Name: Gavali Deshabhakt Nagnath **Subject: Data Structures and Algorithms**

Roll No. 202cd005

Assignment-1

Index

1. Understand the object oriented paradigms. Encapsulation, polymorphism, inheritance, Abstraction. Implement using an object oriented programming language.

Programs:

1.1 Program Demonstrating Encapsulation

```
// Program Demonstrating Encapsulation
// Roll No. 202cd005
// Assignment 1
#include<iostream>
using namespace std;
class encapdemo
{
   private:
   int x;
   public:
   void set(int a)
   {
      x=a;
   }
   void get()
   {
      cout << “x = “ << x<<endl;
   }
};
int main()
{
   encapdemo demo;  // creating object 'demo' of encapdemo 'class'.
   demo.set(5);     // setting x=5 using set function.
                    // As x is declared private, hence we cannot access it directly.
                    // We can only access it using function of encapdemo class.
   demo.get();      // printing x using get function.
```

```
    return 0;
}
```

Output:

x = 5;

1.2.1 Program demonstrating compile time polymorphism

(a) Function Overloading

Program:

```
// Program demonstrating Compile time Polymorphism in c++;
// (a) Function Overloading
// Roll No. 202cd005
// Assingment 1
#include<iostream>
using namespace std;
class functionOverloadingDemo
{
    public:
    void print(int x)
    {
        cout << "Given Interger is " << x << endl;
    }
    void print(char x)
    {
        cout << "Given Character is " << x << endl;
    }
    void print(int x, int y)
    {
        cout << "Given intergers are " << x << " and " << y << endl;
    }
    void print(string s)
```

```
    {
        cout << "Given string is " << s << endl;
    }
};
int main()
{
    functionOverloadingDemo overload;
    overload.print(5);      // assingns value to print(int x) function
    overload.print('X');    // assingns value to print(char x) function
    overload.print(10,15);  // assingns value to print(int x, int y) function
    overload.print("'’Fuction overloading Demonstration’");    //assings value to print(string s)
    return 0;
}
```

Output:

```
Given Interger is 5
Given Character is X
Given intergers are 10 and 15
Given string is 'Fuction overloading Demonstration'
```

(b) Operator Overloading

Program:

```
// Program demonstrating Compile time Polymorphism in c++;
// (b) Operator Overloading
// Roll No. 202cd005
// Assingment 1
#include<iostream>
using namespace std;
class Matrix
{
   private:
   int** mat;
   int rows,columns;
   public:
   Matrix()          // Default Constructor
   {
      rows=0;
      columns=0;
   }
   Matrix(int r, int c) // Parametric Constructor
   {
      this->rows = r;
      this->columns = c;
      mat = new int *[rows];
      for (int i = 0; i < rows; i++)
      {
         mat[i] = new int[columns];
         for (int j = 0; j < columns; j++)
         {
```

```
            mat[i][j] = 0;
        }
    }
}
void matrixIN(int r, int c)
{
    this->rows = r;
    this->columns = c;
    cout << "Enter matrix elements row-wise" << endl;
    mat = new int *[rows];
    for (int i = 0; i < rows; i++)
    {
        mat[i] = new int[columns];
        for (int j = 0; j < columns; j++)
        {
            if (i == 0 && j == 0)
            {
                cout << "Enter First element: ";
                cin >> mat[i][j];
            }
            else
            {
                cout << "Enter next Element: ";
                cin >> mat[i][j];
            }
        }
    }
}
```

```
void matrixOUT(int r, int c)
{
    this->rows = r;
    this->columns = c;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            cout << mat[i][j] << " ";
        }
        cout << "\n";
    }
}
Matrix operator+(Matrix m) // overloading + operator for matrix addition
{
    this->rows = m.rows;
    this->columns = m.columns;
    Matrix temp(m.rows, m.columns);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            temp.mat[i][j] = this->mat[i][j] + m.mat[i][j];
        }
    }
    return temp;
}
Matrix operator-(Matrix m) // overloading - operator for matrix subtraction
```

```
    {

        this->rows = m.rows;
        this->columns = m.columns;
        Matrix temp(m.rows, m.columns);
        for (int i = 0; i < temp.rows; i++)
        {
            for (int j = 0; j < temp.columns; j++)
            {
                temp.mat[i][j] = this->mat[i][j] - m.mat[i][j];
            }
        }
        return temp;
    }
};
int main()
{
    cout << "Demonstration of Operator overloading using Matrix Operations " << endl;
    Matrix m1,m2,m3,m4;
    cout << “Enter elements of first Matrix “ <<endl;
    m1.matrixIN(2,2);
    cout << “Enter elements of second Matrix “ <<endl;
    m2.matrixIN(2,2);
    cout << "Matrix Addition is " << endl;
    m3=m1+m2;
    m3.matrixOUT(2,2);
    cout << "Matrix subtraction is " << endl;
    m4=m1-m2;
```

```
    m4.matrixOUT(2,2);
    return 0;
}
```

Output:

```
Demonstration of Operator overloading using Matrix Operations
Enter elements of first Matrix
Enter matrix elements row-wise
Enter First element: 1
Enter next Element: 2
Enter next Element: 3
Enter next Element: 4
Enter elements of second Matrix
Enter matrix elements row-wise
Enter First element: 5
Enter next Element: 6
Enter next Element: 7
Enter next Element: 8
Matrix Addition is
6 8
10 12
Matrix subtraction is
-4 -4
-4 -4
```

1.2.2 Program Demonstrating Runtime Polymorphism (function overriding)

Program:

```
// Program on Demonstration of Runtime polymorphism (i.e. implementation of function
    overriding)
// Roll No. 202cd005
// Assignment 1
#include<iostream>
using namespace std;
class base
```

```
{
   public:
   virtual void print()
   {
      cout << "Base class print fuction called." << endl;
   }
   void  get()
   {
      cout << "Base class get function called." << endl;
   }
};
class derived : public base
{
   public:
   void print()
   {
      cout << "Derived class print function called."<< endl;
   }
   void get()
   {
      cout << "Derived class get function called."<< endl;
   }
};
int main()
{
   cout << "Demonstration of Runtime polymorphism (Function overriding)"<<endl;
   base *b;
   derived d;
   b=&d;
```

```
    b->print();  // As print() function of base class is declared as "virtual" hence it does not have
                    execution
                 // Therefore, here derived class print() function overrides base class print()
                    function.
    b->get();    // As get() function of base class is not declared as "virtual" hence it is execute.
    return 0;
}
```

Output:

```
Demonstration of Runtime polymorphism (Function overriding)
Derived class print function called.
Base class get function called.
```

1.3 Program Demonstrating inheritance

Program:

```
// Program on Demonstration of Inheritance
// Roll No. 202cd005
// Assingment 1
#include<iostream>
using namespace std;
class parent
{
    public:
    int x,y;
    void setX(int a)
    {
        x=a;
    }
};
class child : public parent
{
    public:
```

```
    void printX()
    {
        cout << "X = " << x << endl;
    }
};
int main()
{
    child c;
    c.setX(5);  // Calling Parent Class setX() function using child class object
    c.printX();
    return 0;
}
```

Output:

X = 5

1.4 Program Demonstrating abstraction

Program:

```
// Program on demonstration of Abstraction
// Abstraction avoids code duplication and increases code reusibility
// Roll No. 202cd005
// Assingment 1
#include<iostream>
using namespace std;
#include <iostream>
using namespace std;
class abstractionDemo
{
    private:
        int a;
    public:
```

```
    void set(int x) // function to set values of private members
    {
        a = x;
    }
    void display()
    {
        cout << "a = " << a << endl;
    }
};
int main()
{
    abstractionDemo obj;
    obj.set(10);
    obj.display();
    obj.set(20);
    obj.display();
    return 0;
}
```

Output:

```
a = 10
a = 20
```

2. Design a data structure "Matrix" with all valid operations on it. Use operator/function overloading facilities. Use template class to implement the same to make it generic.

