

```

// Vahe Karamian - Project I - Dr. Lee - CS 241
// Filename: dllistnd.h
// =====
// DLLISTND.H
// Doubly Linked List Node template definition

#ifndef DLLISTND_H
#define DLLISTND_H

template<class NODETYPE>
class DLList; // This is needed for "Forward visibility"

template<class NODETYPE>
class DLListNode
{
    friend class DLList<NODETYPE>; // make DLList a friend

public:
    DLListNode(const NODETYPE &); // constructor
    NODETYPE getData() const; // return the data in the node

private:
    NODETYPE data; // data
    DLListNode *prevPtr, *nextPtr; // pointers to prev and next nodes
};

// Constructor
template<class NODETYPE>
DLListNode<NODETYPE>::DLListNode(const NODETYPE &info)
{
    data = info; // Put the data
    prevPtr=nextPtr = 0; // Set the pointers to null
}

// Return a copy of the data in the node
template<class NODETYPE>
NODETYPE DLListNode<NODETYPE>::getData() const { return data;}

#endif

// Vahe Karamian - Project I - Dr. Lee - CS 241
// Filename: dllist.h
// =====
// DDLIST.H
// Template Doubly Link List class definition

#ifndef DDLIST_H //just a precaution -- avoid multiple entry of this
#define DDLIST_H //class definition.

#include <iostream.h> // For console I/O
#include <fstream.h> // for file I/O
#include <assert.h> // Not essential but simpler than exception handler
#include "dllistnd.h" //Doubly Linked list node definition file

typedef float Matrix3x3[3][3]; // 2D Matrix
typedef float Matrix4x4[4][4]; // 3D Matrix

typedef struct { int s; float x, y; } POINT; // point structure 2D
typedef struct { int s; float x, y, z; } POINT3D; // point structure 3D

// defines
#define PI 3.1415926535897932384626433832795
#define WIDTH 640
#define HEIGHT 480

template<class NODETYPE> // template class is much more flexible
class DLList
{
public:
    DLList(); // constructor
    ~DLList(); // destructor
    void insert(const NODETYPE &); // "general" insert--calls above two
    int remove(DLListNode<NODETYPE> *); // delete this node
    DLListNode<NODETYPE> * find(NODETYPE &); // find the location of this value
    void toFile(char *); // Store to a file whose name is given later
    int isEmpty() const; // Non-destructive access function
    void print() const; // Traverse the list and print

```

```

// The following functions have been added because of GRAPHICS
void insertPoint(const NODETYPE &); // used for GRAPHICS 2D POINT (state,x,y)
void displayData() const; // Display the data in the list in GRAPHICS format
void changeResolution(); // change resolution

// Geometric transformation functions 2D
void matrix3x3SetIdentity(Matrix3x3 m);
void matrix3x3SetIdentity(); // for the CLASS

void matrix3x3PreMultiply(Matrix3x3 a, Matrix3x3 b);
void matrix3x3PreMultiply(); // for the CLASS

void translate2(int tx, int ty);
void scale2(float sx, float sy, POINT refPt);
void rotate2(float a, POINT refPt);

void transformPoints2(int npts, POINT *pts);
void transformPoints2(); // for the CLASS
// =====

// Geometric transformation functions 3D
void matrix4x4SetIdentity(Matrix4x4 m);
void matrix4x4SetIdentity(); // for the CLASS

void matrix4x4PreMultiply(Matrix4x4 b, Matrix4x4 c);
void matrix4x4PreMultiply(); // for the CLASS

void translate3(float tx, float ty, float tz);
void scale3(float sx, float sy, float sz, POINT3D center);
void rotate3(POINT3D pl, POINT3D p2, float radianAngle);

void transformPoints3(int nPts, POINT3D *pts);
void transformPoints3(); // for the CLASS
// =====

// I don't think we are going to need this
void setTheMatrix();

// This is a function which I have to fix
void setWcPt3(POINT3D &, float, float, float);

float ToRadians(float a); // convert from degrees to radians

private:
DLLListNode<NODETYPE> *firstPtr; // pointer to first node
DLLListNode<NODETYPE> *lastPtr; // pointer to last node

// Utility function to allocate a new node
DLLListNode<NODETYPE> *getNewNode(const NODETYPE &);

Matrix3x3 theMatrix2D; // create The Matrix 2D
Matrix4x4 theMatrix3D; // create The Matrix 3D
Matrix3x3 a, b; // more matrix for multiplication
Matrix4x4 c, d; // more matrix for multiplication
};

// Default constructor
template<class NODETYPE>
DLLList<NODETYPE>::DLLList() { firstPtr = lastPtr = 0; } // set pointers to null

// Destructor
template<class NODETYPE>
DLLList<NODETYPE>::~DLLList()
{
    if (!isEmpty())
    {
        // DLLList is not empty
        cout << "Remove the nodes before exiting --- " << endl;
        DLLListNode<NODETYPE> *currentPtr = firstPtr, *tempPtr; //define pointers
        while (currentPtr != lastPtr)
        {
            // delete remaining nodes
            tempPtr = currentPtr; // but stop at last node.

            cout << tempPtr->data.s << "\t "; // print the state
            cout << tempPtr->data.x << "\t "; // print x-value
            cout << tempPtr->data.y << endl; // print y-value
        }
    }
}

