness for operational use.

PROJECT CONCLUSION

CONCLUSION

In conclusion, the development and implementation of the College Registration System represent a comprehensive process involving planning, design, coding, testing, and deployment.

Achievements and Summary

Design and Planning: Detailed system design, architecture, and requirement analysis laid the foundation for system development.

Coding and Development: Implementation of core functionalities, backend logic, frontend interfaces, and security measures was executed meticulously in accordance with the system design.

Testing Phase: Rigorous testing, including unit testing, integration testing, system testing, and user acceptance testing, was conducted to ensure the system's functionality, security, and performance.

Implementation: A well-planned deployment strategy ensured a seamless transition from the development environment to the production environment, with careful consideration for user acceptance and training.

Key Takeaways

User-Centric Approach: The system design and functionalities were developed considering the specific needs and usability for students, faculty, and administrators.

Continuous Improvement: Emphasis was placed on post-implementation reviews, feedback collection, and strategies for ongoing enhancements and maintenance, ensuring the system's adaptability and responsiveness to changing needs.

Future Considerations

Scalability and Innovation: Continued focus on scalability to accommodate future growth and the integration of innovative technologies to enhance system capabilities.

User Feedback Integration: Encouragement of user engagement for continuous improvement, gathering feedback, and addressing evolving requirements.

Final Thoughts

The development and implementation of the College Registration System represent a significant milestone, offering a robust and efficient platform for managing course registrations, student information, and administrative processes within the educational institution. The structured approach taken in each phase aimed to deliver a reliable and user-centric system, ultimately contributing to an improved academic experience for all stakeholders involved.

Appendix

1. Technical Diagrams and Models

Figure 1: System Architecture Diagram

Figure 2: UML Class Diagrams

Figure 3: Entity-Relationship Diagram (ERD) for Database Structure

2. Code Snippets and Sample Scripts

Excerpt of Enrollment Validation Algorithm (C++ code)

Sample Script for User Authentication Logic (Python)

3. Testing Documentation

Detailed Test Plan for System Testing

Test Cases and Scenarios for User Acceptance Testing

4. User Guides and Documentation

User Manual for Students: "Navigating the Student Portal"

API Documentation for Backend Services

5. Screenshots and Wireframes

Screenshot Gallery of User Interfaces (Student Portal, Admin Dashboard)

Wireframes Used During System Design

6. Security and Compliance Documents

System Security Protocols and Measures Document

Compliance Certificates for Data Protection Regulations

7. System Performance Reports

Performance Testing Results Summary

Load Testing Reports and Metrics

8. Feedback and Survey Data

Summary of User Feedback from User Acceptance Testing

Survey Results on System Usability and Functionality

9. Deployment Documentation

Deployment Procedures for Production Environment Setup

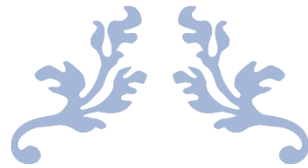
Configuration Settings for Server Deployments

10. References and Citations

List of References to External Sources and Materials Used

11. Glossary of Terms

Definitions of Technical Terms and Acronyms



ASSIGNMENTS



ASSIGNMENT 1

STATEMENT:

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 different 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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","SEPTEMBER","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->SETPMONTH(M);
    THIS->SETPYEAR(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::SETPMONTH(INT M){
    MONTH=((M>=1&&M<=12)?M:1);
}
VOID DATE::SETPYEAR(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();
}

```

ASSIGNMENT 2

STATEMENT:

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.

SOLUTION:

--***** SOURCE CODE *****

```
#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","SEPTEMBER","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 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();
}
```

ASSIGNMENT 3

STATEMENT:

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.

SOLUTION:

```
--***** SOURCE CODE *****

#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<<"COURSE CODE \T\T SUBJECT \T\T MARKS\n";
    COUT<<"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();
}

```

ASSIGNMENT 4

STATEMENT:

Design a C++ program to implement a doubly circular linked list. Define 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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.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;
}
```

ASSIGNMENT 5

STATEMENT:

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 enqueueing and dequeuing elements of various data types.

SOLUTION:

--***** SOURCE CODE *****

```
#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;
}

```

ASSIGNMENT 6

STATEMENT:

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 enqueueing and dequeuing elements of various data types

SOLUTION:

--***** SOURCE CODE *****

```
#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 ENQUEUEQ(T);
        T DEQUEUE();
        T FRONTE();
        BOOL ISEMPY();
};
TEMPLATE<CLASS T>
QUEUE<T>::QUEUE()
{

```



```

    FRONT=NULL;
    RARE=NULL;
}
TEMPLATE<CLASS T>
INT QUEUE<T>::ENQUEUEQ(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.ISEMPY()<<ENDL;
    COUT<<"ENQUE DATA = "<<Q.ENQUEUQ(1)<<ENDL;
    COUT<<"EMPTY = "<<Q.ISEMPY();
    COUT<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"ENQUEUUE DATA = "<<Q.ENQUEUQ(2)<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"EMPTY = "<<Q.ISEMPY();
    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.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
    COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
    COUT<<"EMPTY = "<<Q.ISEMPY();
    COUT<<"\N\N*****\N*****";
    **"\N\N"<<ENDL;
    QUEUE<STRING> Q1;
    COUT<<"EMPTY = "<<Q1.ISEMPY()<<ENDL;

```

```

COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21218")<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();
COUT<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"ENQUEUEUE DATA = "<<Q1.ENQUEUEQ("FARHAN
AHMAD")<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();
COUT<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21219")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21220")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21221")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21222")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21223")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21224")<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();

}

```

ASSIGNMENT 7

STATEMENT:

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.