Program:

```
// Matrix operations with operator overloading
// Roll No. 202cd005

#include <iostream>
#include <limits>
#include<stdlib.h>
using namespace std;
template <class T>
class Matrix
{
private:
   int rows, columns, x, y;
   T **mat, **sparseMatrix;

public:
   Matrix()
   {
      rows = 0;
      columns = 0;
      x = 0;
      y = 0;
   }
   Matrix(int r, int c)
   {
      this->rows = r;
      this->columns = c;
      mat = new T *[rows];
```

```
    for (int i = 0; i < rows; i++)
    {
        mat[i] = new T[columns];
        for (int j = 0; j < columns; j++)
        {
            mat[i][j] = 0;
        }
    }
}
void matrixIN(int r, int c)
{
    this->rows = r;
    this->columns = c;
    mat = new T *[rows];
    for (int i = 0; i < rows; i++)
    {
        mat[i] = new T[columns];
        for (int j = 0; j < columns; j++)
        {
            if (i == 0 && j == 0)
            {
                cout << "Enter First element: ";
                cin >> mat[i][j];
            }
            else
            {
                cout << "Enter next Element: ";
                cin >> mat[i][j];
```

```
            }
        }
    }
}
void matrixOUT(int r, int c)
{
    this->rows = r;
    this->columns = c;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            cout << mat[i][j] << " ";
        }
        cout << "\n";
    }
}
T &operator()(int i, int j) const
{
    return mat[i - 1][j - 1];
}
Matrix operator+(Matrix m)
{
    this->rows = m.rows;
    this->columns = m.columns;
    Matrix temp(m.rows, m.columns);
    for (int i = 0; i < rows; i++)
    {
```

```
        for (int j = 0; j < columns; j++)
        {
            temp.mat[i][j] = this->mat[i][j] + m.mat[i][j];
        }
    }
    return temp;
}
Matrix operator-(Matrix m)
{

    this->rows = m.rows;
    this->columns = m.columns;
    Matrix temp(m.rows, m.columns);
    for (int i = 0; i < temp.rows; i++)
    {
        for (int j = 0; j < temp.columns; j++)
        {
            temp.mat[i][j] = this->mat[i][j] - m.mat[i][j];
        }
    }
    return temp;
}
Matrix operator*(Matrix m)
{
    this->rows = m.rows;
    this->columns = m.columns;
    Matrix temp(rows, columns);
    for (int i = 0; i < rows; i++)
```

```
    {
        for (int j = 0; j < columns; j++)
        {
            for (int k = 0; k < rows; k++)
            {
                temp.mat[i][j] += this->mat[i][k] * m.mat[k][j];
            }
        }
    }
    return temp;
}
Matrix elementwisemultiplication(Matrix m)
{
    this->rows = m.rows;
    this->columns = m.columns;
    Matrix temp(rows, columns);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            temp.mat[i][j] += this->mat[i][j] * m.mat[i][j];
        }
    }
    return temp;
}
Matrix Transpose(Matrix m)
{
    this->rows = m.rows;
```

```
    this->columns = m.columns;
    Matrix temp(rows, columns);
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            temp.mat[i][j] = this->mat[j][i];
        }
    }
    return temp;
}
bool checkSparse()
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            if (mat[i][j] == 0)
            {
                x++;
            }
            else
            {
                y++;
            }
        }
    }
    if (x > y)
```

```
    {
        return true;
    }
    else
    {
        return false;
    }
}
void setSparse()
{
    int c = y;
    int k = 0;
    sparseMatrix = new T *[3];
    for (int i = 0; i < 3; i++)
    {
        sparseMatrix[i] = new T[c];
        for (int j = 0; j < c; j++)
        {
            sparseMatrix[i][j] = 0;
        }
    }
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
            if (mat[i][j] != 0)
            {
                sparseMatrix[0][k] = i;
                sparseMatrix[1][k] = j;
```

```
                sparseMatrix[2][k] = mat[i][j];
                k++;
            }
        }
    }
    void displaySparse()
    {
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < y; j++)
            {
                cout << sparseMatrix[i][j] << " ";
            }
            cout << "\n";
        }
    }
    void generalBlockforMatrixOperationsInMainFuction(Matrix m1, Matrix m2, Matrix m3, Matrix
m4)
    {
        int r1, c1, r2, c2, s;
        char ch;
        cout << "Enter number of rows of first matrix: ";
        cin >> r1;
        cout << "Enter number of columns of first matrix: ";
        cin >> c1;
        if (r1 <= 0 || c1 <= 0 || isdigit(r1) == true || isdigit(c1) == true)
        {
            cout << "Number of rows and columns should be positive intergers";
```

```
    }
    else
    {
        cout << "Enter elements of first matrix row-wise: " << endl;
        m1.matrixIN(r1, c1);
    }
    cout << "Enter number of rows of second matrix: ";
    cin >> r2;
    cout << "Enter number of columns of second matrix: ";
    cin >> c2;
    if (r2 <= 0 || c2 <= 0 || isdigit(r2) == true || isdigit(c2) == true)
    {
        cout << "Number of rows and columns should be positive intergers" << endl;
    }
    else
    {
        cout << "Enter elements of second matrix row-wise: " << endl;
        m2.matrixIN(r2, c2);
    }
    do
    {
        cout << "1.Addition\n2.Subtraction\n3.Multiplication\n4.Element wise multiplication\
n5.Transpose\n6.Check if Sparse or not\n7.Clear Screen\n8.Go To previous Menu" << endl;
        cout << "Which operation do you want to perform?: ";
        cin >> s;
        switch (s)
        {
        case 1:
```

```
        {
            if (r1 == r2 && c1 == c2)
            {
                m3 = m1 + m2;
                cout << "Matrix addition is " << endl;
                m3.matrixOUT(r1, c1);
            }
            else
            {
                cout << "Matrix addition cannot be done because matrices are of different size." << endl;
            }
            break;
        }
        case 2:
        {
            if (r1 == r2 && c1 == c2)
            {
                m3 = m1 - m2;
                cout << "Matrix subtraction is " << endl;
                m3.matrixOUT(r1, c1);
            }
            else
            {
                cout << "Matrix subtraction cannot be done because matrices are of different size." <<
endl;
            }
            break;
        }
```

```
        case 3:
        {
            if (c1 = r2)
            {
                m3 = m1 * m2;
                cout << "Matrix multiplication is " << endl;
                m3.matrixOUT(r1, c2);
            }
            else
            {
                cout << "Matrix multiplication cannot be done.\nCheck size of matrices." << endl;
            }
            break;
        }
        case 4:
        {
            if (r1 == r2 && c1 == c2)
            {
                m3 = m1.elementwisemultiplication(m2);
                cout << "Matrix Element wise multiplication is " << endl;
                m3.matrixOUT(r1, c1);
            }
            else
            {
                cout << "Element wise multiplication cannot be done.\nMatrices are of different size." <<
endl;
            }
            break;
```

```
}
case 5:
{
    cout << "Transpose of first matrix is: " << endl;
    m3 = m1.Transpose(m1);
    m3.matrixOUT(c1, r1);
    cout << "Transpose of second matrix is: " << endl;
    m4 = m2.Transpose(m2);
    m4.matrixOUT(c2, r2);
    break;
}
case 6:
{
    if (m1.checkSparse() == true)
    {
        cout << "First Matrix is a sparse matrix." << endl;
        m1.setSparse();
        m1.displaySparse();
    }
    else if (m1.checkSparse() == false)
    {
        cout << "First Matrix is not a sparse matrix." << endl;
    }
    if (m2.checkSparse() == true)
    {
        cout << "Second matrix is a sparse matrix." << endl;
        m2.setSparse();
        m2.displaySparse();
```

```
            }
            else if (m2.checkSparse() == false)
            {
                cout << "Second matrix is not a sparse matrix." << endl;
            }
            break;
        }
        case 7:
        {
            system("clear");
            break;
        }
        case 8:
        {
            break;
        }
        default:
        {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin.clear();
            cout << "Invalid input.\nTry again! :)" << endl;
            break;
        }
        }
    }while(s!=8);
  }
};
int main()
```

```
{
    char ch;
    int s;
    do
    {
        cout << "\
n================================================================================
=============" << endl;
        cout << "************************** Matrix Operations using 2D Array
**************************" << endl;
        cout <<
"================================================================================
=============" << endl;
        cout << "\nSelect Data type: " << endl;
        cout << "1.INT      2.DOUBLE       3.FLOAT       4.Clear Screen       5.End Program" << endl;
        cout << ">";
        cin >> s;
        if (s == 1)
        {
            cout << "Selected data type: INT" << endl;
            Matrix<int> m1, m2, m3, m4, callobj;
            callobj.generalBlockforMatrixOperationsInMainFuction(m1, m2, m3, m4);
        }
        else if (s == 2)
        {
            cout << "Selected data type: DOUBLE" << endl;
            Matrix<double> m1, m2, m3, m4, callobj;
            callobj.generalBlockforMatrixOperationsInMainFuction(m1, m2, m3, m4);
        }
```

```
        else if (s == 3)
        {
            cout << "Selected data type: FLOAT" << endl;
            Matrix<float> m1, m2, m3, m4, callobj;
            callobj.generalBlockforMatrixOperationsInMainFuction(m1, m2, m3, m4);
        }
        else if(s==4)
        {
            system("clear");
        }
        else if(s==5)
        {
            cout << "\nProgram Ended"<< endl;
            cout << "\
n**********************************************************************************
**" << endl;
            break;
        }
        else
        {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin.clear();
            cout << "Invalid Input!\nTry Again:)" << endl;
        }
    }while(s!=5);
    return 0;
}
```

Output:

```
================================================================================
============
************************* Matrix Operations using 2D Array **************************
================================================================================
============

Select Data type:
1.INT     2.DOUBLE     3.FLOAT     4.Clear Screen     5.End Program
>1
Selected data type: INT
Enter number of rows of first matrix: 2
Enter number of columns of first matrix: 2
Enter elements of first matrix row-wise:
Enter First element: 1
Enter next Element: 2
Enter next Element: 3
Enter next Element: 4
Enter number of rows of second matrix: 2
Enter number of columns of second matrix: 2
Enter elements of second matrix row-wise:
Enter First element: 5
Enter next Element: 6
Enter next Element: 7
Enter next Element: 8
1.Addition
2.Subtraction
3.Multiplication
4.Element wise multiplication
5.Transpose
6.Check if Sparse or not
7.Clear Screen
```

8.Go To previous Menu

Which operation do you want to perform?: 1

Matrix addition is

6 8

10 12

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 2

Matrix subtraction is

-4 -4

-4 -4

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 3

Matrix multiplication is

19 22

43 50

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 4

Matrix Element wise multiplication is

5 12

21 32

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 5

Transpose of first matrix is:

1 3

2 4

Transpose of second matrix is:

5 7

6 8

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 6

First Matrix is not a sparse matrix.

Second matrix is not a sparse matrix.

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?: 7

1.Addition

2.Subtraction

3.Multiplication

4.Element wise multiplication

5.Transpose

6.Check if Sparse or not

7.Clear Screen

8.Go To previous Menu

Which operation do you want to perform?:8

==========

********** Matrix Operations using 2D Array **********

==========

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.Clear Screen 5.End Program

>5

Program Ended

****
*

Note:

Same operations can be repeated with data types DOUBLE and FLOAT.