```

```

        currentPtr = currentPtr->nextPtr; //move to next node
        delete tempPtr; // recycle this space
    }
    // Now take care of last node.
    cout << currentPtr->data.s << "\t "; // print the state
    cout << currentPtr->data.x << "\t "; // print x-value
    cout << currentPtr->data.y << endl; // print y-value
    delete currentPtr; // recycle this space
}
cout << "\nMemory spaces occupied by nodes are freed \n\n";
}

// Insert a node to the list
template<class NODETYPE>
void DLLList<NODETYPE>::insert(const NODETYPE &value)
{
    DLLListNode<NODETYPE> *newPtr = getNewNode(value); // create new node
    if (isEmpty())
    {
        // DLLList is empty
        firstPtr = lastPtr = newPtr; // Pointers from the list points to this node.
        newPtr->prevPtr = newPtr->nextPtr = newPtr; // Both pointers points itself
    }
    else
    {
        // DLLList is not empty
        // insert somewhere in the middle
        DLLListNode<NODETYPE> *tempPtr = firstPtr; // temporary pointer for traversal

        if(value.x < firstPtr->data.x) // new element is smallest
            firstPtr = newPtr; // So, insert before first
        else if (value.x > lastPtr->data.x) // new element is the largest one.
            lastPtr = newPtr;

        // until it finds a larger value or end of list
        else
            while (newPtr->data.x > tempPtr->data.x && tempPtr!=lastPtr)
                tempPtr=tempPtr->nextPtr; // Keep moving forward.
        // Then insert before the tempPtr node
        newPtr->prevPtr = tempPtr->prevPtr; // previous pointer of new node
        newPtr->nextPtr = tempPtr; // next of new node is temp node.
        tempPtr->prevPtr-> nextPtr= newPtr; // previous node links up
        tempPtr->prevPtr = newPtr; //next node links up
    } // else of general insert
    // Update first and last pointer if necessary.
} // insert function

// Insert a node into the list (2D Vertex (state,x,y))
template<class NODETYPE>
void DLLList<NODETYPE>::insertPoint(const NODETYPE &value)
{
    DLLListNode<NODETYPE> *newPtr = getNewNode(value); // create new node
    if (isEmpty())
    {
        // DLLList is empty
        firstPtr = lastPtr = newPtr; // Pointers from the list points to this node.
        newPtr->prevPtr = newPtr->nextPtr = newPtr; // Both pointers points itself
    }
    else
    {
        DLLListNode<NODETYPE> *tempPtr = firstPtr; // temporary pointer for traversal
        lastPtr = newPtr;

        // Then insert before the tempPtr node
        newPtr->prevPtr = tempPtr->prevPtr; // previous pointer of new node
        newPtr->nextPtr = tempPtr; // next of new node is temp node.
        tempPtr->prevPtr->nextPtr = newPtr; // previous node links up
        tempPtr->prevPtr = newPtr; //next node links up
    }
} // insertPoint function

// Delete a node from the list. Pass the pointer.
template <class NODETYPE>
int DLLList<NODETYPE>::remove(DLLListNode<NODETYPE> *ptr)
{
    if (ptr)
    {
        cout << ptr->data.s << "\t "; // print the state
    }
}

