DSA ASSIGNMENT 3

BT22CSH011
Devashish Athawale

Problem Statement:

Implementation of Sparse Matrix using Linked List

Sparse Matrix can be implemented using either a singly linked list or multi-linked list structure**.** You can use any representation of your choice.

Q1. Write a C program to read and display the linked list representation of the sparse matrix. Print the row, column, and value of each non-zero element.

Implementation of Multi-precision Arithmetic using Linked List

Q2. Write a C program to add 2 long integers which are represented using linked list and store the result in the resultant linked list.

Q1.)

```
#include<iostream>
class SparseMatrix
{
class MatrixNode
{
    public:
    int Data;
    unsigned int RowNum, ColumnNum;
    MatrixNode* NextLoc;
    MatrixNode()
    {
        Data = 0;
        RowNum = ColumnNum = 0;
        NextLoc = NULL;
    }
    MatrixNode(int data)
    {
        Data = data;
        RowNum = ColumnNum = 0;
        NextLoc = NULL;
    }
    MatrixNode(int data, unsigned int row, unsigned int column)
    {
        Data = data;
        RowNum = row;
        ColumnNum = column;
        NextLoc = NULL;
    }
    MatrixNode(int data, unsigned int row, unsigned int column,
    MatrixNode* nextLoc)
    {
        Data = data;
        RowNum = row;
        ColumnNum = column;
        NextLoc = nextLoc;
    }
};
```

