/*** *

- Author: Samuel Campbell
- Email: Sccampbell1019@my.msutexas.edu
- Label: P02
- Title: MyVector Class 2
- Course: CMPS 2143
- Semester: Fall 2021
-
- Description:

```
   Adds aditional functionality by overloading basic         *
operators
```

-
- Usage:

```
    MyVector V;
```

```
    V.function
```

-
- Files: in1.dat

```
                output.txt
```

**/

#include #include #include

using namespace std;

struct Node { //struct containing the basic variables needed for int data; //list traversal and modification Node *next; Node *prev; Node(int x) { data = x; next = NULL; } };

/**

- Class MyVector
-
- Description:

- contains the constructor and functions to modify the linked list
-
- Public Methods:
- - MyVector()
- - MyVector(int* Arr, int size)
- - MyVector(string filename)
- - void PushFront(int val)
- - void PushFront(MyVector V2)
- - void PushFront(int val)
- - void PushRear(int val)
- - void PushRear(MyVector V2)
- - int popAt(int x)
- - bool PushAt(int index, int val)
- - int PopFront()

- `- int PopRear()`
- `- int FindAt(int val)`
- `- void Print()`
- `- ~MyVector()`
-
- Usage:
-
- `MyVector V("in1.dat");`
- `MyVector V;`
- `MyVector V2;`
- `V.PushRear(19);`
- `V2.PushRear(10)`
- `V.Print();`
-
-

*/ class MyVector { private: Node *head; Node *tail; int size;

public: /** * Public : MyVector * * Description: * Default Constructor * * Params: *
*
* Returns: * none */ MyVector() { head = NULL; tail = NULL; size = 0; }

```
/**
* Public : MyVector
*
* Description:
*  constructor reads array and pushes into back of list

*
* Params:
*     int*  Arr
*     int   size
*
* Returns:
*     none
*/
MyVector(int *Arr, int asize)
{
    head = NULL;
    tail = NULL;
    size = 0;

    for (int i = 0; i < asize; i++)
    {
        PushRear(Arr[i]);
    }
}

/**
* Public : MyVector( string filename)
*
* Description:
*  constructor using data from file

*
* Params:
*     string filename

*
* Returns:
*     none
*/
MyVector(string filename)
{
    head = NULL;
    tail = NULL;
    size = 0;
```

```
    ifstream fin;
    int x;
    fin.open(filename);
    while (!fin.eof())
    {
        fin >> x;
        PushRear(x);
    }
}

// head
//+-----+   +-----+   +-----+
//|  8  |-> |  2  |-> |  3  |->NULL
//+-----+   +-----+   +-----+
//      0         1         2
//                          T
// index = 0
// cout<<V[2]<<endl;
```

