```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node * link ;
} ;

void append ( struct node **, int ) ;
void addatbeg ( struct node **, int ) ;
void addafter ( struct node *, int, int ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
void del ( struct node **, int ) ;

int main( )
{
    struct node *p ;
    p = NULL ;  /* empty linked list */

    printf ( "No. of elements in the Linked List = %d\n", count ( p ) ) ;
    append ( &p, 14 ) ;
    append ( &p, 30 ) ;
```

```
    append ( &p, 25 ) ;
    append ( &p, 42 ) ;
    append ( &p, 17 ) ;

    system ( "cls" ) ;

    display ( p ) ;

    addatbeg ( &p, 999 ) ;
    addatbeg ( &p, 888 ) ;
    addatbeg ( &p, 777 ) ;

    display ( p ) ;

    addafter ( p, 7, 0 ) ;
    addafter ( p, 2, 1 ) ;
    addafter ( p, 5, 99 ) ;

    display ( p ) ;
    printf ( "No. of elements in the Linked List = %d\n", count ( p ) ) ;

    del ( &p, 99 ) ;
    del ( &p, 1 ) ;
    del ( &p, 10 ) ;

    display ( p ) ;
    printf ( "No. of elements in the linked list = %d\n", count ( p ) ) ;
    return 0 ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
    struct node *temp, *r ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
```

```
        temp -> data = num ;
        temp -> link = NULL ;
        *q = temp ;
    }
    else
    {
        temp = *q ;

        /* go to last node */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        r -> data = num ;
        r -> link = NULL ;
        temp -> link = r ;
    }
}

/* adds a new node at the beginning of the linked list */
void addatbeg ( struct node **q, int num )
{
    struct node *temp ;

    /* add new node */
    temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;

    temp -> data = num ;
    temp -> link = *q ;
    *q = temp ;
}

/* adds a new node after the specified number of nodes */
void addafter ( struct node *q, int loc, int num )
{
    struct node *temp, *r ;
    int i ;
```

```
    temp = q ;
    /* skip to desired portion */
    for ( i = 0 ; i < loc ; i++ )
    {
        temp = temp -> link ;

        /* if end of linked list is encountered */
        if ( temp == NULL )
        {
            printf ( "There are less than %d elements in list\n", loc ) ;
            return ;
        }
    }

    /* insert new node */
    r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    r -> data = num ;
    r -> link = temp -> link ;
    temp -> link = r ;
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
int count ( struct node * q )
{
    int c = 0 ;
```

```
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }

    return c ;
}

/* deletes the specified node from the linked list */
void del ( struct node **q, int num )
{
    struct node *old, *temp ;

    temp = *q ;

    while ( temp != NULL )
    {
        if ( temp -> data == num )
        {
            /* if node to be deleted is the first node in the linked list */
            if ( temp == *q )
                *q = temp -> link ;

            /* deletes the intermediate nodes in the linked list */
            else
                old -> link = temp -> link ;

            /* free the memory occupied by the node */
            free ( temp ) ;
            return ;
        }

        /* traverse the linked list till the last node is reached */
        else
        {
```

```
            old = temp ;  /* old points to the previous node */
            temp = temp -> link ;  /* go to the next node */
        }
    }

    printf ( "Element %d not found\n", num ) ;
}
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
```

```
    struct node *link ;
};

void add ( struct node **, int ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;

int main( )
{
    struct node *p ;
    p = NULL ;  /* empty linked list */

    add ( &p, 5 ) ;
    add ( &p, 1 ) ;
    add ( &p, 6 ) ;
    add ( &p, 4 ) ;
    add ( &p, 7 ) ;

    display ( p ) ;
    printf ( "No. of elements in Linked List = %d\n", count ( p ) ) ;
    return 0 ;
}

/* adds node to an ascending order linked list */
void add ( struct node **q, int num )
{
    struct node *r, *temp = *q ;

    r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    r -> data = num ;

    /* if list is empty or if new node is to be inserted before the first node */
    if ( *q == NULL || ( *q ) -> data > num )
    {
        *q = r ;
        ( *q ) -> link = temp ;
    }
    else
```

```
    {
        /* traverse the entire linked list to search the position to insert the
            new node */
        while ( temp != NULL )
        {
            if ( temp -> data <= num && (temp -> link == NULL ||
                    temp -> link -> data > num ) )
            {
                r -> link = temp -> link ;
                temp -> link = r ;
                return ;
            }
            temp = temp -> link ;  /* go to the next node */
        }
    }
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
int count ( struct node *q )
{
    int c = 0 ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> link ;
```

```
            c++ ;
        }

        return c ;
}
```

Output:

```
1 4 5 6 7
No. of elements in Linked List = 5
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node *link ;
} ;

void addatbeg ( struct node **, int ) ;
void reverse ( struct node ** ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;

int main( )
{
    struct node *p ;
    p = NULL ;  /* empty linked list */

    addatbeg ( &p, 7 ) ;
    addatbeg ( &p, 43 ) ;
    addatbeg ( &p, 17 ) ;
    addatbeg ( &p, 3 ) ;
    addatbeg ( &p, 23 ) ;
    addatbeg ( &p, 5 ) ;

    system ( "cls" ) ;

    display ( p ) ;
    printf ( "No. of elements in the linked list = %d\n", count ( p ) ) ;
```

```
    reverse ( &p ) ;
    display ( p ) ;
    printf ( "No. of elements in the linked list = %d\n", count ( p ) ) ;
    return 0 ;
}

/* adds a new node at the beginning of the linked list */
void addatbeg ( struct node **q, int num )
{
    struct node *temp ;

    /* add new node */
    temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    temp -> data = num ;
    temp -> link = *q ;
    *q = temp ;
}

void reverse ( struct node **x )
{
    struct node *q, *r, *s ;

    q = *x ;
    r = NULL ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        s = r ;
        r = q ;
        q = q -> link ;
        r -> link = s ;
    }

    *x = r ;
}

/* displays the contents of the linked list */
```

```
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
int count ( struct node * q )
{
    int c = 0 ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }

    return c ;
}
```

