USN | | | | | | | | | |

Internal Assessment Test II – Oct 2019

Sub:	Data Structures & Algorithms					Sub Code:	18CS32	Branch:	CSE
Date:	15/10/2019	Duration:	90 min’s	Max Marks:	50	Sem / Sec:	3 ‘A’,’B’ & ‘C’		

Answer any FIVE FULL Questions MARKS

1 What is a polynomial list? Represent the polynomial $3x^3yz + 6x^2y^2z+2xy^5z-2xyz^3-4yz^5$ in a linked list diagram. Write C function to add two polynomials. [10]

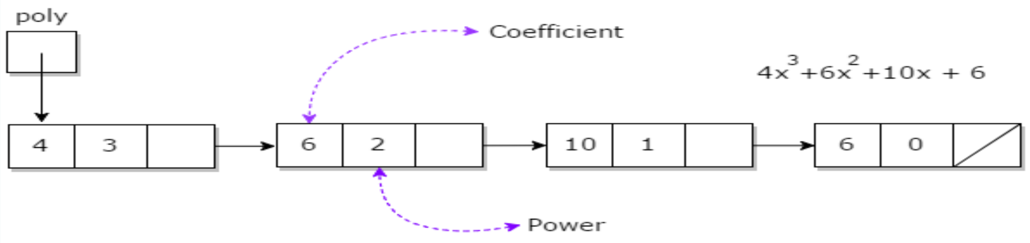
Solution:

A polynomial can be thought of as an ordered list of non zero terms. Each non zero term is a two-tuple which holds two pieces of information:

The exponent part

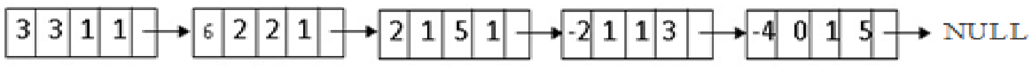
The coefficient part

Example:



The polynomial $3x^3yz + 6x^2y^2z+2xy^5z-2xyz^3-4yz^5$ in a linked is represented as follows:

Coefficients, powers of x,y & z & pointer to next



C function to add two polynomials:

```
// Function Adding two polynomial numbers
void polyadd(struct Node *poly1, struct Node *poly2, struct Node *poly)
{
while(poly1->next && poly2->next)
```

```
{
        // If power of 1st polynomial is greater then 2nd, then store 1st as it is
            // and move its pointer
        if(poly1->pow > poly2->pow)
        {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
        // If power of 2nd polynomial is greater then 1st, then store 2nd as it is
        // and move its pointer
        else if(poly1->pow < poly2->pow)
        {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
           // If power of both polynomial numbers is same then add their coefficients
        else
        {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff+poly2->coeff;
            poly1 = poly1->next;
            poly2 = poly2->next;
        }
            // Dynamically create new node
        poly->next = (struct Node *)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
     while(poly1->next || poly2->next)
    {
        if(poly1->next)
        {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }

        if(poly2->next)
        {
            poly->pow = poly2->pow;
            poly->coeff = poly2->coeff;
            poly2 = poly2->next;
        }
        poly->next = (struct Node *)malloc(sizeof(struct Node));
        poly = poly->next;
        poly->next = NULL;
    }
}
```

2 Differentiate between singly linked list and doubly linked list. Write a C program to concatenate two Singly Linked List [10]

Solution:

Singly linked list	**Doubly linked list**
A singly linked list is a linked list where the node contains some data and a pointer to the next node in the list	A doubly linked list is complex type of linked list where the node contains some data and a pointer to the next as well as the previous node in the list
It allows traversal only in one way	It allows a two way traversal
It uses less memory per node .	It uses more memory per node.
Complexity of insertion and deletion at a known position is O(n)	Complexity of insertion and deletion at a known position is O(1)

If we need to save memory and searching is not required, we use singly linked list	If we need better performance while searching and memory is not a limitation, we go for doubly linked list
If we know that an element is located towards the end section, eg. ‘zebra’ still we need to begin from start and traverse the whole list	If we know that an element is located towards the end section e.g. ’zebra’ we can start searching from the Back.
Singly linked list can mostly be used for stacks .They can be used to implement stacks	They can be used to implement stacks, heaps, binary trees.

C program to concatenate two Singly Linked List :

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

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

struct node *create_list(struct node *);
struct node *concat( struct node *start1,struct node *start2);
struct node *addatbeg(struct node *start, int data);
struct node *addatend(struct node *start,int data);
void display(struct node *start);

int main()
{
    struct node *start1=NULL,*start2=NULL;
    start1=create_list(start1);
    start2=create_list(start2);
    printf("First list is  : ");
    display(start1);
```

```
  printf("Second list is  : ");
    display(start2);
    start1=concat(start1, start2);
    printf("Concatenated list is  : ");
    display(start1);
}

struct node *concat( struct node *start1,struct node *start2)
{
    struct node *ptr;
    if(start1==NULL)
    {
        start1=start2;
        return start1;
    }
    if(start2==NULL)
        return start1;
    ptr=start1;
    while(ptr->link!=NULL)
        ptr=ptr->link;
    ptr->link=start2;
    return start1;
}

struct node *create_list(struct node *start)
{
    int i,n,data;
    printf("Enter the number of nodes : ");
    scanf("%d",&n);
    start=NULL;
    if(n==0)
        return start;
    printf("Enter the element to be inserted : ");
  scanf("%d",&data);
   start=addatbeg(start,data);
    for(i=2;i<=n;i++)
    {
        printf("Enter the element to be inserted : ");
        scanf("%d",&data);
        start=addatend(start,data);
    }
    return start;
}

void display(struct node *start)
{
    struct node *p;
    if(start==NULL)
    {
        printf("List is empty\n");
        return;
    }
    p=start;
    while(p!=NULL)
    {
        printf("%d ", p->info);
        p=p->link;
    }
    printf("\n");
}

struct node *addatbeg(struct node *start,int data)
{
    struct node *tmp;
    tmp=(struct node *)malloc(sizeof(struct node));
    tmp->info=data;
    tmp->link=start;
    start=tmp;
    return start;
}
struct node *addatend(struct node *start, int data)
{
    struct node *p,*tmp;
    tmp= (struct node *)malloc(sizeof(struct node));
    tmp->info=data;
    p=start;
    while(p->link!=NULL)
        p=p->link;
    p->link=tmp;
    tmp->link=NULL;
    return start;
}
```

Output:

```
 Enter the number of nodes : 3
 Enter the element to be inserted : 4 5 6
 Enter the element to be inserted :3
 Enter the element to be inserted : 7 8 9
 First list is  : 4 5 6
 Second list is  : 7 8 9
 Concatenated list is  : 4 5 6 7 8 9
```

3 Write a C program to perform the operations for a circular queue with encountering error message (when circular queue is full or empty) [10]

Solution:

```
    #include <stdio.h>
    #define SIZE 5
```

```
int items[SIZE];
int front = -1, rear =-1;

int isFull()
{
    if( (front == rear + 1) || (front == 0 && rear == SIZE-1)) return 1;
    return 0;
}

int isEmpty()
{
    if(front == -1) return 1;
   return 0;
}

void enQueue(int element)
{
    if(isFull()) printf("\n Queue is full!! \n");
    else
    {
       if(front == -1) front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("\n Inserted -> %d", element);
    }
}

int deQueue()
{
    int element;
    if(isEmpty()) {
        printf("\n Queue is empty !! \n");
        return(-1);
```

```
  }
    else
   {
       element = items[front];
       if (front == rear){
           front = -1;
           rear = -1;
       } /* Q has only one element, so we reset the queue after dequeing it. ? */
       else {
           front = (front + 1) % SIZE;

       }
       printf("\n Deleted element -> %d \n", element);
       return(element);
   }
}

void display()
{
   int i;
   if(isEmpty()) printf(" \n Empty Queue\n");
   else
   {
       printf("\n Front -> %d ",front);
       printf("\n Items -> ");
       for( i = front; i!=rear; i=(i+1)%SIZE) {
           printf("%d ",items[i]);
       }
       printf("%d ",items[i]);
       printf("\n Rear -> %d \n",rear);
   }
}
```

```
int main()
{
    // Fails because front = -1
    deQueue();
    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);
    // Fails to enqueue because front == 0 && rear == SIZE - 1
    enQueue(6);
    display();
    deQueue();
    display();
    enQueue(7);
    display();
    // Fails to enqueue because front == rear + 1
    enQueue(8);
    deQueue();
   deQueue();
   deQueue();
   deQueue();
   deQueue();
   deQueue();
    return 0;
}Output:
```

```
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Queue is full!!

