// Factorial Using recurssion

```
#include<stdio.h>

int factorial(int a)
{
   if(a==0 || a==1)
    return 1;

   return a*factorial(a-1);
}


int main()
{
   int a;
   printf("Enter Number to find factorial\n");
   scanf("%d",&a);

   printf("Factorial of given number is %d",factorial(a));
   return 0;
}
```

// Fibonacci Series Using Recurssion

```
#include<stdio.h>

int fibonacci(int a)
{
   if(a<=0)
   return 0;
   else if(a==1)
   return 1;
   else
   return fibonacci(a-1)+fibonacci(a-2);
}

int main()
{
   int a,i;
   printf("Enter Number To Find Fibonacci Series\n");
   scanf("%d",&a);
   fibonacci(a);
   printf("Fibonacci series:\n");
   for(i=0; i<a; i++)
   {
   printf("%d\t",fibonacci(i));
   }
   return 0;
}
```

// TOH

```
#include <stdio.h>

int moves = 0;
void TowerOfHanoi(int disk_num, char source, char temptower, char destination) {
  if (disk_num == 1) {
    printf("Move disk 1 from %c to %c\n", source, destination);
    moves++;
    return;
  }

  TowerOfHanoi(disk_num - 1, source, destination, temptower);
  printf("Move disk %d from %c to %c\n", disk_num, source, destination);
  moves++;
  TowerOfHanoi(disk_num - 1, temptower, source, destination);
}
int main() {
  int disk_num;
  printf("Enter Number Of Disks\n");
  scanf("%d", &disk_num);

  TowerOfHanoi(disk_num, 'A', 'B', 'C');
  printf("\nTotal Number Of Moves:%d\n", moves);
  return 0;
}
```

// Single reference pointer

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

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

void Traversal() {
  int node_no = 1;
  q = start; // Initialize q with start
  if (q == NULL)
    printf("Link List Is EMPTY\n");
  while (q != NULL) {
    printf("\nData of node %d is %d", node_no, q->data);
    q = q->next;
    node_no++;
```

```
  }
}

void create_at_start() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    p->next = NULL;
  } else {
    p->next = start;
    start = p;
  }
  printf("Node is Succesfully Created\n");
}

void create_at_end() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  p->next = NULL;
  if (start == NULL) {
    start = p;
  }

  else {
    q = start;
    while (q->next != NULL) {
      q = q->next;
    }
    q->next = p;
    // update current last node with adding p node
  }
  printf("Node is Succesfully Created\n");
}

void create_at_location() {
  int location, i = 1;
  p = (struct node *)malloc(sizeof(struct node));
  printf("Enter Location to create Node\n");
  scanf("%d", &location);
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  // At Start if Start==NuLL
```

```
  if (start == NULL) {
    if (location == 1) {
      start = p;
      p->next = NULL;
      printf("Node is succesfully created at start\n");
    } else {
      printf("Link List Is Empty Enter Vallid Location\n");
      free(p);
    }
  } else if (location == 1) // create at start if link list have some elements
  {
    p->next = start;
    start = p;
    printf("Node is succesfully created at start\n");

  } else // Create at any location (except start) at the end also
  {
    q = start;
    while (i < location - 1 && q != NULL) {
      q = q->next;
      i++;
    }
    if (q != NULL) {
      p->next = q->next;
      q->next = p;
      printf("Node is succesfully created at location %d\n", location);
    }

    else {
      printf("Enter Vallid Location\n");
      free(p);
    }
  }
}

void delete_at_start() {
  if (start == NULL)
    printf("Delete can not be perfprmed,Link List is empty\n");
  else {
    p = start;
    start = p->next;
    printf("%d is deleted", p->data);
    free(p);
  }
}

void delete_at_end() {
  // LL is empty
```

```
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having one element
  else if (start->next == NULL) {
    p = start;
    start = NULL;
    printf("%d is deleted at end\n", p->data);
    free(p);
  }
  // Link List having more than one element
  else {
    p = start;
    while (p->next != NULL) {
      q = p;
      p = p->next;
    }
    q->next = NULL;
    printf("%d is deleted\n", p->data);
    free(p);
  }
}

void delete_at_location() {
  int location, i;
  printf("Which Location's node fo you want to delete\n");
  scanf("%d", &location);