Merging Of Linked Lists

Suppose we have two linked lists pointed to by two independent pointers and we wish to merge the two lists into a third list. While carrying out this merging we wish to ensure that those elements which are common to both the lists occur only once in the third list. The program to achieve this is given below. It is assumed that within a list all elements are unique.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node *link ;
};

void add ( struct node **; int ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
void merge ( struct node *, struct node *, struct node ** ) ;

int main( )
{
    struct node *first, *second, *third ;
    first = second = third = NULL ;  /* empty linked lists */

    add ( &first, 9 ) ;
    add ( &first, 12 ) ;
    add ( &first, 14 ) ;
    add ( &first, 17 ) ;
    add ( &first, 35 ) ;
    add ( &first, 61 ) ;
```

```
    add ( &first, 79 ) ;

    system ( "cls" ) ;

    printf ( "First linked list:\n" ) ;
    display ( first ) ;
    printf ( "No. of elements in Linked List: %d\n\n", count ( first ) ) ;

    add ( &second, 12 ) ;
    add ( &second, 17 ) ;
    add ( &second, 24 ) ;
    add ( &second, 36 ) ;
    add ( &second, 59 ) ;
    add ( &second, 64 ) ;
    add ( &second, 87 ) ;

    printf ( "Second linked list:\n" ) ;
    display ( second ) ;
    printf ( "No. of elements in Linked List: %d\n\n", count ( second ) ) ;

    merge ( first, second, &third ) ;

    printf ( "The merged list:\n" ) ;
    display ( third ) ;
    printf ( "No. of elements in Linked List: %d\n", count ( third ) ) ;
    return 0 ;
}

/* adds node to an ascending order linked list */
void add ( struct node **q, int num )
{
    struct node *r, *temp = *q ;

    r = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    r -> data = num ;

    /* if list is empty or if new node is to be inserted before the first node */
    if ( *q == NULL || ( *q ) -> data > num )
```

```
    {
        *q = r ;
        ( *q ) -> link = temp ;
    }
    else
    {
        /* traverse the entire linked list to search the position to insert the
           new node */
        while ( temp != NULL )
        {
            if ( temp -> data < num && ( temp -> link == NULL ||
                                         temp -> link -> data > num ))
            {
                r -> link = temp -> link ;
                temp -> link = r ;
                return ;
            }
            temp = temp -> link ;  /*go to next node */
        }

        r -> link = NULL ;
        temp -> link = r ;
    }
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
```

```
int count ( struct node * q )
{
    int c = 0 ;
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q.-> link ;
        c++ ;
    }

    return c ;
}

/* merges the two linked lists, restricting the common elements to occur only
   once in the final list */
void merge ( struct node *p, struct node *q, struct node **s )
{
    struct node *z ;

    z = NULL ;

    /* if both lists are empty */
    if ( p == NULL && q == NULL )
        return ;

    /* traverse both linked lists till the end. If end of any one list is reached
       loop is terminated */
    while ( p != NULL && q != NULL )
    {
        /* if node being added in the first node */
        if ( *s == NULL )
        {
            *s = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
            z = *s ;
        }
        else
        {
            z -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
```

```
            z = z -> link ;
        }

        if ( p -> data < q -> data )
        {
            z -> data = p -> data ;
            p = p -> link ;
        }
        else
        {
            if ( q -> data < p -> data )
            {
                z -> data = q -> data ;
                q = q -> link ;
            }
            else
            {
                if ( p -> data == q -> data )
                {
                    z -> data = q -> data ;
                    p = p -> link ;
                    q = q -> link ;
                }
            }
        }
    }

    /* if end of first list has not been reached */
    while ( p != NULL )
    {
        z -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        z = z -> link ;
        z -> data = p -> data ;
        p = p -> link ;
    }

    /* if end of second list has not been reached */
    while ( q != NULL )
```

```
    {
        z -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        z = z -> link ;
        z -> data = q -> data ;
        q = q -> link ;
    }

    z -> link = NULL ;
}
```

```
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node *link ;
} *newnode, *start, *visit ;

void getdata( ) ;
void append ( struct node **, int ) ;
void displaylist( ) ;
int count ( struct node * ) ;
void selection_sort ( int ) ;
void bubble_sort ( int ) ;

int main( )
{
    int n ;

    getdata( ) ;

    printf ( "Linked list Before Sorting:\n" ) ;
    displaylist( ) ;

    n = count ( start ) ;

    selection_sort ( n ) ;
    printf ( "Linked list After Selection Sorting:\n" ) ;
    displaylist( ) ;

    getdata( ) ;
    printf ( "Linked list Before Sorting:\n" ) ;
    displaylist( ) ;

    n = count ( start ) ;
```

```
    bubble_sort ( n ) ;
    printf ( "Linked list After Bubble Sorting:\n" ) ;
    displaylist( ) ;
    return 0 ;
}

void getdata( )
{
    int val ;
    char ch ;
    struct node *new ;

    system ( "cls" ) ;

    new = NULL ;
    do
    {
        printf ( "Enter a value: " ) ;
        scanf ( "%d", &val ) ;

        append ( &new, val ) ;

        printf ( "Any More Nodes (Y/N): " ) ;
        fflush ( stdin ) ;
        ch = getchar( ) ;
        printf ( "\n" ) ;
    } while ( ch == 'y' || ch == 'Y' ) ;

    start = new ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL ) /* if the list is empty, create first node */
```