Front -> 0
Items -> 1 2 3 4 5
Rear -> 4
```

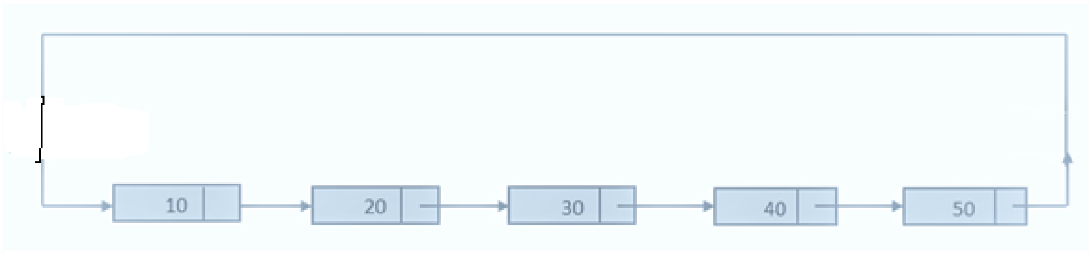
```
  Deleted element -> 1


 Front -> 1
  Items -> 2 3 4 5
  Rear -> 4

  Inserted -> 7
  Front -> 1
  Items -> 2 3 4 5 7
  Rear -> 0

  Queue is full!!

Deleted element -> 2

  Front -> 2
  Items -> 3 4 5
  Rear -> 0

Deleted element -> 3

  Front -> 3
  Items ->  4 5
  Rear -> 0

Deleted element -> 4

  Front -> 4
  Items ->  5
  Rear -> 0

Deleted element -> 5

  Front -> 0
  Items ->
  Rear -> 0

Queue is empty !!
```

4 What is circular linked list? Write a C program to perform the following operations on circular doubly linked list. I)Insert front   ii)Insert end   III) Delete end [10]

Solution:

A circular linked list is a linked list in which the last node points to the head or front node making the data structure to look like a circle. A circularly linked list node can be implemented using singly linked or doubly linked list.

The below representation shows how a circular linked list . Unlike the linear linked list, the last node is pointed to the start node using a rear pointer.



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

struct node
{
   int val;
   struct node *next;
   struct node *prev;
};
typedef struct node n;

n* create_node(int);
void add_node();
void insert_at_first();
void insert_at_end();
void display_from_beg();
n *new, *ptr, *prev;
n *first = NULL, *last = NULL;
int number = 0;
void main()
{
   int ch;

  printf("\n linked list\n");
  printf("1.insert at beginning \n 2.insert at end\n  3.displaylist from beginning\n4.delete end\n 5.exit ");

   while (1)
  {

      printf("\n enter your choice:");
      scanf("%d", &ch);
      switch (ch)
      {
      case 1 :
         insert_at_first();
         break;
      case 2 :
```