  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having only one element
  else if (location == 1) {
    p = start;
    start = NULL;
    printf("%d is deleted at location %d\n", p->data, location);
    free(p);
  }
  // LL having more than one element (logic work at end also)
  else if (location > 1) {
    i = 1;
    p = start;
    // setting p at given location
    while (i < location && p != NULL) {
      q = p;
      p = p->next;
      i++;
    }
    // deleting node
```

```
    if (p != NULL) {
      q->next = p->next;
      printf("%d is deleted at location %d\n", p->data, location);
      free(p);
    }
    else {
    printf("Enter Vallid Location");
  }
  }
}

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

  // Allocate memory for nodes in link list
  first = (struct node *)malloc(sizeof(struct node));
  second = (struct node *)malloc(sizeof(struct node));
  third = (struct node *)malloc(sizeof(struct node));

  // linking nodes
  start = first;
  first->data = 1;
  first->next = second;

  second->data = 2;
  second->next = third;

  third->data = 3;
  third->next = NULL;

  // Choices for user
  do {
    printf("\nEnter Choice:\n 1.Create At Start\n 2.Create At End \n 3.Create "
          "At Given Location\n 4.Delete At Start\n 5.Delete At End \n "
          "6.Delete At Given Location\n 7.Traverse\n 8.Exit\n");
    scanf("%d", &i);

    switch (i) {
    case 1:
      create_at_start();
      break;
    case 2:
      create_at_end();
      break;
    case 3:
      create_at_location();
      break;
```

```
    case 4:
      delete_at_start();
      break;
    case 5:
      delete_at_end();
      break;
    case 6:
      delete_at_location();
      break;
    case 7:
      Traversal();
      break;
    case 8:
      break;
    default:
      printf("Invalid Input\n");
    }
  } while (i != 8);

  return 0;
}
```

// Double reference pointer

```
// Singly Link List using two external reference pointer
#include <stdio.h>
#include <stdlib.h>

struct node {
  int data;
  struct node *next;
};
struct node *start = NULL, *p, *q, *last = NULL;

void Traversal() {
  int node_no = 1;
  q = start; // Initialize q with start
  if (q == NULL)
    printf("Link List Is EMPTY\n");
  while (q != NULL) {
    printf("\nData of node %d is %d", node_no, q->data);
    q = q->next;
    node_no++;
  }
}
```

```
void create_at_start() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    last = p;
    p->next = NULL;
  } else {
    p->next = start;
    start = p;
  }
  printf("Node is Successfully Created\n");
}

void create_at_end() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  p->next = NULL;
  if (start == NULL) {
    start = p;
    last = p;
  } else {
    last->next = p;
    last = p;
    // update current last node with adding p node
  }
  printf("Node is Successfully Created\n");
}

void create_at_location() {
  int location, i = 1;
  p = (struct node *)malloc(sizeof(struct node));
  printf("Enter Location to create Node\n");
  scanf("%d", &location);
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  // At Start if Start==NuLL
  if (start == NULL) {
    if (location == 1) {
```

```
      start = p;
      p->next = NULL;
      last = p;
      printf("Node is successfully created at start\n");
    } else {
      printf("Link List Is Empty Enter Valid Location\n");
      free(p);
    }
  } else if (location == 1) // create at start if link list have some elements
  {
    p->next = start;
    start = p;
    printf("Node is successfully created at start\n");

  } else // Create at any location (except start) at the end also
  {
    q = start;
    while (i < location - 1 && q != NULL) {
      q = q->next;
      i++;
    }
    if (q != NULL) {
      p->next = q->next;
      q->next = p;
      printf("Node is successfully created at location %d\n", location);
    }

    else {
      printf("Enter Vallid Location\n");
      free(p);
    }
  }
}

void delete_at_start() {
  // Link List is empty
  if (start == NULL)
    printf("Delete can not be performed,Link List is empty\n");
  // Link list have only one element
  else if (start->next == NULL) {
    start = NULL;
    last = NULL;
    printf("%d is deleted\n", p->data);
  }
```

