LINKED LISTS

LINKED DATA STRUCTURES

A linked data structure is a dynamically allocated group of nodes that each contain node pointers and are linked together. The manner in which they are linked together defines the kind of data structure. A node is an object that is an instantiation of a C structure that has one or more member variables that are pointers to nodes of the same type. Some of the more common linked data structures are described below.

- Singly Linked List – A data structure that starts from a head node and has nodes connected in a serial fashion to the last node in the list.

- Doubly Linked List – A data structure that has nodes connected in a serial fashion that can go from head to last and also from last to head. That is, the doubly linked list is bi-directional. The doubly linked list uses two pointers to accomplish the bi-directionality.

- Singly Circular Linked List – A data structure that is serially connected in a circular fashion and goes in one direction only, connecting the head node with the last node.

- Doubly Circular Linked list – A data structure that is serially connected in a circular fashion that is bi-directional, connecting the head node with the last node. The doubly circular linked list uses two pointers to accomplish the bi-directionality.

- Binary Tree – A data structure that is arranged like a tree in which nodes branch off to the left and right. The binary tree uses two pointers: a left pointer and a right pointer.

For this class, we will only use the Singly Linked List as it is the most common and most easily-understood data structure.

point.db

```
5 6 4 1
8 8 8 2
0 2 -3 3
-10 10 8 4
```

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

typedef struct point
{
      int x;
      int y;
      int z;
      int pointNum;
      struct point *pNext;
}POINT;


/*-------------------------------------------------------------
 * Function Name: HeadInsert()
 *
 * Description:   This function will insert a POINT node at the
 *                head of a POINT linked list.
 *
 * Inputs:        pHead - Pointer to the head of the linked list
 *                x     - X-Position of the point
 *                y     - Y-Position of the point
 *                z     - Z-Position of the point
 *                num   - Unique tag identifying the point
 *
 * Outputs:       None.
 *
 * Return:        A Pointer to the new head of the linked list.
 *-------------------------------------------------------------*/
POINT *HeadInsert(POINT *pHead, int x, int y, int z, int num)
{
      POINT *pTemp = (POINT *)malloc(sizeof(POINT));

      /* Make sure the memory was allocated */
      if(!pTemp)
      {
            printf("\nERROR: Could not allocate memory...\n");
            exit(-1);
      }

      /* Fill in the node */
      pTemp->x = x;
      pTemp->y = y;
```

```
      pTemp->z = z;
      pTemp->pointNum = num;
      pTemp->pNext = pHead;

      /* Return the new head pointer */
      return pTemp;
}


/*-------------------------------------------------------------
 * Function Name: HeadDelete()
 *
 * Description:   This function will delete a POINT node at the
 *                head of a POINT linked list.
 *
 * Inputs:        pHead - Pointer to the head of the linked list
 *
 * Outputs:       None.
 *
 * Return:        A Pointer to the new head of the linked list.
 *-------------------------------------------------------------*/
POINT *HeadDelete(POINT *pHead)
{
      POINT *pDiscard = pHead; /* Save the head pointer */

      /* Make sure the node exists */
      if(!pHead)
            return pHead;

      /* Move the head pointer to the next node */
      pHead = pHead->pNext;

      /* Delete the head node */
      free(pDiscard);

      /* Return the new head pointer */
      return pHead;
}
```

```
/*-------------------------------------------------------------
 * Function Name: InsertNode()
 *
 * Description:   This function will insert a POINT node anywhere
 *                in a POINT linked list other than the head.
 *
 * Inputs:        pPutAfterMe - Pointer to the node before
 *                the new node to add.
 *                x     - X-Position of the point
 *                y     - Y-Position of the point
 *                z     - Z-Position of the point
 *                num   - Unique tag identifying the point
 *
 * Outputs:       None.
 *
 * Return:        None.
 *-------------------------------------------------------------*/
void InsertNode(POINT *pPutAfterMe, int x, int y, int z, int num)
{
      POINT *pTemp = (POINT *)malloc(sizeof(POINT));

      /* Make sure the memory was allocated */
      if(!pTemp)
      {
            printf("\nERROR: Could not allocate memory...\n");
            exit(-1);
      }

      /* Fill in the node */
      pTemp->x = x;
      pTemp->y = y;
      pTemp->z = z;
      pTemp->pointNum = num;

      /* Connect the temp node tail to the linked list */
      pTemp->pNext = pPutAfterMe->pNext;

      /* Connect the current node to the temp node */
      pPutAfterMe->pNext = pTemp;
}
```