```
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = temp -> link ;
    }

    /* assign data to the last node */
    temp -> data = num ;
    temp -> link = NULL ;
}

/* displays the contents of the linked list */
void displaylist( )
{
    visit = start ;

    /* traverse the entire linked list */
    while ( visit != NULL )
    {
        printf ( "%d ", visit -> data ) ;
        visit = visit -> link ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
int count ( struct node * q )
{
    int c = 0 ;
```

```
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }

    return c ;
}

void selection_sort ( int n )
{
    int i, j, temp ;
    struct node *p, *q ;

    p = start ;
    for ( i = 0 ; i < n - 1 ; i++ )
    {
        q = p -> link ;

        for ( j = i + 1 ; j < n ; j++ )
        {
            if ( p -> data > q -> data )
            {
                temp = p -> data ;
                p -> data = q -> data ;
                q -> data = temp ;
            }
            q = q -> link ;
        }
        p = p -> link ;
    }
}

void bubble_sort ( int n )
{
    int i, j, k, temp ;
```

```
    struct node *p, *q ;

    k = n ;
    for ( i = 0 ; i < n - 1 ; i++, k-- )
    {
        p = start ;
        q = p -> link ;

        for ( j = 1 ; j < k ; j++ )
        {
            if ( p -> data > q -> data )
            {
                temp = p -> data ;
                p -> data = q -> data ;
                q -> data = temp ;
            }
            p = p -> link ;
            q = q -> link ;
        }
    }
}
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>

/* structure containing a data part and link part */
struct node
{
      int data ;
      struct node *link ;
} *start, *visit ;

void getdata( ) ;
void append ( struct node **q, int num ) ;
void displaylist( ) ;
void selection_sort( ) ;
void bubble_sort( ) ;

int main( )
{
      getdata( ) ;
      printf ( "Linked List Before Sorting:\n" ) ;
      displaylist( ) ;

      selection_sort( ) ;
      printf ( "Linked List After Selection Sorting:\n" ) ;
      displaylist( ) ;

      getdata( ) ;
      printf ( "Linked List Before Sorting:\n" ) ;
      displaylist( ) ;

      bubble_sort( ) ;
      printf ( "Linked List After Bubble Sorting:\n" ) ;
      displaylist( ) ;
      return 0 ;
}
```

```
void getdata( )
{
    int val ;
    char ch ;
    struct node *newnode;

    newnode = NULL ;
    do
    {
        printf ( "Enter a value: " ) ;
        scanf ( "%d", &val ) ;
        append ( &newnode, val ) ;

        printf ( "Any More Nodes (Y/N): " ) ;
        fflush ( stdin ) ;
        ch = getchar( ) ;
        printf ( "\n" ) ;
    } while ( ch == 'y' || ch == 'Y' ) ;

    start = newnode ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
        while ( temp -> link != NULL )
        temp = temp -> link ;
```

```
            /* add node at the end */
            temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
            temp = temp -> link ;
        }

        /* assign data to the last node */
        temp -> data = num ;
        temp -> link = NULL ;
}

/* displays the contents of the linked list */
void displaylist( )
{
        visit = start ;

        /* traverse the entire linked list */
        while ( visit != NULL )
        {
            printf ( "%d ", visit -> data ) ;
            visit = visit -> link ;
        }
        printf ( "\n" ) ;
}

void selection_sort( )
{
        struct node *p, *q, *r, *s, *temp ;

        p = r = start ;
        while ( p -> link != NULL )
        {
            s = q = p -> link ;
            while ( q != NULL )
            {
                if ( p -> data > q -> data )
                {
                    if ( p -> link == q )  /* Adjacent Nodes */
```

```
{
    if ( p == start )
    {
        p -> link = q -> link ;
        q -> link = p ;

        temp = p ;
        p = q ;
        q = temp ;

        start = p ;
        r = p ;
        s = q ;
        q = q -> link ;
    }
    else
    {
        p -> link = q -> link ;
        q -> link = p ;
        r -> link = q ;

        temp = p ;
        p = q ;
        q = temp ;

        s = q ;
        q = q -> link ;
    }
}
else
{
    if ( p == start )
    {
        temp = q -> link ;
        q -> link = p -> link ;
        p -> link = temp ;

        s -> link = p ;
```

```
                    temp = p ;
                    p = q ;
                    q = temp ;

                    s = q ;
                    q = q -> link ;
                    start = p ;
                }
                else
                {
                    temp = q -> link ;
                    q -> link = p -> link ;
                    p -> link = temp ;
                    r -> link = q ;
                    s -> link = p ;

                    temp = p ;
                    p = q ;
                    q = temp ;

                    s = q ;
                    q = q -> link ;
                }
            }
        }
        else
        {
            s = q ;
            q = q -> link ;
        }
    }
    r = p ;
    p = p -> link ;
  }
}

void bubble_sort( )
```

```
{
    struct node *p, *q, *r, *s, *temp ;
    s = NULL ;