```
  // Link List have 2 or more elements
  else {
    p = start;
    start = p->next;
    printf("%d is deleted", p->data);
    free(p);
  }
}

void delete_at_end() {
  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having one element
  else if (start->next == NULL) {
    p = start;
    start = NULL;
    last = NULL;
    printf("%d is deleted at end\n", p->data);
    free(p);
  }
  // Link List having more than one element
  else {
    p = start;
    while (p->next != NULL) {
      q = p;
      p = p->next;
    }
    q->next = NULL;
    printf("%d is deleted\n", p->data);
    free(p);
    last =q; // update the last pointer when last node is deleted
  }
}

void delete_at_location() {
  int location, i;
  printf("Which Location's node fo you want to delete\n");
  scanf("%d", &location);

  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having only one element
```

```
  else if (location == 1) {
    p = start;
    start = NULL;
    last = NULL;
    printf("%d is deleted at location %d\n", p->data, location);
    free(p);
  }
  // LL having more than one element (logic work at end also)
  else if (location > 1) {
    i = 1;
    p = start;
    // setting p at given location
    while (i < location && p != NULL) {
      q = p;
      p = p->next;
      i++;
    }
    // deleting node
    if (p != NULL) {
      q->next = p->next;
      printf("%d is deleted at location %d\n", p->data, location);
      free(p);
    } else {
      printf("Enter Valid Location");
    }
  }
}

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

  // Allocate memory for nodes in link list
  first = (struct node *)malloc(sizeof(struct node));
  second = (struct node *)malloc(sizeof(struct node));
  third = (struct node *)malloc(sizeof(struct node));

  // linking nodes
  start = first;
  first->data = 1;
  first->next = second;

  second->data = 2;
  second->next = third;
```

```
third->data = 3;
third->next = NULL;

// Initialize last pointer to last node in LINK LIST
last = third;

// Choices for user
do {
  printf("\nEnter Choice:\n 1.Create At Start\n 2.Create At End \n 3.Create "
         "At Given Location\n 4.Delete At Start\n 5.Delete At End \n "
         "6.Delete At Given Location\n 7.Traverse\n 8.Exit\n");
  scanf("%d", &i);

  switch (i) {
  case 1:
    create_at_start();
    break;
  case 2:
    create_at_end();
    break;
  case 3:
    create_at_location();
    break;
  case 4:
    delete_at_start();
    break;
  case 5:
    delete_at_end();
    break;
  case 6:
    delete_at_location();
    break;
  case 7:
    Traversal();
    break;
  case 8:
    break;
  default:
    printf("Invalid Input\n");
    break;
  }
} while (i != 8);
```