3. Design a "List" data structure with all valid operations. Implement using Template class.

Program:

```
// List operations
// Roll No. 202cd005

#include <iostream>
#include <limits>
#include <stdlib.h>
using namespace std;
template <class T>
class mylist
{
private:
   char ch;
   int user_input_for_switch, arraysize;
   T element;
   T *list;

public:
   mylist(T arr[])
   {
      arraysize=arraySize(arr);
   }
```

```
void setArray(const int &size)
{
    list = new T[size];
    for (int i = 0; i < size; i++)
    {
        cout << ">";
        cin >> list[i];
    }
}
void getArray(unsigned int x, T arr[])
{
    cout << "{";
    for (int i = 0; i < x; i++)
    {
        cout << arr[i];
        if (i < x - 1)
        {
            cout << ",";
        }
    }
    cout << "}" << endl;
}
void deleteElement(const int index, T arr[])
{
    if (index > arraysize)
    {
        cout << "Index out of range";
    }
```

```
    else
    {
        for (int i = 0; i < arraysize; i++)
        {
            if (i >= index)
            {
                arr[i] = arr[i + 1];
            }
        }
        cout << "Array after deletion : ";
        arraysize--;
        getArray(arraysize, arr);
    }
}
unsigned int arraySize(T arr[])
{
    int i = 0;
    while (arr[i] != '\0')
    {
        i++;
    }
    return i;
}
const int fetchIndex(T element_whose_index_to_be_fetched, T arr[])
{
    int s = '\0', i;
    T x = element_whose_index_to_be_fetched;
    for (i = 0; i < arraysize; i++)
```

```
    {
        if (x == arr[i])
        {
            s = arr[i];
            break;
        }
    }
    if (s != '\0')
    {
        cout << "index of element " << x << " is " << i << endl;
    }
    else
    {
        cout << "Element not in list" << endl;
    }

    return i;
}
void fetchElement(const int index, T arr[])
{
    if (index < arraysize)
    {
        cout << "Element present at index " << index << " is " << arr[index] << "." << endl;
    }
    else
    {
        cout << "Index out of range." << endl;
    }
```

```
}
void insertElement(const int index, T element_to_be_added, T arr[])
{
    if (index > arraysize)
    {
        cout << "Index out of range." << endl;
    }
    else if (checkIfElementToBeAddedIsPresentInListORnot(element_to_be_added,arr) >0)
    {
        cout << "Same element already present in list." << endl;
        cout << "Original list is : ";
        getArray(arraysize,arr);
    }
    else
    {
        for (int i = 0; i <= arraysize; i++)
        {
            if(i==index)
            {
                for (int j = arraysize - 1; j >= i; j--)
                {
                    arr[j + 1] = arr[j];
                }
                arr[i] = element_to_be_added;
                cout << "Updated list is : ";
                getArray(++arraysize, arr);
                break;
            }
```

```
        }
    }
}
int checkIfElementToBeAddedIsPresentInListORnot(T element_to_be_added,T arr[])
{
    int temp=0;
    for (int i = 0; i <= arraysize; i++)
    {
        if (arr[i] == element_to_be_added)
        {
            temp++;
        }
        else
        {
            continue;
        }
    }
    return temp;
}
bool empty(T arr[])
{
    if (arraysize == 0)
    {
        return true;
    }
    else
    {
        return false;
```

```
    }
}
void commonSwitchCaseBlockforBuiltInList(mylist ls, T builtInList[])
{
    do
    {
        cout << "Using buit-in list...." << endl;
        cout << "Which Operation do you want to perform?: "
             << " \n 1.Check if list is empty"
             << "\n 2.Print List"
             << "\n 3.Size of List"
             << "\n 4.Delete element from list"
             << "\n 5.Fetch index of an element "
             << "\n 6.Fetch element from list "
             << "\n 7.Insert element in list"
             << "\n 8.Clear Screen"
             << "\n 9.Go To Previous menu" <<endl;
        cout << ">";
        cin >> user_input_for_switch;
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.clear();
        if (ls.arraySize(builtInList) > 0)
        {
            switch (user_input_for_switch)
            {
            case 1:
                if(ls.empty(ls.list))
                {
```

```
        cout << "List is empty."<<endl;
    }
    else
    {
        cout << "List is not empty."<<endl;
    }
    break;
case 2:
    cout << "The list is ";
    ls.getArray(ls.arraySize(builtInList), builtInList);
    break;
case 3:
    cout << "Size of list is " << ls.arraySize(builtInList) << "." << endl;
    break;
case 4:
    int del_idx;
    cout << "Enter index number of element which you want to delete: ";
    cin >> del_idx;
    ls.deleteElement(del_idx, builtInList);
    break;
case 5:
    cout << "Enter element who's index number you want to find: ";
    cin >> ls.element;
    ls.fetchIndex(ls.element, builtInList);
    break;
case 6:
    int fetch_element_at_index;
    cout << "Enter index number of element which you want to find: ";
```

```
            cin >> fetch_element_at_index;
            ls.fetchElement(fetch_element_at_index, builtInList);
            break;
        case 7:
            int in_idx;
            cout << "Enter index at which you want to insert element: " << endl;
            cin >> in_idx;
            cout << "Enter element which you want to add: " << endl;
            cin >> ls.element;
            ls.insertElement(in_idx, ls.element, builtInList);
            break;
        case 8:
            system("clear");
            break;
        case 9:
            break;
        default:
            cout << "Invalid input! please try agian." << endl;
        }
    }
    else
    {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.clear();
        cout << "List is empty.\nCannot perform opeartions." << endl;
    }
  }while(user_input_for_switch!=9);
}
```

```
void commonSwitchCaseBlockforDynamicList(mylist ls)
{
   do
   {
      cout << "Which Operation do you want to perform?: "
          << " \n 1.Check if list is empty"
          << "\n 2.Print List"
          << "\n 3.Size of List"
          << "\n 4.Delete element from list"
          << "\n 5.Fetch index of an element "
          << "\n 6.Fetch element from list "
          << "\n 7.Insert element in list"
          << "\n 8.Clear Screen"
          << "\n 9.Go To Previous menu" << endl;
      cout << ">";
      cin >> user_input_for_switch;
      if (ls.arraySize(ls.list) > 0)
      {
         switch (user_input_for_switch)
         {
         case 1:
            if(ls.empty(ls.list))
            {
               cout << "List is empty."<<endl;
            }
            else
            {
               cout << "List is not empty."<<endl;
```

```
    }
    break;
case 2:
    cout << "The list is ";
    ls.getArray(ls.arraySize(ls.list), ls.list);
    break;
case 3:
    cout << "Size of list is " << ls.arraySize(ls.list) << "." << endl;
    break;
case 4:
    int del_idx;
    cout << "Enter index number of element which you want to delete: ";
    cin >> del_idx;
    ls.deleteElement(del_idx, ls.list);
    break;
case 5:
    cout << "Enter element who's index number you want to find: ";
    cin >> ls.element;
    ls.fetchIndex(ls.element, ls.list);
    break;
case 6:
    int fetch_element_at_index;
    cout << "Enter index number of element which you want to find: ";
    cin >> fetch_element_at_index;
    ls.fetchElement(fetch_element_at_index, ls.list);
    break;
case 7:
    int in_idx;
```

```
                cout << "Enter index at which you want to insert element: " << endl;
                cin >> in_idx;
                cout << "Enter element which you want to add: " << endl;
                cin >> ls.element;
                ls.insertElement(in_idx, ls.element, ls.list);
                break;
            case 8:
                system("clear");
                break;
            case 9:
                break;
            default:
                cout << "Invalid input! please try agian." << endl;
            }
        }
        else
        {
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin.clear();
            cout << "List is empty.\nCannot perform opeartions." << endl;
        }
    }while(user_input_for_switch!=9);
  }
};
int main()
{
  char ch;
  int index, store_size, varForDataTypeSelection, varForListSizeInputFromUser;
```

```
    do
    {
        cout <<"\
n================================================================================
=============="<<endl;
        cout <<"******************************** Linear list Operations
*****************************"<<endl;
        cout
<<"================================================================================
==============="<<endl;
        cout << "\nSelect Data type: " << endl;
        cout << "1.INT       2.CHAR      3.FLOAT      4.Clear Screen        5.End Program" << endl;
        cout << ">";
        cin >> varForDataTypeSelection;
        if (varForDataTypeSelection == 1)
        {
            cout <<"Selected data type: INT"<<endl;
            int builtInList[10] = {1,2,3,4,5,6,7};
            mylist<int> ls(builtInList), callobj(builtInList);
            cout << "Chose between following options:"
                << "\nEnter 1 if you want to use buit-in list"
                << "\nEnter 2 if you want to use your own list"
                << "\n> ";
            int opt_user;
            cin >> opt_user;
            if (opt_user == 1)
            {
                callobj.commonSwitchCaseBlockforBuiltInList(ls, builtInList);
```

```
    }
    else if (opt_user == 2)
    {
        cout << "Enter size of list: ";
        cin >> varForListSizeInputFromUser;
        cout << "\nEnter List elements: " << endl;
        ls.setArray(varForListSizeInputFromUser);
        callobj.commonSwitchCaseBlockforDynamicList(ls);
    }
}
else if (varForDataTypeSelection == 2)
{
    cout <<"Selected data type: CHAR"<<endl;
    char builtInList[10] = {'a','b','c','d','e','f','g'};
    mylist<char> ls(builtInList), callobj(builtInList);
    cout << "Chose between following options:"
         << "\nEnter 1 if you want to use buit-in list"
         << "\nEnter 2 if you want to use your own list"
         << "\n> ";
    int opt_user;
    cin >> opt_user;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin.clear();
    if (opt_user == 1)
    {
        callobj.commonSwitchCaseBlockforBuiltInList(ls, builtInList);
    }
    else if (opt_user == 2)
```

```
    {
        cout << "Enter size of list: ";
        cin >> varForListSizeInputFromUser;
        cout << "\nEnter List elements: " << endl;
        ls.setArray(varForListSizeInputFromUser);
        callobj.commonSwitchCaseBlockforDynamicList(ls);
    }
}
else if (varForDataTypeSelection == 3)
{
    cout <<"Selected data type: FLOAT"<<endl;
    float builtInList[10] = {1.5,2.5,3.5,4.5,5.5};
    mylist<float> ls(builtInList), callobj(builtInList);
    cout << "Chose between following options:"
         << "\nEnter 1 if you want to use buit-in list"
         << "\nEnter 2 if you want to use your own list"
         << "\n> ";
    int opt_user;
    cin >> opt_user;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin.clear();
    if (opt_user == 1)
    {
        callobj.commonSwitchCaseBlockforBuiltInList(ls, builtInList);
    }
    else if (opt_user == 2)
    {
        cout << "Enter size of list: ";
```

```
            cin >> varForListSizeInputFromUser;
            cout << "\nEnter List elements: " << endl;
            ls.setArray(varForListSizeInputFromUser);
            callobj.commonSwitchCaseBlockforDynamicList(ls);
        }
    }
    else if(varForDataTypeSelection == 4)
    {
        system("clear");
    }
    else if(varForDataTypeSelection == 5)
    {
        cout << "\nProgram Ended"<<endl;
        cout<<"\
n==========================================================================
============="<<endl;
        break;
    }
    else
    {
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.clear();
        cout << "Invalid input!\nTry again :)";
    }
  }while(varForDataTypeSelection!=5);
  return 0;
}
```

Output:

```
================================================================================
============
****************************** Linear list Operations ******************************
================================================================================
============

Select Data type:
1.INT      2.CHAR      3.FLOAT      4.Clear Screen      5.End Program
>1
Selected data type: INT
Chose between following options:
Enter 1 if you want to use buit-in list
Enter 2 if you want to use your own list
> 1
Using buit-in list....
Which Operation do you want to perform?:
 1.Check if list is empty
 2.Print List
 3.Size of List
 4.Delete element from list
 5.Fetch index of an element
 6.Fetch element from list
 7.Insert element in list
 8.Clear Screen
 9.Go To Previous menu
>1
List is not empty.
Using buit-in list....
Which Operation do you want to perform?:
```

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>2

The list is {1,2,3,4,5,6,7}

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>3

Size of list is 7.

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>4

Enter index number of element which you want to delete: 1

Array after deletion : {1,3,4,5,6,7}

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>5

Enter element who's index number you want to find: 2

Element not in list

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>6

Enter index number of element which you want to find: 4

Element present at index 4 is 6.

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>7

Enter index at which you want to insert element:

4

Enter element which you want to add:

10

Updated list is : {1,3,4,5,10,6,7,0}

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>8

Using buit-in list....

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>9

==========

********** Linear list Operations **********

==========

Select Data type:

1.INT 2.CHAR 3.FLOAT 4.Clear Screen 5.End Program

>1

Selected data type: INT

Chose between following options:

Enter 1 if you want to use buit-in list

Enter 2 if you want to use your own list

> 2

Enter size of list: 4

Enter List elements:

>1

>2

>3

>4

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>1

List is not empty.

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>2

The list is {1,2,3,4}

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>3

Size of list is 4.

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>4

Enter index number of element which you want to delete: 2

Array after deletion : {1,2,4}

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>5

Enter element who's index number you want to find: 1

index of element 1 is 0

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>6

Enter index number of element which you want to find: 2

Element present at index 2 is 4.

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>7

Enter index at which you want to insert element:

2

Enter element which you want to add:

10

Updated list is : {1,2,10,4}

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>8

Which Operation do you want to perform?:

1.Check if list is empty

2.Print List

3.Size of List

4.Delete element from list

5.Fetch index of an element

6.Fetch element from list

7.Insert element in list

8.Clear Screen

9.Go To Previous menu

>9

==========

********** Linear list Operations **********

==========

Select Data type:

1.INT 2.CHAR 3.FLOAT 4.Clear Screen 5.End Program

>5

Program Ended

==========

Note:

Same operations can be repeated with data types CHAR and FLOAT.