    /* r precedes p and s points to the node up to which comparisons are to
       be made */
    while ( s != start -> link )
    {
        r = p = start ;
        q = p -> link ;

        while ( p != s )
        {
            if ( p -> data > q -> data )
            {
                if ( p == start )
                {
                    temp = q -> link ;
                    q -> link = p ;
                    p -> link = temp ;

                    start = q ;
                    r = q ;
                }
                else
                {
                    temp = q -> link ;
                    q -> link = p ;
                    p -> link = temp ;

                    r -> link = q ;
                    r = q ;
                }
            }
            else
            {
                r = p ;
                p = p -> link ;
```

```
            }
            q = p -> link ;
            if ( q == s )
                s = p ;
        }
    }
}
```

```
/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node * link ;
};

void addcirq ( struct node **, struct node **, int ) ;
int delcirq ( struct node **, struct node ** ) ;
void cirq_display ( struct node * ) ;

int main( )
{
    struct node *front, *rear ;

    front = rear = NULL ;

    addcirq ( &front, &rear, 10 ) ;
    addcirq ( &front, &rear, 17 ) ;
    addcirq ( &front, &rear, 18 ) ;
    addcirq ( &front, &rear, 5 ) ;
    addcirq ( &front, &rear, 30 ) ;
    addcirq ( &front, &rear, 15 ) ;

    system ( "cls" ) ;

    printf ( "Before deletion:\n" ) ;
    cirq_display ( front ) ;

    delcirq ( &front, &rear ) ;
    delcirq ( &front, &rear ) ;
    delcirq ( &front, &rear ) ;

    printf ( "After deletion:\n" ) ;
    cirq_display ( front ) ;
    return 0 ;
}
```

```
/* adds a new element at the end of queue */
void addcirq ( struct node **f, struct node **r, int item )
{
    struct node *q ;

    /* create new node */
    q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
    q -> data = item ;

    /* if the queue is empty */
    if ( *f == NULL )
        *f = q ;
    else
        ( *r ) -> link = q ;

    *r = q ;
    ( *r ) -> link = *f ;
}

/* removes an element from front of queue */
int delcirq ( struct node **f, struct node **r )
{
    struct node *q ;
    int item ;

    /* if queue is empty */
    if ( *f == NULL )
        printf ( "queue is empty\n" ) ;
    else
    {
        if ( *f == *r )
        {
            item = ( *f ) -> data ;
            free ( *f ) ;
            *f = NULL ;
            *r = NULL ;
        }
        else
```

```
        {
            /* delete the node */
            q = *f ;
            item = q -> data ;
            *f = ( *f ) -> link ;
            ( *r ) -> link = *f ;
            free ( q ) ;
        }
        return ( item ) ;
    }
    return NULL ;
}

/* displays whole of the queue */
void cirq_display ( struct node *f )
{
    struct node *q = f, *p = NULL ;

    /* traverse the entire linked list */
    while ( q != p )
    {
        printf ( "%d ", q -> data ) ;

        q = q -> link ;
        p = f ;
    }
    printf ( "\n" ) ;
}
```

Output:

```
Before deletion:
10    17    18    5    30    15
After deletion:
5    30    15
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

struct node
{
    int data ;
    struct node *link ;
};

void append ( struct node **, int ) ;
void concat ( struct node **, struct node ** ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
struct node * erase ( struct node * ) ;

int main( )
{
    struct node *first, *second ;

    first = second = NULL ;  /* empty linked lists */
```

```
    append ( &first, 1 ) ;
    append ( &first, 2 ) ;
    append ( &first, 3 ) ;
    append ( &first, 4 ) ;

    system ( "cls" ) ;
    printf ( "First List:\n" ) ;
    display ( first ) ;
    printf ( "No. of elements in the first Linked List = %d\n", count ( first ) ) ;

    append ( &second, 5 ) ;
    append ( &second, 6 ) ;
    append ( &second, 7 ) ;
    append ( &second, 8 ) ;

    printf ( "Second List:\n" ) ;
    display ( second ) ;
    printf ( "No. of elements in the second Linked List = %d\n",
              count ( second ) ) ;

    /* the result obtained after concatenation is in the first list */
    concat ( &first, &second ) ;

    printf ( "Concatenated List:\n" ) ;
    display ( first ) ;

    printf ( "No. of elements in Linked List before erasing = %d\n",
              count ( first ) ) ;

    first = erase ( first ) ;
    printf ( "No. of elements in Linked List after erasing = %d\n",
              count ( first ) ) ;
    return 0 ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
```

```
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = temp -> link ;
    }

    /* assign data to the last node */
    temp -> data = num ;
    temp -> link = NULL ;
}

/* concatenates two linked lists */
void concat ( struct node **p, struct node **q )
{
    struct node *temp ;

    /* if the first linked list is empty */
    if ( *p == NULL )
        *p = *q ;
    else
    {
        /* if both linked lists are non-empty */
        if ( *q != NULL )
        {
```