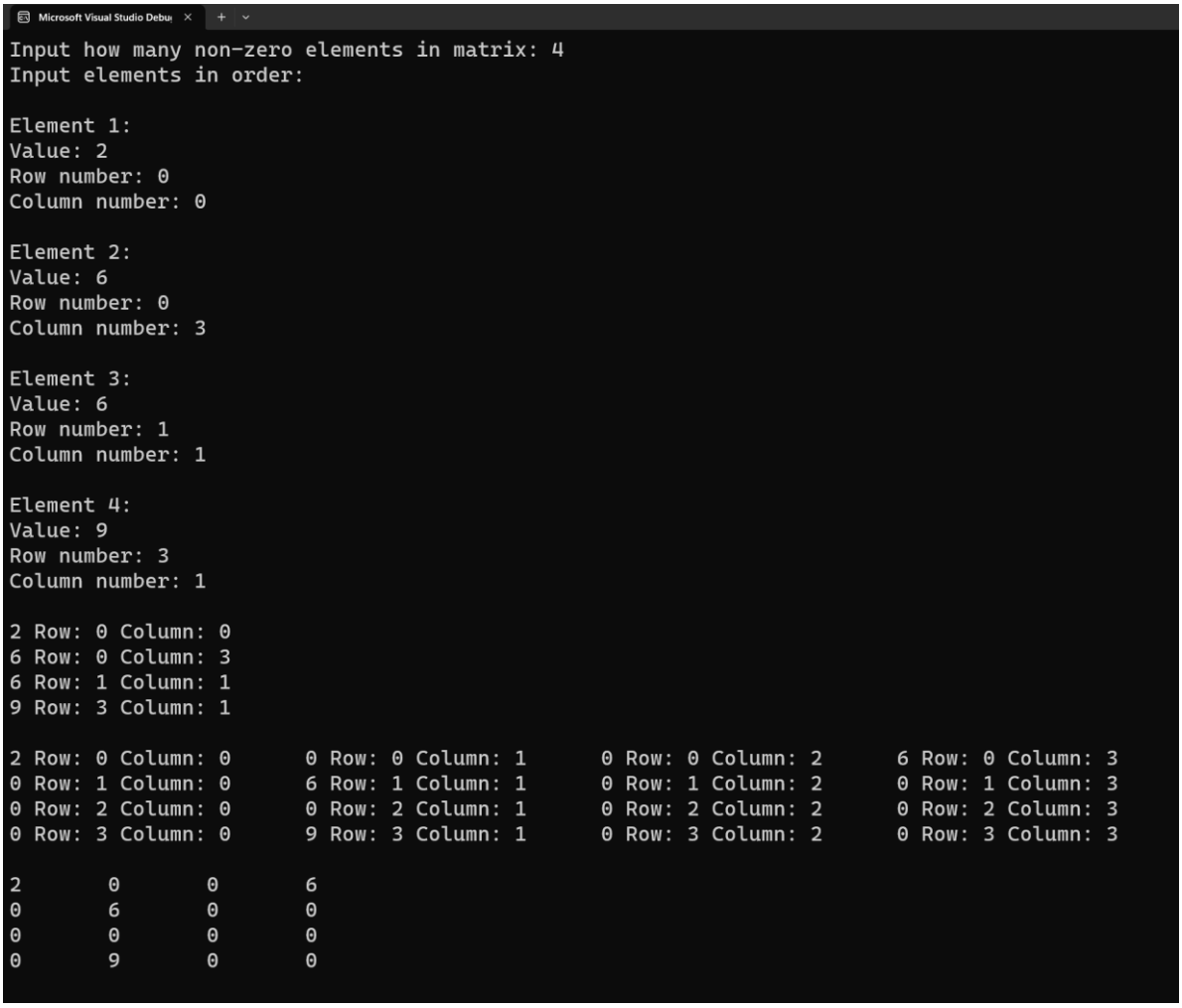
```
public:
MatrixNode* HEAD;
MatrixNode* TAIL;
unsigned int Rows, Columns;
SparseMatrix()
{
   HEAD = TAIL = NULL;
   Rows = Columns = 0;
}
void InsertElement(int INdata, unsigned int INrow, unsigned int INcolumn)
{
      if (HEAD)
      {
            MatrixNode* TEMP = new MatrixNode(INdata, INrow, INcolumn);
            TAIL->NextLoc = TEMP;
            TAIL = TEMP;
      }
      else
      {
            HEAD = new MatrixNode(INdata, INrow, INcolumn);
            TAIL = HEAD;
      }

      if (Rows < (INrow + 1))
      {
            Rows = INrow + 1;
      }
      if (Columns < (INcolumn + 1))
      {
            Columns = INcolumn + 1;
      }
      if (Columns > Rows)
      {
      Rows = Columns;
            }
      else
      {
            Columns = Rows;
      }
}
```

```
void Display()
{
	unsigned int CurrentRow = 0, CurrentColumn = 0;
	MatrixNode* CURRENT = HEAD;
	for (int i = 0; i < Rows; i++)
	{
	for (int j = 0; j < Columns; j++)
	{
	if (CURRENT)
	{
	if ((CURRENT->RowNum == i) && (CURRENT->ColumnNum == j))
	{
	std::cout << CURRENT->Data << " Row: " << i << " Column: " << j << "\t";
	CURRENT = CURRENT->NextLoc;
	}
	else
	{
	std::cout << "0 Row: " << i << " Column: " << j << "\t";
	}
	}
	else
	{
	std::cout << "0 Row: " << i << " Column: " << j << "\t";
	}
	}
	std::cout << std::endl;
	}
}


void DisplayNonZero()
{
	MatrixNode* CURRENT = HEAD;
	while (CURRENT)
	{
	std::cout << CURRENT->Data << " Row: " << CURRENT->RowNum << " Column: "
	<< CURRENT->ColumnNum << std::endl;
	CURRENT = CURRENT->NextLoc;
	}
}
```

```
    void DisplayOnlyElements()
    {
    unsigned int CurrentRow = 0, CurrentColumn = 0;
    MatrixNode* CURRENT = HEAD;
    for (int i = 0; i < Rows; i++)
    {
    for (int j = 0; j < Columns; j++)
    {
    if (CURRENT)
    {
    if ((CURRENT->RowNum == i) && (CURRENT->ColumnNum ==
    j))
    {
    std::cout << CURRENT->Data << "\t";
    CURRENT = CURRENT->NextLoc;
    }
    else
    {
    std::cout << "0" << "\t";
    }
    }
    else
    {
    std::cout << "0" << "\t";
    }
    }
    std::cout << std::endl;
    }
    }
};
```

```
void main()
{
	SparseMatrix matrix1;
	unsigned int inputColumn, inputRow;
	int inputData;
	int num = 0;
	std::cout << "Input how many non-zero elements in matrix: ";
	std::cin >> num;
	std::cout << "Input elements in order: " << std::endl;
	for (int i = 0; i < num; i++)
	{
		std::cout << "\nElement " << i + 1 << ":" << std::endl;
		std::cout << "Value: ";
		std::cin >> inputData;
		std::cout << "Row number: ";
		std::cin >> inputRow;
		std::cout << "Column number: ";
		std::cin >> inputColumn;
		matrix1.InsertElement(inputData, inputRow, inputColumn);
	}

	std::cout << std::endl;
	matrix1.DisplayNonZero();
	std::cout << std::endl;
	matrix1.Display();
	std::cout << std::endl;
	matrix1.DisplayOnlyElements();
}
```

Output:


Microsoft Visual Studio Debu
Input how many non-zero elements in matrix: 4
Input elements in order:
Element 1:
Value: 2
Row number: 0
Column number: 0
Element 2:
Value: 6
Row number: 0
Column number: 3
Element 3:
Value: 6
Row number: 1
Column number: 1
Element 4:
Value: 9
Row number: 3
Column number: 1
2 Row: 0 Column: 0
6 Row: 0 Column: 3
6 Row: 1 Column: 1
9 Row: 3 Column: 1
2 Row: 0 Column: 0 0 Row: 0 Column: 1 0 Row: 0 Column: 2 6 Row: 0 Column: 3
0 Row: 1 Column: 0 6 Row: 1 Column: 1 0 Row: 1 Column: 2 0 Row: 1 Column: 3
0 Row: 2 Column: 0 0 Row: 2 Column: 1 0 Row: 2 Column: 2 0 Row: 2 Column: 3
0 Row: 3 Column: 0 9 Row: 3 Column: 1 0 Row: 3 Column: 2 0 Row: 3 Column: 3
2 0 0 6
0 6 0 0
0 0 0 0
0 9 0 0

Q2.)

```
#include<iostream>
#include<string>
class LongInteger
{
    class Node
    {
        public:
        Node* nextLoc;
        int data;
        Node()
        {
        nextLoc = NULL;
        data = 0;
        }
        Node(int INData)
        {
        nextLoc = NULL;
        data = INData;
        }
    };
    public:
    Node* HEAD;
    Node* TAIL;
    int Size;
    LongInteger()
    {
        HEAD = NULL;
        TAIL = NULL;
        Size = 0;
    }
    void enterData(const char* data)
    {
        int currentLoc = 0;
        while (data[currentLoc] != '\0')
        {
        enterData(((int)data[currentLoc]) - 48);
        currentLoc++;
        }
    }
```

```
void enterData(char* data)
{
    int currentLoc = 0;
    while (data[currentLoc] != '\0')
    {
    enterData(((int)data[currentLoc]) - 48);
    currentLoc++;
    }
}
void enterData(int data)
{
    if (HEAD)
    {
    Node* TEMP = new Node(data);
    TEMP->nextLoc = HEAD;
    HEAD = TEMP;
    }
    else
    {
    HEAD = new Node(data);
    TAIL = HEAD;
    }
    Size++;
}
void enterDataAtBack(int data)
{
    if (HEAD)
    {
    Node* TEMP = new Node(data);
    TAIL->nextLoc = TEMP;
    TAIL = TEMP;
    }
    else
    {
    HEAD = new Node(data);
    TAIL = HEAD;
    }
    Size++;
}
```

```
bool isEmpty()
{
    if (HEAD)
    return false;
    else
    return true;
    }
    void displayBackwards()
    {
    if (HEAD)
    {
    Node* CURRENT = HEAD;
    while (CURRENT)
    {
    std::cout << CURRENT->data;
    CURRENT = CURRENT->nextLoc;
    }
    }
    std::cout << std::endl;
}
void display()
{
    char* temp = (char*)std::malloc(Size + 1);
    Node* TEMP = HEAD;
    temp[Size] = '\0';
    int currentLoc = Size - 1;
    while (TEMP)
    {
    temp[currentLoc] = (TEMP->data + 48);
    currentLoc--;
    TEMP = TEMP->nextLoc;
    }
    std::cout << temp << std::endl;
}
```

```
void erase(){
    while (HEAD)
    {
    Node* TEMP = HEAD;
    HEAD = HEAD->nextLoc;
    delete TEMP;
    }
    HEAD = NULL;
    TAIL = HEAD;
    Size = 0;
}
//returns a LongInteger that is sum of two input LongIntegers
static LongInteger* add(LongInteger* numA, LongInteger* numB) {
    int carry = 0;
    LongInteger* result = new LongInteger();
    Node* TEMPA = numA->HEAD;
    Node* TEMPB = numB->HEAD;
    while (TEMPA && TEMPB)
    {
    result->enterDataAtBack(((TEMPA->data + TEMPB->data) % 10) +
    carry);
    carry = (TEMPA->data + TEMPB->data) / 10;
    TEMPA = TEMPA->nextLoc;
    TEMPB = TEMPB->nextLoc;
    }
    while (TEMPA)
    {
    result->enterDataAtBack((TEMPA->data + carry) % 10);
    carry = (TEMPA->data + carry) / 10;
    TEMPA = TEMPA->nextLoc;
    }
    while (TEMPB)
    {
    result->enterDataAtBack((TEMPB->data + carry) % 10);
    carry = (TEMPB->data + carry) / 10;
    TEMPB = TEMPB->nextLoc;
    }
    if (carry==1)
    result->enterDataAtBack(1);
    return result;
}
};
```

```
void main()
{
    LongInteger number1, number2;
    number1.enterData("543467");
    number2.enterData("48315");number1.display();
    number2.display();


    LongInteger result = *LongInteger::add(&number1, &number2);
    result.display();
}
```

Output: