/*

Experiment No: 05

Author:Harsheet Sujit Chordiya
Roll No.: 37
Sem & Sec: CSE 3B
Source file: expt05.c
Date Compiled: Jan 2021

==========

Aim: To study singly linked linear lists and implement various operations on it – Create, Insert, Delete, Reverse, Order, Locate, Merge.

Problem Statement: Create a self-referential structure, node to represent a node of a singly linked linear list.
Implement the routines to (1) find length of the list, (2) create a list 3) insert an element – at the beginning, at the end and at a specified position in the list, (4) delete an element–from the front, rear, or a specific position at the list, (5) reverse the list, and (6) search the list.
Create a menu-driven C program to test these routines.
Use the singly linked linear list routines to implement a linked stack and a linked queue.

==========

THEORY

==========

Dynamic Memory Allocation:

Linked lists are inherently dynamic data structures; they rely on new and delete (or malloc and free) for their operation. Normally, dynamic memory management is provided by the C/C++ standard library, with help from the operating system. However, nothing stops us from writing our own allocator, providing the same services as malloc and free.
The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.
malloc()
calloc()
realloc()
free()

Linked Lists:

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Linked List:It is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array. Following are the important terms to understand the concept of Linked List.
Types of Linked List

Following are the various types of linked list.
Simple Linked List − Item navigation is forward only.
Doubly Linked List − Items can be navigated forward and backward.
Circular Linked List − Last item contains link of the first element as next and the first element has a link to the last element as previous.
Basic Operations
Following are the basic operations supported by a list.
Insertion − Adds an element at the beginning of the list.
Deletion − Deletes an element at the beginning of the list.
Display − Displays the complete list.
Search − Searches an element using the given key.
Delete − Deletes an element using the given key.

==========

Algorithm

==========

Algorithm to traverse the linked list:
Step 1: [INITIALIZE] SET PTR = START
Step 2: Repeat Steps 3 and 4 while PTR != NULL
Step 3: Apply Process to PTR DATA
Step 4: SET PTR = PTR NEXT
[END OF LOOP]
Step 5: EXIT

Algorithm to print the number of nodes in a linked list:
Step 1: [INITIALIZE] SET =
Step 2: [INITIALIZE] SET PTR = START
Step 3: Repeat Steps 4 and 5 while PTR != NULL
Step 4: SET=+ 1
Step 5: SET PTR = PTR NEXT
[END OF LOOP]
Step 6: Write COUNT
Step 7: EXIT

Algorithm to insert node at specified position:
Step 1: IF AVAIL = NULL
Write OVERFLOW
Go to Step 12
[END OF IF]
Step 2: SET = AVAIL
Step 3: SET AVAIL = AVAIL NEXT
Step 4: SET DATA = VAL
Step 5: SET PTR = START
Step 6: SET PREPTR = PTR
Step 7: Repeat Steps 8 and 9 while PTR DATA != NUM
Step 8: SET PREPTR = PTR
Step 9: SET PTR = PTR NEXT
[END OF LOOP]
Step 1 : PREPTR NEXT =
Step 11: SET NEXT = PTR
Step 12: EXIT

