2.1.2 Write a program to find out factorial of number using stack.

```
#include <stdio.h>
int stack[50];
int top=-1;
void push(int val){
   stack[++top]= val;
}
void pop(){
  top--;
}
int peep(){
   if(top<0){
      return -1;
   }
   else{
      return stack[top];
}
}
int main() {
   int n;
   scanf("%d",&n);
   push(1);
   for(int i=2; i<=n;i++){
      push(peep()*i);
   }
   printf("factorial of %d is %d",n, peep());

   return 0;
}
```

2.1.3 Write a program to print string in reverse order using stack.

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

#define max 100

int top,stack[max];

void push(char x){

// Push(Inserting Element in stack) operation
if(top == max-1){
   printf("stack overflow");
} else {
   stack[++top]=x;
}
```

```
}

void pop(){
   // Pop (Removing element from stack)
     printf("%c",stack[top--]);
}


main()
{
  char str[]="sri lanka";
  int len = strlen(str);
  int i;

  for(i=0;i<len;i++)
      push(str[i]);

  for(i=0;i<len;i++)
    pop();
}
```

2.1.4 Write a Tower of Hanoi program in C using Recursion

```
#include <stdio.h>

void toH(int n, char rodA, char rodC, char rodB)
{
        if (n == 1)
        {
                printf("\n Move disk 1 from rod %c to rod %c",rodA ,rodC );
                return;
        }
        toH(n-1, rodA, rodB, rodC);
        printf("\n Move disk %d from rod %c to rod %c", n, rodA, rodC);
        toH(n-1, rodB, rodC,rodA);
}

int main()
{
        int no_of_disks ;
        printf("Enter number of disks: ");
        scanf("%d", &no_of_disks);
        toH(no_of_disks, 'A','C','B');
        return 0;
}
```

2.2.2 Write a program which performs following operations using circular queue.
Insert() -> delete() -> display()

```
#include <stdio.h>

#define MAX_SIZE 5

int queue[MAX_SIZE];

int front = -1, rear = -1;

int isFull() {
   return (rear + 1) % MAX_SIZE == front;
}

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

void enqueue(int data) {
   if (isFull()) {
      printf("Queue overflow\n");
      return;
   }
   if (front == -1) {
      front = 0;
   }
   rear = (rear + 1) % MAX_SIZE;
   queue[rear] = data;
   printf("Element %d inserted\n", data);
}

int dequeue() {
   if (isEmpty()) {
      printf("Queue underflow\n");
      return -1;
   }
   int data = queue[front];
   if (front == rear) {
      front = rear = -1;
   } else {
      front = (front + 1) % MAX_SIZE;
   }
   return data;
}

void display() {
   if (isEmpty()) {
```

```
        printf("Queue is empty\n");
        return;
    }
    printf("Queue elements: ");
    int i = front;
    while (i != rear) {
        printf("%d ", queue[i]);
        i = (i + 1) % MAX_SIZE;
    }
    printf("%d\n", queue[rear]);
}

int main() {
    enqueue(10);
    enqueue(20);
    enqueue(30);

    display();

    printf("Dequeued element: %d\n", dequeue());

    display();