```

```

    cout << ptr->data.x << "\t "; // print x-value
    cout << ptr->data.y << endl; // print y-value
}
if (ptr == NULL)
{
    // There is no node to delete
    cout << " This value does not exist and can not be deleted\n\a";
    return -1;
}
if (firstPtr == lastPtr && ptr == lastPtr)
{
    // single element
    firstPtr = lastPtr = 0; // make it empty
    delete ptr; // and recycle this node
}
else
{
    // at least two elements--do pointer operation
    ptr->nextPtr->prevPtr = ptr-> prevPtr; // next node get updated
    ptr->prevPtr->nextPtr = ptr-> nextPtr; // previous node get updated
}
if (ptr == firstPtr) // was this the first node?
    firstPtr=ptr->nextPtr; // then say so

if (ptr == lastPtr) // was this the last node?
    lastPtr=ptr->prevPtr; // then say so

delete ptr; // Now recycle this node.
return 1; // delete successful
} // delete a node

//find the location of this value
template<class NODETYPE>
DLLListNode<NODETYPE> * DLLList<NODETYPE>::find(NODETYPE &value)
{
    if(!firstPtr) // empty list--no answer
        return NULL; // Exit with null value

    if(firstPtr->data.x == value.x) // The value is at the first node
        return firstPtr; // exit and return first pointer

    if(lastPtr->data.x == value.x) // The value is at the last node
        return lastPtr; // exit and return lastpointer

    DLLListNode<NODETYPE> *tempPtr = firstPtr; // temporary pointer for traverse
    // Better yet, make temptr=firstPtr->nextPtr;
    while(tempPtr != lastPtr)
    {
        if(value.x == tempPtr->data.x) // Eureka! found the node having this value!
            return tempPtr;
        if(value.x < tempPtr->data.x) // Pointer passed the area--obviously this
            return NULL; // value does not exist.
        tempPtr=tempPtr->nextPtr; // Still more to go
    }
    return NULL; // if not found, null pointer is returned.
} // find function

// Is the DLLlist empty?
template<class NODETYPE>
int DLLList<NODETYPE>::isEmpty() const { return firstPtr == 0; }

// Return a pointer to a newly allocated node
template<class NODETYPE>
DLLListNode<NODETYPE> *DLLList<NODETYPE>::getNewNode(const NODETYPE &value)
{
    DLLListNode<NODETYPE> *ptr = new DLLListNode<NODETYPE>(value);
    assert(ptr != 0);
    return ptr;
}

// save this list to a file
template<class NODETYPE>
void DLLList<NODETYPE>::toFile(char * filename)
{
    // caller supplies fn
    ofstream odatfl (filename, ios::out); // File definition

    if (isEmpty()) // no data to store

```

```

    return;
DLLListNode<NODETYPE> *currentPtr = firstPtr; // a temporat pointer
while (currentPtr != lastPtr)
{
    // Keep traverse the list
    odatfl << currentPtr->data.s <<"\t "; // store this data
    odatfl << currentPtr->data.x <<"\t ";
    odatfl << currentPtr->data.y <<endl;
    currentPtr = currentPtr->nextPtr; // and move over
}

odatfl << currentPtr->data.s <<"\t "; // store Last Data
odatfl << currentPtr->data.x <<"\t ";
odatfl << currentPtr->data.y <<endl;

odatfl.close(); // Close the file. Not mandatory but a good habit.
}

// Display the contents of the DLLList -- forward and backward
template<class NODETYPE>
void DLLList<NODETYPE>::print() const
{
    if (isEmpty())
    {
        cout << "The list is empty" << endl << endl;
        return;
    }

    DLLListNode<NODETYPE> *currentPtr = firstPtr; //a temporat pointer

    cout << "\n\nThe list in forward order is: ";

    while (currentPtr != lastPtr)
    {
        // Keep traverse the list
        cout << currentPtr->data.s << "\t "; // you can use manipulation routines
        cout << currentPtr->data.x << "\t "; // you can use manipulation routines
        cout << currentPtr->data.y << endl; // you can use manipulation routines

        currentPtr = currentPtr->nextPtr; // move to next node.
    }
    cout << currentPtr->data.s << "\t "; // you can use manipulation routines
    cout << currentPtr->data.x << "\t "; // you can use manipulation routines
    cout << currentPtr->data.y << endl; // you can use manipulation routines
    cout << endl; // line skip

    cout << "The list in reverse order is: ";

    while (currentPtr != firstPtr)
    {
        // Keep traverse the list
        cout << currentPtr->data.s << "\t "; // you can use manipulation routines
        cout << currentPtr->data.x << "\t "; // you can use manipulation routines
        cout << currentPtr->data.y << endl; // you can use manipulation routines
        currentPtr = currentPtr->prevPtr; // move to next node.
    }
    cout << currentPtr->data.s << "\t "; // you can use manipulation routines
    cout << currentPtr->data.x << "\t "; // you can use manipulation routines
    cout << currentPtr->data.y << endl; // you can use manipulation routines
    cout << endl << endl; // a couple of line skips
}

// Display the data file in a graphics format
template<class NODETYPE>
void DLLList<NODETYPE>::displayData() const
{
    int x1, y1, x2, y2;
    int temp_x, temp_y;

    if (isEmpty())
    {
        cout << "The list is empty" << endl << endl;
        return;
    }

    DLLListNode<NODETYPE> *currentPtr = firstPtr; //a temporat pointer

    while (currentPtr != lastPtr)
    {