/** * Public : int &operator[](int index) * * Description: * overloading the []operator will loop until travel is positioned in the correct node

```
*
* Params:
*    int index

*
* Returns:
*     int travel->data
*/
// return type name (params list)
int &operator[](int index)
{
    Node *travel = head;
    //cout<<travel->data<<endl;
    while (travel)
    {
        travel = travel->next;
        index--;
        if (index <= 0)
        {
            break;
        }
    }
    return travel->data;
}

/**
* Public : PushFront
```

```
*
* Description:
*  Takes in int val and pushes to the front of the list

*
* Params:
*     int val

*
* Returns:
*     Void
*/
void PushFront(int val)
{
    Node *Temp = new Node(val);
    if (head == NULL)
    {
        head = Temp;
        tail = head;
        size++;
    }
    else
    {
        Temp->next = head;
        head = Temp;
        size++;
    }
}

/**
* Public : PushFront
*
* Description:
*  Takes in int val and pushes to the front of the list

*
* Params:
*     MyVector V2

*
* Returns:
*     Void
*/
void PushFront(MyVector V2)
{
    int x = V2.PopRear();
    while (x != -1)
    {
        PushFront(x);
        x = V2.PopRear();
        cout << x;
```

```
        size++;
    }
}

// cout<<(V1 + V2)<<endl;
// V1.Add(V2); synonym
// V1 is `this` and V2 is `rhs`
// V1 or `this` = {  1,  2,  3, 4, 5 }; +
// V2 or `rhs` = { 10, 20, 30       };
// { 11, 22, 33, 4, 5 }

/**
* Public : MyVector operator+
*
* Description:
*  Adds 2 lists of equal size. Stores results in new vector

*
* Params:
*     const MyVector &rhs

*
* Returns:
*     NewVector
*/
MyVector operator+(const MyVector &rhs)
{
    MyVector NewVector;

    Node *travelThis = head;
    Node *travelRhs = rhs.head;

    while (travelThis != nullptr && travelRhs != nullptr)
    {
        NewVector.PushRear(travelThis->data + travelRhs->data);
        travelThis = travelThis->next;
        travelRhs = travelRhs->next;
    }

    while (travelThis != nullptr)
    {
        NewVector.PushRear(travelThis->data);
        travelThis = travelThis->next;
    }

    while (travelRhs != nullptr)
    {
        NewVector.PushRear(travelRhs->data);
        travelRhs = travelRhs->next;
    }
```

```
    return NewVector;
}

/**
* Public : MyVector operator-
*
* Description:
*  Subtracts 2 lists of equal size. Stores results in new vector

*
* Params:
*     const MyVector &rhs

*
* Returns:
*     NewVector
*/
MyVector operator-(const MyVector &rhs)
{
    MyVector NewVector;

    Node *travelThis = head;
    Node *travelRhs = rhs.head;

    while (travelThis != nullptr && travelRhs != nullptr)
    {
        NewVector.PushRear(travelThis->data - travelRhs->data);
        travelThis = travelThis->next;
        travelRhs = travelRhs->next;
    }

    while (travelThis != nullptr)
    {
        NewVector.PushRear(travelThis->data);
        travelThis = travelThis->next;
    }

    while (travelRhs != nullptr)
    {
        NewVector.PushRear(travelRhs->data);
        travelRhs = travelRhs->next;
    }

    return NewVector;
}

/**
* Public : MyVector operator*
*
* Description:
*  Multiplies 2 lists of equal size. Stores results in new vector
```

```
*
* Params:
*     const MyVector &rhs

*
* Returns:
*     NewVector
*/
MyVector operator*(const MyVector &rhs)
{
    MyVector NewVector;

    Node *travelThis = head;
    Node *travelRhs = rhs.head;

    while (travelThis != nullptr && travelRhs != nullptr)
    {
        NewVector.PushRear(travelThis->data * travelRhs->data);
        travelThis = travelThis->next;
        travelRhs = travelRhs->next;
    }

    while (travelThis != nullptr)
    {
        NewVector.PushRear(travelThis->data);
        travelThis = travelThis->next;
    }

    while (travelRhs != nullptr)
    {
        NewVector.PushRear(travelRhs->data);
        travelRhs = travelRhs->next;
    }

    return NewVector;
}

/**
* Public : MyVector operator/
*
* Description:
*  Divides 2 lists of equal size. Stores results in new vector

*
* Params:
*     const MyVector &rhs

*
* Returns:
*     NewVector
```

```
*/
MyVector operator/(const MyVector &rhs)
{
    MyVector NewVector;

    Node *travelThis = head;
    Node *travelRhs = rhs.head;

    while (travelThis != nullptr && travelRhs != nullptr)
    {
        NewVector.PushRear(travelThis->data / travelRhs->data);
        travelThis = travelThis->next;
        travelRhs = travelRhs->next;
    }

    while (travelThis != nullptr)
    {
        NewVector.PushRear(travelThis->data);
        travelThis = travelThis->next;
    }

    while (travelRhs != nullptr)
    {
        NewVector.PushRear(travelRhs->data);
        travelRhs = travelRhs->next;
    }

    return NewVector;
}

/**
* Public : MyVector operator==
*
* Description:
*  Overloads the comparing operator, allowing use check if two lists are equal to
one another. If so it returns true, if not false

*
* Params:
*     const MyVector &rhs

*
* Returns:
*     bool
*/
bool operator==(const MyVector &rhs)
{
    MyVector NewVector;

    Node *travelThis = head;
    Node *travelRhs = rhs.head;
```

```
    if (travelThis == nullptr && travelRhs == nullptr)
    {
        return true;
    }
    while (travelThis != nullptr && travelRhs != nullptr)
    {
        if (travelThis->data == travelRhs->data)
        {
            // NewVector.PushRear(travelThis->data==travelRhs->data);
        }
        else
        {
            // NewVector.PushRear(travelThis->data!=travelRhs->data);
            return false;
        }

        travelThis = travelThis->next;
        travelRhs = travelRhs->next;
    }

    // while(travelThis != nullptr ){
    //   NewVector.PushRear(travelThis->data);
    //   travelThis = travelThis->next;
    // }

    // while(travelRhs != nullptr){
    //   NewVector.PushRear(travelRhs->data);
    //   travelRhs = travelRhs->next;
    // }

    return true;
}
/**
* Public : PushRear
*
* Description:
*  Takes in int val and pushes to the back of the list

*
* Params:
*     int val

*
* Returns:
*     Void
*/
void PushRear(int val)
{
    Node *Temp = new Node(val);
    if (head == NULL)
```

```
    {
        head = Temp;
        tail = head;
        size++;
    }
    else
    {
        tail->next = Temp;
        tail = Temp;
        size++;
    }
}

/**
* Public : PushRear
*
* Description:
*  Takes in MyVector and pushes to the back of the list

*
* Params:
*     MyVector V2

*
* Returns:
*     Void
*/
void PushRear(MyVector V2)
{
    int x = V2.PopFront();
    while (x != -1)
    {
        PushRear(x);
        x = V2.PopFront();
        size++;
        // cout<<x;
    }
}

/**
* Public : PopAt
*
* Description:
*  takes in a index and pops at that location in list(use size for this then loop
index amount of times then place node there)

*
* Params:
*     int size