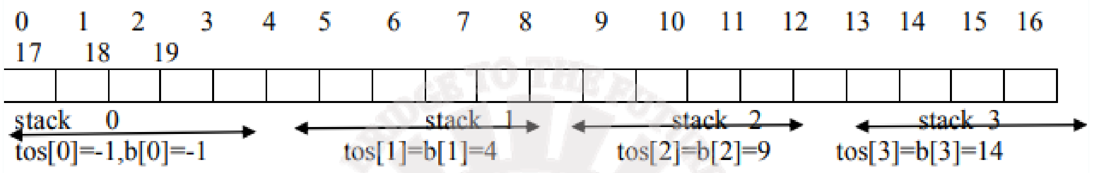
```
   insert_at_end();
            break;
        case 3 :
            display_from_beg();
            break;
        case 4 :
            Delete_end();
            break;
       case 5 :
   exit(0);
case 6 :
            add_node();
            break;
        default:
            printf("\ninvalid choice");
        }
    }
  }
  /*
   *MEMORY ALLOCATED FOR NODE DYNAMICALLY
   */
  n* create_node(int info)
  {
     number++;
     new = (n *)malloc(sizeof(n));
     new->val = info;
     new->next = NULL;
     new->prev = NULL;
     return new;
  }
  /*
   *ADDS NEW NODE
   */
  void add_node()
  {

     int info;

    printf("\nenter the value you would like to add:");
     scanf("%d", &info);
     new = create_node(info);

     if (first == last && first == NULL)
     {

        first = last = new;
       first->next = last->next = NULL;
        first->prev = last->prev = NULL;
     }
     else
     {
        last->next = new;
```

```
            new->prev = last;
            last = new;
             last->next = first;
            first->prev = last;
         }
      }
      /*
       *INSERTS ELEMENT AT FIRST
       */
      void insert_at_first()
{

         int info;

         printf("\nenter the value to be inserted at first:");
         scanf("%d",&info);
         new = create_node(info);

         if (first == last && first == NULL)
         {
            printf("\ninitially it is empty linked list later insertion is done");
            first = last = new;
            first->next = last->next = NULL;
            first->prev = last->prev = NULL;
         }
        else
         {
            new->next = first;
            first->prev = new;
            first = new;
            first->prev = last;
            last->next = first;
            printf("\n the value is inserted at begining");
         }
      }
      /*
       *INSERTS ELEMNET AT END
       */
      void insert_at_end()
      {

         int info;

         printf("\nenter the value that has to be inserted at last:");
         scanf("%d", &info);
         new = create_node(info);

         if (first == last && first == NULL)
         {
            printf("\ninitially the list is empty and now new node is inserted but
      at first");
            first = last = new;
        first->next = last->next = NULL;
        first->prev = last->prev = NULL;
     }
    else
    {
        last->next = new;
        new->prev = last;
        last = new;
        first->prev = last;
        last->next = first;
     }
}
void delete_end()
{
    n *temp, *prevnode;


    if (first == last && first == NULL)
        printf("\n empty linked list you cant delete");

        else
        {
          While(ptr->next != first)
           {
               prevnode = ptr;
              ptr = ptr->next;
           }

                    prevnode->next = ptr->next;
                    first->prev = prevnode;
                    printf("%d is deleted", ptr->val);
   free(ptr);
                    break;
               }
           }
        }
    }
}
/*
 *DISPLAYING IN BEGINNING
 */
void display_from_beg()
{
    int i;
    if (first == last && first == NULL)
        printf("\nlist is empty no elemnts to print");
    else
    {
        printf("\n%d number of nodes are there", number);
        for (ptr = first, i = 0;i < number;i++,ptr = ptr->next)
            printf("\n %d", ptr->val);
    }}
```

5 Explain in detail about multiple stacks, with relevant C functions to perform push and pop operations. [10]

Solution:

Multiple stacks A sequential representation of a single stack using array is simple since only the top of the stack needs to be maintained and kept track. A linear structure like array can be used to represent multiple stacks. If multiple stacks are to be implemented, the array can be approximately divided into equal sized segments, each segment denoting a stack. The top and bottom of each stack are to be kept track of to manage insertions and deletions into the individual stacks. Consider a set of N stacks to be implemented using an array. The array can be divided into N equal sized segments. Say if the array can hold 20 elements, and 4 stacks are to be implemented then each individual stack holds 5 elements.



If i denotes an individual stack ,to establish multiple stacks, an array of top(tos[i]) and bottom pointers (b[i])are maintained to keep track of the top and bottom of every stack. Every stack‘s bottom and top pointer is set to B[i]=tos[i]=(size/n)*i-1 which enables dividing the stack to be divided into equal sized segments.

Overflow in any stack

tos[i]=b[i+1] // top pointer of one stack points to the bottom position of the following stack

Underflow in any stack tos[i]=b[i] //top and bottom pointer of a stack in the same position

Implementation of multiple stacks using array

```
#define memsize 20 //size of array
 #define maxstack 4
//number of stacks
int s[memsize],tos[maxstack],b[maxstack],n;
int main()
{
int i;
scanf(“%d”,&n);
//number of stacks
for(i=0;i<=tos[i];j++)
      tos[i]=b[i]=(memsize/n)*i-1;
      b[n]=memsize-1;
      // use switch case to call the different operations push, pop and display
      }
      void push()
       {
             int ele;
             scanf(“%d”,&i); //stack number on which operation is to be done
            if(tos[i]==b[i+1])
            {
                printf(“stack %d is full”, i);
                return;
            }
           printf(“enter the value to be inserted\n”);
            scanf(“%d”,&ele);
             s[++tos[i]]=ele;
         }
       void pop()
       {
           printf(“enter the stack number\n”);
           scanf(“%d”,&i);
            if(tos[i]==b[i])
            {
                printf(“empty stack\n”);
                return;
           }
```

```
  printf("deleted element is %d",s[tos[i]--];
 }
void disp()
{
       printf("enter the stack number\n");
        scanf("%d",&i);

       if (tos[i]==b[i])
      {
             printf("empty stack\n");return;
      }
       printf("contents are \n");
       for(j=b[i]+1;j<=tos[i];j++)
       printf("%d",s[j]);
}
```