SOLUTION:

--***** SOURCE CODE *****

```
#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());
    }
}
//DEFINITION OF THE MEMBER FUNCTION
VOID INORDER(NODE *R)
{
    IF(R!=NULL)
    {
        INORDER(R->GET_LEFT());
        R->SHOW_DATA();
        INORDER(R->GET_RIGHT());
    }
}
//DEFINITION 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", "BACKBON
E", "BACKSIDE", "BAD", "BECOMING", "CANDY", "CHARTER", "CHEESY
", "CHORUS", " DUNE ", " DAZZLING ", " DOMAIN ", " DIGNITY ",
EXCITE ", " ERADICATE ", " EVIL ", " ETHICAL ", " FAITH ", " FIRM ",
FORSAKE ", " FOREIGN ", " GALVANIZED ", "
GEARED", "GARBAGE", "GAY", "HALL", "HANDSOME", "HAPPILY", "HA
RD", "IDLER", "IF", "ILLIBERAL", "IMMEDIATE", "JEALOUS", "JOY", "JE
RKWEED", "KNOWINGLY", "LACKING", "LAST", "LEADING", "LEGITIM
ATE", "MADNESS", "MAGICIAN", "MATERIAL", "MANMADE", "NAKED"
, "NAMELESS", "NAPKIN", "NECESSARY", "OBDURATE", "OBJECT", "OB
LIGATORY", "OBLIQUE", "PAINTING", "PARTICULAR", "PASSABLE", "P
ATTERN", "QUITE", "REASONABLE", "REFRAIN", "RELIABLE", "RELIGI
OUS", "REMAINDER", "REMARK", "REMINISCENCE", "SACRITY", "SAT
ISFIED", "SCARCITY", "SCRUMPTIOUS", "SECOND", "SELECT", "SELEC
TION", "SIGNAL", "TAILORED", "TEMPER", "TERROR", "TESTAMENT",
"ULTIMATE", "UNCOMMON", "UNCOOKED", "UNDENIABLE", "VACAN
CY", "VAGUE", "VAIN", "VALUELESS", "WARRANTY", "WELL-
TIMED", "WINERY", "WELL-
MANNERED", "UNHURT", "UNFORTUNATE", "UNLAWFUL", "UNMAR
RIED", "TRUSTWORTHY", "TWISTER", "TWOsome", "ZENITH"};
    STRING
B[100]={ "APPROXIMATE", "SUMMARY", "NEARLY", "LIVELY", "SPINE
", "BEHIND", "POOR", "FITTING", "SWEET", "CONSTITUTION", "CORN
Y", "REFRAIN", " MOUND ", " BLURRING ", " FIELD ", " GRACE ",
ACTIVATE ", " REMOVE ", " VICE ", " PROPER ", " BELIEF ", " STAUNCH
", " DESERT ", " ALIEN ", " AROUSE ", " READY
", "RUBBISH", "HOMOSEXUAL", "CORRIDOR", "GOOD-
LOOKING", "FORTUNATELY", "TOUGH", "LOAFER", "WHETHER", "INT
OLERANT", "INSTANT", "ENVIOUS", "DETESTABLE", "DELIGHT", "DEL
IBERATELY", "MISSING", "FINAL", "MAIN", "VALID", "INSANITY", "CON

```



```

JUROR","FABRIC","ARTIFICIAL","BARE","ANONYMOUS","SERVIETT
E","ESSENTIAL","STUBBORN","THING","COMPULSORY","INDIRECT
","PORTRAY","SPECIFIC","SATISFACTORY","SAMPLE","FAIR","FAIR
","CHORUS","DEPENDABLE","DEVOUT","REST","COMMENT","MEMO
RY","VESTRY","CONVINCED","SHORTAGE","DELICIOUS","MOME
NT","CHOOSE","CHOICE","SIGN","TAILOR-
MADE","MOOD","TERRORISM","TESTIMONY","FINAL","UNUSUAL",
"RAW","INDISPUTABLE","EMPTINESS","INDISTINCT","USELESS","
WORTHLESS","GUARANTEE","TIMELT","VINEYARD","POLITE","UN
HARMED","UNLUCKY","ILLEGAL","SINGLE","RELIABLE","TORNAD
O","PAIR","PEAK",};
    NODE *ROOT=NEW NODE("NULL","NULL");
//    NODE<INT> *ROOT=NEW NODE<INT>(o,o);
    ROOT->SET_DATA(A[o],B[o]);
//    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);
}

```

ASSIGNMENT 8

STATEMENT:

A case where circularly linked list comes in handy is the solution of the Josephus Problem. Consider there are 55 persons. They would like to choose two leaders "CR and GR". The way they decide is that all 55 sit in a circle. They start a count with person 1 and go in clockwise direction and skip 3. Person 4 reached is eliminated. The count starts with the fifth and the next person to go is the fourth in count. Eventually, two persons remain.