4. Design a Linked list (with single and double link) with all valid operations.

(a) Singly Linked List operations

Program:

```
// Singly Linked List Execution and its operations
// Roll No. 202cd005

#include <iostream>
#include <stdlib.h>
#include <limits>
using namespace std;

template <class T>
class Node
{
public:
   int key;
   T data;
   Node<T> *next;
   Node()
   {
      key = 0;
      data = 0;
      next = NULL;
   }
   Node(int k, T d)
   {
      key = k;
      data = d;
   }
```

```
};
template <class T>
class SinglyLinkedList
{
public:
    Node<T> *head;
    SinglyLinkedList()
    {
        head = NULL;
    }
    SinglyLinkedList(Node<T> *n)
    {
        head = n;
    }
    Node<T> *nodeExists(int k)
    {
        Node<T> *temp = NULL;
        Node<T> *ptr = head;
        while (ptr != NULL)
        {
            if (ptr->key == k)
            {
                temp = ptr;
            }
            ptr = ptr->next;
        }
        return temp;
    }
```

```
void appendNode(Node<T> *n)
{
    if (nodeExists(n->key) != NULL)
    {
        cout << "Node already exists with key value" << n->key << "." << endl;
    }
    else
    {
        if (head == NULL)
        {
            head = n;
            cout << "Node Appended Successfully." << endl;
        }
        else
        {
            Node<T> *ptr = head;
            while (ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = n;
            cout << "Node Appended Successfully." << endl;
        }
    }
}
void prependNode(Node<T> *n)
{
    if (nodeExists(n->key) != NULL)
```

```
    {
        cout << "Node already exists with key value" << n->key << "." << endl;
    }
    else
    {
        n->next = head;
        head = n;
        cout << "Node Prepended Successfully." << endl;
    }
}
void insertNodeAfter(int k, Node<T> *n)
{
    Node<T> *ptr = nodeExists(k);
    if (ptr == NULL)
    {
        cout << "No node Exists with key value " << k << "." << endl;
    }
    else
    {
        if (nodeExists(n->key) != NULL)
        {
            cout << "Node already exists with key value" << n->key << "." << endl;
        }
        else
        {
            n->next = ptr->next;
            ptr->next = n;
        }
```

```
    }
}
void deleteNodeByKey(int k)
{
    if (head == NULL)
    {
        cout << "Singly Linked List is empty.\nCannot perform deletion." << endl;
    }
    else if (head != NULL)
    {
        if (head->key == k)
        {
            head = head->next;
            cout << "Node unlinked with key value " << k << "." << endl;
        }
        else
        {
            Node<T> *temp = NULL;
            Node<T> *prevptr = head;
            Node<T> *currentptr = head->next;
            while (currentptr != NULL)
            {
                if (currentptr->key == k)
                {
                    temp = currentptr;
                    currentptr = NULL;
                }
                else
```

```
            {
                prevptr = prevptr->next;
                currentptr = currentptr->next;
            }
        }
        if (temp != NULL)
        {
            prevptr->next = temp->next;
            cout << "Node unlinked with key value " << k << " successfully." << endl;
        }
        else
        {
            cout << "Node doesn't exists with key value " << k << "." << endl;
        }
    }
  }
}
void updateNodeByKey(int k, T d)
{
  Node<T> *ptr = nodeExists(k);
  if (ptr != NULL)
  {
    ptr->data = d;
    cout << "Node data updated successfully." << endl;
  }
  else
  {
    cout << "Node doesn't exist with key value " << k << "." << endl;
```

```
        }
    }
    void printLinkedList()
    {
        if (head == NULL)
        {
            cout << "No nodes in singly linked list" << endl;
        }
        else
        {
            Node<T> *temp = head;
            while (temp != NULL)
            {
                cout << "(" << temp->key << "," << temp->data << ")   ";
                temp = temp->next;
            }
        }
    }
};
int main()
{
    int varForDataTypeSelection, varForSwitchCase, key1, k1;
    do
    {
        cout << "\
n================================================================================
=============" << endl;
        cout << "***************************** Singly Linked List Operations
************************" << endl;
```

```
        cout <<
"================================================================================
=============" << endl;
        cout << "\nSelect Data type: " << endl;
        cout << "1.INT     2.DOUBLE    3.FLOAT     4.CHAR     5.Clear Screen     6.Exit from program"
<< endl;
        cout << ">";
        cin >> varForDataTypeSelection;
        std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        std::cin.clear();
        if (varForDataTypeSelection == 1)
        {
            int data1;
            cout << "Selected Data type: INT" << endl;
            cout << "Which Operation Do you want to perform?" << endl;
            SinglyLinkedList<int> s, callobj;
            do
            {
                cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print singly linked list\n7.Clear Screen\n8.Go to previous menu\n> ";
                cin >> varForSwitchCase;
                Node<int> *n1 = new Node<int>();
                switch (varForSwitchCase)
                {
                case 1:
                {
                    cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                    cout << ">";
```

```
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.nodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
```

```
            cout << ">";
            cin >> key1;
            cout << ">";
            cin >> data1;
            n1->key = key1;
            n1->data = data1;
            s.insertNodeAfter(k1, n1);
        }
        else
        {
            cout << "No node exists with entered key value." << endl;
        }
        break;
    }
    case 4:
    {
        cout << "Delete Node Operation (Deletion by key of node)" << endl;
        cout << "Enter key value of node to be deleted:" << endl;
        cout << ">";
        cin >> k1;
        s.deleteNodeByKey(k1);
        break;
    }
    case 5:
    {
        cout << "Update data of Node with key value" << endl;
        cout << "Enter key of node who's data is to be updated: ";
        cin >> k1;
```

```
            cout << "\nEnter new data: ";
            cin >> data1;
            cout << endl;
            s.updateNodeByKey(k1, data1);
            break;
        }
        case 6:
        {
            cout << "Linked list is: " << endl;
            s.printLinkedList();
            break;
        }
        case 7:
        {
            system("clear");
            break;
        }
        case 8:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
```

```
            }
            }
        } while (varForSwitchCase != 8);
    }
    else if (varForDataTypeSelection == 2)
    {
        double data1;
        cout << "Selected Data type: DOUBLE" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        SinglyLinkedList<double> s;
        do
        {
            Node<double> *n1 = new Node<double>();
            cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print singly linked list\n7.Clear Screen\n8.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin.clear();
            switch (varForSwitchCase)
            {
            case 1:
            {
                cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                cout << ">";
                cin >> key1;
                cout << ">";
                cin >> data1;
                n1->key = key1;
```

```
    n1->data = data1;
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.nodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
        cin >> data1;
```

```
            n1->key = key1;
            n1->data = data1;
            s.insertNodeAfter(k1, n1);
        }
        else
        {
            cout << "No node exists with entered key value." << endl;
        }
        break;
    }
    case 4:
    {
        cout << "Delete Node Operation (Deletion by key of node)" << endl;
        cout << "Enter key value of node to be deleted:" << endl;
        cout << ">";
        cin >> k1;
        s.deleteNodeByKey(k1);
        break;
    }
    case 5:
    {
        cout << "Update data of Node with key value" << endl;
        cout << "Enter key of node who's data is to be updated: ";
        cin >> k1;
        cout << "\nEnter new data: ";
        cin >> data1;
        cout << endl;
        s.updateNodeByKey(k1, data1);
```

```
            break;
        }
        case 6:
        {
            cout << "Linked list is: " << endl;
            s.printLinkedList();
            break;
        }
        case 7:
        {
            system("clear");
            break;
        }
        case 8:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
        }
        }
    } while (varForSwitchCase != 8);
}
```

```
    else if (varForDataTypeSelection == 3)
    {
        float data1;
        cout << "Selected Data type: FLOAT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        SinglyLinkedList<float> s;
        do
        {
            Node<float> *n1 = new Node<float>();
            cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print singly linked list\n7.Clear Screen\n8.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cin.clear();
            switch (varForSwitchCase)
            {
            case 1:
            {
                cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                cout << ">";
                cin >> key1;
                cout << ">";
                cin >> data1;
                n1->key = key1;
                n1->data = data1;
                s.appendNode(n1);
                break;
            }
```

```
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.nodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
        cin >> data1;
        n1->key = key1;
        n1->data = data1;
        s.insertNodeAfter(k1, n1);
    }
```

```
    else
    {
        cout << "No node exists with entered key value." << endl;
    }
    break;
}
case 4:
{
    cout << "Delete Node Operation (Deletion by key of node)" << endl;
    cout << "Enter key value of node to be deleted:" << endl;
    cout << ">";
    cin >> k1;
    s.deleteNodeByKey(k1);
    break;
}
case 5:
{
    cout << "Update data of Node with key value" << endl;
    cout << "Enter key of node who's data is to be updated: ";
    cin >> k1;
    cout << "\nEnter new data: ";
    cin >> data1;
    cout << endl;
    s.updateNodeByKey(k1, data1);
    break;
}
case 6:
{
```

```
                cout << "Linked list is: " << endl;
                s.printLinkedList();
                break;
            }
            case 7:
            {
                system("clear");
                break;
            }
            case 8:
            {
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            default:
            {
                cout << "Invalid Input.\nTry Again:)" << endl;
                break;
            }
            }
        } while (varForSwitchCase != 8);
    }
    else if (varForDataTypeSelection == 4)
    {
        char data1;
        cout << "Selected Data type: CHAR" << endl;
```

```
cout << "Which Operation Do you want to perform?" << endl;
SinglyLinkedList<char> s;
do
{
   Node<char> *n1 = new Node<char>();
   cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print singly linked list\n7.Clear Screen\n8.Go to previous menu\n> ";
   cin >> varForSwitchCase;
   std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
   std::cin.clear();
   switch (varForSwitchCase)
   {
   case 1:
   {
      cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
      cout << ">";
      cin >> key1;
      cout << ">";
      cin >> data1;
      n1->key = key1;
      n1->data = data1;
      s.appendNode(n1);
      break;
   }
   case 2:
   {
      cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
      cout << ">";
```

```
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.nodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
        cin >> data1;
        n1->key = key1;
        n1->data = data1;
        s.insertNodeAfter(k1, n1);
    }
    else
    {
        cout << "No node exists with entered key value." << endl;
    }
```

```
    break;
}
case 4:
{
    cout << "Delete Node Operation (Deletion by key of node)" << endl;
    cout << "Enter key value of node to be deleted:" << endl;
    cout << ">";
    cin >> k1;
    s.deleteNodeByKey(k1);
    break;
}
case 5:
{
    cout << "Update data of Node with key value" << endl;
    cout << "Enter key of node who's data is to be updated: ";
    cin >> k1;
    cout << "\nEnter new data: ";
    cin >> data1;
    cout << endl;
    s.updateNodeByKey(k1, data1);
    break;
}
case 6:
{
    cout << "Linked list is: " << endl;
    s.printLinkedList();
    break;
}
```

```
            case 7:
            {
               system("clear");
               break;
            }
            case 8:
            {
               cout <<
"================================================================================
=============" << endl;
               break;
            }
            default:
            {
               cout << "Invalid Input.\nTry Again:)" << endl;
               break;
            }
            }
         } while (varForSwitchCase != 8);
      }
      else if (varForDataTypeSelection == 5)
      {
         system("clear");
      }
      else if (varForDataTypeSelection == 6)
      {
         cout << "\n\nThanks for Using Program!" << endl;
         cout << "Program Ended\n\n" << endl;
```

```
            cout <<
"================================================================================
=============" << endl;
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        else
        {
            cout << "Invalid input!\nTry Again:)" << endl;
        }

    } while (varForDataTypeSelection != 4);
    return 0;
}
```

Output:

```
================================================================================
============
**************************** Singly Linked List Operations ************************
================================================================================
============

Select Data type:
1.INT     2.DOUBLE    3.FLOAT     4.CHAR     5.Clear Screen     6.Exit from program
>1
Selected Data type: INT
Which Operation Do you want to perform?
```

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 1

Append Node Operation

Enter Key & data of node to be appended:

>1

>10

Node Appended Successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 2

Prepend Node Operation

Enter key and data of node to be prepended:

>0

>12

Node Prepended Successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 3

Insert Node Operation

Enter key of node after which you want to insert a node:

>1

Enter key and data of node to be inserted:

>2

>10

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 1

Append Node Operation

Enter Key & data of node to be appended:

>4

>10

Node Appended Successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 6

Linked list is:

(0,12) (1,10) (2,10) (4,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 3

Insert Node Operation

Enter key of node after which you want to insert a node:

>2

Enter key and data of node to be inserted:

>3

>25

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 6

Linked list is:

(0,12) (1,10) (2,10) (3,25) (4,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 4

Delete Node Operation (Deletion by key of node)

Enter key value of node to be deleted:

>3

Node unlinked with key value 3 successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 6

Linked list is:

(0,12) (1,10) (2,10) (4,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 5

Update data of Node with key value

Enter key of node who's data is to be updated: 2

Enter new data: 25

Node data updated successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 6

Linked list is:

(0,12) (1,10) (2,25) (4,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

> 7

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print singly linked list

7.Clear Screen

8.Go to previous menu

>8

==========

==========

==========

********** Singly Linked List Operations **********

==========

==========

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>6

Thanks for Using Program!

Program Ended

==========

==========

==========

==========

Note:

Same operations can be repeated with data types DOUBLE, FLOAT and CHAR.