6 Explain priority queue & write C program for insert & sort operations in a priority queue. [10]

Solution:

Priority queue is like a queue, but where additionally each element has a "priority" associated with it. For example jobs/processes in operating system can be processed according to its priority. In real life example also some objects has priority to get itself processed/operated/deleted first.

```
          Delete/process <---                                  <-----
Insert
                              P1 3   p2 2  p3 1  p4 5  p5 4
```

A priority Queue with processes and its priority number

In a priority queue, an element with high priority (generally the least priority number will have highest priority) is served before an element with low priority. So, items in priority queue are sorted according its priority. If two elements have the same priority, they are served according to their order in the queue.

}

C program:

```
#include <stdio.h>

#include <string.h>

#define MAX 5

struct data {        char job[10] ;     int prno ; };

int front, rear ;

struct data d[MAX];

void pque( )

{

                int i;       front = rear = -1 ;

              for ( i = 0 ; i < MAX ; i++ )

             { strcpy ( d[i].job, '\0' ) ; d[i].prno = 0 ; }

}

void add ( struct data dt )

{

                struct data temp;

                int i,j;

                if ( rear == MAX - 1 )       { printf("\nQueue is full") ;          return ; }

                rear++ ;                d[rear] = dt ;

                if ( front == -1 ) front = 0 ;

                for ( i = front ; i <= rear ; i++ )

                 for ( j = i + 1 ; j <= rear ; j++ )

                  if ( d[i].prno > d[j].prno ) { temp = d[i] ; d[i] = d[j] ; d[j] = temp
;                }

}
```

```
 struct data del()

{

                struct data  t ;    strcpy ( t.job, "" ) ;           t.prno = 0 ;

                if ( front == -1 )

                {

                     printf("\nQueue is Empty\n");

                     return t ; }

                      t = d[front] ;

                     d[front] = t ;

                    if ( front == rear )  front = rear = -1 ;

                    else

                   front++ ;

                  return  t ;

}

main( )
{
                 struct data dt,temp ;

                int i ;

                pque();

                printf("Enter Job description and its priority number:");

                printf("Lower the priority number, higher the priority:") ;

                printf("\nJob Description    Priority\n");

                 for ( i = 0 ; i < MAX ; i++ )

               {

                        scanf("%s%d",dt.job,&dt.prno) ; add ( dt ) ;

 }
```

```
            printf("\n");

              printf("Process jobs priority wise\n");

               printf("Job    Priority\n");

                for ( i = 0 ; i < MAX ; i++ )

               {

             temp  =  del( ) ;
             printf("%s    %d\n",temp.job,temp.prno);

          }

              printf("\n");

   return 0;

}
```

7 Define a MAZE problem with solution design. Write a C program to implement the same. [10]

Solution:

In this problem, there is a given maze of size N x N. The source and the destination location is top-left cell and bottom right cell respectively. Some cells are valid to move and some cells are blocked. If one rat starts moving from start vertex to destination vertex, we have to find that is there any way to complete the path, if it is possible then mark the correct path for the rat.

The maze is given using a binary matrix, where it is marked with 1; it is a valid path, otherwise 0 for a blocked cell.