```
/*-------------------------------------------------------------
 * Function Name: DiscardNode()
 *
 * Description:   This function will delete a POINT node anywhere
 *                in a POINT linked list.
 *
 * Inputs:        pDiscard - Pointer to the node to delete
 *                pHead    - Pointer to the head of the list
 *
 * Outputs:       None.
 *
 * Return:        A Pointer to the new head of the linked list.
 *-------------------------------------------------------------*/
POINT *DiscardNode(POINT *pDiscard, POINT * pHead)
{
      POINT *pBeforeNode = NULL;

      /* If the node is the head node, delete it */
      if(pDiscard == pHead)
      {
            pHead = HeadDelete(pHead);
            return pHead;
      }

      /*
       * Start at the beginning of the list and search until
       * there is a match with the desired node to discard.
       */
      pBeforeNode = pHead;

      while((pBeforeNode->pNext != pDiscard) &&
            (pBeforeNode->pNext != NULL))
      {
            pBeforeNode = pBeforeNode->pNext;
      }

      /* If there was no match, the pNext pointer will be NULL */
      if(pBeforeNode->pNext == NULL)
      {
            printf("Cannot find node to discard\n");
            return pHead;
      }

      /*
       * Set the pBeforeNode pNext pointer to point to the
       * node after the pDiscard node.
       */
      pBeforeNode->pNext = pDiscard->pNext;

      /* Delete the node */
      free(pDiscard);

      return pHead;
}
```

```
/*-------------------------------------------------------------
 * Function Name: SearchList()
 *
 * Description:   This function will search through a POINT
 *                linked list until there is a match with a
 *                tag identifying the point or until the list
 *                is completely searched without a match.
 *
 * Inputs:        pFound - Pointer to the head of the list
 *
 * Outputs:       None.
 *
 * Return:        pFound - Pointer to the found node (or NULL)
 *-------------------------------------------------------------*/
POINT *SearchList(POINT *pFound, int num)
{
      /* Check for the empty list case */
      if(pFound == NULL)
      {
            printf("The list is empty.\n");
            return NULL;
      }

      /* Find the point num in the list */
      while((pFound->pointNum != num) && (pFound->pNext != NULL))
            pFound = pFound->pNext;

      if(pFound->pointNum == num)
            return pFound;
      else
      {
            printf("Cannot find the point in the list.\n");
            return NULL;
      }

      return NULL;
}
```

```
/*-------------------------------------------------------------
 * Function Name: DisplayPoint()
 *
 * Description:   This function will display one point node in a
 *                POINT linked list
 *
 * Inputs:        pCurrent - Pointer to the node to display
 *
 * Outputs:       None.
 *
 * Return:        None.
 *-------------------------------------------------------------*/
void DisplayPoint(POINT *pCurrent)
{
      /* Check for the empty list case */
      if(pCurrent == NULL)
      {
            printf("The list is empty.\n");
            return;
      }

      printf(“Point #%03d: [%d, %d, %d]\n”, pCurrent->pointNum,
                  pCurrent->x, pCurrent->y, pCurrent->z);
}
```

```
/*-------------------------------------------------------------
 * Function Name: DisplayList()
 *
 * Description:   This function will display all point nodes in
 *                a POINT linked list.
 *
 * Inputs:        pCurrent - Pointer to the head of the list
 *
 * Outputs:       None.
 *
 * Return:        None.
 *-------------------------------------------------------------*/
void DisplayList(POINT *pCurrent)
{
      /* Check for the empty list case */
      if(pCurrent == NULL)
      {
            printf("The list is empty.\n");
            return;
      }

      do
      {
            DisplayPoint(pCurrent);

            pCurrent = pCurrent->pNext;
      } while (pCurrent!= NULL);
}
```

```
/*---------------------------------------------------------------
 * Function Name: DeleteList ()
 *
 * Description:   This function delete a POINT linked list
 *
 * Inputs:        pHead - Pointer to the head of the list
 *
 * Outputs:       None.
 *
 * Return:        A Pointer to the new head of the list (NULL).
 *---------------------------------------------------------------*/
POINT *DeleteList(POINT *pHead)
{
      POINT *pCurrent = NULL;

      /* Check for the empty list case */
      if(pHead == NULL)
      {
            printf("The list is empty.\n");
            return pHead;
      }

      while(pHead != NULL)
      {
            pCurrent = pHead->pNext;

            /* Delete the head node */
            free(pHead);

            /* Set the new head node */
            pHead = pCurrent;
      }

      return pHead;
}
```