(b) Doubly linked list operations

Program:

```
// Doubly Linked list
// Roll No. 202cd005
#include <iostream>
#include <stdlib.h>
#include <limits>
using namespace std;
template <class T>
class Node
{
public:
   int key;
   int data;
   Node<T> *next;
   Node<T> *previous;

   Node()
   {
      key = 0;
      data = 0;
      next = NULL;
      previous = NULL;
   }
```

```
    Node(int k, int d)
    {
        key = k;
        data = d;
    }
};
template <class T>
class DoublyLinkedList
{
public:
    Node<T> *head;
    DoublyLinkedList()
    {
        head = NULL;
    }
    DoublyLinkedList(Node<T> *n)
    {
        head = n;
    }
    Node<T> *checkIfNodeExists(int k)
    {
        Node<T> *temp = NULL;
        Node<T> *ptr = head;
        while (ptr != NULL)
        {
            if (ptr->key == k)
            {
                temp = ptr;
```

```
        }
        ptr = ptr->next;
    }
    return temp;
}
void appendNode(Node<T> *n)
{
    if (checkIfNodeExists(n->key) != NULL)
    {
        cout << "Node exists alreday in doubly linked list with key value " << n->key << "." << endl;
    }
    else
    {
        if (head == NULL)
        {
            head = n;
            cout << "Node appended as head node" << endl;
        }
        else
        {
            Node<T> *ptr = head;
            while (ptr->next != NULL)
            {
                ptr = ptr->next;
            }
            ptr->next = n;
            n->previous = ptr;
            cout << "Node appended successfully." << endl;
```

```
        }
    }
}
void prependNode(Node<T> *n)
{
    if (checkIfNodeExists(n->key) != NULL)
    {
        cout << "Node already exists in doubly linked list with key value " << n->key << "." << endl;
    }
    else
    {
        if (head == NULL)
        {
            head = n;
            cout << "Node prepended as head node" << endl;
        }
        else
        {
            head->previous = n;
            n->next = head;
            n->previous = NULL;
            head = n;
            cout << "Node prepended successfully." << endl;
        }
    }
}
void insertNodeAfter(int k, Node<T> *n)
{
```

```
    Node<T> *ptr = checkIfNodeExists(k);
    if (ptr == NULL)
    {
        cout << "Cannot perform insertion." << endl;
        cout << "Node with key value " << k << " does not exists in doubly linked list." << endl;
    }
    else
    {
        if (checkIfNodeExists(n->key) != NULL)
        {
            cout << "Node with key value " << n->key << " already exists in doubly linked list." <<
endl;
        }
        else
        {
            Node<T> *nextNode = ptr->next;
            if (nextNode == NULL)
            {
                ptr->next = n;
                n->previous = ptr;
                cout << "Node inserted at the end." << endl;
            }
            else
            {
                nextNode->previous = n;
                n->next = nextNode;
                n->previous = ptr;
                ptr->next = n;
```

```
            }
        }
    }
}
void deleteNodeByKey(int k)
{
    Node<T> *ptr = checkIfNodeExists(k);
    if (ptr == NULL)
    {
        cout << "Cannot perform deletion." << endl;
        cout << "Node with key value " << k << " does not exists in linked list." << endl;
    }
    else
    {
        if (head->key == k)
        {
            head = head->next;
            cout << "head node with key value " << k << " unlinked successfully." << endl;
        }
        else
        {
            Node<T> *previousNode = ptr->previous;
            Node<T> *nextNode = ptr->next;
            if (nextNode == NULL)
            {
                previousNode->next = NULL;
                cout << "Last node with key value " << k << " unlinked successfully." << endl;
            }
```

```
            else
            {
                previousNode->next = nextNode;
                nextNode->previous = previousNode;
                cout << "Node with key value " << k << " unlinked successfully." << endl;
            }
        }
    }
}
void updateNodeByKey(int k, T data)
{
    Node<T> *ptr = checkIfNodeExists(k);
    if (ptr == NULL)
    {
        cout << "Cannot perform updation." << endl;
        cout << "Node with key value " << k << " does not exists in linked list." << endl;
    }
    else
    {
        Node<T> *temp = ptr;
        ptr->data = data;
        cout << "Node before updation: (" << temp->key << "," << temp->data << ")" << endl;
        cout << "Node after updation: (" << ptr->key << "," << ptr->data << ")" << endl;
    }
}
void printDoublyLinkedList()
{
    if (head == NULL)
```

```
    {
        cout << "Doubly linked list is empty." << endl;
    }
    {
        Node<T> *ptr = head;
        while (ptr != NULL)
        {
            cout << "(" << ptr->key << "," << ptr->data << ")    ";
            ptr = ptr->next;
        }
    }
}
void printDoublyLinkedListInReverseOrder()
{
    if (head == NULL)
    {
        cout << "Doubly linked list is empty." << endl;
    }
    {
        Node<T> *temp = head;
        Node<T> *ptrbwd = NULL;
        while (temp != NULL)
        {
            ptrbwd = temp;
            temp = temp->next;
        }
        while (ptrbwd != NULL)
        {
```

```
            cout << "(" << ptrbwd->key << "," << ptrbwd->data << ")    ";
            ptrbwd = ptrbwd->previous;
        }
    }
  }
};
int main()
{
  int varForDataTypeSelection, varForSwitchCase, key1, k1;
  do
  {
    cout << "\
n==========================================================================
============" << endl;
    cout << "***************************** Doubly Linked list Operations
*************************" << endl;
    cout <<
"==========================================================================
============" << endl;
    cout << "\nSelect Data type: " << endl;
    cout << "1.INT    2.DOUBLE   3.FLOAT    4.CHAR    5.Clear Screen    0.Exit from program"
<< endl;
    cout << ">";
    cin >> varForDataTypeSelection;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin.clear();
    if (varForDataTypeSelection == 1)
    {
      int data1;
```

```
        cout << "Selected Data type: INT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        DoublyLinkedList<int> s;
        do
        {
            cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete Node\n5.Update Node data\n6.Print doubly linked list\n7.Print doubly linked list in reverse order\n8.Clear Screen\n0.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<int> *n1 = new Node<int>();
            switch (varForSwitchCase)
            {
            case 0:
            {
                cout << "================================================================================" << endl;
                break;
            }
            case 1:
            {
                cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                cout << ">";
                cin >> key1;
                cout << ">";
                cin >> data1;
                n1->key = key1;
                n1->data = data1;
```

```
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.checkIfNodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
        cin >> data1;
        n1->key = key1;
```

```
            n1->data = data1;
            s.insertNodeAfter(k1, n1);
        }
        else
        {
            cout << "No node exists with entered key value." << endl;
        }
        break;
    }
    case 4:
    {
        cout << "Delete Node Operation (Deletion by key of node)" << endl;
        cout << "Enter key value of node to be deleted:" << endl;
        cout << ">";
        cin >> k1;
        s.deleteNodeByKey(k1);
        break;
    }
    case 5:
    {
        cout << "Update data of Node with key value" << endl;
        cout << "Enter key of node who's data is to be updated: ";
        cin >> k1;
        cout << "\nEnter new data: ";
        cin >> data1;
        cout << endl;
        s.updateNodeByKey(k1, data1);
        break;
```

```
        }
        case 6:
        {
            cout << "Doubly linked list is: " << endl;
            s.printDoublyLinkedList();
            break;
        }
        case 7:
        {
            cout << "Doubly linked list in reverse order: " << endl;
            s.printDoublyLinkedListInReverseOrder();
            break;
        }
        case 8:
        {
            system("clear");
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
        }
        }

    } while (varForSwitchCase != 0);
}
else if (varForDataTypeSelection == 2)
```

```
    {
        double data1;
        cout << "Selected Data type: DOUBLE" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        DoublyLinkedList<double> s;
        do
        {
            cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print doubly linked list\n7.Print doubly linked list in reverse order\
n8.Clear Screen\n0.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<double> *n1 = new Node<double>();
            switch (varForSwitchCase)
            {
            case 0:
            {
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            case 1:
            {
                cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                cout << ">";
                cin >> key1;
                cout << ">";
                cin >> data1;
```

```
    n1->key = key1;
    n1->data = data1;
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.checkIfNodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
```

```
            cin >> data1;
            n1->key = key1;
            n1->data = data1;
            s.insertNodeAfter(k1, n1);
        }
        else
        {
            cout << "No node exists with entered key value." << endl;
        }
        break;
    }
    case 4:
    {
        cout << "Delete Node Operation (Deletion by key of node)" << endl;
        cout << "Enter key value of node to be deleted:" << endl;
        cout << ">";
        cin >> k1;
        s.deleteNodeByKey(k1);
        break;
    }
    case 5:
    {
        cout << "Update data of Node with key value" << endl;
        cout << "Enter key of node who's data is to be updated: ";
        cin >> k1;
        cout << "\nEnter new data: ";
        cin >> data1;
        cout << endl;
```

```
            s.updateNodeByKey(k1, data1);
            break;
        }
        case 6:
        {
            cout << "Doubly linked list is: " << endl;
            s.printDoublyLinkedList();
            break;
        }
        case 7:
        {
            cout << "Doubly linked list in reverse order: " << endl;
            s.printDoublyLinkedListInReverseOrder();
            break;
        }
        case 8:
        {
            system("clear");
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
        }
        }

    } while (varForSwitchCase != 0);
```

```
    }
    else if (varForDataTypeSelection == 3)
    {
        float data1;
        cout << "Selected Data type: FLOAT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        DoublyLinkedList<float> s;
        do
        {
            cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print doubly linked list\n7.Print doubly linked list in reverse order\
n8.Clear Screen\n0.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<float> *n1 = new Node<float>();
            switch (varForSwitchCase)
            {
            case 0:
            {
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            case 1:
            {
                cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
                cout << ">";
                cin >> key1;
```

```
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.checkIfNodeExists(k1))
    {
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
```

```
            cin >> key1;
            cout << ">";
            cin >> data1;
            n1->key = key1;
            n1->data = data1;
            s.insertNodeAfter(k1, n1);
        }
        else
        {
            cout << "No node exists with entered key value." << endl;
        }
        break;
    }
    case 4:
    {
        cout << "Delete Node Operation (Deletion by key of node)" << endl;
        cout << "Enter key value of node to be deleted:" << endl;
        cout << ">";
        cin >> k1;
        s.deleteNodeByKey(k1);
        break;
    }
    case 5:
    {
        cout << "Update data of Node with key value" << endl;
        cout << "Enter key of node who's data is to be updated: ";
        cin >> k1;
        cout << "\nEnter new data: ";
```

```
    cin >> data1;
    cout << endl;
    s.updateNodeByKey(k1, data1);
    break;
}
case 6:
{
    cout << "Doubly linked list is: " << endl;
    s.printDoublyLinkedList();
    break;
}
case 7:
{
    cout << "Doubly linked list in reverse order: " << endl;
    s.printDoublyLinkedListInReverseOrder();
    break;
}
case 8:
{
    system("clear");
    break;
}
default:
{
    cout << "Invalid Input.\nTry Again:)" << endl;
    break;
}
}
```

```
    } while (varForSwitchCase != 0);
  }
  else if (varForDataTypeSelection == 4)
  {
    char data1;
    cout << "Selected Data type: CHAR" << endl;
    cout << "Which Operation Do you want to perform?" << endl;
    DoublyLinkedList<char> s;
    do
    {
      cout << "\n1.Append Node\n2.PrependNode\n3.Insert Node after a particular node\n4.Delete
Node\n5.Update Node data\n6.Print doubly linked list\n7.Print doubly linked list in reverse order\
n8.Clear Screen\n0.Go to previous menu\n> ";
      cin >> varForSwitchCase;
      Node<char> *n1 = new Node<char>();
      switch (varForSwitchCase)
      {
      case 0:
      {
        cout <<
"================================================================================
=============" << endl;
        break;
      }
      case 1:
      {
        cout << "Append Node Operation\nEnter Key & data of node to be appended:" << endl;
```

```
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.appendNode(n1);
    break;
}
case 2:
{
    cout << "Prepend Node Operation\nEnter key and data of node to be prepended:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    n1->key = key1;
    n1->data = data1;
    s.prependNode(n1);
    break;
}
case 3:
{
    cout << "Insert Node Operation" << endl;
    cout << "Enter key of node after which you want to insert a node:\n>";
    cin >> k1;
    if (s.checkIfNodeExists(k1))
    {
```

```
        cout << "Enter key and data of node to be inserted:" << endl;
        cout << ">";
        cin >> key1;
        cout << ">";
        cin >> data1;
        n1->key = key1;
        n1->data = data1;
        s.insertNodeAfter(k1, n1);
    }
    else
    {
        cout << "No node exists with entered key value." << endl;
    }
    break;
}
case 4:
{
    cout << "Delete Node Operation (Deletion by key of node)" << endl;
    cout << "Enter key value of node to be deleted:" << endl;
    cout << ">";
    cin >> k1;
    s.deleteNodeByKey(k1);
    break;
}
case 5:
{
    cout << "Update data of Node with key value" << endl;
    cout << "Enter key of node who's data is to be updated: ";
```

```
    cin >> k1;
    cout << "\nEnter new data: ";
    cin >> data1;
    cout << endl;
    s.updateNodeByKey(k1, data1);
    break;
}
case 6:
{
    cout << "Doubly linked list is: " << endl;
    s.printDoublyLinkedList();
    break;
}
case 7:
{
    cout << "Doubly linked list in reverse order: " << endl;
    s.printDoublyLinkedListInReverseOrder();
    break;
}
case 8:
{
    system("clear");
    break;
}
default:
{
    cout << "Invalid Input.\nTry Again:)" << endl;
    break;
```

```
            }
            }
        } while (varForSwitchCase != 0);
    }
    else if (varForDataTypeSelection == 5)
    {
        system("clear");
    }
    else if (varForDataTypeSelection == 0)
    {
        cout << "\n\nThanks for Using Program!" << endl;
        cout << "Program Ended\n\n" << endl;
        cout <<
"================================================================================
=============" << endl;
        cout <<
"================================================================================
=============" << endl;
        break;
    }
    else
    {
        cout << "Invalid input!\nTry Again:)" << endl;
    }
  } while (varForDataTypeSelection != 0);
  return 0;
}
```

Output:

```
================================================================================
============
***************************** Doubly Linked list Operations ************************
================================================================================
============

Select Data type:
1.INT     2.DOUBLE    3.FLOAT     4.CHAR     5.Clear Screen     0.Exit from program
>1
Selected Data type: INT
Which Operation Do you want to perform?

1.Append Node
2.PrependNode
3.Insert Node after a particular node
4.Delete Node
5.Update Node data
6.Print doubly linked list
7.Print doubly linked list in reverse order
8.Clear Screen
0.Go to previous menu
> 1
Append Node Operation
Enter Key & data of node to be appended:
>1
>10
Node appended as head node

1.Append Node
2.PrependNode
```

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 2

Prepend Node Operation

Enter key and data of node to be prepended:

>0

>25

Node prepended successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 1

Append Node Operation

Enter Key & data of node to be appended:

>3

>10

Node appended successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 3

Insert Node Operation

Enter key of node after which you want to insert a node:

>1

Enter key and data of node to be inserted:

>2

>20

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 6

Doubly linked list is:

(0,25) (1,10) (2,20) (3,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 4

Delete Node Operation (Deletion by key of node)

Enter key value of node to be deleted:

>2

Node with key value 2 unlinked successfully.

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 6

Doubly linked list is:

(0,25) (1,10) (3,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 5

Update data of Node with key value

Enter key of node who's data is to be updated: 1

Enter new data: 23

Node before updation: (1,23)

Node after updation: (1,23)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 6

Doubly linked list is:

(0,25) (1,23) (3,10)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 7

Doubly linked list in reverse order:

(3,10) (1,23) (0,25)

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 8

1.Append Node

2.PrependNode

3.Insert Node after a particular node

4.Delete Node

5.Update Node data

6.Print doubly linked list

7.Print doubly linked list in reverse order

8.Clear Screen

0.Go to previous menu

> 0

==========

==========

==========

********** Doubly Linked list Operations **********

==========

==========

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 0.Exit from program

>0

Thanks for Using Program!

Program Ended

==========

==========

==========

==========

Note:

Same operations can be repeated with data types DOUBLE, FLOAT and CHAR.

5. Use a Singly Linked list to reverse a string, to find whether a string is a palindrome or not.

Program:

```
// Reversing a string using Singly Linked List and checking if a list is Palindrome or not
// Roll NO. 202cd05
#include <iostream>
#include<stdlib.h>
using namespace std;
template <class T>
class Node
{
public:
   int key;
   T data;
   Node<T> *next;
   Node()
   {
      key = 0;
      data = 0;
      next = NULL;
   }
   Node(int k, int d)
   {
      key = k;
      data = d;
   }
};
template <class T>
class String
{
public:
```

```
Node<T> *head;
int count;
String()
{
   head = NULL;
}
void append(Node<T> *n)
{
   if (head == NULL)
   {
      head = n;
   }
   else
   {
      Node<T> *ptr = head;
      while (ptr->next != NULL)
      {
         ptr = ptr->next;
      }
      ptr->next = n;
   }
}
void displayString()
{
   Node<T> *ptr = head;
   if (ptr == NULL)
   {
      cout << "String is empty." << endl;
```

```
    }
    else
    {
        while (ptr != NULL)
        {
            cout << ptr->data << " ";
            ptr = ptr->next;
        }
    }
}
void reverseString()
{
    //reversing string
    Node<T> *current = head;
    Node<T> *prev = NULL;
    Node<T> *next = NULL;
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
    //displaying reversed string
    displayString();
    //re-reversing string so that original string remains intact
    current = head;
```

```
    prev = NULL;
    next = NULL;
    while (current != NULL)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    head = prev;
}
bool checkIfPalindrome() //by spliting list from middle and comparing the two parts
{
    Node<T> *temp = NULL;
    Node<T> *sp = head, *fp = head, *mid = NULL;
    while (fp != NULL && fp->next != NULL)
    {
        sp = sp->next;
        fp = fp->next->next;
    }
    if (fp != NULL) //fp will not be null only if linked list has odd number of elements
    {
        temp = sp;
        mid = sp->next;
    }
    else
    {
        mid = sp;
```

```
        }
        //Reversing second part of list i.e. from middle to end
        Node<T> *prev = NULL;
        Node<T> *next = NULL;
        while (mid != NULL)
        {
            next = mid->next;
            mid->next = prev;
            prev = mid;
            mid = next;
        }
        while (prev != NULL)
        {
            if (prev->data != head->data) //checking if same elements are present at mirror location i.e
palindrome
                return false;
            prev = prev->next;
            head = head->next;
        }
    return true;
    }
};
int main()
{
    int varForSwitchCase;
    String<char> s;
    string st;
```

```
    cout << "\
n================================================================================
=============" << endl;
    cout << "******** Program on Reversing String and Checking If String is Palindrome
**********" << endl;
    cout << "\n***************************** (Using Singly Linked List)
*****************************" << endl;
    cout <<
"================================================================================
=============" << endl;
    cout << "Enter String: ";
    cin >> st;
    for (int i = 0; i < st.length();i++)
    {
        Node<char> *newNode = new Node<char>();
        {
            newNode->key = i;
            newNode->data = st[i] ;
            s.append(newNode);
        }
    }
    cout << "\nWhich Operation Do you want to perform?" << endl;
    do
    {
        cout << "\n1.Diplay String\n2.Reverse String\n3.Check if string is Palindrome\n4.Clear Screen\
n5.Exit\n>";
        cin >> varForSwitchCase;
        switch (varForSwitchCase)
        {
```

```
case 1:
{
    cout << "Linked list is :";
    s.displayString();
    break;
}
case 2:
{
    cout << "Reverse String Operation" << endl;
    s.reverseString();
    break;
}
case 3:
{
    if (s.checkIfPalindrome())
    {
        cout << "String is Palindrome." << endl;
    }
    else
    {
        cout << "String is not Palindrome" << endl;
    }
    break;
}
case 4:
{
    system("clear");
    break;
```

```
        }
        case 5:
        {
            break;
        }
        default:
        {
            cout << "Invalide input.\nTry Again:)" << endl;
        }
        }
    } while (varForSwitchCase != 5);
    return 0;
}
```

Output with palindrome string input:

```
================================================================================
============
******** Program on Reversing String and Checking If String is Palindrome ***********

***************************** (Using Singly Linked List) ****************************
================================================================================
============
Enter String: ABCDDCBA

Which Operation Do you want to perform?

1.Diplay String
2.Reverse String
```

3.Check if string is Palindrome

4.Clear Screen

5.Exit

>1

Linked list is :A B C D D C B A

1.Diplay String

2.Reverse String

3.Check if string is Palindrome

4.Clear Screen

5.Exit

>2

Reverse String Operation

A B C D D C B A

1.Diplay String

2.Reverse String

3.Check if string is Palindrome

4.Clear Screen

5.Exit

>3

String is Palindrome.

1.Diplay String

2.Reverse String

3.Check if string is Palindrome

4.Clear Screen

5.Exit

>5

Output with non-palindrome string input:

==========

==========

******** Program on Reversing String and Checking If String is Palindrome ***********

********** (Using Singly Linked List) **********

==========

==========

Enter String: ABCDEFG

Which Operation Do you want to perform?

1.Diplay String
2.Reverse String
3.Check if string is Palindrome
4.Clear Screen
5.Exit
>1
Linked list is :A B C D E F G
1.Diplay String
2.Reverse String
3.Check if string is Palindrome
4.Clear Screen
5.Exit
>2
Reverse String Operation
G F E D C B A
1.Diplay String
2.Reverse String
3.Check if string is Palindrome
4.Clear Screen
5.Exit
>3
String is not Palindrome