Algorithm to delete node at specified position:
Step 1: IF START = NULL
Write UNDERFLOW
Go to Step 1
[END OF IF]
Step 2: SET PTR = START
Step 3: SET PREPTR = PTR
Step 4: Repeat Steps 5 and 6 while PREPTR DATA != NUM
Step 5: SET PREPTR = PTR
Step 6: SET PTR = PTR NEXT
[END OF LOOP]
Step 7: SET TEMP = PTR
Step 8: SET PREPTR NEXT = PTR NEXT
Step 9: FREE TEMP
Step 10 : EXIT

```
================================================================================
                              CODE
================================================================================
*/
//Header file delecrations
#include <stdio.h>
#include <stdlib.h>

struct nodeLL
{
   int data;
   struct nodeLL *link;
};
typedef struct nodeLL node;
typedef node *list;

//Function Declaration/Signatures/Prototypes
list createNodeSLL();
list insertBegSLL(list, int);
list insertEndSLL(list, int);
list insertPosSLL(list, int, int);
list deleteBegSLL(list, int *);
list deleteEndSLL(list, int *);
list deletePosSLL(list, int *, int);
void displaySLL(list);
list reverseSLL(list);
void sortSLL(list);
int searchnodeSLL(list, int);
void mergeSLL(list, list);

int largestSLL(list);
int sumSLL(list);
int nodeCountSLL(list);

//Driver Functions
int main()
```

```
{

   list first = NULL; //struct nodeLL *first=NULL;
   list second = NULL;
   int key, responce, choice, pos, n;
   do
   {
      printf("\t1.insert\t2.delete\t3.reverse\t4.order\n");
      printf("\t5.locate\t6.merge \t7.show   \t0.exit\n");
      printf("\tresponce??  ");
      scanf("%d", &responce);
      printf("\n");
      switch (responce)
      {
      case 0:
         printf("\tyou opted to Exit...\n");
         break;
      case 1:
         do
         {
            printf("\t\t1.insert at begin   \t2.insert at end\n");
            printf("\t\t3.insert at position\t0.exit\n");
            printf("\t\tchoice??  ");
            scanf("%d", &choice);
            printf("\n");
            switch (choice)
            {
            case 0:
               printf("\t\tyou opted to Exit...\n");
               break;
            case 1:
               printf("\t\tenter the element to be inserted: ");
               scanf("%d", &key);
               first = insertBegSLL(first, key);
               break;
            case 2:
               printf("\t\tenter the element to be inserted: ");
               scanf("%d", &key);
               first = insertEndSLL(first, key);
               break;
            case 3:
               printf("\t\tenter the element to be inserted: ");
               scanf("%d", &key);
               printf("\t\tenter the position at it is to be inserted: ");
               scanf("%d", &pos);
               first = insertPosSLL(first, key, pos);
               break;
            default:
               printf("\t\tInvalid operation code\n ");
               break;
            }
         } while (choice != 0);
```

```
    break;
case 2:
    do
    {
        printf("\t\t1.delete at begin   \t2.delete at end\n");
        printf("\t\t3.delete at position\t0.exit\n");
        printf("\t\tchoice??  ");
        scanf("%d", &choice);
        printf("\n");
        key = 0;
        switch (choice)
        {
        case 0:
            printf("\t\tyou opted to Exit...\n");
            break;
        case 1:
            first = deleteBegSLL(first, &key);
            printf("\t\tthe value %d is deleted\n", key);
            break;
        case 2:
            first = deleteEndSLL(first, &key);
            printf("\t\tthe value %d is deleted\n", key);
            break;
        case 3:
            // printf("\t\tenter the element to be inserted\n");
            // scanf("%d", &key);
            printf("\t\tenter the position at it is to be inserted: ");
            scanf("%d", &pos);
            first = deletePosSLL(first, &key, pos);
            printf("\t\tthe value %d is deleted\n", key);
            break;
        default:
            printf("\t\tInvalid operation code\n ");
            break;
        }
    } while (choice != 0);

    break;
case 3: //reverse
    first = reverseSLL(first);
    printf("\tlist is reversed\n");
    break;
case 4: //order
    sortSLL(first);
    printf("\tlist is ordered\n");
    break;
case 5: //locate
    printf("\tenter the element to be searched: ");
    scanf("%d", &key);
    pos = searchnodeSLL(first, key);
    printf("\tthe element %d is found at index %d\n", key, pos+1);
```

```
            break;
        case 6: //merge
            //to create another list
            printf("\tthe total no. of elements you want to insert in second list: ");
            scanf("%d", &n);
            for (int i = 1; i <= n; i++)
            {
                printf("\tenter the %d element : ",i);
                scanf("%d", &key);
                second = insertBegSLL(second, key);
            }
            mergeSLL(first, second);
            printf("\tboth the list is merged\n");
            break;
        case 7: //show
            displaySLL(first);
            break;

        default:
            printf("\tInvalid operation code\n ");
            break;
        }
    } while (responce != 0);

    return 0;
}

//function defination
list createNodeSLL()
{
    list neww = (list)malloc(sizeof(node)); //returns the address of void type
    if (neww == NULL)
        printf("\t\tMemory underflow\n");
    else
        printf("\t\tnode created\n");
    return (neww);
}
list insertBegSLL(list first, int key)
{
    list neww; //create a node
    neww = createNodeSLL();
    //printf("\taddr(node):=%lu \n", (unsigned long)neww);
    if (neww == NULL) //node is not created
        return (first);
    //populate the node
    neww->data = key;
    neww->link = NULL;
    //if list empty return current node as list.
    if (first == NULL)
        return (neww);
    //insert node and return the list.
    neww->link = first;
```

```
    return (neww);
}
list insertEndSLL(list first, int key)
{
    list neww; //create a node
    neww = createNodeSLL();
    //printf("\taddr(node):=%lu \n", (unsigned long)neww);
    if (neww == NULL) //node is not created
        return (first);
    neww->data = key;
    neww->link = NULL;
    if (first == NULL) //node is not created
        return (neww);
    list temp;
    temp = first;
    while (temp->link != NULL)
    {
        temp = temp->link;
    }
    temp->link = neww;
    return (first);
}

list deleteBegSLL(list first, int *key)
{
    if (first == NULL)
    {
        printf("\tthe list is empty. Delete Fails!\n");
        return (first);
    }
    list temp;
    temp = first;
    *key = temp->data;
    first = first->link;
    free(temp);
    return (first);
}
list deleteEndSLL(list first, int *key)
{
    if (first == NULL)
    {
        printf("\tthe list is empty. Delete Fails!\n");
        return (first);
    }
    if (first->link == NULL)
    {
        *key = first->data;
        free(first);
        return (NULL);
    }
    list temp;
    temp = first;
```

```
    while (temp->link->link != NULL)
    {
        temp = temp->link;
    }
    *key = temp->link->data;
    free(temp->link);         //change
    temp->link = NULL;
    return (first);
}
void displaySLL(list first)
{
    if (first == NULL)
    {
        printf("\tlist is empty\n");
        return;
    }
    printf("\n\tList :\n\t");
    while (first != NULL)
    {
        printf("%3d-->[%lu]-->", first->data, (unsigned int)first->link);
        first = first->link;
    }
    printf("NULL\n");
    return;
}
int largestSLL(list first)
{
    list temp = first;
    int max;
    max = temp->data;
    temp = temp->link;
    while (temp != NULL)
    {
        if (temp->data > max)
        {
            max = temp->data;
        }
        temp = temp->link;
    }
    return (max);
}
int sumSLL(list first)
{
    int sum = 0;
    list temp;
    temp = first;
    while (temp != NULL)
    {
        sum = sum + temp->data;
        temp = temp->link;
    }
    return (sum);
```

```
}
int nodeCountSLL(list first)
{
   int count = 0;
   list temp = first;
   while (temp != NULL)
   {
      count++;
      temp = temp->link;
   }
   return (count);
}

int searchnodeSLL(list first, int key)
{
   list temp;
   temp = first;
   while (temp != NULL)
   {
      if (temp->data == key)
         return 1;
      else
         temp = temp->link;
   }
   return 0;
}
void sortSLL(list first)
{
   list current, index;
   current = first;
   index = NULL;
   int temp;
   if (first == NULL)
   {
      printf("\tEmpty\n");
      return;
   }
   else
   {
      while (current != NULL)
      {
         index = current->link; //index points to the node next to current
         while (index != NULL)
         {
            // if current greater than index then swap
            if ((current->data) > (index->data))
            {
               temp = current->data;
               current->data = index->data;
               index->data = temp;
            }
            index = index->link;
```

```
            }
            current = current->link;
        }
    }
}
list insertPosSLL(list first, int key, int pos)
{
    int ndx = 0;
    int len = nodeCountSLL(first);
    if (pos < 1 || pos > len)
    {
        printf("Invalid position. Inserction Fails!\n");
        return (first);
    }
    list neww;
    neww = createNodeSLL();
    if (neww == NULL)
        return (first);
    neww->data = key;
    neww->link = NULL;
    if (pos == 1)
    {
        neww->link = first;
        return (neww);
    }
    list temp;
    temp = first;
    while (ndx < pos - 2)
    {
        ndx = ndx + 1;
        temp = temp->link;
    }
    neww->link = temp->link;
    temp->link = neww;
    return (first);
}
list reverseSLL(list first)
{
    if (first == NULL)
        return (NULL);
    list rev, ptr1, ptr2;
    rev = first;
    ptr2 = first->link->link;
    ptr1 = first->link;
    rev->link = NULL;
    ptr1->link = rev;
    while (ptr2 != NULL)
    {
        rev = ptr1;
        ptr1 = ptr2;
        ptr2 = ptr2->link;
        ptr1->link = rev;
```

```
    }
    return (ptr1);
}
list deletePosSLL(list first, int *key, int pos)
{
    int ndx = 0;
    int len = nodeCountSLL(first);
    if (first == NULL)
    {
        printf("\tthe list is empty. Delete Fails!\n");
        return (first);
    }
    if (pos < 1 || pos > len)
    {
        printf("\tInvalid position. Deletision Fails!\n");
        return (first);
    }
    list temp, save;
    temp = first;
    if (pos == 1)
    {
        first = first->link;
        *key = temp->data;
        free(temp);
        return (first);
    }
    while (ndx < pos - 2)
    {
        ndx = ndx + 1;
        temp = temp->link;
    }
    *key = temp->link->data;
    save = temp->link;
    temp->link = temp->link->link;
    free(save);
    return (first);
}

void mergeSLL(list a, list b)
{
    sortSLL(a);
    sortSLL(b);
    if (a->link == NULL)
        a->link = b;
    else
        mergeSLL(a->link, b);
}
/*
=============================================================================
                            OUTPUT TRAIL
=============================================================================
```

```
C:\Users\Harsheet\CLionProjects\ta4\cmake-build-debug\ta4_second.c.exe
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??1

        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??1

        enter the element to be inserted:1
       node created
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??2

        enter the element to be inserted:2
       node created
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??0

        you opted to Exit...
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??7


List :
  1-->[13047256]-->  2-->[0]-->NULL
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??2

        1.delete at begin       2.delete at end
        3.delete at position    0.exit
        choice??2

        the value 2 is deleted
        1.delete at begin       2.delete at end
        3.delete at position    0.exit
        choice??0

        you opted to Exit...
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??7


List :
  1-->[0]-->NULL
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
```

```
responce??1

        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??2

        enter the element to be inserted:2
       node created
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??2

        enter the element to be inserted:3
       node created
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??3

        enter the element to be inserted:4
       enter the position at it is to be inserted:4
 Invalid posi
tion. Inserction Fails!
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??3

        enter the element to be inserted:4
       enter the position at it is to be inserted:3

node created
        1.insert at begin       2.insert at end
        3.insert at position    0.exit
        choice??0

        you opted to Exit...
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??7


List :
  1-->[13047256]-->  2-->[13047288]-->  4-->[13047272]-->  3-->[0]-->NULL
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??3

list is reversed
1.insert        2.delete        3.reverse       4.order
5.locate        6.merge         7.show          0.exit
responce??7
```

List :
3-->[13047288]--> 4-->[13047256]--> 2-->[13047240]--> 1-->[0]-->NULL
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??4

list is ordered
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??7

List :
1-->[13047288]--> 2-->[13047256]--> 3-->[13047240]--> 4-->[0]-->NULL
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??5

enter the element to be searched:2
the element 2 is found at index 2
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??6

the total no. of elements you want to insert in second list:3
enter the 1 element :11
node created
enter the 2 element :22
node created
enter the 3 element :33
node created
both the list is merged
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??7

List :
1-->[13047288]--> 2-->[13047256]--> 3-->[13047240]--> 4-->[13047336]--> 11--
>[13047320]--> 22-->[13047304]-
-> 33-->[0]-->NULL
1.insert 2.delete 3.reverse 4.order
5.locate 6.merge 7.show 0.exit
responce??0

==========

VIVA Questions

==========

1] How does linked list different from array?

Ans: Arrays store elements in contiguous memory locations, resulting in easily calculable addresses for the elements stored and this allows fasteraccess to an element at a specific index. Linked

lists are less rigid in their storage structure and elements are usually not stored in contiguous locations,
hence they need to be stored with additional tags giving a reference to the next element. This difference in the data storage scheme decides which data structure would be more suitable for a given situation.
Size: Since data can only be stored in contiguous blocks of memory in an array, its size cannot be altered at runtime due to the risk of overwriting other data. However, in a linked list, each node points to the next one such that data can exist at scattered (non-contiguous) addresses; this allows for a dynamic size that can change at runtime.
Memory allocation: For arrays at compile time and at runtime for linked lists. but, a dynamically allocated array also allocates memory at runtime.
Memory efficiency: For the same number of elements, linked lists use more memory as a reference to the next node is also stored along with the data.
However, size flexibility in linked lists may make them use less memory overall; this is useful when there is uncertainty about size or there arelarge variations in the size of data elements;
memory equivalent to the upperlimit on the size has to be allocated (even if not all of it is being used)
while using arrays, whereas linked lists can increase their sizes step-by-step proportionately to the amount of data.

2] What is circular linked lists? List its merits.
Ans: A circular linked list is a sequence of nodes arranged such a way that each node can be retraced to itself. Here a “node” is a self-referential element with pointers to one or two nodes in it’s immediate vicinity.
Some of the advantages of circular linked lists are:
1) No requirement for a NULL assignment in the code. The circular list never points to a NULL pointer unless fully deallocated.
2) Circular linked lists are advantageous for end operations since beginning and end coincide. Algorithms such as the Round Robin scheduling can neatly eliminate processes which are queued in a circular fashion without encountering dangling or NULL-referential pointers.
3) Circular linked list also performs all regular functions of a singly linked list. In fact, circular doubly linked lists discussed below can even eliminate the need for a full-length traversal to locate an element. That element would at most only be exactly opposite to the start, completing just half the linked list.

3] State advantages of using doubly linked lists.
Ans: Advantages of a Doubly Linked List
- Allows us to iterate in both directions.
- We can delete a node easily as we have access to its previous node.
- Reversing is easy.
- Can grow or shrink in size dynamically.
- Useful in implementing various other data structures.

4] Does linked list facilitate random access? Does it better the memory requirement as compared to array?
Ans: To implement file system, for separate chaining in hash-tables and to

implement non-binary trees linked lists are used. Elements are accessed sequentially in linked list. Random access of elements is not an applications of linked list.
The memory required to store data in the linked list is more than that of an array because of additional memory used to store the address/references of the next node. In an array, memory is assigned during compile time while in a Linked list it is allocated during execution or runtime.

*/