```

```

    x1 = (int) currentPtr->data.x;
    y1 = (int) currentPtr->data.y;

    if(currentPtr->data.s == 0)
    {
        x2 = (int) currentPtr->data.x;
        y2 = (int) currentPtr->data.y;
    }
    else if(currentPtr->data.s == 1)
    {
        temp_x = x2;
        temp_y = y2;
        x2 = (int) currentPtr->data.x;
        y2 = (int) currentPtr->data.y;
        x1 = temp_x;
        y1 = temp_y;
    }

    line(x1,y1,x2,y2);          // draw the line
    currentPtr = currentPtr->nextPtr; // move to next node
}

x1 = (int) currentPtr->data.x;
y1 = (int) currentPtr->data.y;

// Handle the last node
if(currentPtr->data.s == 0)
{
    x2 = (int) currentPtr->data.x;
    y2 = (int) currentPtr->data.y;
}
else if(currentPtr->data.s == 1)
{
    temp_x = x2;
    temp_y = y2;
    x2 = (int) currentPtr->data.x;
    y2 = (int) currentPtr->data.y;
    x1 = temp_x;
    y1 = temp_y;
}

// Turn this option on before you execute the program in DOS
line(x1,y1,x2,y2);          // draw the line
}

// Geometric Transformation Functions 2D
// original matrix identity function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix3x3SetIdentity(Matrix3x3 m)
{
    int i, j;
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            m[i][j] = (i==j);
}

// New matrix identity function!!!
template<class NODETYPE>
void DLLList<NODETYPE>::matrix3x3SetIdentity()
{
    int i, j;
    for(i=0; i<3; i++)
        for(j=0; j<3; j++)
            theMatrix2D[i][j] = (i==j);
}

// original matrix multiplication function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix3x3PreMultiply(Matrix3x3 a, Matrix3x3 b)
{
    int r, c;
    Matrix3x3 temp;

    for(r=0; r<3; r++)
        for(c=0; c<3; c++)
            temp[r][c] = a[r][0]*b[0][c] + a[r][1]*b[1][c] + a[r][2]*b[2][c];

    for(r=0; r<3; r++)
        for(c=0; c<3; c++)

```

```

        b[r][c] = temp[r][c];
    }

// New matrix multiplication function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix3x3PreMultiply()
{
    int r, c;
    Matrix3x3 temp;

    for(r=0; r<3; r++)
        for(c=0; c<3; c++)
            temp[r][c] = a[r][0]*b[0][c] + a[r][1]*b[1][c] + a[r][2]*b[2][c];

    for(r=0; r<3; r++)
        for(c=0; c<3; c++)
            b[r][c] = temp[r][c];
}

// original translation function
template<class NODETYPE>
void DLLList<NODETYPE>::translate2(int tx, int ty)
{
    Matrix3x3 m;

    matrix3x3SetIdentity(m);
    m[0][2] = (float) tx;
    m[1][2] = (float) ty;
    matrix3x3PreMultiply(m, theMatrix2D);
}

// original scale function
template<class NODETYPE>
void DLLList<NODETYPE>::scale2(float sx, float sy, POINT refPt)
{
    Matrix3x3 m;

    matrix3x3SetIdentity(m);
    m[0][0] = sx;
    m[0][2] = (1-sx) * refPt.x;
    m[1][1] = sy;
    m[1][2] = (1-sy) * refPt.y;
    matrix3x3PreMultiply(m, theMatrix2D);
}

// original rotation function
template<class NODETYPE>
void DLLList<NODETYPE>::rotate2(float a, POINT refPt)
{
    Matrix3x3 m;

    matrix3x3SetIdentity(m);
    a = ToRadians(a);
    m[0][0] = (float) cos(a);
    m[0][1] = (float) -sin(a);
    m[0][2] = (float) (refPt.x*(1-cos(a)) + refPt.y*sin(a));
    m[1][0] = (float) sin(a);
    m[1][1] = (float) cos(a);
    m[1][2] = (float) (refPt.y*(1-cos(a)) - refPt.x*sin(a));
    matrix3x3PreMultiply(m, theMatrix2D);
}

// original transformation function
template<class NODETYPE>
void DLLList<NODETYPE>::transformPoints2(int npts, POINT *pts)
{
    int k;
    float tmp;

    for(k=0; k<npts; k++)
    {
        tmp = theMatrix2D[0][0] * pts[k].x + theMatrix2D[0][1] * pts[k].y + theMatrix2D[0][2];
        pts[k].y = theMatrix2D[1][0] * pts[k].x + theMatrix2D[1][1] * pts[k].y + theMatrix2D[1][2];
        pts[k].x = tmp;
    }
}

// New implemented transformation function!!!
template<class NODETYPE>

