Name – Adya Singh

Reg No – RA2211003010181

21CSC203P

Advanced Programming Practice

Assignment 3

1. Simple mini calculator program in C++ that uses subroutines for basic arithmetic operations.

Code –

main.cpp Run

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

int add(int a,int b)
{
 return a+b;
}
int sub(int a,int b)
{
if (a>b){
 return a-b;
}
else{
    return b-a;
}
}
int mul(int a,int b)
{
 return a*b;
}
int divd(int a,int b)
{
if (b!=0){
 return a/b;}
 else{
     return b=0;
```

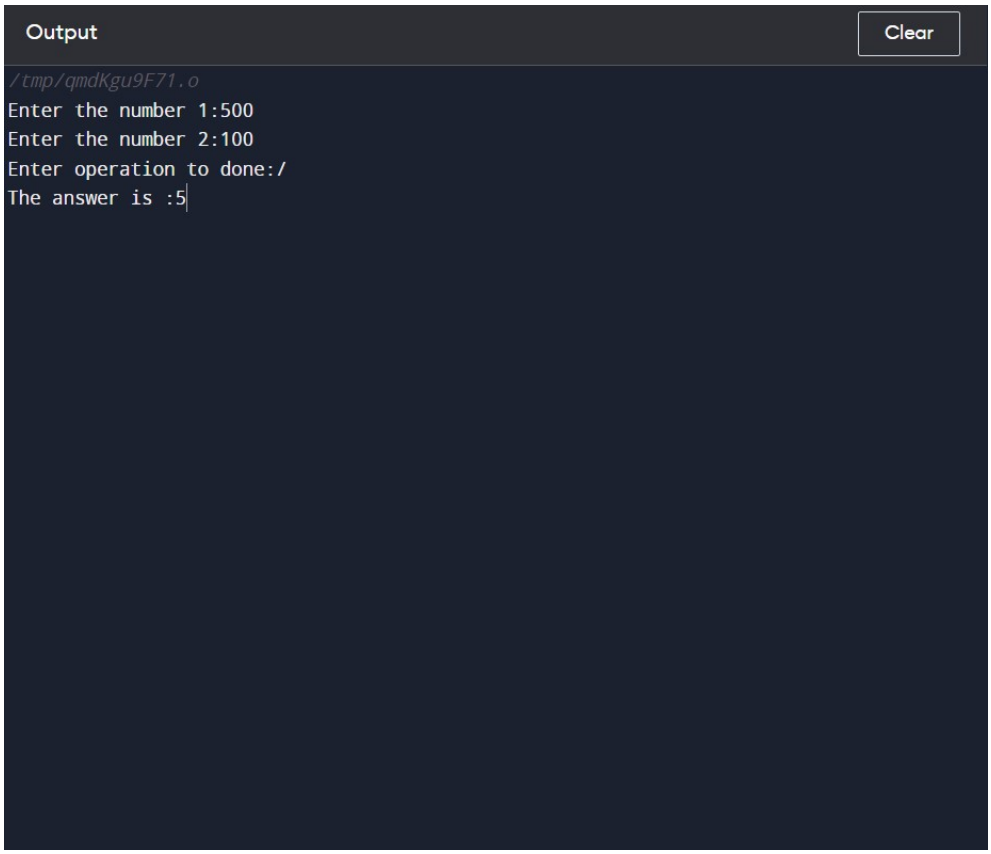
main.cpp

```
26        return b=0;
27    }
28  }
29  int main() {
30      int a,b;
31      cout<<"Enter the number 1:";
32      cin>>a;
33      cout<<"Enter the number 2:";
34      cin>>b;
35      char op;
36      cout<<"Enter operation to done:";
37      cin>>op;
38      int result;
39      switch(op){
40          case'+':
41          result=add(a,b);
42          break;
43          case'-':
44          result=sub(a,b);
45          break;
46          case'*':
47          result=mul(a,b);
48          break;
49          case'/':
50          result=divd(a,b);
51          break;
```

main.cpp