```
            temp = *p ;  /* points to the starting of the first list */

            /* traverse the entire first linked list */
            while ( temp -> link != NULL )
                temp = temp -> link ;

            temp -> link = *q ;  /* concatenate the second list after the
                                    first */
        }
    }
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n\n" ) ;
}

/* counts the number of nodes present in the linked list */
int count ( struct node *q )
{
    int c = 0 ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> link ;
        c++ ;
    }

    return c ;
}
```

```
/* erases all the nodes from a linked list */
struct node * erase ( struct node *q )
{
    struct node *temp ;

    /* traverse till the end erasing each node */
    while ( q != NULL )
    {
        temp = q ;
        q = q -> link ;
        free ( temp ) ;  /* free the memory occupied by the node */
    }

    return NULL ;
}
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
      int data ;
      struct node *link ;
} ;

void append ( struct node **, int ) ;
int length ( struct node * ) ;

int main( )
{
```

```
    struct node *p ;
    p = NULL ;  /* empty linked list */

    append ( &p, 1 ) ;
    append ( &p, 2 ) ;
    append ( &p, 3 ) ;
    append ( &p, 4 ) ;
    append ( &p, 5 ) ;

    system ( "cls" ) ;

    printf ( "Length of linked list = %d\n", length ( p ) ) ;
    return 0 ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = temp -> link ;
    }

    /* assign data to the last node */
```

```
    temp -> data = num ;
    temp -> link = NULL ;
}

/* counts the number of nodes in a linked list */
int length ( struct node *q )
{
    static int l ;

    /* if list is empty or if NULL is encountered */
    if ( q == NULL )
        return ( 0 ) ;
    else
    {
        /* go to next node */
        l = 1 + length ( q -> link ) ;
        return ( l ) ;
    }
}
```

Output:

```
Length of linked list = 5
```

(b) Program to compare two linked lists using recursion

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

struct node
{
    int data ;
    struct node *link ;
} ;

void append ( struct node **, int ) ;
```

```
int compare ( struct node *, struct node * ) ;

int main( )
{
    struct node *first, *second ;
    first = second = NULL ;  /* empty linked lists */

    append ( &first, 1 ) ;
    append ( &first, 2 ) ;
    append ( &first, 3 ) ;

    append ( &second, 1 ) ;
    append ( &second, 2 ) ;
    append ( &second, 3 ) ;

    system ( "cls" ) ;

    if ( compare ( first, second ) )
        printf ( "Both linked lists are EQUAL\n" ) ;
    else
        printf ( "Linked lists are DIFFERENT\n" ) ;
    return 0 ;
}

/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
```

```
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = temp -> link ;
    }

    /* assign data to the last node */
    temp -> data = num ;
    temp -> link = NULL ;
}

/* compares 2 linked lists and returns 1 if linked lists are equal and 0 if
    unequal */
int compare ( struct node *q, struct node *r )
{
    static int flag ;

    if ( ( q == NULL ) && ( r == NULL ) )
        flag = 1 ;
    else
    {
        if ( q == NULL || r == NULL )
            flag = 0 ;

        if ( q -> data != r -> data )
            flag = 0 ;
        else
            compare ( q -> link, r -> link ) ;
    }
    return ( flag ) ;
}
```

(c) Program to copy one linked list into another using recursion.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure containing a data part and link part */
struct node
{
    int data ;
    struct node *link ;
};

void append ( struct node **, int ) ;
void copy ( struct node *, struct node ** ) ;
void display ( struct node * ) ;

int main( )
{
    struct node *first, *second ;
    first = second = NULL ;  /* empty linked lists */

    append ( &first, 1 ) ;
    append ( &first, 2 ) ;
    append ( &first, 3 ) ;
    append ( &first, 4 ) ;
    append ( &first, 5 ) ;
    append ( &first, 6 ) ;
    append ( &first, 7 ) ;

    system ( "cls" ) ;

    display ( first ) ;

    copy ( first, &second ) ;
```

```
    display ( second ) ;
    return 0 ;
}

/* adds a node at the end of the linked list */
void append ( struct node **q, int num )
{
    struct node *temp ;
    temp = *q ;

    if ( *q == NULL )  /* if the list is empty, create first node */
    {
        *q = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = *q ;
    }
    else
    {
        /* go to last node */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* add node at the end */
        temp -> link = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        temp = temp -> link ;
    }

    /* assign data to the last node */
    temp -> data = num ;
    temp -> link = NULL ;
}

/* copies a linked list into another */
void copy ( struct node *q, struct node **s )
{
    if ( q != NULL )
    {
        *s = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
```

```
        ( *s ) -> data = q -> data ;
        ( *s ) -> link = NULL ;

        copy ( q -> link, &( ( *s ) -> link ) ) ;
    }
}

/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}
```

Output:

```
1 2 3 4 5 6 7
1 2 3 4 5 6 7
```

(d) Program to add a new node at the end of linked list using recursion.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

struct node
{
    int data ;
    struct node *link ;
```

```
};

void addatend ( struct node **, int ) ;
void display ( struct node * ) ;

int main( )
{
    struct node *p ;

    p = NULL ;

    addatend ( &p, 1 ) ;
    addatend ( &p, 2 ) ;
    addatend ( &p, 3 ) ;
    addatend ( &p, 4 ) ;
    addatend ( &p, 5 ) ;
    addatend ( &p, 6 ) ;
    addatend ( &p, 10 ) ;

    system ( "cls" ) ;

    display ( p ) ;
    return 0 ;
}

/* adds a new node at the end of the linked list */
void addatend ( struct node **s, int num )
{
    if ( *s == NULL )
    {
        *s = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
        ( *s ) -> data = num ;
        ( *s ) -> link = NULL ;
    }
    else
        addatend ( &( ( *s ) -> link ), num ) ;
}
```

```
/* displays the contents of the linked list */
void display ( struct node *q )
{
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
    }
    printf ( "\n" ) ;
}
```

Output:

```
1 2 3 4 5 6 10
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure representing a node of the doubly linked list */
struct dnode
{
      struct dnode *prev ;
      int data ;
      struct dnode * next ;
} ;

void d_append ( struct dnode **, int ) ;
void d_addatbeg ( struct dnode **, int ) ;
void d_addafter ( struct dnode *, int , int ) ;
void d_display ( struct dnode * ) ;
int d_count ( struct dnode *  ) ;
void d_delete ( struct dnode **, int ) ;

int main( )
{
```

```
    struct dnode *p ;

    p = NULL ;  /* empty doubly linked list */

    d_append ( &p , 11 ) ;
    d_append ( &p , 2 ) ;
    d_append ( &p , 14 ) ;
    d_append ( &p , 17 ) ;
    d_append ( &p , 99 ) ;

    system ( "cls" ) ;

    d_display ( p ) ;
    printf ( "No. of elements in the DLL = %d\n\n", d_count ( p ) ) ;

    d_addatbeg ( &p, 33 ) ;
    d_addatbeg ( &p, 55 ) ;

    d_display ( p ) ;
    printf ( "No. of elements in the DLL = %d\n\n", d_count ( p ) ) ;

    d_addafter ( p, 4, 66 ) ;
    d_addafter ( p, 2, 96 ) ;

    d_display ( p ) ;
    printf ( "No. of elements in the DLL = %d\n\n", d_count ( p ) ) ;

    d_delete ( &p, 55 ) ;
    d_delete ( &p, 2 ) ;
    d_delete ( &p, 99 ) ;

    d_display ( p ) ;
    printf ( "No. of elements in the DLL = %d\n\n", d_count ( p ) ) ;
    return 0 ;
}

/* adds a new node at the end of the doubly linked list */
void d_append ( struct dnode **s, int num )
```

```
{
    struct dnode *r, *q = *s ;

    /* if the linked list is empty */
    if ( *s == NULL )
    {
        /*create a new node */
        *s = ( struct dnode * ) malloc ( sizeof ( struct dnode ) ) ;
        ( *s ) -> prev = NULL ;
        ( *s ) -> data = num ;
        ( *s ) -> next = NULL ;
    }
    else
    {
        /* traverse the linked list till the last node is reached */
        while ( q -> next != NULL )
            q = q -> next ;

        /* add a new node at the end */
        r = ( struct dnode * ) malloc ( sizeof ( struct dnode ) ) ;
        r -> data = num ;
        r -> next = NULL ;
        r -> prev = q ;
        q -> next = r ;
    }
}

/* adds a new node at the begining of the linked list */
void d_addatbeg ( struct dnode **s, int num )
{
    struct dnode *q ;

    /* create a new node */
    q = ( struct dnode * ) malloc ( sizeof ( struct dnode ) ) ;

    /* assign data and pointer to the new node */
    q -> prev = NULL ;
    q -> data = num ;
```

```
    q -> next = *s ;

    /* make new node the head node */
    ( *s ) -> prev = q ;
    *s = q ;
}

/* adds a new node after the specified number of nodes */
void d_addafter ( struct dnode *q, int loc, int num )
{
    struct dnode *temp ;
    int i ;

    /* skip to desired portion */
    for ( i = 0 ; i < loc ; i++ )
    {
        q = q -> next ;
        /* if end of linked list is encountered */
        if ( q == NULL )
        {
            printf ( "There are less than %d elements\n", loc );
            return ;
        }
    }

    /* insert new node */
    q = q -> prev ;
    temp = ( struct dnode * ) malloc ( sizeof ( struct dnode ) ) ;
    temp -> data = num ;
    temp -> prev = q ;
    temp -> next = q -> next ;
    temp -> next -> prev = temp ;
    q -> next = temp ;
}

/* displays the contents of the linked list */
void d_display ( struct dnode *q )
{
```

```
    /* traverse the entire linked list */
    while ( q != NULL )
    {
        printf ( "%2d\t", q -> data ) ;
        q = q -> next ;
    }
    printf ( "\n" ) ;
}

/* counts the number of nodes present in the linked list */
int d_count ( struct dnode * q )
{
    int c = 0 ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        q = q -> next ;
        c++ ;
    }

    return c ;
}

/* deletes the specified node from the doubly linked list */
void d_delete ( struct dnode **s, int num )
{
    struct dnode *q = *s ;

    /* traverse the entire linked list */
    while ( q != NULL )
    {
        /* if node to be deleted is found */
        if ( q -> data == num )
        {
            /* if node to be deleted is the first node */
            if ( q == *s )
            {
```

```
            *s = ( *s ) -> next ;
            ( *s ) -> prev = NULL ;
        }
        else
        {
            /* if node to be deleted is the last node */
            if ( q -> next == NULL )
                q -> prev -> next = NULL ;
            else
            /* if node to be deleted is any intermediate node */
            {
                q -> prev -> next = q -> next ;
                q -> next -> prev = q -> prev ;
            }
            free ( q ) ;
        }
        return ;  /* return back after deletion */
    }
    q = q -> next ; /* go to next node */
}
printf ( "%d not found.\n", num ) ;
}
```