C program:

```
#include<stdio.h>

 #include<conio.h>

 #define ROW 7

 #define COL 5
```

```
/* Given a 2d matrix filled with 0s and 1s.

*  Find 1 in adjacent cell, bBlockages is marked with 0.

* if wrong path is selected then backtrack

Example:

1 1 1 1 1

0 0 0 0 1

1 1 1 0 1

1 0 1 1 1

1 0 0 0 0

1 0 1 1 1

1 1 1 0 1

*/

     static int x = -1;

    static int y = -1;

    enum movement {FWD, BWD, UP, DN}move;

    void maze(int arr[ROW][COL], int row, int col,int r1[30],int c1[30],enum
movement move)

    {

      int i;

      if (arr[row][col] == 1)/*If there is a path */

      {

      r1[++x] = row;

      c1[++y] = col;

      if ((col == COL - 1) && (row == ROW-1))/*Rat has reached exit*/

      {

          printf("\nSolution\n");
    for (i = 0; i <= x; i++)
    {
              printf("(%d,%d) ",r1[i], c1[i]);
    }
     getch();
      exit(0);
  }
else if(col == COL-1)/*Last col*/
  {
    if(move == DN)/*Moved from last col down*/
      {
      if(arr[row+1][col] == 0)
        {
      maze(arr, row, col-1, r1, c1,BWD);/*Move to previous col*/
        }
      }
    maze(arr,row+1,col,r1,c1,DN);/*Move to next row*/
  }
else if(row == ROW-1)/*Last row*/
  {
    if(move == FWD)
      {
      if(arr[row][col+1] == 0)
        {
      maze(arr, row-1, col, r1, c1,UP);/*Move to previous col*/
```

```
          }
        }
      maze(arr,row,col+1,r1,c1,FWD);/*Move to next col*/
    }
  else if(move == BWD)
    {
      if(arr[row+1][col] == 1)
       maze(arr, row+1, col, r1, c1,DN);/*Move to next row*/
      else if(arr[row][col-1] == 1)
       maze(arr, row, col-1, r1, c1,BWD);
      else if(arr[row-1][col] == 1)
       maze(arr, row-1, col, r1, c1,UP);
    }
  else if(move == UP)/*Moved up*/
    {
    if(arr[row][col-1] == 1)
      maze(arr, row, col-1, r1, c1,BWD);/*Move to previous col*/
      else
      maze(arr, row, col+1, r1, c1,FWD);
    }
    else
    {
      maze(arr, row, col + 1, r1, c1,FWD);/*Move to next col*/
      maze(arr, row + 1, col, r1, c1,DN);/*Move to next row*/
```

```
        if((arr[row][col+1] == 0) && (arr[row+1][col] == 0))
      {
                maze(arr, row, col-1, r1, c1,BWD);/*Move to previous col*/
      }
    }
  }
      if((r1[x] == row) && (c1[y] == col))/*Erase the paths bactracked*/
    {
        c1[y--]=0;
         r1[x--]=0;
    }
}/*maze*/

void main()
{
  int arr[ROW][COL],i,j;/*maze*/
  int r1[30] = { 0 };/*Row value of solution*/
  int c1[30] = { 0 };/*Column value of solution*/
    clrscr();
  /*Input with either 0/1*/
    for (i = 0; i < ROW; i++)
    {
    for (j = 0; j < COL; j++)
    {
            scanf("%d",&arr[i][j]);
      }
    }
    printf("maze Structure\n");
    for (i = 0; i < ROW; i++)
    {
    for (j = 0; j < COL; j++)
    {
        printf("%d ",arr[i][j]);
    }
    printf("\n");
    }
    maze(arr,0,0,r1,c1,FWD);/*Rat starts at location (0,0)*/
      printf("No solution");
      getch();
  }
```

8 Develop a program to reverse a singly linked list in C [10]

Solution:

```
#include <stdio.h>
#include <stdlib.h>
/* Link list node */
struct Node {
    int data;
    struct Node* next;
};
```

```
{

/* Function to reverse the linked list */
static void reverse(struct Node** head_ref)
{
    struct Node* prev = NULL;
    struct Node* current = *head_ref;
    struct Node* next = NULL;
    while (current != NULL) {
        // Store next
        next = current->next;

        // Reverse current node's pointer
        current->next = prev;

        // Move pointers one position ahead.
        prev = current;
        current = next;
    }
    *head_ref = prev;
}

/* Function to push a node */
void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

/* Function to print linked list */
void printList(struct Node* head)
{
    struct Node* temp = head;
```

```
        while (temp != NULL) {
        printf("%d  ", temp->data);
        temp = temp->next;
    }
}

/* Driver program to test above function*/
int main()
{
    /* Start with the empty list */
    struct Node* head = NULL;

    push(&head, 20);
    push(&head, 4);
    push(&head, 15);
    push(&head, 85);

    printf("Given linked list\n");
    printList(head);
    reverse(&head);
    printf("\nReversed Linked list \n");
    printList(head);
    getchar();
}
```

Output:

```
Given linked list

85 15 4 20

Reversed Linked list

20 4 15 85
```