```
33      cout<<"Enter the number 2:";
34      cin>>b;
35      char op;
36      cout<<"Enter operation to done:";
37      cin>>op;
38      int result;
39      switch(op){
40          case'+':
41          result=add(a,b);
42          break;
43          case'-':
44          result=sub(a,b);
45          break;
46          case'*':
47          result=mul(a,b);
48          break;
49          case'/':
50          result=divd(a,b);
51          break;
52          default:
53          cout<<"Invalid operator";
54          return 1;
55      }
56      cout<<"The answer is :"<<result;
57      return 0;
58  }
```

Output –


Output
Clear
/tmp/qmdKgu9F71.o
Enter the number 1:500
Enter the number 2:100
Enter operation to done:/
The answer is :5

2. Write a complete students database with subroutines involves storing and managing student information using appropriate data structures and providing various functionalities to interact with the database and implement in C++ with subroutines:

Code –

main.cpp

Run

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Define a struct to represent a student
struct Student {
    string name;
    int id;
    double gpa;
};

// Define a vector to store all the students
vector<Student> students;

// Define a subroutine to add a new student to the database
void addStudent() {
    Student student;
    cout << "Enter student name: ";
    getline(cin, student.name);
    cout << "Enter student ID: ";
    cin >> student.id;
    cout << "Enter student GPA: ";
    cin >> student.gpa;
    students.push_back(student);
```

main.cpp

Run

```
    students.push_back(student);
    cout << "Student added successfully!" << endl;
}

// Define a subroutine to print all the students in the database
void printStudents() {
    if (students.empty()) {
        cout << "No students in the database." << endl;
    } else {
        cout << "Students in the database:" << endl;
        for (const auto& student : students) {
            cout << "Name: " << student.name << ", ID: " << student.id << ",
                GPA: " << student.gpa << endl;
        }
    }
}

// Define a subroutine to search for a student by ID
void searchStudent() {
    int id;
    cout << "Enter student ID to search for: ";
    cin >> id;
    for (const auto& student : students) {
        if (student.id == id) {
            cout << "Student found:" << endl;
            cout << "Name: " << student.name << ", ID: " << student.id << ",
```

main.cpp

Run

```
                GPA: " << student.gpa << endl;
            return;
        }
    }
    cout << "Student not found." << endl;
}

// Define a subroutine to delete a student by ID
void deleteStudent() {
    int id;
    cout << "Enter student ID to delete: ";
    cin >> id;
    for (auto it = students.begin(); it != students.end(); ++it) {
        if (it->id == id) {
            students.erase(it);
            cout << "Student deleted successfully!" << endl;
            return;
        }
    }
    cout << "Student not found." << endl;
}

// Define the main function to interact with the user
int main() {
    while (true) {
        cout << "Select an option:" << endl;
```

main.cpp

Run

```
75          cout << "Select an option:" << endl;
76          cout << "1. Add a student" << endl;
77          cout << "2. Print all students" << endl;
78          cout << "3. Search for a student" << endl;
79          cout << "4. Delete a student" << endl;
80          cout << "5. Quit" << endl;
81          int option;
82          cin >> option;
83          cin.ignore(); // Ignore the newline character left by cin
84          switch (option) {
85              case 1:
86                  addStudent();
87                  break;
88              case 2:
89                  printStudents();
90                  break;
91              case 3:
92                  searchStudent();
93                  break;
94              case 4:
95                  deleteStudent();
96                  break;
97              case 5:
98                  return 0;
99              default:
100                 cout << "Invalid option. Please try again." << endl;
```

main.cpp

Run

```
79          cout << "4. Delete a student" << endl;
80          cout << "5. Quit" << endl;
81          int option;
82          cin >> option;
83          cin.ignore(); // Ignore the newline character left by cin
84          switch (option) {
85              case 1:
86                  addStudent();
87                  break;
88              case 2:
89                  printStudents();
90                  break;
91              case 3:
92                  searchStudent();
93                  break;
94              case 4:
95                  deleteStudent();
96                  break;
97              case 5:
98                  return 0;
99              default:
100                 cout << "Invalid option. Please try again." << endl;
101         }
102     }
103 }
104
```

Output –

```
Output                                                    Clear
/tmp/SMlnIj0fRD.o
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
5. Quit
1
Enter student name: Adya Singh
Enter student ID: 181
Enter student GPA: 9.8
Student added successfully!
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
5. Quit
2
Students in the database:
Name: Adya Singh, ID: 181, GPA: 9.8
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
```

```
Output                                                    Clear
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
5. Quit
3
Enter student ID to search for: 181
Student found:
Name: Adya Singh, ID: 181, GPA: 9.8
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
5. Quit
4
Enter student ID to delete: 181
Student deleted successfully!
Select an option:
1. Add a student
2. Print all students
3. Search for a student
4. Delete a student
5. Quit
```

3. Design a subroutine program to calculate the area and perimeter of different geometric shapes (circle, rectangle, triangle, etc.).