Linked Lists And Polynomials

Polynomials like $5x^4 + 2x^3 + 7x^2 + 10x - 8$ can be maintained using a linked list. To achieve this each node should consist of three elements, namely coefficient, exponent and a link to the next term. While maintaining the polynomial it is assumed that the exponent of each successive term is less than that of the previous term. Once we build a linked list to represent a polynomial we can use such lists to perform common polynomial operations like addition and multiplication. The program that can perform these operations is given below.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure representing a node of a linked list. The node can store term of a
polynomial */
struct polynode
{
    float coeff ;
    int exp ;
    struct polynode *link ;
};

void poly_append ( struct polynode **, float, int ) ;
void display_poly ( struct polynode * ) ;
void poly_add ( struct polynode *, struct polynode *, struct polynode ** ) ;

int main( )
{
    struct polynode *first, *second, *total ;
    int i = 0 ;

    first = second = total = NULL ; /* empty linked lists */
```

```
    poly_append ( &first, 1.4f, 5 ) ;
    poly_append ( &first, 1.5f, 4 ) ;
    poly_append ( &first, 1.7f, 2 ) ;
    poly_append ( &first, 1.8f, 1 ) ;
    poly_append ( &first, 1.9f, 0 ) ;

    system ( "cls" ) ;

    display_poly ( first ) ;

    poly_append ( &second, 1.5f, 6 ) ;
    poly_append ( &second, 2.5f, 5 ) ;
    poly_append ( &second, -3.5f, 4 ) ;
    poly_append ( &second, 4.5f, 3 ) ;
    poly_append ( &second, 6.5f, 1 ) ;

    printf ( "\n\n" ) ;
    display_poly ( second ) ;

    /* draws a dashed horizontal line */
    printf ( "\n" ) ;
    while ( i++ < 79 )
        printf ( "-" ) ;
    printf ( "\n\n" ) ;

    poly_add ( first, second, &total ) ;
    display_poly ( total ) ;  /* displays the resultant polynomial */
    return 0 ;
}

/* adds a term to a polynomial */
void poly_append ( struct polynode **q, float x, int y )
{
    struct polynode *temp ;
    temp = *q ;

    /* creates a new node if the list is empty */
```

```
    if ( *q == NULL )
    {
        *q = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
        temp = *q ;
    }
    else
    {
        /* traverse the entire linked list */
        while ( temp -> link != NULL )
            temp = temp -> link ;

            /* create new nodes at intermediate stages */
            temp -> link = ( struct polynode * ) malloc ( sizeof ( struct
polynode ) ) ;
            temp = temp -> link ;
    }

    /* assign coefficient and exponent */
    temp -> coeff = x ;
    temp -> exp = y ;
    temp -> link = NULL ;
}

/* displays the contents of linked list representing a polynomial */
void display_poly ( struct polynode *q )
{
    /* traverse till the end of the linked list */
    while ( q != NULL )
    {
        printf ( "%.1f x^%d  :  ", q -> coeff, q -> exp ) ;
        q = q -> link ;
    }

    printf ( "\b\b\b " ) ;  /* erases the last colon */
}

/* adds two polynomials */
void poly_add ( struct polynode *x, struct polynode *y, struct  polynode **s )
```

```
{
    struct polynode *z ;

    /* if both linked lists are empty */
    if ( x == NULL && y == NULL ) ,
        return ;

    /* traverse till one of the list ends */
    while ( x != NULL && y != NULL )
    {
        /* create a new node if the list is empty */
        if ( *s == NULL )
        {
            *s = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
            z = *s ;
        }
        /* create new nodes at intermediate stages */
        else
        {
            z -> link = ( struct polynode * ) malloc ( sizeof ( struct polynode )
);
            z = z -> link ;
        }

        /* store a term of the larger degree polynomial */
        if ( x -> exp < y -> exp )
        {
            z -> coeff = y -> coeff ;
            z -> exp = y -> exp ;
            y = y -> link ;  /* go to the next node */
        }
        else
        {
            if ( x -> exp > y -> exp )
            {
                z -> coeff = x -> coeff ;
                z -> exp = x -> exp ;
                x = x -> link ;  /* go to the next node */
```

```
            }
            else
            {
                /* add the coefficients, when exponents are equal */
                if ( x -> exp == y -> exp )
                {
                    /* assigning the added coefficient */
                    z -> coeff = x -> coeff + y -> coeff ;
                    z -> exp = x -> exp ;
                    /* go to the next node */
                    x = x -> link ;
                    y = y -> link ;
                }
            }
        }
    }

    /* assign remaining terms of the first polynomial to the result */
    while ( x != NULL )
    {
        if ( *s == NULL )
        {
            *s = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
            z = *s ;
        }
        else
        {
            z -> link = ( struct polynode * ) malloc ( sizeof ( struct polynode )
);
            z = z -> link ;
        }

        /* assign coefficient and exponent */
        z -> coeff = x -> coeff ;
        z -> exp = x -> exp ;
        x = x -> link ;  /* go to the next node */
    }
```