```

```

void DLLList<NODETYPE>::transformPoints2()
{
    float tmp;

    if(!isEmpty())
    {
        // DLLList is not empty
        DLLListNode<NODETYPE> *currentPtr = firstPtr, *tempPtr; // define pointers
        while(currentPtr != lastPtr)
        {
            tempPtr = currentPtr; // stop at last node
            tmp = theMatrix2D[0][0] * tempPtr->data.x + theMatrix2D[0][1] * tempPtr->data.y + theMatrix2D[0][2];
            tempPtr->data.y = theMatrix2D[1][0] * tempPtr->data.x + theMatrix2D[1][1] * tempPtr->data.y + theMatrix2D[1][2];
;
            tempPtr->data.x = tmp;
            currentPtr = currentPtr->nextPtr; // move to next node
        }
        // now take care of the last node
        tmp = theMatrix2D[0][0] * currentPtr->data.x + theMatrix2D[0][1] * currentPtr->data.y + theMatrix2D[0][2];
        currentPtr->data.y = theMatrix2D[1][0] * currentPtr->data.x + theMatrix2D[1][1] * currentPtr->data.y + theMatrix2D
[1][2];
        currentPtr->data.x = tmp;
    } // transformation complete
}
// End of 2D Geometric Transformation Functions ++++++
// =====

// Geometric Transformation Functions 3D
// original matrix identity function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix4x4SetIdentity(Matrix4x4 m)
{
    int r, c;

    for(r=0; r<4; r++)
        for(c=0; c<4; c++)
            m[r][c] = (r==c);
}

// New matrix identity function!!!
template<class NODETYPE>
void DLLList<NODETYPE>::matrix4x4SetIdentity()
{
    int r, c;
    for(r=0; r<4; r++)
        for(c=0; c<4; c++)
            theMatrix3D[r][c] = (r==c);
}

// Multiplies matrix c times d, putting result in d
// original matrix multiplication function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix4x4PreMultiply(Matrix4x4 c, Matrix4x4 d)
{
    int r, c;
    Matrix4x4 temp;

    for(r=0; r<4; r++)
        for(c=0; c<4; c++)
            temp[r][c] = c[r][0]*d[0][c] + c[r][1]*d[1][c] + c[r][2]*d[2][c] + c[r][3]*d[3][c];

    for(r=0; r<4; r++)
        for(c=0; c<4; c++)
            d[r][c] = temp[r][c];
}

// New matrix multiplication function
template<class NODETYPE>
void DLLList<NODETYPE>::matrix4x4PreMultiply()
{
    int r, c;
    Matrix4x4 temp;

    for(r=0; r<4; r++)
        for(c=0; c<4; c++)
            temp[r][c] = c[r][0]*d[0][c] + c[r][1]*d[1][c] + c[r][2]*d[2][c] + c[r][3]*d[3][c];

    for(r=0; r<4; r++)
        for(c=0; c<4; c++)