```
  return 0;
}
```

// Doubly LL

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

struct node {
  int data;
  struct node *next;
  struct node *prev;
};
struct node *start = NULL, *p, *q, *last = NULL;

void Traversal() {
  int node_no = 1;
  q = start; // Initialize q with start
  // q = last; // to traverse in backward direction
  if (q == NULL)
    printf("Link List Is EMPTY\n");
  else {
    while (q != NULL) {
      printf("\nData of node %d is %d", node_no, q->data);
      q = q->next;
      // q = q->prev; // if you want to traverse in backward direction
      node_no++;
    }
  }
}

void create_at_start() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    p->next = NULL;
    p->prev = NULL;

  } else {
    p->next = start;
    p->prev = NULL;
    start->prev = p;
    start = p;
```

```
  }
  printf("Node is Succesfully Created\n");
}

void create_at_end() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    p->next = NULL;
    p->prev = NULL;
  }

  else {
    q = start;
    while (q->next != NULL) {
      q = q->next;
    }
    q->next = p;
    p->prev = q;
    p->next = NULL;
    last = p; // update the last node
    // update current last node with adding p node
  }
  printf("Node is Succesfully Created\n");
}

void create_at_location() {
  int location, i = 1;
  p = (struct node *)malloc(sizeof(struct node));
  printf("Enter Location to create Node\n");
  scanf("%d", &location);
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  // At Start if Start==NULL
  if (start == NULL) {
    if (location == 1) {
      start = p;
      p->next = NULL;
      p->prev = NULL;
      printf("Node is succesfully created at start\n");
```

```
    } else {
      printf("Link List Is Empty Enter Valid Location\n");
      free(p);
    }
  } else if (location == 1) // create at start if link list have some elements
  {
    p->next = start;
    p->prev = NULL;
    start = p;
    printf("Node is succesfully created at start\n");

  } else // Create at any location  (except start) ,at the end also.
  {
    q = start;
    while (i < location - 1 && q != NULL) {
      q = q->next;
      i++;
    }
    if (q != NULL) {
      p->next = q->next;
      if (q->next != NULL) {
        q->next->prev = p;
      } // Adjusted previous pointer of (q's next) to newly added node p.
      q->next = p;
      p->prev = q;
      printf("Node is succesfully created at location %d\n", location);
    }

    else {
      printf("Enter Vallid Location\n");
      free(p);
    }
  }
}

void delete_at_start() {
  if (start == NULL)
    printf("Delete can not be performed,Link List is empty\n");
  else if (start->next == NULL) {
    p = start;
    start = NULL;
    printf("%d is deleted", p->data);
    free(p);
  } else {
```

```
    p = start;
    start = p->next;
    start->prev = NULL;
    printf("%d is deleted", p->data);
    free(p);
  }
}

void delete_at_end() {
  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having one element
  else if (start->next == NULL) {
    p = start;
    start = NULL;
    printf("%d is deleted at end\n", p->data);
    free(p);
  }
  // Link List having more than one element
  else {
    p = start;
    while (p->next != NULL) {
      q = p;
      p = p->next;
    }
    q->next = NULL; // Removed last node from link list
    printf("%d is deleted\n", p->data);
    free(p);
  }
}

void delete_at_location() {
  int location, i;
  printf("Which Location's node fo you want to delete\n");
  scanf("%d", &location);

  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having only one element
  else if (location == 1) {
    p = start;
    start = NULL;
```

```
    printf("%d is deleted at location %d\n", p->data, location);
    free(p);
  }
  // LL having more than one element (logic work at end also)
  else if (location > 1) {
    i = 1;
    p = start;
    // setting p at given location
    while (i < location && p != NULL) {
      q = p;
      p = p->next;
      i++;
    }
    // deleting node
    if (p != NULL) {
      q->next = p->next;
      if (p->next !=
          NULL) // NULL cant point to prev so p->next taken otherwise p!= taken.
      {
        p->next->prev = q;
      }
      printf("%d is deleted at location %d\n", p->data, location);
      free(p);
    } else {
      printf("Enter Vallid Location");
    }
  }
}

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

  // Allocate memory for nodes in link list
  first = (struct node *)malloc(sizeof(struct node));
  second = (struct node *)malloc(sizeof(struct node));
  third = (struct node *)malloc(sizeof(struct node));

  // linking nodes
  start = first;
  last = third;
  first->data = 1;
  first->next = second;
  first->prev = NULL;
```

```
second->data = 2;
second->next = third;
second->prev = first;

third->data = 3;
third->next = NULL;
third->prev = second;

// Choices for user
do {
  printf("\nEnter Choice:\n 1.Create At Start\n 2.Create At End \n 3.Create "
         "At Given Location\n 4.Delete At Start\n 5.Delete At End \n "
         "6.Delete At Given Location\n 7.Traverse\n 8.Exit\n");
  scanf("%d", &i);

  switch (i) {
  case 1:
    create_at_start();
    break;
  case 2:
    create_at_end();
    break;
  case 3:
    create_at_location();
    break;
  case 4:
    delete_at_start();
    break;
  case 5:
    delete_at_end();
    break;
  case 6:
    delete_at_location();
    break;
  case 7:
    Traversal();
    break;
  case 8:
    return 0;
  }
} while (i != 8);

return 0;
```

```
}
```