SOLUTION:

```
--***** SOURCE CODE *****

#include<Iostream>
#include<string.h>
using namespace std;
int limit[13]={0,31,28,31,30,31,30,31,31,30,31,30,31};
char
*m[13]={"NULL","JANUARY","FEBRUARY","MARCH","APRIL","MAY",
"JUNE","JULY","AUGUST","SEPTEMBER","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 printdate();
};
```

```

DATE::DATE()
{
}
DATE::DATE(INT D,INT M,INT Y)
{

    SETMONTH(M);
    SETYEAR(Y);
    SETDAY(D);
}
VOID DATE::SETDAY(INT D)
{
    DAY = ((D>=1&&D<=LIMIT[THIS->GETMONTH()])?D:1);
    IF(THIS->GETYEAR()%4==0&&THIS->GETMONTH()==2)
        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>=1900&&Y<=2023)?Y:1900);
}
INT DATE::GETDAY()
{
    RETURN DAY;
}
INT DATE::GETYEAR()
{
    RETURN YEAR;
}
INT DATE::GETMONTH()
{
    RETURN MONTH;
}
VOID DATE::PRINTDATE()
{

```

```

        COUT<<THIS->GETDAY()<<":"<<M[THIS-
>GETMONTH()]<<":"<<THIS->GETYEAR()<<ENDL;
    }

    CLASS STUDENT
    {
        PRIVATE:
            CHAR * ID;
            CHAR *NAME;
            DATE DOB;
            CHAR G;
        PUBLIC:
            STUDENT();
            STUDENT(CHAR *, CHAR *, DATE, CHAR);
            CHAR GETGENDER();
            VOID SHOWSTUDENT();

};
STUDENT::STUDENT()
{
}
STUDENT::STUDENT(CHAR *I, CHAR *N, DATE D, CHAR GN)
{
    ID = I;
    NAME = N;
    DOB = D;
    G = GN;
}
VOID STUDENT::SHOWSTUDENT()
{
    COUT<<"ID : "<< ID<<ENDL;
    COUT<<"NAME : "<< NAME<<ENDL;
    COUT<<"DATE OF BIRTH : ";
    DOB.PRINTDATE();
    COUT<<"GENDER : "<<G<<ENDL;

}
CHAR STUDENT::GETGENDER()

```

```

{
    RETURN G;
}
CLASS NODE
{
    PRIVATE:
        STUDENT DATA;
        NODE * NEXT;
    PUBLIC:
        NODE(STUDENT );
        VOID SETNEXT(NODE *);
        NODE* GETNEXT();
        VOID SHOWNODE();
        STUDENT GETDATA();
};

NODE::NODE(STUDENT D)
{
    DATA = D;
    NEXT = NULL;
}
VOID NODE::SETNEXT(NODE * PTR)
{
    NEXT = PTR;
}
NODE* NODE::GETNEXT()
{
    RETURN NEXT;
}
VOID NODE::SHOWNODE()
{
    DATA.SHOWSTUDENT();
}
STUDENT NODE::GETDATA()
{
    RETURN DATA;
}
CLASS LIST

```

```

{
    PRIVATE:
        NODE * HEAD;
        NODE * CURRENT;
        INT SIZE;
        INT MALECOUNT;
        INT FEMALECOUNT;
    PUBLIC:
        LIST();
        VOID ADD(STUDENT D);
        VOID REMOVE();
        VOID START();
        INT GETSIZE();
        VOID NEXT();
        VOID SHOWCURRENT();
        VOID SHOWLIST();
        INT GETFEMALECOUNT();
        INT GETMALECOUNT();
        NODE* NODETOREMOVE();
};

LIST::LIST()
{
    HEAD = NULL;
    CURRENT = NULL;
    SIZE = 0;
}

VOID LIST::ADD(STUDENT D)
{
    NODE *PTR = NEW NODE(D);
    IF(SIZE == 0)
    {
        HEAD = PTR;
        CURRENT = PTR;
        HEAD->SETNEXT(HEAD);
    }
    ELSE
    {
        PTR->SETNEXT(CURRENT->GETNEXT());
    }
}

```

```

        CURRENT->SETNEXT(PTR);
        CURRENT = PTR;
    }
    IF(D.GETGENDER()=='M')
        MALECOUNT++;
    IF(D.GETGENDER()=='F')
        FEMALECOUNT++;
    SIZE++;
}
VOID LIST::SHOWLIST()
{
    NODE *PTR = HEAD;
    DO
    {
        PTR->SHOWNODE();
        PTR=PTR->GETNEXT();
    }WHILE(PTR!=HEAD);
}
VOID LIST::REMOVE()
{
    NODE *PTR = CURRENT->GETNEXT();
    CURRENT->SETNEXT(PTR->GETNEXT());
    // CURRENT=PTR->GETNEXT();
    IF(PTR==HEAD)
        HEAD=HEAD->GETNEXT();

    STUDENT S = PTR->GETDATA();
    IF(S.GETGENDER()=='M')
        MALECOUNT--;
    IF(S.GETGENDER()=='F')
        FEMALECOUNT--;

    DELETE PTR;
    SIZE--;
}
VOID LIST::START()
{