```
    /* assign remaining terms of the second polynomial to the result */
    while ( y != NULL )
    {
        if ( *s == NULL )
        {
            *s = ( struct polynode * ) malloc ( sizeof ( struct polynode ) );
            z = *s ;
        }
        else
        {
            z -> link = ( struct polynode * ) malloc ( sizeof ( struct polynode)
);
            z = z -> link ;
        }

        /* assign coefficient and exponent */
        z -> coeff = y -> coeff ;
        z -> exp = y -> exp ;
        y = y -> link ;  /* go to the next node */
    }

    z -> link = NULL ;  /* assign NULL at end of resulting linked list */
}
```

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

/* structure representing a node of a linked list. The node can store a term of
   a polynomial */
struct polynode
{
    float coeff ;
    int exp ;
    struct polynode *link ;
};

void poly_append ( struct polynode **, float, int ) ;
void display_poly ( struct polynode * ) ;
void poly_multiply ( struct polynode *, struct polynode *, struct polynode ** ) ;
void padd ( float, int, struct polynode ** ) ;

int main( )
{
```

```
    struct polynode *first, *second, *mult ;
    int i = 1 ;

    first = second = mult = NULL ;  /* empty linked lists */

    poly_append ( &first, 3, 5 ) ;
    poly_append ( &first, 2, 4 ) ;
    poly_append ( &first, 1, 2 ) ;

    system ( "cls" ) ;

    display_poly ( first ) ;

    poly_append ( &second, 1, 6 ) ;
    poly_append ( &second, 2, 5 ) ;
    poly_append ( &second, 3, 4 ) ;

    printf ( "\n\n" ) ;
    display_poly ( second ) ;

    printf ( "\n" ) ;
    while ( i++ < 79 )
        printf ( "-" ) ;

    poly_multiply ( first, second, &mult ) ;
    printf ( "\n\n" ) ;
    display_poly ( mult ) ;
    return 0 ;
}

/* adds a term to a polynomial */
void poly_append ( struct polynode **q, float x, int y )
{
    struct polynode *temp ;
    temp = *q ;

    /* create a new node if the list is empty */
    if ( *q == NULL )
```

```
    {
        *q = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
        temp = *q ;
    }
    else
    {
        /* traverse the entire linked list */
        while ( temp -> link != NULL )
            temp = temp -> link ;

        /* create new nodes at intermediate stages */
        temp -> link = ( struct polynode * ) malloc ( sizeof ( struct polynode )
);
        temp = temp -> link ;
    }
    /* assign coefficient and exponent */
    temp -> coeff = x ;
    temp -> exp = y ;
    temp -> link = NULL ;
}

/* displays the contents of linked list representing a polynomial */
void display_poly ( struct polynode *q )
{
    /* traverse till the end of the linked list */
    while ( q != NULL )
    {
        printf ( "%.1f x^%d  :  ", q -> coeff, q -> exp ) ;
        q = q -> link ;
    }
    printf ( "\b\b\b " ) ;  /* erases the last colon(:) */
}

/* multiplies the two polynomials */
void poly_multiply ( struct polynode *x, struct polynode *y,
                     struct polynode **m )
{
    struct polynode *y1 ;
```

```
    float coeff1 ;
    int exp1 ;

    y1 = y ;  /* point to the starting of the second linked list */

    if ( x == NULL && y == NULL )
        return ;

    /* if one of the list is empty */
    if ( x == NULL )
        *m = y ;
    else
    {
        if ( y == NULL )
            *m = x ;
        else  /* if both linked lists exist */
        {
            /* for each term of the first list */
            while ( x != NULL )
            {
                /* multiply each term of the second linked list with a
                   term of the first linked list */
                while ( y != NULL )
                {
                    coeff1 = x -> coeff * y -> coeff ;
                    exp1 = x -> exp + y -> exp ;
                    y = y -> link ;

                    /* add the new term to the resultant polynomial */
                    padd ( coeff1, exp1, m ) ;
                }
                y = y1 ;  /* reposition the pointer to the starting of
                             the second linked list */
                x = x -> link ;  /* go to the next node */
            }
        }
    }
}
```

```
/* adds a term to the polynomial in the descending order of the exponent */
void padd ( float c, int e, struct polynode **s )
{
    struct polynode *r = NULL, *temp = *s ;

    /* if list is empty or if the node is to be inserted before the first node */
    if ( *s == NULL || e > ( *s ) -> exp )
    {
        *s = r = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
        ( *s ) -> coeff = c ;
        ( *s ) -> exp = e ;
        ( *s ) -> link = temp ;
    }
    else
    {
        /* traverse the entire linked list to search the position to insert a new
           node */
        while ( temp != NULL )
        {
            if ( temp -> exp == e )
            {
                temp -> coeff += c ;
                return ;
            }
            if ( temp -> exp > e && ( temp -> link == NULL ||
                                      temp -> link -> exp < e ) )
            {
                r = ( struct polynode * ) malloc ( sizeof ( struct polynode ) ) ;
                r -> coeff = c ;
                r -> exp = e ;
                r -> link = temp -> link ;
                temp -> link = r ;
                return ;
            }
            temp = temp -> link ;  /* go to next node */
        }
        r -> link = NULL ;
```

```
            temp -> link = r ;
        }
}
```

Output:

3.0 x^5 : 2.0 x^4 : 1.0 x^2

1.0 x^6 : 2.0 x^5 : 3.0 x^4