Code –

main.cpp Run

```cpp
#include <iostream>
#include <cmath>

// Function to calculate the area and perimeter of a circle
void circleProperties(double radius, double& area, double& perimeter) {
    const double pi = 3.14159;
    area = pi * radius * radius;
    perimeter = 2 * pi * radius;
}

// Function to calculate the area and perimeter of a rectangle
void rectangleProperties(double length, double width, double& area, double&
    perimeter) {
    area = length * width;
    perimeter = 2 * (length + width);
}

// Function to calculate the area and perimeter of a triangle
void triangleProperties(double side1, double side2, double side3, double& area,
    double& perimeter) {
    double s = (side1 + side2 + side3) / 2;
    area = std::sqrt(s * (s - side1) * (s - side2) * (s - side3));
    perimeter = side1 + side2 + side3;
}

int main() {
```

main.cpp

Run

```cpp
int main() {
    int choice;
    double area, perimeter;

    do {
        std::cout << "Select a shape to calculate properties:\n";
        std::cout << "1. Circle\n";
        std::cout << "2. Rectangle\n";
        std::cout << "3. Triangle\n";
        std::cout << "4. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: {
                double radius;
                std::cout << "Enter the radius of the circle: ";
                std::cin >> radius;
                circleProperties(radius, area, perimeter);
                std::cout << "Area: " << area << "\n";
                std::cout << "Perimeter: " << perimeter << "\n";
                break;
            }
            case 2: {
                double length, width;
                std::cout << "Enter the length of the rectangle: ";
```

main.cpp

Run

```cpp
                std::cout << "Enter the length of the rectangle: ";
                std::cin >> length;
                std::cout << "Enter the width of the rectangle: ";
                std::cin >> width;
                rectangleProperties(length, width, area, perimeter);
                std::cout << "Area: " << area << "\n";
                std::cout << "Perimeter: " << perimeter << "\n";
                break;
            }
            case 3: {
                double side1, side2, side3;
                std::cout << "Enter the length of side 1 of the triangle: ";
                std::cin >> side1;
                std::cout << "Enter the length of side 2 of the triangle: ";
                std::cin >> side2;
                std::cout << "Enter the length of side 3 of the triangle: ";
                std::cin >> side3;
                triangleProperties(side1, side2, side3, area, perimeter);
                std::cout << "Area: " << area << "\n";
                std::cout << "Perimeter: " << perimeter << "\n";
                break;
            }
            case 4:
                std::cout << "Exiting the program.\n";
                break;
            default:
```

main.cpp

Run

```
            case 3: {
                double side1, side2, side3;
                std::cout << "Enter the length of side 1 of the triangle: ";
                std::cin >> side1;
                std::cout << "Enter the length of side 2 of the triangle: ";
                std::cin >> side2;
                std::cout << "Enter the length of side 3 of the triangle: ";
                std::cin >> side3;
                triangleProperties(side1, side2, side3, area, perimeter);
                std::cout << "Area: " << area << "\n";
                std::cout << "Perimeter: " << perimeter << "\n";
                break;
            }
            case 4:
                std::cout << "Exiting the program.\n";
                break;
            default:
                std::cout << "Invalid choice. Try again.\n";
        }

        std::cout << std::endl;
    } while (choice != 4);

    return 0;
}
```

Output –

Output

Clear

```
/tmp/Zsz9THHMvC.o
Select a shape to calculate properties:
1. Circle
2. Rectangle
3. Triangle
4. Exit
Enter your choice: 1
Enter the radius of the circle: 6.5
Area: 132.732
Perimeter: 40.8407

Select a shape to calculate properties:
1. Circle
2. Rectangle
3. Triangle
4. Exit
Enter your choice: 4
Exiting the program.
```

4. Implement a subroutine program to check if a given string is a palindrome or not.

Code –

main.cpp

Run

```cpp
#include <iostream>

bool isPalindrome(const char *s) {
    int length = 0;
    while (s[length] != '\0') {
        length++;
    }

    char modifiedString[length + 1];
    int j = 0;
    for (int i = 0; i < length; i++) {
        char c = s[i];
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || (c >= '0' && c
            <= '9')) {
            modifiedString[j++] = tolower(c);
        }
    }
    modifiedString[j] = '\0';

    int left = 0;
    int right = j - 1;
    while (left < right) {
        if (modifiedString[left] != modifiedString[right]) {
            return false;
        }
        left++;
```

main.cpp

Run

```cpp

    int left = 0;
    int right = j - 1;
    while (left < right) {
        if (modifiedString[left] != modifiedString[right]) {
            return false;
        }
        left++;
        right--;
    }
    return true;
}

int main() {
    char input_string[100];
    std::cout << "Enter a string: ";
    std::cin.getline(input_string, sizeof(input_string));

    if (isPalindrome(input_string)) {
        std::cout << "The given string is a palindrome." << std::endl;
    } else {
        std::cout << "The given string is not a palindrome." << std::endl;
    }

    return 0;
}
```

Output –

Output

Clear

```
/tmp/EYbygeXR2s.o
Enter a string: level
The given string is a palindrome.
```

5. Implement a subroutine program to reverse an array of integers in-place.

Code –

main.cpp

Run

```cpp
#include <iostream>

// Function to reverse an array of integers in-place
void reverseArray(int arr[], int size) {
    int left = 0;
    int right = size - 1;

    while (left < right) {
        // Swap elements at left and right indices
        int temp = arr[left];
        arr[left] = arr[right];
        arr[right] = temp;

        left++;
        right--;
    }
}

int main() {
    int size;

    std::cout << "Enter the size of the array: ";
    std::cin >> size;

    int* arr = new int[size];

```

main.cpp

Run

```
    int* arr = new int[size];

    std::cout << "Enter " << size << " integers: ";
    for (int i = 0; i < size; i++) {
        std::cin >> arr[i];
    }

    std::cout << "Original array: ";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    reverseArray(arr, size);

    std::cout << "Reversed array: ";
    for (int i = 0; i < size; i++) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    delete[] arr;

    return 0;
}
```

Output –

Output

Clear

```
/tmp/siYagOqKiM.o
Enter the size of the array: 10
Enter 10 integers: 1 2 3 4 5 6 7 8 9 10
Original array: 1 2 3 4 5 6 7 8 9 10
Reversed array: 10 9 8 7 6 5 4 3 2 1
```

6. Write a program that dynamically allocates memory for an array of integers based on user input and then finds the sum of all elements in the array.

Code –

main.cpp

Run

```c
#include <stdio.h>
#include <stdlib.h>

int main() {
    int n;  // Number of elements in the array
    printf("Enter the number of elements: ");
    scanf("%d", &n);

    // Dynamically allocate memory for the array
    int *arr = (int *)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;  // Exit with an error code
    }

    // Input array elements from the user
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Calculate the sum of array elements
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
```

main.cpp

Run

```c
    int *arr = (int *)malloc(n * sizeof(int));

    if (arr == NULL) {
        printf("Memory allocation failed.\n");
        return 1;  // Exit with an error code
    }

    // Input array elements from the user
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }

    // Calculate the sum of array elements
    int sum = 0;
    for (int i = 0; i < n; i++) {
        sum += arr[i];
    }

    printf("Sum of array elements: %d\n", sum);

    // Free the dynamically allocated memory
    free(arr);

    return 0;  // Exit successfully
}
```

Output –

Output

Clear

```
/tmp/MxIDdR0kLc.o
Enter the number of elements: 5
Enter 5 integers:
4 7 44 76 55
Sum of array elements: 186
```

7. Implement a program that reads a text file and dynamically stores each line as a string in memory. Then, display the content of the file with line numbers.

Code –

main.cpp

Run

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

int main() {
    std::string filename;
    std::cout << "Enter the name of the text file: ";
    std::cin >> filename;

    std::ifstream inputFile(filename);

    if (!inputFile.is_open()) {
        std::cout << "Error: Unable to open the file.\n";
        return 1;
    }

    std::vector<std::string> lines;
    std::string line;

    while (std::getline(inputFile, line)) {
        lines.push_back(line);
    }

    inputFile.close();

```

main.cpp

Run

```
14          std::cout << "Error: Unable to open the file.\n";
15          return 1;
16      }
17
18      std::vector<std::string> lines;
19      std::string line;
20
21      while (std::getline(inputFile, line)) {
22          lines.push_back(line);
23      }
24
25      inputFile.close();
26
27      if (lines.empty()) {
28          std::cout << "The file is empty.\n";
29          return 0;
30      }
31
32      std::cout << "Content of the file with line numbers:\n";
33      for (size_t i = 0; i < lines.size(); i++) {
34          std::cout << i + 1 << ". " << lines[i] << std::endl;
35      }
36
37      return 0;
38  }
39
```

Output –

Output

Clear

```
/tmp/XYyDEBcgUE.o
Enter the name of the text file: addie.txt
ERROR!
Error: Unable to open the file.
```

8. Create a program that uses dynamic memory allocation to implement a stack data structure to push and pop elements.

Code –

main.cpp

```cpp
#include <iostream>

class Stack {
private:
    int *arr;   // Dynamic array to store stack elements
    int capacity;   // Maximum capacity of the stack
    int top;    // Index of the top element (-1 if empty)

public:
    Stack(int capacity) {
        this->capacity = capacity;
        this->arr = new int[capacity];
        this->top = -1;
    }

    ~Stack() {
        delete[] arr;
    }

    bool isEmpty() {
        return top == -1;
    }

    bool isFull() {
        return top == capacity - 1;
    }
```

main.cpp

Run

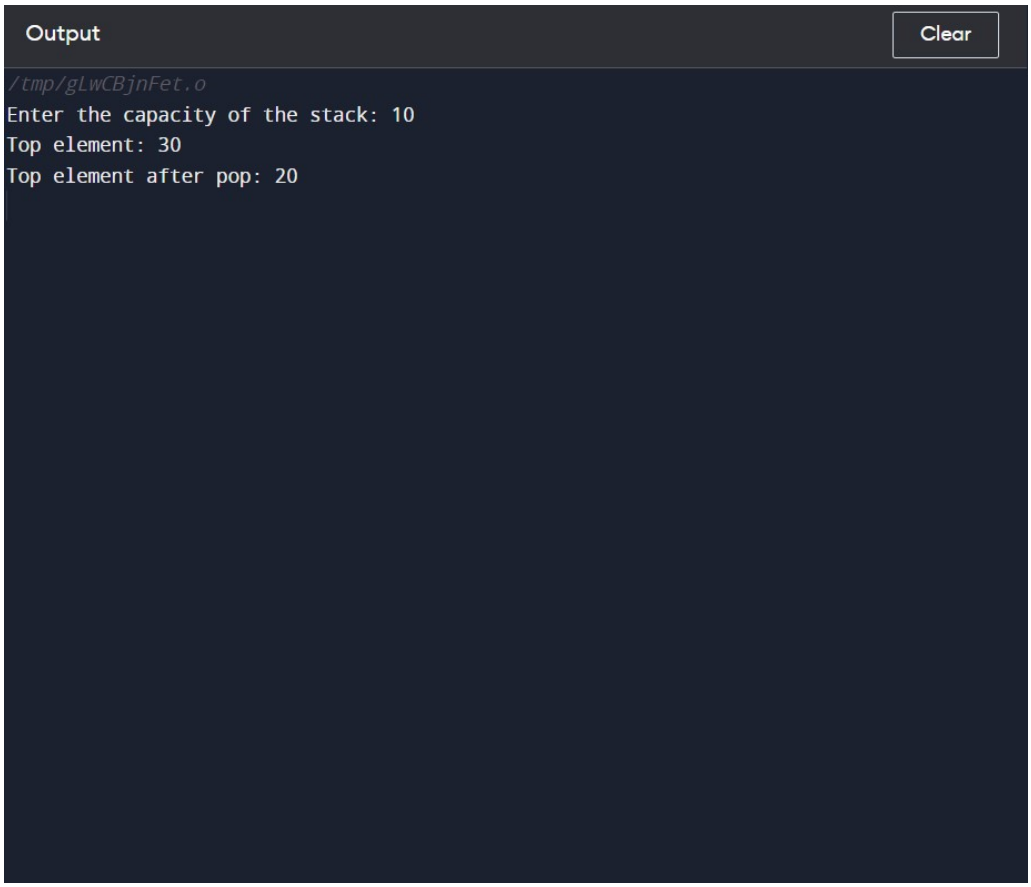
```
    }

    void push(int value) {
        if (isFull()) {
            std::cout << "Stack is full. Cannot push element.\n";
            return;
        }
        top++;
        arr[top] = value;
    }

    void pop() {
        if (isEmpty()) {
            std::cout << "Stack is empty. Cannot pop element.\n";
            return;
        }
        top--;
    }

    int peek() {
        if (isEmpty()) {
            std::cout << "Stack is empty. Cannot peek.\n";
            return -1;
        }
        return arr[top];
    }
```

main.cpp

Run