```

```

    CURRENT = HEAD;
}
INT LIST::GETSIZE()
{
    RETURN SIZE;
}
VOID LIST::NEXT()
{
    CURRENT=CURRENT->GETNEXT();
}
VOID LIST::SHOWCURRENT()
{
    CURRENT->GETNEXT()->SHOWNODE();
}
INT LIST::GETFEMALECOUNT()
{
    RETURN FEMALECOUNT;
}
INT LIST::GETMALECOUNT()
{
    RETURN MALECOUNT;
}
NODE* LIST::NODETOREMOVE()
{
    RETURN CURRENT->GETNEXT();
}
INT MAIN()
{
    DATE D1(15,4,1983);

    //S1.SHOWSTUDENT();
    DATE D2(15,5,1983);

    //S2.SHOWSTUDENT();
    STUDENT S1("MCSFo2E001","ALI", D2,'M');
    STUDENT S2("MCSFo2E002","AHMAD", D2,'M');
    STUDENT S3("MCSFo2E003","HASAN", D2,'M');
    STUDENT S4("MCSFo2E004","AR", D2,'M');

```



```

STUDENT S5("MCSFo2E005","IRHA",D2,'F');
STUDENT S6("MCSFo2E006","ESHAAL",D2,'F');
STUDENT S7("MCSFo2E007","IQRA",D2,'F');
STUDENT S8("MCSFo2E008","UZAIR",D2,'M');
STUDENT S9("MCSFo2E009","SALMAN",D1,'M');
STUDENT S10("MCSFo2E010","MAHNOOR BLOACH",D2,'F');
STUDENT S11("MCSFo2E0011","IJAZ",D1,'M');
STUDENT S12("MCSFo2E0012","IQBAL",D1,'M');
STUDENT S13("MCSFo2E0013","SHARIF",D1,'M');
STUDENT S14("MCSFo2E0014","ROMAISA",D1,'F');
STUDENT S15("MCSFo2E0015","ALEEHA",D1,'F');
STUDENT S16("MCSFo2E0016","ARFA",D1,'F');
STUDENT S17("MCSFo2E017","KHANSA",D1,'F');
STUDENT S18("MCSFo2E0018","HAMID",D1,'M');
STUDENT S19("MCSFo2E0019","MEERAB",D1,'F');
STUDENT S20("MCSFo2E0020","IBRAHIM",D1,'M');
STUDENT S21("MCSFo2E0021","ADIL",D1,'M');
STUDENT S22("MCSFo2E0022","ADNAN",D1,'M');
STUDENT S23("MCSFo2E0023","RABEA",D1,'F');
STUDENT S24("MCSFo2E0024","ASHER",D1,'M');
STUDENT S25("MCSFo2E0025","ARIF",D1,'M');

```

```

LIST L;
L.ADD(S1);
L.ADD(S2);
L.ADD(S3);
L.ADD(S4);
L.ADD(S5);
L.ADD(S6);
L.ADD(S7);
L.ADD(S8);
L.ADD(S9);
L.ADD(S10);
L.ADD(S11);
L.ADD(S12);
L.ADD(S13);
L.ADD(S14);
L.ADD(S15);

```

```

L.ADD(S16);
L.ADD(S17);
L.ADD(S18);
L.ADD(S19);
L.ADD(S20);
L.ADD(S21);
L.ADD(S22);
L.ADD(S23);
L.ADD(S24);
L.ADD(S25);
L.START();
L.SHOWLIST();
INT I;
NODE *PTR;
STUDENT S;
WHILE(L.GETSIZE()>2)
{
    FOR(I=1;I<=3;I++)
    {
        L.NEXT();
    }

    PTR = L.NODETOREMOVE();
    S=PTR->GETDATA();
    IF(L.GETFEMALECOUNT()==1&&S.GETGENDER()=='F')
    {
        L.NEXT();
        CONTINUE;
    }
    IF(L.GETMALECOUNT()==1&&S.GETGENDER()=='M')
    {
        L.NEXT();
        CONTINUE;
    }

    COUT<<"STUDENT TO B REMOVED : -----"<<ENDL;
    L.SHOWCURRENT();
    L.REMOVE();
}

```

```
}  
    COUT<<ENDL<<ENDL<<"CR AND GR DATA :-----"  
    "<<ENDL;  
    L.SHOWLIST();  
  
}
```



HOMWORKS



HOMework 1

STATEMENT:

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.

SOLUTION:

```
--***** SOURCE CODE *****

#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","SEPTEMBER","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);
}

```


HOMework 2

STATEMENT:

A- 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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();
}

```

STATEMENT:

B- 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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();
}

```

STATEMENT:

- C-** 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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();
}

```

STATEMENT:

D- 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.

SOLUTION:

--***** SOURCE CODE *****

```
#INCLUDE<ISTREAM>
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();  
}
```

HOMework 3

STATEMENT:

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

SOLUTION:

--***** SOURCE CODE *****

```
#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);

}

```

HOMEWORK 4

STATEMENT:

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 specified maximum number of elements to prevent overflow. Implement error handling for stack underflow conditions. Design a C++ program to implement a stack using a linked list. Define a node structure for the linked list and include functionalities to push, pop, and check the top element of the stack. Implement error handling for stack underflow conditions.

SOLUTION:

--***** SOURCE CODE *****

```
#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 ISEMPY();
};