//CircularLL

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

struct node {
  int data;
  struct node *next;
};
struct node *start = NULL, *p, *q, *last = NULL;

void Traversal() {
  int node_no = 1;
  q = start; // Initialize q with start
  if (q == NULL)
    printf("Link List Is EMPTY\n");
  else {
    do {
      printf("\nData of node %d is %d", node_no, q->data);
      q = q->next;
      node_no++;
    } while (q!= start);
  }
}

void create_at_start() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    p->next = p;
    last = start; // or //last=p;

  } else {
    p->next = start;
    start = p;
    last->next = start;
  }
  printf("Node is Succesfully Created\n");
}
```

```
void create_at_end() {
  p = (struct node *)malloc(sizeof(struct node));
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  if (start == NULL) {
    start = p;
    p->next = p;
    last = start;
  }

  else {
    last->next = p;
    p->next = start;
    last = p; // update the last node
    // update current last node with adding p node
  }
  printf("Node is Succesfully Created\n");
}

void create_at_location() {
  int location, i = 1;
  p = (struct node *)malloc(sizeof(struct node));
  printf("Enter Location to create Node\n");
  scanf("%d", &location);
  printf("\nEnter data of node to be created:");
  scanf("%d", &p->data);

  // At Start if Start==NULL
  if (start == NULL) {
    if (location == 1) {
      start = p;
      p->next = p;
      last = p; // or //last=start
      printf("Node is succesfully created at start\n");
    } else {
      printf("Link List Is Empty Enter Valid Location\n");
      free(p);
    }
  } else if (location == 1) // create at start if link list have some elements
  {
    p->next = start;
    start = p;
    last->next = p;
```

```
    printf("Node is succesfully created at start\n");

  } else // Create at any location  // Will not work at end and start.
  {
    q = start;
    while (i < location - 1 && q != last) {
      q = q->next;
      i++;
    }
    if (q != last) {
      p->next = q->next;
      q->next = p;
      printf("Node is succesfully created at location %d\n", location);
    }
    // to create at end
    else if (i == location - 1 && q == last) {
      last->next = p;
      last = p;
      p->next = start;
    } else {
      printf("Enter Vallid Location\n");
      free(p);
    }
  }
}

void delete_at_start() {
  if (start == NULL)
    printf("Delete can not be performed,Link List is empty\n");
  else if (start->next == start) {
    p = start;
    start = NULL;
    last = NULL;
    printf("%d is deleted", p->data);
    free(p);
  } else {
    p = start;
    start = p->next;
    last->next = start;
    printf("%d is deleted", p->data);
    free(p);
  }
}
```

```
void delete_at_end() {
  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having one element
  else if (start->next == start) {
    p = start;
    start = NULL;
    last = NULL;
    printf("%d is deleted at end\n", p->data);
    free(p);
  }
  // Link List having more than one element
  else {
    p = start;
    while (p->next != start) {
      q = p;
      p = p->next;
    }
    q->next = start; // Removed last node from link list
    printf("%d is deleted\n", p->data);
    free(p);
  }
}

void delete_at_location() {
  int location, i;
  printf("Which Location's node fo you want to delete\n");
  scanf("%d", &location);

  // LL is empty
  if (start == NULL)
    printf("Link List is empty delete cannot be performed\n");
  // LL having only one element
  else if (location == 1) {
    p = start;
    if (start->next == start) {
      start = NULL;
      last = NULL;
    } else {
      start = start->next;
      last->next = start;
    }
    printf("%d is deleted at location %d\n", p->data, location);
```

```
    free(p);
  }
  // LL having more than one element // will not work at end (last)
  else if (location > 1) {
    i = 1;
    p = start;
    // setting p at given location
    while (i < location && p != last) {
      q = p;
      p = p->next;
      i++;
    }
    // deleting node
    if (p != last) {
      q->next = p->next;
      printf("%d is deleted at location %d\n", p->data, location);
      free(p);
    }
    // Now we have already set our p at location and if location is at end that
    // means location=i;
    // deleting node
    else if (location == i) {
      p = last;
      q->next = start;
      last = q; // update the last pointer
      printf("%d is deleted at location %d\n", p->data, location);
      free(p);
    } else {
      printf("Enter Vallid Location");
    }
  }
}

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

  // Allocate memory for nodes in link list
  first = (struct node *)malloc(sizeof(struct node));
  second = (struct node *)malloc(sizeof(struct node));
  third = (struct node *)malloc(sizeof(struct node));

  // linking nodes
  start = first;
```

```
last = third;
first->data = 1;
first->next = second;

second->data = 2;
second->next = third;

third->data = 3;
third->next = first;

// Choices for user
do {
  printf("\nEnter Choice:\n 1.Create At Start\n 2.Create At End \n 3.Create "
         "At Given Location\n 4.Delete At Start\n 5.Delete At End \n "
         "6.Delete At Given Location\n 7.Traverse\n 8.Exit\n");
  scanf("%d", &i);

  switch (i) {
  case 1:
    create_at_start();
    break;
  case 2:
    create_at_end();
    break;
  case 3:
    create_at_location();
    break;
  case 4:
    delete_at_start();
    break;
  case 5:
    delete_at_end();
    break;
  case 6:
    delete_at_location();
    break;
  case 7:
    Traversal();
    break;
  case 8:
    return 0;
  }
} while (i != 8);
```

```
  return 0;
}
// merge LL
#include <stdio.h>
#include <stdlib.h>

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

void creatingLL(struct node **start, int data); // function declaration

// Function to display the linked list
void displayLL(struct node *start) {
  struct node *q = start;
  int node_no = 1;
  if (q == NULL)
    printf("Link List Is EMPTY\n");
  while (q != NULL) {
    printf("\nData of node %d is %d", node_no, q->data);
    q = q->next;
    node_no++;
  }
}

// Function to input data into linked lists
void Enter_data(struct node **start, int *no_nodes_in_LL) {
  int data;
  printf("\nEnter Number of Nodes To Be Created In Link List: ");
  scanf("%d", no_nodes_in_LL);
  // Invalid input
  if (*no_nodes_in_LL <= 0) {
    printf("Invalid input. Please enter a positive integer.\n");
    return;
  }

  for (int i = 0; i < *no_nodes_in_LL; i++) {
    printf("Enter data of the element %d: ", i + 1);
    scanf("%d", &data);
    creatingLL(start, data); // Pass the address of start to the function
  }
}
```

```
// Function to create a linked list
void creatingLL(struct node **start, int data) {
  struct node *p = (struct node *)malloc(sizeof(struct node));
  p->data = data;
  p->next = NULL;
  if (*start == NULL) {
    *start = p;
  } else {
    struct node *temp = *start;
    while (temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = p;
  }
}

// Function to merge two linked lists alternatively
struct node *merge(struct node *start1, struct node *start2) {
  struct node *p1 = start1;
  struct node *p2 = start2;
  struct node *start3 = NULL;
  struct node *p3 = NULL;

  while (p1 != NULL && p2 != NULL) {
    if (start3 == NULL) {
      start3 = p1;
      p3 = start3;
      p1 = p1->next;
      p3->next = p2;
      p2 = p2->next;
      p3 = p3->next;
    } else {
      p3->next = p1;
      p1 = p1->next;
      p3 = p3->next;
      p3->next = p2;
      p2 = p2->next;
      p3 = p3->next;
    }
  }

  if (p1 != NULL) {
    p3->next = p1;
  }
```

```
  if (p2 != NULL) {
    p3->next = p2;
  }

  return start3;
}

int main() {
  struct node *start1 = NULL, *start2 = NULL, *newstart = NULL;
  int no_nodes_in_LL1, no_nodes_in_LL2;

  Enter_data(&start1, &no_nodes_in_LL1);
  Enter_data(&start2, &no_nodes_in_LL2);

  printf("\nFirst Linked List:");
  displayLL(start1);
  printf("\n");

  printf("\nSecond Linked List:");
  displayLL(start2);
  printf("\n");

  newstart = merge(start1, start2);

  printf("\nMerged Linked List:");
  displayLL(newstart);
  printf("\n");

  return 0;
}
```

// Stack

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

int a[5],top=-1;

void push(){

        if(top==4)
        printf("\n Stack Overflow\n");
        else{
        top=top+1;
```

```
        printf("\nEnter %d element of stack :",top+1);
        scanf("%d",&a[top]);
        }
    }

void pop(){

        if(top==-1)
        printf("\n Stack Underflow\n");
        else{
        printf("\n%d is popped\n",a[top]);
        top=top-1;
        }
    }

void Traverse_Stack(){
        int i;
        if(top==-1)
        printf("\nStack Is Empty\n");
        else{
        for(i=0;i<=top;i++){
        printf("\nElement %d is %d",i+1,a[i]);
        }
        }
        printf("\n");
}

void Top_of_stack(){
        if(top==-1)
        printf("\nStack Is empty\n");
        else
        printf("Top of stack is %d",a[top]);
}

int main(){

        int choice;
        while(choice!=5){
        printf("\nEnter choice:\n 1.Traverse\n 2.Push\n 3.Pop\n 4.Top of stack\n 5.Exit\n");
        scanf("%d",&choice);

        switch(choice){
        case 1: Traverse_Stack();
         break;
```

```
        case 2: push();
         break;
        case 3: pop();
         break;
        case 4:Top_of_stack();
         break;
        case 5: //Exit
         break;
        default:
                printf("Enter Valid Input\n");
                break;
        }
        }
        return 0;
}
```

// MOVE EVEN NODES OF THE LINKED LIST AT THE END OF THE LINKED LIST IN REVERSE ORDER

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

struct Node {
  int data;
  struct Node *next;
} *q, *start = NULL, *start1 = NULL;

typedef struct Node NODE;

void traverse() { // TRAVERSE

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

void createatstart(int a) { // CREATE AT START
  NODE *func;
  func = (NODE *)malloc(sizeof(NODE));
  func->data = a;
  if (start1 == NULL) {
    start1 = func;
```

```
   func->next = NULL;
  } else {
   func->next = start1;
   start1 = func;
  }
}

int main() {
  int ch, i,node;
  NODE *b[60], *p;

  printf("BY WHICH WAY YOU WANT TO PERFORM??\n1.Predefined linked "
        "list\n2.Create new linked list\n");
  scanf("%d", &ch);
  b[0] = (NODE *)malloc(sizeof(NODE));
  switch (ch) {

  case 1:
   start = b[0];
   for (i = 0; i < 8; i++) {
    b[i + 1] = (NODE *)malloc(sizeof(NODE));
    b[i]->next = b[i + 1];
    b[i]->data = i * 10;
   }
   b[8]->data = 80;
   b[8]->next = NULL;
   break;

  case 2:
   printf("How many nodes do you want to create:\n");
   scanf("%d", &node);
   printf("Enter data elements:\n");
   b[0] = (NODE *)malloc(sizeof(NODE));
   start = b[0];
   for (i = 0; i < node - 1; i++) {
    b[i + 1] = (NODE *)malloc(sizeof(NODE));
    b[i]->next = b[i + 1];
    scanf("%d", &b[i]->data);
   }
   scanf("%d", &b[i]->data);
   b[i]->next = NULL;
   break;

  default:
```

```
   printf("ENTER VALID CHOICE!!");
  }

  printf("\nLINKED LIST CREATED SUCCESSFULLY\nYour linked list:\n");
  traverse();
  printf("\n\nEnter 1 to perform operation OR any other number to EXIT\n");
  scanf("%d", &ch);
  if (ch == 1) {
    q = start;
    while (q->next != NULL) {
      p = q->next;
      createatstart(p->data);
      q->next = p->next;
      free(p);
      if (q->next != NULL)
        q = q->next;
    }
    q->next = start1;
    printf("OPERATION PERFORMED SUCCESSFULLY.\nYour new linked list:\n");
    traverse();
  } else {
    printf("Exited from the code");
  }
  return 0;
}
```

// Find Link list in second link list

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

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

// Function to create a linked list
void creatingLL(struct node **start, int data) {
    struct node *p = (struct node *)malloc(sizeof(struct node));
    p->data = data;
    p->next = NULL;
    if (*start == NULL) {
        *start = p;
    } else {
        struct node *temp = *start;
```

```
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = p;
    }
}

// Function to input data into linked lists
void Enter_data(struct node **start, int *no_nodes_in_LL) {
    int data;
    printf("\nEnter Number of Nodes To Be Created In Link List: ");
    scanf("%d", no_nodes_in_LL);
    // Invalid input
    if (*no_nodes_in_LL <= 0) {
        printf("Invalid input. Please enter a positive integer.\n");
        return;
    }

    for (int i = 0; i < *no_nodes_in_LL; i++) {
        printf("Enter data of the element %d: ", i + 1);
        scanf("%d", &data);
        creatingLL(start, data); // Pass the address of start to the function
    }
}

int searchLL(struct node *LL1, struct node *LL2) {
    while (LL2 != NULL) {
        struct node *temp1 = LL1;
        struct node *temp2 = LL2;
        while (temp1 != NULL && temp2 != NULL && temp1->data == temp2->data) {
            temp1 = temp1->next;
            temp2 = temp2->next;
        }
        if (temp1 == NULL) {
            return 1; // Found
        }
        LL2 = LL2->next;
    }
    return 0; // Not found
}

// Function to display the linked list
void displayLL(struct node *start) {
    struct node *q = start;
```

```
    int node_no = 1;
    if (q == NULL)
        printf("Link List Is EMPTY\n");
    while (q != NULL) {
        printf("\nData of node %d is %d", node_no, q->data);
        q = q->next;
        node_no++;
    }
}

int main() {
    struct node *start1 = NULL, *start2 = NULL;
    int no_nodes_in_LL1, no_nodes_in_LL2;

    Enter_data(&start1, &no_nodes_in_LL1);
    Enter_data(&start2, &no_nodes_in_LL2);

    printf("\nFirst Linked List:");
    displayLL(start1);
    printf("\n");

    printf("\nSecond Linked List:");
    displayLL(start2);
    printf("\n");

    if (searchLL(start1, start2)) {
        printf("\nFirst Link List Found In second link List\n");
    } else {
        printf("\nFirst Link List Not Found In second link List\n");
    }

    return 0;
}
```

//Polish Notations

```
#include<stdio.h>

int a[50],top=-1;
int pop(){

        if(top==-1)
        printf("\n Stack Underflow\n");
        else{
        return a[top--];
        }
```

```
}

void push(int new_tos){

            if(top==49)
            printf("\n Stack Overflow\n");
            else{
            top=top+1;
            a[top]= new_tos;
            }
}


int main()
{   char postfix[50],operator;
    int result;
    printf("Enter Postfix Expression:\n");
    scanf("%s",postfix);

    for(int i=0;postfix[i]!='\0';i++) {
        if(postfix[i] >= '0' && postfix[i] <= '9') {
            push(postfix[i]-'0');
        }
        else {
            int b = pop();
            int a = pop();
            switch(postfix[i]){
                case '+' :
                    result=a+b;
                    break;
                case '-' :
                    result=a-b;
                    break;
                case '*' :
                    result=a*b;
                    break;
                case '/' :
                    result=a/b;
                    break;
            }
            push(result);
        }
    }
```

```
    result=pop();
    printf("Result: %d\n",result);
    return 0;
}
```