    return 0;
}
```

1. Singly Linked List

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

struct node {
    int data;
    struct node *next;
};
struct node *start = NULL;

void create_ll();
void display();
void insert_beg();
void insert_end();
void insert_before();
void insert_after();
void delete_beg();
void delete_end();
void delete_node();
void search();
void count();
void sort();
void update();

void main() {
    int option;
    do {
            printf("\n\n *****MAIN MENU *****");
            printf("\n 1: Create a list");
            printf("\n 2: Display the list");
            printf("\n 3: Add a node at the beginning");
            printf("\n 4: Add a node at the end");
            printf("\n 5: Add a node before a given node");
            printf("\n 6: Add a node after a given node");
            printf("\n 7: Delete a node from the beginning");
            printf("\n 8: Delete a node from the end");
            printf("\n 9: Delete a given node");
            printf("\n 10: Search");
            printf("\n 11: Count");
            printf("\n 12: Sort");
            printf("\n 13: Update");
```

```
printf("\n 14: EXIT");
printf("\n\n Enter your option: ");
scanf("%d", &option);
switch(option) {
        case 1:
                create_ll();
                printf("\n LINKED LIST CREATED");
                break;
        case 2:
                display();
                break;
        case 3:
                insert_beg();
                break;
        case 4:
                insert_end();
                break;
        case 5:
                insert_before();
                break;
        case 6:
                insert_after();
                break;
        case 7:
                delete_beg();
                break;
        case 8:
                delete_end();
                break;
        case 9:
                delete_node();
                break;
        case 10:
                search();
                break;
        case 11:
                count();
                break;
        case 12:
                sort();
                break;
        case 13:
```

```
                                update();
                                break;
                        }
        } while(option != 14);
}

void create_ll() {
        struct node *new_node, *ptr;
        int num;
        printf("\n Enter -1 to end");
        printf("\n Enter the data: ");
        scanf("%d", &num);
        while(num != -1) {
                new_node = (struct node *)malloc(sizeof(struct node));
                new_node->data = num;
                if(start == NULL) {
                        new_node->next = NULL;
                        start = new_node;
                } else {
                        ptr = start;
                        while(ptr->next != NULL) {
                                ptr = ptr->next;
                        }
                        ptr->next = new_node;
                        new_node->next = NULL;
                }
                printf("\n Enter the data: ");
                scanf("%d", &num);
        }
}

void display() {
        struct node *ptr;
        ptr = start;
        while(ptr != NULL) {
                printf("\t %d", ptr->data);
                ptr = ptr->next;
        }
}

void insert_beg() {
        struct node *new_node;
```

```
    int num;
    printf("\n Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
}

void insert_end() {
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data: ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    ptr = start;
    while(ptr->next != NULL)
        ptr = ptr->next;
    ptr->next = new_node;
}

void insert_before() {
    struct node *new_node, *ptr, *preptr;
    int num, val, found = 0;
    printf("\n Enter the data: ");
    scanf("%d", &num);
    printf("\n Enter the value before which the data has to be inserted: ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    if(ptr->data == val) {
        insert_beg();
        found = 1;
    } else {
        while(ptr != NULL && ptr->data != val) {
            preptr = ptr;
            ptr = ptr->next;
        }
        if(ptr != NULL) {
```

```
                preptr->next = new_node;
                new_node->next = ptr;
                found = 1;
            }
        }
        if(!found) {
            printf("\n Value %d not found in the list.", val);
        }
}

void insert_after() {
        struct node *new_node, *ptr;
        int num, val, found = 0;
        printf("\n Enter the data: ");
        scanf("%d", &num);
        printf("\n Enter the value after which the data has to be inserted: ");
        scanf("%d", &val);
        new_node = (struct node *)malloc(sizeof(struct node));
        new_node->data = num;
        ptr = start;
        while(ptr != NULL && ptr->data != val) {
            ptr = ptr->next;
        }
        if(ptr != NULL) {
            new_node->next = ptr->next;
            ptr->next = new_node;
            found = 1;
        }
        if(!found) {
            printf("\n Value %d not found in the list.", val);
        }
}

void delete_beg() {
        struct node *ptr;
        ptr = start;
        start = start->next;
        free(ptr);
}

void delete_end() {
        struct node *ptr, *preptr;
```

```
	ptr = start;
	while(ptr->next != NULL) {
		preptr = ptr;
		ptr = ptr->next;
	}
	preptr->next = NULL;
	free(ptr);
}

void delete_node() {
	struct node *ptr, *preptr;
	int val, found = 0;
	printf("\n Enter the value of the node to be deleted: ");
	scanf("%d", &val);
	ptr = start;
	if(ptr->data == val) {
		delete_beg();
		found = 1;
	} else {
		while(ptr != NULL && ptr->data != val) {
			preptr = ptr;
			ptr = ptr->next;
		}
		if(ptr != NULL) {
			preptr->next = ptr->next;
			free(ptr);
			found = 1;
		}
	}
	if(!found) {
		printf("\n Value %d not found in the list.", val);
	}
}

void search() {
	int val, f = 0;
	struct node *ptr;
	printf("\n Enter Value: ");
	scanf("%d", &val);
	ptr = start;
	while(ptr != NULL) {
		if(ptr->data == val) {
```

```
                printf("\n Value Found: %d", ptr->data);
                f = 1;
                break;
            }
            ptr = ptr->next;
        }
        if(!f) {
            printf("\n Value is not found");
        }
}

void count() {
        struct node *ptr;
        int count = 0;
        ptr = start;
        while(ptr != NULL) {
            printf("\t %d", ptr->data);
            ptr = ptr->next;
            count++;
        }
        printf("\n Total Nodes: %d", count);
}

void sort() {
        struct node *ptr1, *ptr2;
        int temp;
        ptr1 = start;
        while(ptr1->next != NULL) {
            ptr2 = ptr1->next;
            while(ptr2 != NULL) {
                if(ptr1->data > ptr2->data) {
                    temp = ptr1->data;
                    ptr1->data = ptr2->data;
                    ptr2->data = temp;
                }
                ptr2 = ptr2->next;
            }
            ptr1 = ptr1->next;
        }
}

void update() {
```

```
    int idx, num, count = 1, found = 0;
    struct node *ptr;
    printf("\n Enter Index to be updated: ");
    scanf("%d", &idx);
    printf("\n Enter Updated Value: ");
    scanf("%d", &num);
    ptr = start;
    while(ptr != NULL) {
        if(count == idx) {
            ptr->data = num;
            found = 1;
            break;
        }
        ptr = ptr->next;
        count++;
    }
    if(!found) {
        printf("\n Index is not available");
    }
}
```

2. Singly Circular Linked List

```
//CIRCULAR SINGLY  LINKED LIST
#include <stdio.h>
#include <malloc.h>

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

struct node *start = NULL;

void create_cll();
void display();
void insert_beg();
void insert_end();
void insert_before();
void insert_after();
void delete_beg();
void delete_end();
void delete_node();

void main() {
   int option;
   do {
      printf("\n\n *****MAIN MENU *****");
      printf("\n 1: Create a list");
      printf("\n 2: Display the list");
      printf("\n 3: Add a node at the beginning");
      printf("\n 4: Add a node at the end");
      printf("\n 5: Add a node before a specific node");
      printf("\n 6: Add a node after a specific node");
      printf("\n 7: Delete a node from the beginning");
      printf("\n 8: Delete a node from the end");
      printf("\n 9: Delete a specific node");
      printf("\n 10: EXIT");
      printf("\n\n Enter your option : ");
      scanf("%d", &option);
      switch(option) {
         case 1:
            create_cll();
```

```
            printf("\n CIRCULAR LINKED LIST CREATED");
            break;
         case 2:
            display();
            break;
         case 3:
            insert_beg();
            break;
         case 4:
            insert_end();
            break;
         case 5:
            insert_before();
            break;
         case 6:
            insert_after();
            break;
         case 7:
            delete_beg();
            break;
         case 8:
            delete_end();
            break;
         case 9:
            delete_node();
            break;
      }
   } while(option != 10);
}

void create_cll() {
   struct node *new_node, *ptr;
   int num;
   printf("\n Enter -1 to end");
   printf("\n Enter the data : ");
   scanf("%d", &num);
   while(num != -1) {
      new_node = (struct node*)malloc(sizeof(struct node));
      new_node->data = num;
      if(start == NULL) {
         new_node->next = new_node;
         start = new_node;
```

```
        } else {
            ptr = start;
            while(ptr->next != start)
                ptr = ptr->next;
            ptr->next = new_node;
            new_node->next = start;
        }
        printf("\n Enter the data : ");
        scanf("%d", &num);
    }
}

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

void insert_beg() {
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    if(start == NULL) {
        new_node->next = new_node;
        start = new_node;
    } else {
        ptr = start;
        while(ptr->next != start)
            ptr = ptr->next;
        ptr->next = new_node;
        new_node->next = start;
```

```
        start = new_node;
    }
}

void insert_end() {
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    if(start == NULL) {
        new_node->next = new_node;
        start = new_node;
    } else {
        ptr = start;
        while(ptr->next != start)
            ptr = ptr->next;
        ptr->next = new_node;
        new_node->next = start;
    }
}

void insert_before() {
    struct node *new_node, *ptr, *preptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value before which the data has to be inserted : ");
    scanf("%d", &val);

    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;

    if(start == NULL) {
        printf("\n List is empty.");
        return;
    }

    if(ptr->data == val) {
        insert_beg();
```

```
    } else {
        preptr = NULL;
        while(ptr->next != start && ptr->data != val) {
            preptr = ptr;
            ptr = ptr->next;
        }

        if(ptr->data == val) {
            preptr->next = new_node;
            new_node->next = ptr;
        } else {
            printf("\n Value %d not found in the list.", val);
        }
    }
}

void insert_after() {
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value after which the data has to be inserted : ");
    scanf("%d", &val);

    new_node = (struct node*)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;

    if(start == NULL) {
        printf("\n List is empty.");
        return;
    }

    while(ptr->next != start && ptr->data != val) {
        ptr = ptr->next;
    }

    if(ptr->data == val) {
        new_node->next = ptr->next;
        ptr->next = new_node;
    } else {
        printf("\n Value %d not found in the list.", val);
```

```
    }
}

void delete_beg() {
    struct node *ptr;
    if(start == NULL) {
        printf("\n List is empty");
        return;
    }
    ptr = start;
    while(ptr->next != start)
        ptr = ptr->next;
    if(ptr == start) {
        free(start);
        start = NULL;
    } else {
        ptr->next = start->next;
        free(start);
        start = ptr->next;
    }
}

void delete_end() {
    struct node *ptr, *preptr;
    if(start == NULL) {
        printf("\n List is empty");
        return;
    }
    ptr = start;
    if(ptr->next == start) {
        free(start);
        start = NULL;
    } else {
        while(ptr->next != start) {
            preptr = ptr;
            ptr = ptr->next;
        }
        preptr->next = start;
        free(ptr);
    }
}
```

```
void delete_node() {
   struct node *ptr, *preptr;
   int val;
   printf("\n Enter the value of the node which has to be deleted : ");
   scanf("%d", &val);

   if(start == NULL) {
      printf("\n List is empty.");
      return;
   }

   ptr = start;

   if(ptr->data == val) {
      delete_beg();
   } else {
      preptr = NULL;
      while(ptr->next != start && ptr->data != val) {
         preptr = ptr;
         ptr = ptr->next;
      }

      if(ptr->data == val) {
         preptr->next = ptr->next;
         free(ptr);
      } else {
         printf("\n Value %d not found in the list.", val);
      }
   }
}
```

3. Doubly Linked List

```
//DOUBLY LINKED LIST
#include <stdio.h>
#include <malloc.h>

struct node {
   struct node *next;
   int data;
   struct node *prev;
};

struct node *start = NULL;

void create_ll();
void display();
void insert_beg();
void insert_end();
void insert_before();
void insert_after();
void delete_beg();
void delete_end();
void delete_specific();

void main() {
   int option;

   do {
      printf("\n\n *****MAIN MENU *****");
      printf("\n 1: Create a list");
      printf("\n 2: Display the list");
      printf("\n 3: Add a node at the beginning");
      printf("\n 4: Add a node at the end");
      printf("\n 5: Add a node before a given node");
      printf("\n 6: Add a node after a given node");
      printf("\n 7: Delete a node from the beginning");
      printf("\n 8: Delete a node from the end");
      printf("\n 9: Delete a specific node");
      printf("\n 10: EXIT");
      printf("\n\n Enter your option : ");
      scanf("%d", &option);
      switch(option) {
```

```
            case 1:
               create_ll();
               printf("\n DOUBLY LINKED LIST CREATED");
               break;
            case 2:
               display();
               break;
            case 3:
               insert_beg();
               break;
            case 4:
               insert_end();
               break;
            case 5:
               insert_before();
               break;
            case 6:
               insert_after();
               break;
            case 7:
               delete_beg();
               break;
            case 8:
               delete_end();
               break;
            case 9:
               delete_specific();
               break;
         }
      } while(option != 10);
}

void create_ll() {
   struct node *new_node, *ptr;
   int num;
   printf("\n Enter -1 to end");
   printf("\n Enter the data : ");
   scanf("%d", &num);
   while(num != -1) {
      if(start == NULL) {
         new_node = (struct node*)malloc(sizeof(struct node));
         new_node->prev = NULL;
```

```
            new_node->data = num;
            new_node->next = NULL;
            start = new_node;
         } else {
            ptr = start;
            new_node = (struct node*)malloc(sizeof(struct node));
            new_node->data = num;
            while(ptr->next != NULL)
               ptr = ptr->next;
            ptr->next = new_node;
            new_node->prev = ptr;
            new_node->next = NULL;
         }
         printf("\n Enter the data : ");
         scanf("%d", &num);
      }
   }

   void display() {
      struct node *ptr;
      if(start == NULL) {
         printf("\n List is empty.");
         return;
      }
      ptr = start;
      while(ptr != NULL) {
         printf("\t %d", ptr->data);
         ptr = ptr->next;
      }
   }

   void insert_beg() {
      struct node *new_node;
      int num;
      printf("\n Enter the data : ");
      scanf("%d", &num);
      new_node = (struct node *)malloc(sizeof(struct node));
      new_node->data = num;
      new_node->prev = NULL;
      if(start == NULL) {
         new_node->next = NULL;
         start = new_node;
```

```
    } else {
        new_node->next = start;
        start->prev = new_node;
        start = new_node;
    }
}

void insert_end() {
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    new_node->next = NULL;
    if(start == NULL) {
        new_node->prev = NULL;
        start = new_node;
    } else {
        ptr = start;
        while(ptr->next != NULL)
            ptr = ptr->next;
        ptr->next = new_node;
        new_node->prev = ptr;
    }
}

void insert_before() {
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value before which the data has to be inserted : ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while(ptr != NULL && ptr->data != val)
        ptr = ptr->next;
    if(ptr == NULL) {
        printf("\n Value not found.");
    } else if(ptr == start) {
```

```
        insert_beg();
    } else {
        new_node->next = ptr;
        new_node->prev = ptr->prev;
        ptr->prev->next = new_node;
        ptr->prev = new_node;
    }
}

void insert_after() {
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value after which the data has to be inserted : ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while(ptr != NULL && ptr->data != val)
        ptr = ptr->next;
    if(ptr == NULL) {
        printf("\n Value not found.");
    } else {
        new_node->prev = ptr;
        new_node->next = ptr->next;
        if(ptr->next != NULL)
            ptr->next->prev = new_node;
        ptr->next = new_node;
    }
}

void delete_beg() {
    struct node *ptr;
    if(start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    start = start->next;
    if(start != NULL)
        start->prev = NULL;
```

```
    free(ptr);
}

void delete_end() {
    struct node *ptr;
    if(start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    if(ptr->next == NULL) {
        start = NULL;
        free(ptr);
    } else {
        while(ptr->next != NULL)
            ptr = ptr->next;
        ptr->prev->next = NULL;
        free(ptr);
    }
}

void delete_specific() {
    struct node *ptr;
    int val;
    printf("\n Enter the value of the node to be deleted: ");
    scanf("%d", &val);
    if(start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    while(ptr != NULL && ptr->data != val)
        ptr = ptr->next;
    if(ptr == NULL) {
        printf("\n Node not found.");
    } else if(ptr == start) {
        delete_beg();
    } else if(ptr->next == NULL) {
        delete_end();
    } else {
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
```

```
        free(ptr);
    }
}
```

4. Circular Doubly Linked List

```
//CIRCULAR DOUBLY LINKED  LIST
#include <stdio.h>
#include <malloc.h>

struct node {
   struct node *next;
   int data;
   struct node *prev;
};

struct node *start = NULL;

void create_ll();
void display();
void insert_beg();
void insert_end();
void insert_before();
void insert_after();
void delete_beg();
void delete_end();
void delete_node();

void main() {
   int option;
   do {
      printf("\n\n *****MAIN MENU *****");
      printf("\n 1: Create a list");
      printf("\n 2: Display the list");
      printf("\n 3: Add a node at the beginning");
      printf("\n 4: Add a node at the end");
      printf("\n 5: Insert before a given node");
      printf("\n 6: Insert after a given node");
      printf("\n 7: Delete a node from the beginning");
      printf("\n 8: Delete a node from the end");
      printf("\n 9: Delete a given node");
      printf("\n 10: EXIT");
      printf("\n\n Enter your option : ");
      scanf("%d", &option);
      switch(option) {
         case 1:
```

```
            create_ll();
            printf("\n CIRCULAR DOUBLY LINKED LIST CREATED");
            break;
         case 2:
            display();
            break;
         case 3:
            insert_beg();
            break;
         case 4:
            insert_end();
            break;
         case 5:
            insert_before();
            break;
         case 6:
            insert_after();
            break;
         case 7:
            delete_beg();
            break;
         case 8:
            delete_end();
            break;
         case 9:
            delete_node();
            break;
      }
   } while(option != 10);
}

void create_ll() {
   struct node *new_node, *ptr;
   int num;
   printf("\n Enter -1 to end");
   printf("\n Enter the data : ");
   scanf("%d", &num);
   while(num != -1) {
      if(start == NULL) {
         new_node = (struct node*)malloc(sizeof(struct node));
         new_node->prev = NULL;
         new_node->data = num;
```

```
            start = new_node;
            new_node->next = start;
        } else {
            new_node = (struct node*)malloc(sizeof(struct node));
            new_node->data = num;
            ptr = start;
            while(ptr->next != start)
               ptr = ptr->next;
            new_node->prev = ptr;
            ptr->next = new_node;
            new_node->next = start;
            start->prev = new_node;
        }
        printf("\n Enter the data : ");
        scanf("%d", &num);
    }
}

void display() {
    struct node *ptr;
    if (start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    while(ptr->next != start) {
        printf("\t %d", ptr->data);
        ptr = ptr->next;
    }
    printf("\t %d", ptr->data);
}

void insert_beg() {
    struct node *new_node, *ptr;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while(ptr->next != start)
        ptr = ptr->next;
```

```
    new_node->prev = ptr;
    ptr->next = new_node;
    new_node->next = start;
    start->prev = new_node;
    start = new_node;
}

void insert_end() {
    struct node *ptr, *new_node;
    int num;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;
    while(ptr->next != start)
        ptr = ptr->next;
    ptr->next = new_node;
    new_node->prev = ptr;
    new_node->next = start;
    start->prev = new_node;
}

void insert_before() {
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value before which the data has to be inserted : ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;

    // Check for empty list
    if (start == NULL) {
        printf("\n List is empty.");
        free(new_node);
        return;
    }

    // Traverse the list to find the value
```

```
    while (ptr->data != val) {
        ptr = ptr->next;
        // If we circle back to the start, the value is not found
        if (ptr == start) {
            printf("\n Value not found.");
            free(new_node);
            return;
        }
    }

    // Insert the new node before the found node
    new_node->next = ptr;
    new_node->prev = ptr->prev;
    ptr->prev->next = new_node;
    ptr->prev = new_node;

    // If inserting before the start, update the start pointer
    if (ptr == start) {
        start = new_node;
    }
}

void insert_after() {
    struct node *new_node, *ptr;
    int num, val;
    printf("\n Enter the data : ");
    scanf("%d", &num);
    printf("\n Enter the value after which the data has to be inserted : ");
    scanf("%d", &val);
    new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = num;
    ptr = start;

    // Check for empty list
    if (start == NULL) {
        printf("\n List is empty.");
        free(new_node);
        return;
    }

    // Traverse the list to find the value
    while (ptr->data != val) {
```

```
        ptr = ptr->next;
        // If we circle back to the start, the value is not found
        if (ptr == start) {
            printf("\n Value not found.");
            free(new_node);
            return;
        }
    }

    // Insert the new node after the found node
    new_node->prev = ptr;
    new_node->next = ptr->next;
    if (ptr->next != NULL)
        ptr->next->prev = new_node;
    ptr->next = new_node;
}

void delete_beg() {
    struct node *ptr, *temp;
    if (start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
    ptr->next = start->next;
    temp = start;
    start = start->next;
    start->prev = ptr;
    free(temp);
}

void delete_end() {
    struct node *ptr;
    if (start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    while (ptr->next != start)
        ptr = ptr->next;
```

```
    ptr->prev->next = start;
    start->prev = ptr->prev;
    free(ptr);
}

void delete_node() {
    struct node *ptr;
    int val;
    printf("\n Enter the value of the node to be deleted: ");
    scanf("%d", &val);
    if (start == NULL) {
        printf("\n List is empty.");
        return;
    }
    ptr = start;
    while (ptr->data != val) {
        ptr = ptr->next;
        if (ptr == start) {
            printf("\n Value not found.");
            return;
        }
    }
    if (ptr == start) {
        delete_beg();
    } else {
        ptr->prev->next = ptr->next;
        ptr->next->prev = ptr->prev;
        free(ptr);
    }
}
```

5. Write a program to perform polynomial addition.

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

struct PolyNode {
   int coeff;
   int exp;
   struct PolyNode* next;
};

struct PolyNode* createNode(int coeff, int exp) {
   struct PolyNode* newNode = (struct PolyNode*)malloc(sizeof(struct PolyNode));
   newNode->coeff = coeff;
   newNode->exp = exp;
   newNode->next = NULL;
   return newNode;
}

struct PolyNode* addPolynomials(struct PolyNode* poly1, struct PolyNode* poly2) {
   struct PolyNode* result = NULL;
   struct PolyNode* last = NULL;

   while (poly1 != NULL && poly2 != NULL) {
      struct PolyNode* newNode;

      if (poly1->exp > poly2->exp) {
         newNode = createNode(poly1->coeff, poly1->exp);
         poly1 = poly1->next;
      } else if (poly1->exp < poly2->exp) {
         newNode = createNode(poly2->coeff, poly2->exp);
         poly2 = poly2->next;
      } else {
         newNode = createNode(poly1->coeff + poly2->coeff, poly1->exp);
         poly1 = poly1->next;
         poly2 = poly2->next;
      }

      if (result == NULL) {
         result = newNode;
      } else {
         last->next = newNode;
```

```
        }
        last = newNode;
    }

    while (poly1 != NULL) {
        struct PolyNode* newNode = createNode(poly1->coeff, poly1->exp);
        if (result == NULL) {
            result = newNode;
        } else {
            last->next = newNode;
        }
        last = newNode;
        poly1 = poly1->next;
    }

    while (poly2 != NULL) {
        struct PolyNode* newNode = createNode(poly2->coeff, poly2->exp);
        if (result == NULL) {
            result = newNode;
        } else {
            last->next = newNode;
        }
        last = newNode;
        poly2 = poly2->next;
    }

    return result;
}

void printPolynomial(struct PolyNode* poly) {
    while (poly != NULL) {
        printf("%dx^%d", poly->coeff, poly->exp);
        poly = poly->next;
        if (poly != NULL) {
            printf(" + ");
        }
    }
    printf("\n");
}

int main()
{
```

```
    struct PolyNode* poly1 = createNode(7, 2);
    poly1->next = createNode(4, 1);
    poly1->next->next = createNode(2, 0);

    struct PolyNode* poly2 = createNode(3, 2);
    poly2->next = createNode(2, 1);
    poly2->next->next = createNode(5, 0);


    printf("First Polynomial: ");
    printPolynomial(poly1);

    printf("Second Polynomial: ");
    printPolynomial(poly2);

    struct PolyNode* result = addPolynomials(poly1, poly2);

    printf("Resultant Polynomial: ");
    printPolynomial(result);

    return 0;
}
```

6. Write a program to check the given matrix is sparse or not.

```
#include <stdio.h>

int main()
{
   int rows, cols,matrix[50][50] ;
   int zeroCount = 0;

   printf("Enter the number of rows and columns of the matrix: ");
   scanf("%d %d", &rows, &cols);

   printf("Enter the elements of the matrix:\n");
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         scanf("%d", &matrix[i][j]);
         if (matrix[i][j] == 0) {
            zeroCount++;
         }
      }
   }

   int totalElements = rows * cols;

   if (zeroCount > totalElements / 2) {
      printf("The matrix is sparse.\n");
   } else {
      printf("The matrix is not sparse.\n");
   }

   return 0;
}
```