```



```

        d[r][c] = temp[r][c];
    }

// original translation function
template<class NODETYPE>
void DLLList<NODETYPE>::translate3(float tx, float ty, float tz)
{
    Matrix4x4 m;

    matrix4x4SetIdentity(m);
    m[0][3] = tx;
    m[1][3] = ty;
    m[2][3] = tz;
    matrix4x4PreMultiply(m, theMatrix3D);
}

// original scaling function
template<class NODETYPE>
void DLLList<NODETYPE>::scale3(float sx, float sy, float sz, POINT3D center)
{
    Matrix4x4 m;

    matrix4x4SetIdentity(m);
    m[0][0] = sx;
    m[0][3] = (1-sx)*center.x;
    m[1][1] = sy;
    m[1][3] = (1-sy)*center.y;
    m[2][2] = sz;
    m[2][3] = (1-sz)*center.z;
    matrix4x4PreMultiply(m, theMatrix3D);
}

// original rotation function
template<class NODETYPE>
void DLLList<NODETYPE>::rotate3(POINT3D p1, POINT3D p2, float radianAngle)
{
    float length = sqrt((p2.x-p1.x) * (p2.x-p1.x) + (p2.y-p1.y) * (p2.y-p1.y) + (p2.z-p1.z) * (p2.z-p1.z));

    float cosA2 = cos(radianAngle/2.0);
    float sinA2 = sin(radianAngle/2.0);

    float a = sinA2 * (p2.x-p1.x) / length;
    float b = sinA2 * (p2.y-p1.y) / length;
    float c = sinA2 * (p2.z-p1.z) / length;

    Matrix4x4 m;

    translate3(-p1.x, -p1.y, -p1.z);
    matrix4x4SetIdentity(m);
    m[0][0] = 1.0 - 2*b*b - 2*c*c;
    m[0][1] = 2*a*b - 2*cosA2*c;
    m[0][2] = 2*a*c + 2*cosA2*b;
    m[1][0] = 2*a*b + 2*cosA2*c;
    m[1][1] = 1.0 - 2*a*a - 2*c*c;
    m[1][2] = 2*b*c - 2*cosA2*a;
    m[2][0] = 2*a*c - 2*cosA2*b;
    m[2][1] = 2*b*c + 2*cosA2*a;
    m[2][2] = 1.0 - 2*a*a - 2*b*b;
    matrix4x4PreMultiply(m, theMatrix3D);
    translate3(p1.x, p1.y, p1.z);
}

// original transformation function
template<class NODETYPE>
void DLLList<NODETYPE>::transformPoints3(int nPts, POINT3D *pts)
{
    int k, j;
    float temp[3];

    for(k=0; k<nPts; k++)
    {
        for(j=0; j<3; j++)
            temp[j] = theMatrix3D[j][0] * pts[k].x + theMatrix3D[j][1] * pts[k].y + theMatrix3D[j][2] * pts[k].z + theMatrix3D[j][3];
        setWcPt3(&pts[k], temp[0], temp[1], temp[2]);
    }
}

// New implemented transformation function

```

```

template<class NODETYPE>
void DLLList<NODETYPE>::transformPoints3()
{
    int j;
    float temp[3];

    if(!isEmpty())
    {
        // DLLList is not empty
        DLLListNode<NODETYPE> *currentPtr = firstPtr, *tempPtr; // define pointers
        while(currentPtr != lastPtr)
        {
            tempPtr = currentPtr; // stop at last node
            for(j=0; j<3; j++)
                temp[j] = theMatrix3D[j][0] * tempPtr->data.x + theMatrix3D[j][1] * tempPtr->data.y + theMatrix3D[j][2] * tempPtr->data.z + theMatrix3D[j][3];
            // setWcPt3(&pts[k], temp[0], temp[1], temp[2]);
            tempPtr->data.x = temp[0]; // set x point
            tempPtr->data.y = temp[1]; // set y point
            tempPtr->data.z = temp[2]; // set z point
            currentPtr = currentPtr->nextPtr; // move to next node
        }
        // now take care of the last node
        for(j=0; j<3; j++)
            temp[j] = theMatrix3D[j][0] * currentPtr->data.x + theMatrix3D[j][1] * currentPtr->data.y + theMatrix3D[j][2] * currentPtr->data.z + theMatrix3D[j][3];
        // setWcPt3(&pts[k], temp[0], temp[1], temp[2]);
        currentPtr->data.x = temp[0]; // set x point, last!
        currentPtr->data.y = temp[1]; // set y point, last!
        currentPtr->data.z = temp[2]; // set z point, last!
    } // transformation complete
}
// End of 3D Geometric Transformation Functions ++++++
// =====

// convert degrees to radians
template<class NODETYPE>
float DLLList<NODETYPE>::ToRadians(float a) { return (float)(a*PI/180); }

// convert to normal coordinates
template<class NODETYPE>
void DLLList<NODETYPE>::changeResolution()
{
    if(isEmpty())
    {
        cout<<"The list is empty"<<endl<<endl;
        return;
    }

    DLLListNode<NODETYPE> *currentPtr = firstPtr; // a temporary pointer

    while(currentPtr != lastPtr)
    {
        currentPtr->data.x = currentPtr->data.x * WIDTH;
        currentPtr->data.y = currentPtr->data.y * HEIGHT;
        currentPtr = currentPtr->nextPtr;
    }
    // For last node
    currentPtr->data.x = currentPtr->data.x * WIDTH;
    currentPtr->data.y = currentPtr->data.y * HEIGHT;
}

#endif

// Vahe Karamian - Project III - Dr. Lee - CS 245
// Filename: proj3.cpp
// =====
// Project III - Geometric Transformation

#include <iostream.h>
// #include <graphics.h>
#include <fstream.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>

#include "dllist.h"