1.Diplay String

2.Reverse String

3.Check if string is Palindrome

4.Clear Screen

5.Exit

>5

6. Design a Stack and a queue using linked list (use singly linked list).

(a) Stack using Singly linked list

```
// Stack operations using Singly Linked List
// Roll NO. 202cd005
#include<iostream>
#include<stdlib.h>
#include<limits>
using namespace std;
template<class T>
class Node
{
   public:
   int key;
   T data;
   Node<T>* next;
   Node()
   {
      key=0;
      data=0;
      next=NULL;
   }
   Node(int k,int d)
```

```
    {
        key=k;
        data=d;
    }
};
template<class T>
class stack
{
    public:
    Node<T>* top;
    stack()
    {
        top=NULL;
    }
    bool isEmpty()
    {
        if(top==NULL)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    bool checkIfNodeExists(Node<T>* n)
    {
        Node<T>* temp=top;
```

```
    bool exists = false;
    while(temp!=NULL)
    {
        if(temp->key==n->key)
        {
            exists = true;
            break;
        }
        temp=temp->next;
    }
    return exists;
}
void push(Node<T> *n)
{
    if(top==NULL)
    {
        top=n;
    }
    else if(checkIfNodeExists(n))
    {
        cout << "Node with key value "<< n->key << " already exists in stack.";
    }
    else
    {
        Node<T>* temp = top;
        top = n;
        n->next=temp;
        cout << "Node pushed successfully."<<endl;
```

```
    }
}
Node<T>* pop()
{
    Node<T>* temp=NULL;
    if(isEmpty())
    {
        cout << "Cannot perform pop operation!"<<endl;
        cout << "Stack is empty."<<endl;
        return temp;
    }
    else
    {
        temp=top;
        top = top->next;
        return temp;
    }
}
Node<T>* peek()
{

    if(isEmpty())
    {
        cout << "Stack Underflow."<<endl;
        return NULL;
    }
    else
    {
```

```
        cout << "(" << top->key <<","<<top->data<<")    "<<endl;
        return top;
    }
}
int count()
{
    int count =0;
    Node<T>* temp=top;
    while(temp!=NULL)
    {
        count++;
        temp=temp->next;
    }
    return count;
}
void displayStack()
{
    Node<T>* temp=top;
    if(isEmpty())
    {
        cout << "Nothing to Print."<<endl;
        cout << "Stack is empty!"<<endl;
    }
    else
    {
        while(temp!=NULL)
        {
            cout << "("<<temp->key<<","<<temp->data<<")   ";
```

```
            temp=temp->next;
        }
    }
  }
};
int main()
{
  int key1,varForDataTypeSelection, varForSwitchCase,count;
  do
  {
    cout << "\
n==========================================================================
============" << endl;
    cout << "********************* Stack Operations using Singly Linked list
*******************" << endl;
    cout << "\nSelect Data type: " << endl;
    cout << "1.INT   2.DOUBLE   3.FLOAT   4.CHAR   5.Clear Screen   6.Exit from program"
<< endl;
    cout << ">";
    cin >> varForDataTypeSelection;
    std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    std::cin.clear();
    if (varForDataTypeSelection == 1)
    {
      int data1;
      cout << "Selected Data type: INT" << endl;
      cout << "Which Operation Do you want to perform?" << endl;
      stack<int> s;
      do
```

```
{
    cout << "\n1.Check If stack is Empty\n2.Push Node\n3.Pop\n4.Peek\n5.Check number of
elements in Stack\n6.Print stack\n7.Clear Screen\n8.Go to previous menu\n> ";
    cin >> varForSwitchCase;
    Node<int> *newNode = new Node<int>();
    switch (varForSwitchCase)
    {
    case 1:
    {
       if(s.isEmpty())
       {
          cout << "Stack is Empty."<<endl;
       }
       else
       {
          cout << "Stack is not Empty."<<endl;
       }
       break;
    }
    case 2:
    {
       cout << "Push Node Operation\nEnter key and data of node to be pushed:" << endl;
       cout << ">";
       cin >> key1;
       cout << ">";
       cin >> data1;
       newNode->key = key1;
       newNode->data = data1;
```

```
    s.push(newNode);
    break;
}
case 3:
{
    cout << "Pop Operation" << endl;
    newNode=s.pop();
    cout << "Node popped out ("<<newNode->key<<","<<newNode->data<<")"<<endl;
    delete newNode;
    break;
}
case 4:
{
    if(s.isEmpty())
    {
        cout << "Stack is Empty"<<endl;
    }
    else
    {
        cout << "Peek Operation" << endl;
    newNode=s.peek();
    cout << "Top of stack is ("<<newNode->key<<","<<newNode->data<<")"<<endl;
    }
    break;
}
case 5:
{
    count=s.count();
```

```
            cout << "Number of elements in stack are "<<count <<"."<<endl;
            break;
        }
        case 6:
        {
            cout << "Stack is: " << endl;
            s.displayStack();
            break;
        }
        case 7:
        {
            system("clear");
            break;
        }
        case 8:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
        }
        }
    } while (varForSwitchCase != 8);
```

```
    }
    else if (varForDataTypeSelection == 2)
    {
        double data1;
        cout << "Selected Data type: DOUBLE" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        stack<double> s;
        do
        {
            cout << "\n1.Check If stack is Empty\n2.Push Node\n3.Pop\n4.Peek\n5.Check number of
elements in Stack\n6.Print stack\n7.Clear Screen\n8.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<double> *newNode = new Node<double>();
            switch (varForSwitchCase)
            {
            case 1:
            {
                if(s.isEmpty())
                {
                    cout << "Stack is Empty."<<endl;
                }
                else
                {
                    cout << "Stack is not Empty."<<endl;
                }
                break;
            }
            case 2:
```

```
{
    cout << "Push Node Operation\nEnter key and data of node to be pushed:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    newNode->key = key1;
    newNode->data = data1;
    s.push(newNode);
    break;
}
case 3:
{
    cout << "Pop Operation" << endl;
    newNode=s.pop();
    cout << "Node popped out ("<<newNode->key<<","<<newNode->data<<")"<<endl;
    delete newNode;
    break;
}
case 4:
{
    if(s.isEmpty())
    {
        cout << "Stack is Empty"<<endl;
    }
    else
    {
        cout << "Peek Operation" << endl;
```

```
            newNode=s.peek();
            cout << "Top of stack is ("<<newNode->key<<","<<newNode->data<<")"<<endl;
            }
            break;
        }
        case 5:
        {
            count=s.count();
            cout << "Number of elements in stack are "<<count <<"."<<endl;
            break;
        }
        case 6:
        {
            cout << "Stack is: " << endl;
            s.displayStack();
            break;
        }
        case 7:
        {
            system("clear");
            break;
        }
        case 8:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
```

```
            }
            default:
            {
               cout << "Invalid Input.\nTry Again:)" << endl;
               break;
            }
            }
        } while (varForSwitchCase != 8);
    }
    else if (varForDataTypeSelection == 3)
    {
        float data1;
        cout << "Selected Data type: FLOAT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        stack<float> s;
        do
        {
            cout << "\n1.Check If stack is Empty\n2.Push Node\n3.Pop\n4.Peek\n5.Check number of
elements in Stack\n6.Print stack\n7.Clear Screen\n8.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<float> *newNode = new Node<float>();
            switch (varForSwitchCase)
            {
            case 1:
            {
               if(s.isEmpty())
               {
                  cout << "Stack is Empty."<<endl;
```

```
    }
    else
    {
        cout << "Stack is not Empty."<<endl;
    }
    break;
}
case 2:
{
    cout << "Push Node Operation\nEnter key and data of node to be pushed:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    newNode->key = key1;
    newNode->data = data1;
    s.push(newNode);
    break;
}
case 3:
{
    cout << "Pop Operation" << endl;
    newNode=s.pop();
    cout << "Node popped out ("<<newNode->key<<","<<newNode->data<<")"<<endl;
    delete newNode;
    break;
}
case 4:
```

```
{
   if(s.isEmpty())
   {
      cout << "Stack is Empty"<<endl;
   }
   else
   {
      cout << "Peek Operation" << endl;
   newNode=s.peek();
   cout << "Top of stack is ("<<newNode->key<<","<<newNode->data<<")"<<endl;
   }
   break;
}
case 5:
{
   count=s.count();
   cout << "Number of elements in stack are "<<count <<"."<<endl;
   break;
}
case 6:
{
   cout << "Stack is: " << endl;
   s.displayStack();
   break;
}
case 7:
{
   system("clear");
```

```
            break;
        }
        case 8:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
            break;
        }
        }
    } while (varForSwitchCase != 8);
}
else if (varForDataTypeSelection == 4)
{
    char data1;
    cout << "Selected Data type: CHAR" << endl;
    cout << "Which Operation Do you want to perform?" << endl;
    stack<char> s;
    do
    {
        cout << "\n1.Check If stack is Empty\n2.Push Node\n3.Pop\n4.Peek\n5.Check number of
elements in Stack\n6.Print stack\n7.Clear Screen\n8.Go to previous menu\n> ";
        cin >> varForSwitchCase;
```

```
Node<char> *newNode = new Node<char>();
switch (varForSwitchCase)
{
case 1:
{
   if(s.isEmpty())
   {
      cout << "Stack is Empty."<<endl;
   }
   else
   {
      cout << "Stack is not Empty."<<endl;
   }
   break;
}
case 2:
{
   cout << "Push Node Operation\nEnter key and data of node to be pushed:" << endl;
   cout << ">";
   cin >> key1;
   cout << ">";
   cin >> data1;
   newNode->key = key1;
   newNode->data = data1;
   s.push(newNode);
   break;
}
case 3:
```

```
{
   cout << "Pop Operation" << endl;
   newNode=s.pop();
   cout << "Node popped out ("<<newNode->key<<","<<newNode->data<<")"<<endl;
   delete newNode;
   break;
}
case 4:
{
   if(s.isEmpty())
   {
      cout << "Stack is Empty"<<endl;
   }
   else
   {
      cout << "Peek Operation" << endl;
   newNode=s.peek();
   cout << "Top of stack is ("<<newNode->key<<","<<newNode->data<<")"<<endl;
   }
   break;
}
case 5:
{
   count=s.count();
   cout << "Number of elements in stack are "<<count <<"."<<endl;
   break;
}
case 6:
```

```
            {
                cout << "Stack is: " << endl;
                s.displayStack();
                break;
            }
            case 7:
            {
                system("clear");
                break;
            }
            case 8:
            {
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            default:
            {
                cout << "Invalid Input.\nTry Again:)" << endl;
                break;
            }
            }
        } while (varForSwitchCase != 8);
    }
    else if (varForDataTypeSelection == 5)
    {
        system("clear");
```

```
        }
        else if (varForDataTypeSelection == 6)
        {
            cout << "\n\nThanks for Using Program!" << endl;
            cout << "Program Ended\n\n" << endl;
            cout <<
"================================================================================
=============" << endl;
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        else
        {
            cout << "Invalid input!\nTry Again:)" << endl;
        }
    }while (varForDataTypeSelection != 6);
    return 0;
}
```

Output:

```
================================================================================
============
********************* Stack Operations using Singly Linked list  *********************
================================================================================
============

Select Data type:
```

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>1

Selected Data type: INT

Which Operation Do you want to perform?

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 1

Stack is Empty.

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 2

Push Node Operation

Enter key and data of node to be pushed:

>1

>10

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 2

Push Node Operation

Enter key and data of node to be pushed:

>2

>20

Node pushed successfully.

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 6

Stack is:

(2,20) (1,10)

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 2

Push Node Operation

Enter key and data of node to be pushed:

>3

>30

Node pushed successfully.

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 6

Stack is:

(3,30) (2,20) (1,10)

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 3

Pop Operation

Node popped out (3,30)

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 6

Stack is:

(2,20) (1,10)

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 4

Peek Operation

(2,20)

Top of stack is (2,20)

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 5

Number of elements in stack are 2.

1.Check If stack is Empty

2.Push Node

3.Pop

4.Peek

5.Check number of elements in Stack

6.Print stack

7.Clear Screen

8.Go to previous menu

> 8

==========

********** Stack Operations using Singly Linked list **********

==========

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>6

Thanks for Using Program!

Program Ended

==========

==========

Note:

Same operations can be repeated with data types DOUBLE, FLOAT and CHAR.