*
```

```
* Returns:
*      value if the node at the said index
*/
int popAt(int x)
{
    if (x >= size)
    {
        return -1;
    }
    else
    {
        Node *prev = NULL;
        Node *temp = head;
        int loc = 0;
        while (loc != x)
        {
            prev = temp;
            temp = temp->next;
            loc++;
        }
        prev->next = temp->next;
        int cont = temp->data;
        delete temp;
        return cont;
    }
}

/**
* Public : PushAt
*
* Description:
*      Pushes a node carying a value at a certain index
   takes in a index and puts at that location in list(use size for this then loop
index amount of times then place node there)
*
* Params:
*     int index
      int val
*
* Returns:
*      bool  : whether it can successfuly enter in a value
*/

bool PushAt(int index, int val)
{
    Node *prev = head; // get previous and next pointers
    Node *current = head;
    Node *nNode = new Node(val); // needed ne memory for new value

    while (index > 0)
    {
```

```
            prev = current;
            current = current->next;
            index--;
        }
        cout << prev->data << "," << current->data << endl;
        prev->next = nNode;    // Need to point prev (next) to the new memory.
        nNode->next = current; // Need to point nNode's next to current.

        size++;
        return true;
    }

    /**
    * Public : PopFront
    *
    * Description:
    *      Pops front value from list
    *
    * Params:
    *      None
    *
    * Returns:
    *      int  : value at front
    */
    int PopFront()
    {
        if (head == NULL)
        {
            return -1;
        }
        else
        {
            int value = head->data;
            Node *Temp = head;
            head = head->next;
            delete Temp;
            size--;
            return value;
        }
    }

    /**
    * Public : PopRear
    *
    * Description:
    *      Pops rear value from list
    *
    * Params:
    *      None
    *
    * Returns:
```

```
*      int  : value at rear
*/
int PopRear()
{
    if (head == NULL)
    {
        return -1;
    }
    else
    {
        int value = tail->data;
        Node *Temp = tail;
        tail = tail->next;
        delete Temp;
        size--;
        return value;
    }
}

/**
* Public : Find
*
* Description:
*      trys to see if the value is in list if not return -1
*
* Params:
*      int val
*
* Returns:
*      int  : index
*/
int Find(int val)
{
    Node *current = head;
    int size = 0;
    while (current != NULL)
    {
        current = current->next;
        size++;
    }
    current = head;
    for (int i = 0; i < size; i++)
    {
        if (current->data == val)
        {
            cout << val << " found at index: " << i << endl;
            return -1;
        }
        else
            current = current->next;
    }
```

```
    cout << val << " not found" << endl;
    return -1;
}
```