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::ISEMPY()
{
    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.ISEMPY()<<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.ISEMPY()<<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(o) AND FULL (o)*****
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;

}

```

HOMEWORK 5

STATEMENT:

A- Design a C++ program to implement the evaluation of a postfix expression using a stack. Implement a stack using a linked list, and provide functionalities to push, pop, and check the top element of the stack. The program should read a postfix expression as input and use the stack to evaluate the result. Support the basic arithmetic operators (+, -, *, /) and handle operand and operator errors appropriately.

SOLUTION:

--***** SOURCE CODE *****

```
#INCLUDE<Iostream>
#include<math.h>
using namespace std;
class STACK
{
    private:
        int *PTR;
        int current;
        int length;
    public:
        STACK(int);
        void push(int);
        int pop();
        int top();
        bool isfull();
        bool isempty();
};
STACK::STACK(int L)
{
    length=L;
    PTR=new int[length];
    current=-1;
```

```

}
VOID STACK::PUSH(INT D)
{
    IF(CURRENT==LENGTH-1)
    {
        COUT<<"STACK IS FULL"<<ENDL;
    }
    ELSE
    {
        PTR[++CURRENT]=D;
    }
}
INT STACK::POP()
{
    IF(CURRENT== -1)
    {
        RETURN -1;
    }
    ELSE
    {
        RETURN PTR[CURRENT--];
    }
}
INT STACK::TOP()
{
    IF(CURRENT== -1)
        RETURN -1;
    ELSE
        RETURN PTR[CURRENT];
}
BOOL STACK::ISEMPTY()
{
    IF(CURRENT== -1)

```



```

    {
        RETURN TRUE;
    }
    ELSE
        RETURN FALSE;
}

BOOL STACK::ISFULL()
{
    IF(CURRENT==LENGTH-1)
    {
        RETURN TRUE;
    }
    ELSE
        RETURN FALSE;
}

INT MAIN()
{
    INT I = 0, OP1,OP2,X;
    CHAR INPUT[100];
    STACK S(10);
    COUT<<"ENTER A VALID POSTFIX EXPRESSION: "<<ENDL;
    CIN>>INPUT;

    WHILE(INPUT[I] != '\0')
    {
        IF(INPUT[I]>='0' && INPUT[I]<='9')
        {
            X = INPUT[I] - 48;
            S.PUSH(X);
        }
        ELSE IF(INPUT[I]=='+' || INPUT[I]=='-' || INPUT[I]=='*' ||
INPUT[I]=='/'||INPUT[I]=='^')

```

```

{
    OP2 = S.POP();
    OP1 = S.POP();
    SWITCH(INPUT[I]){
        CASE '+':
            S.PUSH(OP1+OP2);
            BREAK;
        CASE '-':
            S.PUSH(OP1-OP2);
            BREAK;
        CASE '*':
            S.PUSH(OP1*OP2);
            BREAK;
        CASE '/':
            S.PUSH(OP1/OP2);
            BREAK;
        CASE '^':
            S.PUSH(POW(OP1,OP2));
            BREAK;
    }
}
I++;
}
COUT<<"ANSWER = "<<S.POP();
RETURN o;
}

```

STATEMENT:

B- Design a C++ program to implement the evaluation of an infix expression using a stack. Implement a stack using a linked list, and provide functionalities to push, pop, and check the top element of the stack. The program should read an infix expression as input, convert it to postfix notation, and then use the stack to evaluate the result. Support the basic arithmetic operators (+, -, *, /) and parentheses. Handle operand and operator errors appropriately.

SOLUTION:

--***** SOURCE CODE *****

```
#INCLUDE<Iostream>
#include<string>
#include<math.h>
using namespace std;
template <class T>
class stack
{
    private:
        int top;
        T arr[10];
    public:
        stack();
        void push(T);
        T pop();
        T topelement();
        bool isempty();
};
template <class T>
stack<T>::stack()
{
    top=-1;
}
template <class T>
void stack<T>::push(T val)
{
    arr[++top]=val;
}
template <class T>
T stack<T>::topelement()
{
```

```

        RETURN ARR[TOP];
    }
    TEMPLATE <CLASS T>
    T STACK<T>::POP()
    {
        RETURN ARR[TOP--];
    }
    TEMPLATE <CLASS T>
    BOOL STACK<T>::ISEMPTY()
    {
        IF(TOP== -1)
        {
            RETURN TRUE;
        }
        ELSE
            RETURN FALSE;
    }
    INT PREC (CHAR C){
        IF (C == '^'){
            RETURN 3;
        }
        ELSE IF (C == '*' || C == '/'){
            RETURN 2;
        }
        ELSE IF (C == '+' || C == '-'){
            RETURN 1;
        }
        ELSE {
            RETURN -1;
        }
    }
    INT MAIN()
    {

```

```

INT I = 0;
CHAR INPUT[100];
STRING RES;
STACK<CHAR> S;
COUT<<"ENTER AN INFIX EXPRESSION: "<<ENDL;
CIN>>INPUT;

WHILE(INPUT[I] != '\0')
{
    IF(INPUT[I]>='A' && INPUT[I]<='Z' || INPUT[I]>='a' &&
INPUT[I]<='z')
    {
        RES += INPUT[I];
    }
    ELSE IF(INPUT[I]=='(')
    {
        S.PUSH(INPUT[I]);
    }
    ELSE IF(INPUT[I]==')')
    {
        WHILE(!S.ISEMPTY() && S.TOPELEMENT()!='(')
        {
            RES += S.TOPELEMENT();
            S.POP();
        }
        IF(!S.ISEMPTY()){
            S.POP();
        }
    }
    ELSE {
        WHILE (!S.ISEMPTY() && PREC(S.TOPELEMENT()) >
PREC(INPUT[I]))
        {

```

```
        RES += S.TOPELEMENT();
        S.POP();
    }
    S.PUSH(INPUT[I]);
}
I++;
}
    WHILE(!S.ISEMPY()){
        RES += S.TOPELEMENT();
        S.POP();
    }
    COUT<<"POSTFIX FORM IS: ";
    COUT<<RES<<ENDL;
    RETURN o;
}
```

HOMework 6

STATEMENT:

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 specified maximum number of elements to prevent overflow. Implement error handling for queue underflow conditions.

Design a C++ program to implement a queue using a linked list. 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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 ISEMPY();
    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 ENQUEUEQ(T);
        T DEQUEUE();
        T FRONTE();
        BOOL ISEMPY();
};
TEMPLATE<CLASS T>
QUEUE<T>::QUEUE()
{
    FRONT=NULL;
    RARE=NULL;
}
TEMPLATE<CLASS T>
INT QUEUE<T>::ENQUEUEQ(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.ISEMPY();
COUT<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"ENQUEUE DATA = "<<Q.ENQUEUQ(2)<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q.ISEMPY();
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.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q.ISEMPY();
COUT<<"\N\N*****\N*****
**\N\N"<<ENDL;
QUEUE<STRING> Q1;
COUT<<"EMPTY = "<<Q1.ISEMPY()<<ENDL;

```

```

COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21218")<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();
COUT<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"ENQUEUEUE DATA = "<<Q1.ENQUEUEQ("FARHAN
AHMAD")<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();
COUT<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21219")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21220")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21221")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21222")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21223")<<ENDL;
COUT<<"ENQUE DATA = "<<Q1.ENQUEUEQ("BIT21224")<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"FRONT ELEMENT = "<<Q1.FRONT()<<ENDL;
COUT<<"DEQUEUE = "<<Q1.DEQUEUE()<<ENDL;
COUT<<"EMPTY = "<<Q1.ISEMPY();
}

```

HOMWORK 7

STATEMENT:

A- Design a C++ program to implement a binary tree. Define 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.

SOLUTION:

--***** SOURCE CODE *****

```
#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]);
    }
}

```

STATEMENT:

B- Demonstrate the usage of your program by inserting nodes into the tree and then calling these recursive functions.

SOLUTION:

--***** SOURCE CODE *****

```

#include<Iostream>
#include<conio.h>
using namespace std;
class NODE
{
    private:
        int data;
        NODE *left;
        NODE *right;
    public:
        NODE(int );
        void set_data(int );
        int get_data();
        void set_left(NODE *);
        NODE* get_left();
        void set_right( NODE *);

```

```

    NODE* GET_RIGHT();

};
NODE::NODE(INT D)
{
    DATA = D;
    LEFT = NULL;
    RIGHT = NULL;
}
VOID NODE::SET_DATA(INT D)
{
    DATA = D;
}
INT NODE::GET_DATA()
{
    RETURN DATA;
}
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;
}

```

```
////////////////////////////////////
```

```
//DEFINTION OF INSERT FUNCTION
```

```
VOID INSERT(NODE *R,INT VAL)
{
    NODE *NEWNODE=NEW NODE(o);
    NEWNODE->SET_DATA(VAL);
    NODE *P,*F;
    P=F=R;
    WHILE(P->GET_DATA()!=VAL&&F!=NULL)
    {
        P=F;
        IF(F->GET_DATA()>VAL)
            F=F->GET_RIGHT();
        ELSE
            F=F->GET_LEFT();
    }
    IF(VAL==P->GET_DATA())
    {
        COUT<<"\NSAME VALUE....."<<VAL<<ENDL;
        DELETE NEWNODE;
        RETURN;
    }
    ELSE IF(P->GET_DATA()>VAL)
        P->SET_RIGHT(NEWNODE);
    ELSE
        P->SET_LEFT(NEWNODE);
}
```

```
//DEFINITION THE THE PREORDER FUNCTION
```

```
VOID PREORDER(NODE *R)
{
    IF(R!=NULL)
    {
        COUT<<" "<<R->GET_DATA();
```

```

        PREORDER(R->GET_LEFT());
        PREORDER(R->GET_RIGHT());
    }
}

//DEFINITION OF THE MEMBER FUNCTION
VOID INORDER(NODE *R)
{
    IF(R!=NULL)
    {
        INORDER(R->GET_LEFT()) ;
        COUT<<R->GET_DATA()<<" ";
        INORDER(R->GET_RIGHT());
    }
}

//DEFINITION OF THE MEMBER FUNCTION POST ORDER
VOID POSTORDER(NODE *R)
{
    IF(R!=NULL)
    {
        POSTORDER(R->GET_LEFT()) ;
        POSTORDER(R->GET_RIGHT());
        COUT<<R->GET_DATA()<<" ";
    }
}

////////////////////////////////////
//***MAIN BODEY*****
INT MAIN()
{
    INT A[7]={14,4,16,3,9,15,18};
    NODE *ROOT=NEW NODE(o);
    ROOT->SET_DATA(A[o]);
    FOR(INT I=1;I<7;I++)
    {

```

```

    INSERT(ROOT,A[I]) ;
}
    COUT<<"\NDISPLAY DATA IN PREORDER FORMAT...\N";
    PREORDER(ROOT);
    COUT<<"\NDISPLAY DATA IN INORDER FORMAT....\N";
    INORDER(ROOT);
    COUT<<"\NDISPLAY DATA IN POSTORDER FORMAT....\N";
    POSTORDER(ROOT);
    GETCH();
}

```

STATEMENT:

- C-** Demonstrate the usage of your program by inserting nodes into the tree and then calling these recursive functions.

SOLUTION:

--***** SOURCE CODE *****

```

#include<Iostream>
using namespace std;
template<class T>
class NODE
{
    public:
        NODE(T D);
        void setData(T D);
        T getData();
        void setNext(NODE<T> * N);
        NODE<T> * getNext();
    private:
        T data;
        NODE<T> *next;
};
template <class T>

```

```

NODE<T>::NODE(T D)
{
    DATA = 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>::SETNEXT(NODE<T> *N)
{
    NEXT = N;
}
TEMPLATE <CLASS T>
NODE<T> * NODE<T>::GETNEXT()
{
    RETURN NEXT;
} //END OF CLASS NODE

//CLASS STACK
TEMPLATE <CLASS T>
CLASS STACK
{
    PRIVATE:
        NODE<T> * HEAD;
    PUBLIC:

```



```

    STACK();
    VOID PUSH(T);
    T POP();
    T TOP();
    INT IS_EMPTY();
    VOID SHOW();
};
TEMPLATE <CLASS T>
STACK<T>::STACK()
{
    HEAD = NULL;
}
TEMPLATE <CLASS T>
VOID STACK<T>::PUSH(T VAL)
{
    NODE<T> * PTR = NEW NODE<T>(VAL);
    PTR ->SETNEXT(HEAD);
    HEAD = PTR;
}
TEMPLATE <CLASS T>
T STACK<T>::POP()
{
    NODE<T> *PTR = HEAD;
    T VAL = HEAD ->GETDATA();
    HEAD = HEAD->GETNEXT();
    DELETE PTR;
    RETURN VAL;
}
TEMPLATE <CLASS T>
T STACK<T>::TOP()
{
    RETURN HEAD ->GETDATA();
}

```

```

TEMPLATE <CLASS T>
INT STACK<T>::IS_EMPTY()
{
    IF(HEAD==NULL)
        RETURN TRUE;
    ELSE
        RETURN FALSE;
}
TEMPLATE <CLASS T>
VOID STACK <T>::SHOW()
{
    WHILE(HEAD!=NULL)
    {
        COUT<<POP();
        COUT<<ENDL;
    }
}
//END OF STACK
TEMPLATE <CLASS T>
CLASS TREENODE
{
    PRIVATE:
        T DATA;
        TREENODE<T> * LEFT;
        TREENODE<T> * RIGHT;
    PUBLIC:
        TREENODE(T);
        VOID SETDATA(T);
        T GETDATA();
        VOID SETLEFT(TREENODE<T> *);
        TREENODE<T> * GETLEFT();
        VOID SETRIGHT(TREENODE<T> *);
        TREENODE<T> * GETRIGHT();
}

```

```

        INT ISLEAF();
};
TEMPLATE <CLASS T>
TREENODE<T>::TREENODE(T D)
{
    DATA=D;
    LEFT=NULL;
    RIGHT=NULL;
}
TEMPLATE <CLASS T>
VOID TREENODE<T>::SETDATA(T D)

{
    DATA=D;
}
TEMPLATE <CLASS T>
T TREENODE<T>::GETDATA()
{
    RETURN DATA;
}
TEMPLATE <CLASS T>
VOID TREENODE<T>::SETLEFT(TREENODE<T> *L)
{
    LEFT=L;
}
TEMPLATE <CLASS T>
TREENODE<T> * TREENODE<T>::GETLEFT()
{
    RETURN LEFT;
}
TEMPLATE <CLASS T>
VOID TREENODE<T>::SETRIGHT(TREENODE<T> *R)
{

```

```

    RIGHT=R;
}
TEMPLATE <CLASS T>
TREENODE<T> * TREENODE<T>::GETRIGHT()
{
    RETURN RIGHT;
}
TEMPLATE <CLASS T>
INT TREENODE<T>::ISLEAF()
{
    IF(THIS->LEFT==NULL && THIS->RIGHT==NULL)
        RETURN 1;
    RETURN 0;
} // END OF TREENODE CLASS
////////////////////////////////////
////////////////////////////////////

//GLOBAL FN OF INSERT
TEMPLATE <CLASS T>
VOID INSERT(TREENODE<T> * ROOT, T D)
{
    TREENODE<T> * NODE=NEW TREENODE<T>(D);
    TREENODE<T> * P, * Q;
    P=Q=ROOT;
    WHILE(D!=P->GETDATA() && Q!= NULL)
    {
        P=Q;
        IF(D<P->GETDATA())
            Q=P->GETLEFT();
        ELSE
            Q=P->GETRIGHT();
    }
    IF(D==P->GETDATA())

```

```

    {
        COUT<<"\nATTEMPT TO INSERT DUPLICATE :
"<<D<<ENDL;
        DELETE NODE;
    }
    ELSE IF(D<P->GETDATA())
        P->SETLEFT(NODE);
    ELSE
        P->SETRIGHT(NODE);
} //END OF INSERT FN

```

```

//GLOBAL FN OF PREORDER
TEMPLATE <CLASS T>
VOID PREORDER(TREENODE<T> * NODE)
{
    STACK <TREENODE<T> *> S;
    TREENODE<T> *P;
    P=NODE;
    DO
    {
        WHILE(P!=NULL)
        {
            COUT<<P->GETDATA()<<" ";
            S.PUSH(P);
            P=P->GETLEFT();
        }
        IF(!S.IS_EMPTY())
        {
            P=S.POP();
            P=P->GETRIGHT();
        }
    } WHILE(P!=NULL || !(S.IS_EMPTY()));
}

```

```

//GLOBAL FN OF INORDER
TEMPLATE <CLASS T>
VOID INORDER(TREENODE<T> * NODE)
{
    STACK<TREENODE<T>*> S;
    TREENODE<T> * P;
    P=NODE;
    DO
    {
        WHILE(P!=NULL)
        {
            S.PUSH(P);
            P=P->GETLEFT();
        }
        IF(!S.IS_EMPTY())
        {
            P=S.POP();
            COUT<<P->GETDATA()<<" ";
            P=P->GETRIGHT();
        }
    }WHILE(P!=NULL || !(S.IS_EMPTY()));
}

//GLOBAL FN OF POSTORDER
TEMPLATE <CLASS T>
VOID POSTORDER(TREENODE<T> * NODE)
{
    STACK<TREENODE<T>*> S;
    STACK<TREENODE<T>*> OUTPUT;
    S.PUSH(NODE);
    WHILE (!S.IS_EMPTY())
    {
        TREENODE<T> * CURR = S.TOP();
        OUTPUT.PUSH(CURR);
    }
}

```

```

        S.POP();
        IF (CURR->GETLEFT())
        S.PUSH(CURR->GETLEFT());
        IF (CURR->GETRIGHT())
        S.PUSH(CURR->GETRIGHT());
    }
    WHILE (!OUTPUT.IS_EMPTY())
    {
        COUT << OUTPUT.TOP()->GETDATA() << " ";
        OUTPUT.POP();
    }

}

INT MAIN()
{
    INT X[]={14,4,3,9,16,15,18,-1};
    TREENODE<INT> * ROOT=NEW TREENODE<INT>(14);
    FOR(INT I=1;X[I]!=-1;I++)
    {
        INSERT(ROOT,X[I]);
    }
    COUT<<"\NPREORDER : ";
    PREORDER(ROOT);
    COUT<<"\NINORDER : ";
    INORDER(ROOT);
    COUT<<"\NPOSTORDER : ";
    POSTORDER(ROOT);
}

```



LABS



STATEMENT:

1. Suppose, we have a Shape class having following function. virtual void display area() = 0;
2. Shape class is inherited by three different classes, i.e. Square, Rectangle, and Triangle.
3. Each child class must implement the area function. (You may use your own logic to implement it). Formula of triangle area = (length * base) / 2 Formula of rectangle area = length * width Formula of square area = length * length

SOLUTION:

--***** SOURCE CODE *****

```
#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 o;  
}
```

STATEMENT:

Suppose, we have the following Stack class.

1. You have to implement all its functions.
2. Now, you write the following main function and verify the output.

SOLUTION:

--***** SOURCE CODE *****

```
#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;  
}
```

STATEMENT:

In last lecture, we have learnt the Stack data structure by dynamically allocate the memory. Today, we will implement it in the C++ language.

Suppose, we have the following Stack class.

1. You have to implement all its functions.
2. Now, you write the following main function and verify the output.

SOLUTION:

--***** SOURCE CODE *****

```
#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::TOTALEMENTS()
{
    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;
}

```

STATEMENT:

Today, you will implement the Link List class as we have discussed it in the previous lectures.

Suppose, we have the following Node and LinkList class.

1. You have to implement all functions of the both classes.
2. Now, you write the following main function and verify the output.

SOLUTION:

__***** SOURCE CODE *****

```
#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::TOTALEMENTS()
{
    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;
}

```

STATEMENT:

Today, you will implement the Circular Single Link List class as we have discussed it in the previous lectures.

Suppose, we have the following Node and Circular-Single-Link-List class.

1. You have to implement all functions of the both classes.
2. Now, you write the following main function and verify the output.

SOLUTION:

--***** SOURCE CODE *****

```
#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;
}

```


STATEMENT:

In last online session, we have learnt to implement the single link list class. Now, in lab 10, you have to implement the Stack by using single link list functions in the C++ language. Suppose, we have the following Stack class.

1. You have to implement all its functions.
2. Now, you write the following main function and verify the output.

SOLUTION:

--***** SOURCE CODE *****

```
#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;  
}
```