(b) Queue operations using singly linked list

Program:

```
// Queue operations using singly linked list
// Roll No. 202cd005
#include <iostream>
#include <stdlib.h>
using namespace std;
template <class T>
class Node
{
public:
   int key;
   T data;
   Node<T> *next;
   Node()
   {
      key = 0;
      data = 0;
      next = NULL;
   }
   Node(int k, T d)
   {
      key = k;
      data = d;
```

```
        next = NULL;
    }
};
template <class T>
class queue
{
public:
    Node<T> *front;
    Node<T> *rear;
    queue()
    {
        front = NULL;
        rear = NULL;
    }
    bool isEmpty()
    {
        Node<T> *tempf = front;
        Node<T> *tempr = rear;
        if (tempf == NULL)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
    Node<T> *checkIfNodeExists(Node<T> *n)
```

```
{
    Node<T> *ptr = NULL;
    Node<T> *temp = front;
    if (temp == n)
    {
        ptr = temp;
        return ptr;
    }
    else
    {
        temp = rear;
        while (temp != NULL)
        {
            if (temp->key == n->key)
            {
                ptr = temp;
            }
            temp = temp->next;
        }
        return ptr;
    }
}
void enqueue(Node<T> *n)
{
    Node<T> *temp = checkIfNodeExists(n);
    if (temp != NULL)
    {
        cout << "Node with key value " << temp->key << " already exists in queue." << endl;
```

```
    }
    else if (front == NULL)
    {
        front = n;
        rear = n;
        cout << "First element Enqueued successfully." << endl;
    }
    else
    {
        Node<T> *tempr = rear;
        while (tempr->next != NULL)
        {
            tempr = tempr->next;
        }
        tempr->next = n;
        cout << "Next element Enqueued successfully." << endl;
    }
}
void dequeue()
{
    if (front != NULL)
    {
        front = front->next;
        rear = rear->next;
        cout << "Node dequeued successfully."<<endl;
    }
    else
    {
```

```
        cout << "Queue is empty.\nCannot perform dequeue operation." << endl;
    }
}
int count()
{
    int count = 0;
    Node<T> *ptr = rear;
    while (ptr != NULL)
    {
        count++;
        ptr = ptr->next;
    }
    return count;
}
void displayQueue()
{
    Node<T> *tempr = rear;
    if (isEmpty())
    {
        cout << "Nothing to Print.\nQueue is Empty:(" << endl;
    }
    else
    {
        while (tempr != NULL)
        {
            if (tempr == front)
            {
                cout << "(" << front->key << "," << front->data << ")   ";
```

```
            }
            else
            {
                cout << "(" << tempr->key << "," << tempr->data << ")   ";
            }
            tempr = tempr->next;
          }
       }
   }
};
int main()
{
   int key1, varForDataTypeSelection, varForSwitchCase, count;
   do
   {
         cout << "\
n==========================================================================
=============" << endl;
       cout << "********************** Queue operations using singly linked list
**********************" << endl;
       cout <<
"==========================================================================
=============" << endl;
       cout << "\nSelect Data type: " << endl;
       cout << "1.INT     2.DOUBLE    3.FLOAT     4.CHAR     5.Clear Screen     6.Exit from program"
<< endl;
       cout << ">";
       cin >> varForDataTypeSelection;
```

```
    if (varForDataTypeSelection == 1)
    {
        int data1;
        cout << "Selected Data type: INT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        queue<int> s;
        do
        {
            cout << "\n1.Check If queue is Empty\n2.Enqueue\n3.dequeue\n4.Check number of elements in queue\n5.Print queue\n6.Clear Screen\n7.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<int> *newNode = new Node<int>();
            switch (varForSwitchCase)
            {
            case 1:
            {
                if (s.isEmpty())
                {
                    cout << "queue is Empty." << endl;
                }
                else
                {
                    cout << "queue is not Empty." << endl;
                }
                break;
            }
            case 2:
            {
```

```
    cout << "Enqueue Operation\nEnter key and data of node to be enqueued:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    newNode->key = key1;
    newNode->data = data1;
    s.enqueue(newNode);
    break;
}
case 3:
{
    cout << "dequeue Operation" << endl;
    s.dequeue();
    break;
}
case 4:
{
    count = s.count();
    cout << "Number of elements in queue are " << count << "." << endl;
    break;
}
case 5:
{
    cout << "queue is: " << endl;
    s.displayQueue();
    break;
}
```

```
            case 6:
            {
               system("clear");
               break;
            }
            case 7:
            {
               cout <<
"================================================================================
=============" << endl;
               break;
            }
            default:
            {
               cout << "Invalid Input.\nTry Again:)" << endl;
               break;
            }
            }
         } while (varForSwitchCase != 7);
      }
      else if (varForDataTypeSelection == 2)
      {
         double data1;
         cout << "Selected Data type: DOUBLE" << endl;
         cout << "Which Operation Do you want to perform?" << endl;
         queue<double> s;
         do
         {
```

```
        cout << "\n1.Check If queue is Empty\n2.Enqueue\n3.dequeue\n4.Check number of elements in queue\n5.Print queue\n6.Clear Screen\n7.Go to previous menu\n> ";
        cin >> varForSwitchCase;
        Node<double> *newNode = new Node<double>();
        switch (varForSwitchCase)
        {
        case 1:
        {
            if (s.isEmpty())
            {
                cout << "queue is Empty." << endl;
            }
            else
            {
                cout << "queue is not Empty." << endl;
            }
            break;
        }
        case 2:
        {
            cout << "Enqueue Operation\nEnter key and data of node to be enqueued:" << endl;
            cout << ">";
            cin >> key1;
            cout << ">";
            cin >> data1;
            newNode->key = key1;
            newNode->data = data1;
            s.enqueue(newNode);
```

```
    break;
}
case 3:
{
    cout << "dequeue Operation" << endl;
    s.dequeue();
    break;
}
case 4:
{
    count = s.count();
    cout << "Number of elements in queue are " << count << "." << endl;
    break;
}
case 5:
{
    cout << "queue is: " << endl;
    s.displayQueue();
    break;
}
case 6:
{
    system("clear");
    break;
}
case 7:
{
```

```
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            default:
            {
                cout << "Invalid Input.\nTry Again:)" << endl;
                break;
            }
            }
        } while (varForSwitchCase != 7);
    }
    else if (varForDataTypeSelection == 3)
    {
        float data1;
        cout << "Selected Data type: FLOAT" << endl;
        cout << "Which Operation Do you want to perform?" << endl;
        queue<float> s;
        do
        {
            cout << "\n1.Check If queue is Empty\n2.Enqueue\n3.dequeue\n4.Check number of elements
in queue\n5.Print queue\n6.Clear Screen\n7.Go to previous menu\n> ";
            cin >> varForSwitchCase;
            Node<float> *newNode = new Node<float>();
            switch (varForSwitchCase)
            {
            case 1:
```

```
{
    if (s.isEmpty())
    {
        cout << "queue is Empty." << endl;
    }
    else
    {
        cout << "queue is not Empty." << endl;
    }
    break;
}
case 2:
{
    cout << "Enqueue Operation\nEnter key and data of node to be enqueued:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    newNode->key = key1;
    newNode->data = data1;
    s.enqueue(newNode);
    break;
}
case 3:
{
    cout << "dequeue Operation" << endl;
    s.dequeue();
    break;
```

```
        }
        case 4:
        {
            count = s.count();
            cout << "Number of elements in queue are " << count << "." << endl;
            break;
        }
        case 5:
        {
            cout << "queue is: " << endl;
            s.displayQueue();
            break;
        }
        case 6:
        {
            system("clear");
            break;
        }
        case 7:
        {
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        default:
        {
            cout << "Invalid Input.\nTry Again:)" << endl;
```

```
            break;
        }
        }
    } while (varForSwitchCase != 7);
}
else if (varForDataTypeSelection == 4)
{
    char data1;
    cout << "Selected Data type: CHAR" << endl;
    cout << "Which Operation Do you want to perform?" << endl;
    queue<char> s;
    do
    {
        cout << "\n1.Check If queue is Empty\n2.Enqueue\n3.dequeue\n4.Check number of elements in queue\n5.Print queue\n6.Clear Screen\n7.Go to previous menu\n> ";
        cin >> varForSwitchCase;
        Node<char> *newNode = new Node<char>();
        switch (varForSwitchCase)
        {
        case 1:
        {
            if (s.isEmpty())
            {
                cout << "Queue is Empty." << endl;
            }
            else
            {
                cout << "Queue is not Empty." << endl;
```

```
    }
    break;
}
case 2:
{
    cout << "Enqueue Operation\nEnter key and data of node to be enqueued:" << endl;
    cout << ">";
    cin >> key1;
    cout << ">";
    cin >> data1;
    newNode->key = key1;
    newNode->data = data1;
    s.enqueue(newNode);
    break;
}
case 3:
{
    cout << "Dequeue Operation" << endl;
    s.dequeue();
    break;
}
case 4:
{
    count = s.count();
    cout << "Number of elements in queue are " << count << "." << endl;
    break;
}
case 5:
```

```
            {
                cout << "Queue is: " << endl;
                s.displayQueue();
                break;
            }
            case 6:
            {
                system("clear");
                break;
            }
            case 7:
            {
                cout <<
"================================================================================
=============" << endl;
                break;
            }
            default:
            {
                cout << "Invalid Input.\nTry Again:)" << endl;
                break;
            }
            }
        } while (varForSwitchCase != 7);
    }
    else if (varForDataTypeSelection == 5)
    {
        system("clear");
```

```
        }
        else if (varForDataTypeSelection == 6)
        {
            cout << "\n\nThanks for Using Program!" << endl;
            cout << "Program Ended\n\n"
                 << endl;
            cout <<
"================================================================================
=============" << endl;
            cout <<
"================================================================================
=============" << endl;
            break;
        }
        else
        {
            cout << "Invalid input!\nTry Again:)" << endl;
        }
    } while (varForDataTypeSelection != 6);
    return 0;
}
```

Output:

```
================================================================================
============
********************* Queue operations using singly linked list **********************
================================================================================
============

Select Data type:
```

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>1

Selected Data type: INT

Which Operation Do you want to perform?

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 1

queue is Empty.

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 2

Enqueue Operation

Enter key and data of node to be enqueued:

>1

>10

First element Enqueued successfully.

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 2

Enqueue Operation

Enter key and data of node to be enqueued:

>2

>20

Next element Enqueued successfully.

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 5

queue is:

(1,10) (2,20)

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 2

Enqueue Operation

Enter key and data of node to be enqueued:

>3

>30
Next element Enqueued successfully.

1.Check If queue is Empty
2.Enqueue
3.dequeue
4.Check number of elements in queue
5.Print queue
6.Clear Screen
7.Go to previous menu
> 5
queue is:
(1,10) (2,20) (3,30)
1.Check If queue is Empty
2.Enqueue
3.dequeue
4.Check number of elements in queue
5.Print queue
6.Clear Screen
7.Go to previous menu
> 3
dequeue Operation
Node dequeued successfully.

1.Check If queue is Empty
2.Enqueue
3.dequeue
4.Check number of elements in queue
5.Print queue
6.Clear Screen
7.Go to previous menu
> 5

queue is:

(2,20) (3,30)

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 4

Number of elements in queue are 2.

1.Check If queue is Empty

2.Enqueue

3.dequeue

4.Check number of elements in queue

5.Print queue

6.Clear Screen

7.Go to previous menu

> 7

==========

==========

==========

==========

********** Queue operations using singly linked list **********

==========

==========

Select Data type:

1.INT 2.DOUBLE 3.FLOAT 4.CHAR 5.Clear Screen 6.Exit from program

>6

Thanks for Using Program!

Program Ended

==========

==========

==========

==========

Note:

Same operations can be repeated with data types DOUBLE, FLOAT and CHAR.

7. Convert a infix expression to post-fix expression and evaluate the same.

Program:

```
#include<iostream>
#include<math.h>
using namespace std;
class Node
{
   public:
   char data;
   Node* next;
   Node()
   {
      data=0;
      next=NULL;
   }
   Node(char d)
   {
```

```
        data=d;
    }
};
class stack
{
    public:
    int strlen;
    Node* top;
    stack()
    {
        top=NULL;
        strlen=0;
    }
    void push(Node* n)
    {
        if(top==NULL)
        {
            top=n;
        }
        else
        {
            Node* temp=top;
            top=n;
            n->next=temp;

        }
    }
    char pop()
```

```
{
    Node* temp=NULL;
    if(top==NULL)
    {
        cout<<"stack underflow";
    }
    else
    {
        temp=top;
        top=top->next;
    }
    return temp->data;
}
int getstrlen(string s)
{
    while(s[this->strlen]!='\0')
    {
        this->strlen++;
    }
    return this->strlen;
}
int prec(char c)
{
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
```

```
        return 1;
    else
        return -1;
}
string infixToPostFix(string s)
{
int l = getstrlen(s);
string ns;
for(int i = 0; i < l; i++)
{
    Node* temp=new Node();
    if((s[i] >= 'a' && s[i] <= 'z') ||  (s[i] >= 'A' && s[i] <= 'Z') || isdigit(s[i]))
    {
        ns+=s[i];
    }
    else if(s[i] == '(')
    {
        temp->data='(';
        push(temp);
    }
    else if(s[i] == ')')
    {
        while(top!= NULL && top->data != '(')
        {
            char c = top->data;
            pop();
           ns += c;
        }
```

```
        if(top->data == '(')
        {
            char c = top->data;
            pop();
        }
    }
    else
    {
        while(top!= NULL && prec(s[i]) <=prec(top->data))
        {
            char c = top->data;
            pop();
            ns += c;
        }
        temp->data=s[i];
        push(temp);
    }
}
 while(top!=NULL)
{
    char c = top->data;
    pop();
    ns += c;
}
return ns;
}
void findvalueofpostfix(string s)
{
```

```
string ns;
for(int i = 0; s[i]; i++)
{
    Node* temp=new Node();
    if(isdigit(s[i]))
    {
        temp->data=s[i]-'0';
        push(temp);
    }
    else
    {
        int a=pop();
        int b=pop();
        switch (s[i])
        {
        case '+':
            temp->data=b+a;
            push(temp);
            break;
        case '-':
            temp->data=b-a;
            push(temp);

            break;
        case '*':
            temp->data=b*a;
            push(temp);
            break;
```

```
            case '/':
                temp->data=b/a;
                push(temp);
                break;
            case '^':
                temp->data =pow(b,a);
                push(temp);
                break;
            }
        }
    }
    cout << "Value of expression is " <<  (int) pop() <<endl;
}
bool checkIfDigit(string s)
{
    bool digit=false;
    for(int i=0;s[i];i++)
    {
        if(isdigit(s[i]))
            digit=true;
        else
            continue;
    }
    return digit;
}
};
int main()
{
```

```
    stack st;
    string exp,postfix,eval;
    cout << "\
n================================================================================
=============" << endl;
    cout << "********************* Infix To Postfix Conversion and evaluation
**********************" << endl;
    cout <<
"================================================================================
=============" << endl;
    cout << "Enter Expression:";
    cin >> exp;
    postfix=st.infixToPostFix(exp);
    cout << "Postfix Expression is " << postfix<<endl;
    if(st.checkIfDigit(postfix))
        st.findvalueofpostfix(postfix);
    return 0;
}
```

Output-1:

```
================================================================================
============
********************* Infix To Postfix Conversion and evaluation **********************
================================================================================
============
Enter Expression:(A+B)*(C+D)*(A+B+C)^K
Postfix Expression is AB+CD+*AB+C+K^*
```

Output-2:

==========

==========

********** Infix To Postfix Conversion and evaluation **********

==========

==========

Enter Expression:(1+2)*(3+4)+5

Postfix Expression is 12+34+*5+

Value of expression is 26

==========**End of Assignment**==========