/** * Public : MyVector operator= * * Description: * allows us to assign

```
*
* Params:
*     const MyVector &rhs

*
* Returns:
*     NewVector
*/
MyVector& operator=(const MyVector &rhs)
{   //reset this object
    head = tail = nullptr;
    size = 0;

    // get all the values from rhs
    Node *travel = rhs.head;

    while (travel != nullptr)
    {
        cout<<travel->data<<endl;
        PushRear(travel->data);
        travel = travel->next;
    }
    return *this;
```

}

```
/**
* Public : Print
*
* Description:
*      prints the list
*
* Params:
*      none
*
* Returns:
*      void
*/
void Print()
{
    Node *Temp = head;
```

```
        while (Temp != NULL)
        {
            cout << Temp->data << "->";
            Temp = Temp->next;
        }
        cout << endl;
    }

    friend ostream &operator<<(ostream &os, const MyVector &other)
    {
        Node *Temp = other.head;
        while (Temp != NULL)
        {
            os << Temp->data << "->";
            Temp = Temp->next;
        }
        os << endl;
        return os;
    }

    friend fstream &operator<<(fstream &fs, const MyVector &other)
    {
        Node *Temp = other.head;
        while (Temp != NULL)
        {
            fs << Temp->data << "->";
            Temp = Temp->next;
        }
        fs << endl;
        return fs;
    }

    ~MyVector()
    {
        Node *curr = head;
        Node *prev = head;
        while (curr)
        {
            prev = curr;
            curr = curr->next;
            delete prev;
        }
    }
```

};

// { 1, 2, 3, 4, 5 }; + // { 10, 20, 30 }; // { 11, 22, 33, 4, 5 }

int main() { ofstream fs; fs.open("outfile.txt"); // MyVector V("in1.dat"); //MyVector V; int a1[] = {1, 2, 3, 4, 5}; int a2[] = {10, 20, 30};

```
    MyVector V1(a1, 5);
    MyVector V2(a2, 3);

    cout << V1 << endl;
    fs << V1 << endl;
    V1[2] = 9;
    cout << "V1: " << V1 << endl;
    cout << "V2: " << V2 << endl;
    cout << "V1-V2: " << (V1 - V2) << endl;
    cout << "V1+V2: " << (V1 + V2) << endl;
    //print to file
    fs << "V1: " << V1 << endl;
    fs << "V2: " << V2 << endl;
    fs << "V1-V2: " << (V1 - V2) << endl;
    fs << "V1+V2: " << (V1 + V2) << endl;

    MyVector V3 = V1 + V2;
    MyVector V5 = V3 - V2;

    cout << "V3: " << V3 << endl;
    cout << "V5: " << V5 << endl;
    //print to file
    fs << "V3: " << V3 << endl;
    fs << "V5: " << V5 << endl;


    V2[1] = 99;

    V3 = V3 + V1;

    cout << "V3: " << V3 << endl;
    //print
    fs << "V3: " << V3 << endl;

    MyVector V4;
    V4 = V1;
    cout << (V4 == V1)<<endl;
    cout << "V4: " << V4 << endl;
    //print file
    fs << (V4 == V1)<<endl;
    fs << "V4: " << V4 << endl;

    fs.close();
    return 0;
```

}