```
            return -1;
        }
        return arr[top];
    }
};

int main() {
    int capacity;
    std::cout << "Enter the capacity of the stack: ";
    std::cin >> capacity;

    Stack stack(capacity);

    stack.push(10);
    stack.push(20);
    stack.push(30);

    std::cout << "Top element: " << stack.peek() << std::endl;

    stack.pop();

    std::cout << "Top element after pop: " << stack.peek() << std::endl;

    return 0;
}
```

Output –


Output
Clear
/tmp/gLwCBjnFet.o
Enter the capacity of the stack: 10
Top element: 30
Top element after pop: 20

9. Implement a program that uses dynamic memory allocation to simulate a banking system that stores customer information, account details, and transactions.

Code –

main.cpp

Run

```cpp
#include <iostream>
#include <string>
#include <vector>

struct Transaction {
    std::string type;
    double amount;
};

struct Account {
    int accountNumber;
    std::string customerName;
    double balance;
    std::vector<Transaction> transactions;
};

// Function to create a new account
Account* createAccount(int accountNumber, const std::string& customerName) {
    Account* newAccount = new Account;
    newAccount->accountNumber = accountNumber;
    newAccount->customerName = customerName;
    newAccount->balance = 0.0;
    return newAccount;
}

// Function to deposit money into an account
```

main.cpp

Run

```cpp
// Function to deposit money into an account
void deposit(Account* account, double amount) {
    account->balance += amount;
    Transaction transaction;
    transaction.type = "Deposit";
    transaction.amount = amount;
    account->transactions.push_back(transaction);
    std::cout << "Amount deposited successfully.\n";
}

// Function to withdraw money from an account
bool withdraw(Account* account, double amount) {
    if (account->balance >= amount) {
        account->balance -= amount;
        Transaction transaction;
        transaction.type = "Withdrawal";
        transaction.amount = amount;
        account->transactions.push_back(transaction);
        std::cout << "Amount withdrawn successfully.\n";
        return true;
    } else {
        std::cout << "Insufficient balance.\n";
        return false;
    }
}

```

main.cpp

Run

```cpp
// Function to display account information
void displayAccountInfo(const Account* account) {
    std::cout << "Account Number: " << account->accountNumber << "\n";
    std::cout << "Customer Name: " << account->customerName << "\n";
    std::cout << "Account Balance: " << account->balance << "\n";
}

// Function to display transaction history
void displayTransactionHistory(const Account* account) {
    std::cout << "Transaction History for Account Number " << account
        ->accountNumber << ":\n";
    for (const auto& transaction : account->transactions) {
        std::cout << "Type: " << transaction.type << "\tAmount: " <<
            transaction.amount << "\n";
    }
}

// Function to delete an account and free its memory
void deleteAccount(Account* account) {
    delete account;
}

int main() {
    std::vector<Account*> accounts;

    int accountNumber = 1001;
```

main.cpp

Run

```
    int accountNumber = 1001;
    int choice;
    do {
        std::cout << "\nBanking System Menu:\n";
        std::cout << "1. Create Account\n";
        std::cout << "2. Deposit\n";
        std::cout << "3. Withdraw\n";
        std::cout << "4. View Account Information\n";
        std::cout << "5. View Transaction History\n";
        std::cout << "6. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1: {
                std::string customerName;
                std::cout << "Enter customer name: ";
                std::cin.ignore(); // Ignore the newline character left by
                    previous input
                std::getline(std::cin, customerName);

                Account* newAccount = createAccount(accountNumber,
                    customerName);
                accounts.push_back(newAccount);
                std::cout << "Account created successfully. Account number: "
                    << accountNumber << "\n";
```

main.cpp

Run

```
                    << accountNumber << "\n";
                accountNumber++;
                break;
            }
            case 2: {
                int accountNum;
                double amount;
                std::cout << "Enter account number: ";
                std::cin >> accountNum;

                Account* account = nullptr;
                for (const auto& acc : accounts) {
                    if (acc->accountNumber == accountNum) {
                        account = acc;
                        break;
                    }
                }

                if (account == nullptr) {
                    std::cout << "Account not found.\n";
                    break;
                }

                std::cout << "Enter amount to deposit: ";
                std::cin >> amount;
                deposit(account, amount);
```

main.cpp

Run

```
            deposit(account, amount);
            break;
        }
        case 3: {
            int accountNum;
            double amount;
            std::cout << "Enter account number: ";
            std::cin >> accountNum;

            Account* account = nullptr;
            for (const auto& acc : accounts) {
                if (acc->accountNumber == accountNum) {
                    account = acc;
                    break;
                }
            }

            if (account == nullptr) {
                std::cout << "Account not found.\n";
                break;
            }

            std::cout << "Enter amount to withdraw: ";
            std::cin >> amount;
            withdraw(account, amount);
            break;
```

main.cpp

Run

```
            break;
        }
        case 4: {
            int accountNum;
            std::cout << "Enter account number: ";
            std::cin >> accountNum;

            Account* account = nullptr;
            for (const auto& acc : accounts) {
                if (acc->accountNumber == accountNum) {
                    account = acc;
                    break;
                }
            }

            if (account == nullptr) {
                std::cout << "Account not found.\n";
                break;
            }

            displayAccountInfo(account);
            break;
        }
        case 5: {
            int accountNum;
            std::cout << "Enter account number: ";
```

main.cpp

Run

```
172                 std::cout << "Enter account number: ";
173                 std::cin >> accountNum;
174
175                 Account* account = nullptr;
176                 for (const auto& acc : accounts) {
177                     if (acc->accountNumber == accountNum) {
178                         account = acc;
179                         break;
180                     }
181                 }
182
183                 if (account == nullptr) {
184                     std::cout << "Account not found.\n";
185                     break;
186                 }
187
188                 displayTransactionHistory(account);
189                 break;
190             }
191             case 6:
192                 // Free memory for all accounts
193                 for (const auto& account : accounts) {
194                     deleteAccount(account);
195                 }
196                 std::cout << "Exiting the program.\n";
197                 break;
```

main.cpp

Run

```
181                 }
182
183                 if (account == nullptr) {
184                     std::cout << "Account not found.\n";
185                     break;
186                 }
187
188                 displayTransactionHistory(account);
189                 break;
190             }
191             case 6:
192                 // Free memory for all accounts
193                 for (const auto& account : accounts) {
194                     deleteAccount(account);
195                 }
196                 std::cout << "Exiting the program.\n";
197                 break;
198             default:
199                 std::cout << "Invalid choice. Try again.\n";
200         }
201     } while (choice != 6);
202
203     return 0;
204 }
205
206
```

Output –

```
Output                                                    Clear
/tmp/VHYEiuzTAf.o
Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. View Account Information
5. View Transaction History
6. Exit
Enter your choice: 1
Enter customer name: Adya
Account created successfully. Account number: 1001

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. View Account Information
5. View Transaction History
6. Exit
Enter your choice: 2
Enter account number: 1001
Enter amount to deposit: 5000
Amount deposited successfully.

Banking System Menu:
1. Create Account
```

```
Output                                                    Clear
Amount deposited successfully.

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. View Account Information
5. View Transaction History
6. Exit
Enter your choice: 3
Enter account number: 1001
Enter amount to withdraw: 2000
Amount withdrawn successfully.

Banking System Menu:
1. Create Account
2. Deposit
3. Withdraw
4. View Account Information
5. View Transaction History
6. Exit
Enter your choice: 5
Enter account number: 1001
Transaction History for Account Number 1001:
Type: Deposit   Amount: 5000
Type: Withdrawal    Amount: 2000
```

10. Design a program that dynamically allocates memory for an image processing application, allowing users to resize and manipulate images.

Code –

main.cpp

Run

```cpp
#include <iostream>
#include <cstdlib>

// Structure to represent an image
struct Image {
    int width;
    int height;
    unsigned char* data; // Pixel data (assuming grayscale images for simplicity)
};

// Function to create a new image
Image* createImage(int width, int height) {
    Image* img = new Image;
    img->width = width;
    img->height = height;
    img->data = static_cast<unsigned char*>(calloc(width * height, sizeof(unsigned char)));
    return img;
}