```

```

#define RADS 0.017453293

// Prototypes for the program menu system and data handling
int menu(); // Display the menu
void storeData(); // Read data and store it in the list
void SetGraphicsMode(); // Initialize graphics mode
void EndGraphicsMode(); // Terminate graphics Mode

// tessellation pattern drawing
void tessellation(POINT, float, float, float, int);

// The following are 2D objects since POINT is 2D
DLList<POINT> List; // create a GENERAL linked list object

DLList<POINT> Dino; // Dino data file linked list
DLList<POINT> House; // House data file linked list
DLList<POINT> Tree; // Tree data file linked list
DLList<POINT> Car; // Car datafile linked list

// The following are 3D objects since POINT3D is 3D
DLList<POINT3D> Test;// Just for a test

// Global variable
POINT refPt; // 2D reference point
POINT3D refPt3; // 3D reference point
POINT3D P1 = { 0, 10, 10, 0 };
POINT3D P2 = { 0, 10, 10, 10 };

float FanRatio = .22;
float LengthRatio = 0.5;
int NumBranches = 5;

int main()
{
    int choice; // variable for selection

    do
    {
        switch(choice = menu())
        {
            case 0:
                cout<<"Ok, option number 1"<<endl;
                storeData();
                Dino.changeResolution();
                House.changeResolution();
                Tree.changeResolution();
                Car.changeResolution();
                Test.changeResolution();
                break;

            case 1:
                Dino.matrix3x3SetIdentity(); // Done
                House.matrix3x3SetIdentity(); // Done
                Tree.matrix3x3SetIdentity(); // Done
                Car.matrix3x3SetIdentity(); // Done

                // Display each individual object in a Modeling Coordinate System
                // Initialize graphics mode here and display the data
                SetGraphicsMode(); // Initialize graphics mode

                setcolor(15);
                Dino.displayData();
                getch();
                setcolor(0);
                Dino.displayData();

                setcolor(15);
                House.displayData();
                getch();
                setcolor(0);
                House.displayData();

                setcolor(15);
                Tree.displayData();
                getch();
                setcolor(0);
                Tree.displayData();

                setcolor(15);

```

```

Car.displayData();
getch();
setcolor(0);
Car.displayData();

setcolor(15);

// Display all of the object inthe World Coordinate System
// set the (x,y) reference point for rotation and scale
refPt.x = 150;
refPt.y = 400;
Dino.scale2(0.5, 0.5, refPt);
Dino.transformPoints2();

refPt.x = 10;
refPt.y = 100;
House.scale2(0.5, 0.5, refPt);
House.transformPoints2();

refPt.x = 350;
refPt.y = 100;
Tree.scale2(0.5, 0.5, refPt);
Tree.transformPoints2();

refPt.x = 400;
refPt.y = 250;
Car.scale2(0.5, 0.5, refPt);
Car.transformPoints2();

Dino.displayData(); // show data
House.displayData(); // show data
Car.displayData(); // show data
Tree.displayData(); // show data
getch(); // wait for key-press

EndGraphicsMode(); // terminate graphics mode
break;

case 2:
// Display each individual object in a Modeling Coordinate System
// Initialize graphics mode here and display the data
SetGraphicsMode(); // Initialize graphics mode

// Display all of the object inthe World Coordinate System
// set the (x,y) reference point for rotation and scale
refPt.x = 150; // set x-value for refPt
refPt.y = 400; // set y-value for refPt
Dino.scale2(0.5, 0.5, refPt); // scale the object (shrink)
Dino.rotate2(180, refPt); // rotate the object 180 degrees
Dino.transformPoints2(); // transform the points

refPt.x = 10; // set x-value for refPt
refPt.y = 100; // set y-value for refPt
House.scale2(0.5, 0.5, refPt); // scale the house
House.transformPoints2(); // transform the points

refPt.x = 400; // set x-value for refPt
refPt.y = 250; // set y-value for refPt
Car.translate2(10, 10); // translate the object to (x,y)
Car.transformPoints2();

Dino.displayData(); // show data
House.displayData(); // show data
Car.displayData(); // show data
Tree.displayData(); // show data
getch(); // wait for key-press

EndGraphicsMode(); // terminate graphics mode
break;

case 3:
SetGraphicsMode();
refPt.x = 320;
refPt.y = 200;
tesselation(refPt, 90, 100, 50, 10);
getch();
EndGraphicsMode();
break;

```

```

    case 4:
        SetGraphicsMode();
        refPt3.x = 68.0;
        refPt3.y = 30.0;
        refPt3.z = 0.0;
        Test.matrix4x4SetIdentity();
        Test.rotate3(P1, P2, PI/4.0);
        Test.scale3(0.75, 0.75, 1.0, refPt3);
        Test.translate3(25, 40, 0);
        Test.transformPoints3();

        Test.displayData();

    case 9:
        cout<<"\nThank you, please come back soon!\n"<<endl;
        Dino.toFile("dinolist.dat");
        House.toFile("housetlist.dat");
        Tree.toFile("treelist.dat");
        Car.toFile("carlist.dat");
        break;

    default:
        cout<<"\n invalid option--try again!\n";
}
}while(choice != 9);

return 0;
}

void storeData()
{
    int    state;    // state of the coordinate
    float  x, y, z;  // x & y values to read in
    POINT  data;    // 2D data record
    POINT3D data3D; // 3D data record

    ifstream datafile1("house.dat", ios::in);
    ifstream datafile2("dino.dat", ios::in);
    ifstream datafile3("tree.dat", ios::in);
    ifstream datafile4("car.dat", ios::in);
    ifstream datafile5("3d.dat", ios::in);

    if(!datafile1 && !datafile2 && !datafile3 && !datafile4 && !datafile5)
    {
        cout<<"\n Sorry -- one or more data file missing"<<endl;
        return;
    }

    // create the list
    while(datafile1>>state>>x>>y)
    {
        data.s = state;
        data.x = x;
        data.y = y;

        House.insertPoint(data);
    }

    datafile1.close();

    while(datafile2>>state>>x>>y)
    {
        data.s = state;
        data.x = x;
        data.y = y;

        Dino.insertPoint(data);
    }

    datafile2.close();

    while(datafile3>>state>>x>>y)
    {
        data.s = state;
        data.x = x;
        data.y = y;

        Tree.insertPoint(data);
    }
}

```

```

datafile3.close();

while(datafile4>>state>>x>>y)
{
    data.s = state;
    data.x = x;
    data.y = y;

    Car.insertPoint(data);
}

datafile4.close();

while(datafile5>>state>>x>>y>>z)
{
    data3D.s = state;
    data3D.x = x;
    data3D.y = y;
    data3D.z = z;

    Test.insertPoint(data3D);
}

datafile5.close();
cout<<"\nList created."<<endl;
}

// Menu System function
int menu()
{
    int selection;

    cout<<"Main Menu:"<<endl;
    cout<<"======"<<endl;
    cout<<" 0) Create the Linked List from the data file"<<endl;
    cout<<" 1) Display the World Coordinate System"<<endl;
    cout<<" 2) Display the World Coordinate System - After Geometric Transformation"<<endl;
    cout<<" 3) tessellation patterns"<<endl;
    cout<<" 4) Test 3D stuff"<<endl;
    cout<<" 9) Quit this program"<<endl<<endl;
    cout<<"> ";
    cin>>selection;

    return selection;
}

// The following only works with BORLAND C++
// Initialize the graphics mode
void SetGraphicsMode()
{
    int GraphicsBoard = DETECT;
    int GraphicsMode;
    int ErrorCode;

    // read result of initialization
    ErrorCode = graphresult();
    if(ErrorCode != grOk) // an error ocured
    {
        cout<<"Graphics Error: "<<grapherrormsg(ErrorCode)<<endl;
        cout<<"Press any key to halt: "<<endl;
        getch();
        exit(1); // terminate with an error code
    }

    // initialize graphics mode
    initgraph(&GraphicsBoard, &GraphicsMode, "");
}

// Terminate graphics Mode
void EndGraphicsMode() { closegraph(); }

// tessellation drawing pattern
void tessellation(POINT p, float A, float L, float F, int n)
{
    int i, num;
    POINT p2;
    float delang, ang;

```

```

if(n>0)
{
  p2.x = p.x + L * cos(RADS*A);    // find node point
  p2.y = p.y + L * sin(RADS*A);

  line((int)p.x, 480-(int)p.y, (int)p2.x, 480-(int)p2.y);

  num = NumBranches;

  if(num>1)
    delang = F/(num-1.0);
  else
    delang = 0;

  for(i=1; i<=num; i++)
  {
    ang = ang + delang;
    tessellation(p2, ang, L*LengthRatio, F*FanRatio, n-1);
  }
}
}

```