// Function to resize an image using realloc
void resizeImage(Image* &img, int newWidth, int newHeight) {
    unsigned char* newData = static_cast<unsigned char*>(realloc(img->data, newWidth * newHeight * sizeof(unsigned char)));
    if (newData) {
```

main.cpp

Run

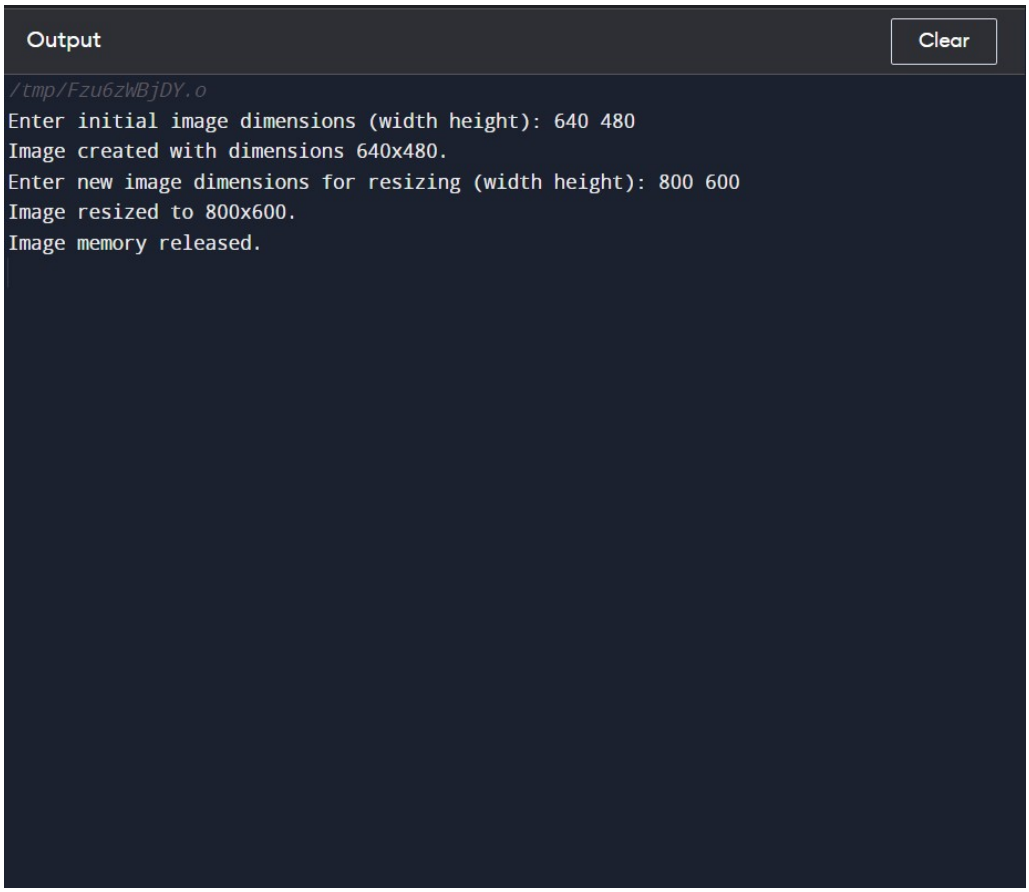
```
    if (newData) {
        img->data = newData;
        img->width = newWidth;
        img->height = newHeight;
        std::cout << "Image resized to " << newWidth << "x" << newHeight << "."
            << std::endl;
    } else {
        std::cout << "Memory reallocation failed. Image not resized." << std
            ::endl;
    }
}

// Function to release memory occupied by an image
void releaseImage(Image* &img) {
    if (img) {
        free(img->data);
        delete img;
        img = nullptr;
        std::cout << "Image memory released." << std::endl;
    }
}

int main() {
    int initialWidth, initialHeight, newWidth, newHeight;

    // Get initial image dimensions from the user
```

main.cpp

Run

```
    int initialWidth, initialHeight, newWidth, newHeight;

    // Get initial image dimensions from the user
    std::cout << "Enter initial image dimensions (width height): ";
    std::cin >> initialWidth >> initialHeight;

    // Create a new image
    Image* image = createImage(initialWidth, initialHeight);
    std::cout << "Image created with dimensions " << initialWidth << "x" <<
        initialHeight << "." << std::endl;

    // Get new image dimensions for resizing from the user
    std::cout << "Enter new image dimensions for resizing (width height): ";
    std::cin >> newWidth >> newHeight;

    // Resize the image
    resizeImage(image, newWidth, newHeight);

    // Perform more image processing operations...

    // Release memory occupied by the image
    releaseImage(image);

    return 0;
}
```

Output –


Output
Clear
/tmp/Fzu6zWBjDY.o
Enter initial image dimensions (width height): 640 480
Image created with dimensions 640x480.
Enter new image dimensions for resizing (width height): 800 600
Image resized to 800x600.
Image memory released.