```
/* 1. malloc() */

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

int main()
{
     int i, n;
     printf("Enter the number of intergers: ");
     scanf("%d", &n);
     int* ptr = (int*)malloc(n * sizeof(int));
     if (ptr == NULL)
     {
          printf("Memory not available.");
          exit(1);
     }
     for (i = 0; i < n; i++)
     {
          printf("Enter an integer: ");
          scanf("%d", ptr + i);
     }
     for (i = 0; i < n; i++)
          printf("%d ", *(ptr + i));
     return 0;
}
```

```
Output:
Enter the number of intergers : 3
Enter an integer : 43
Enter an integer : 56
Enter an integer : 78
43 56 78
```

```c
/* 2. calloc() */

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

int main()
{
     int i, n;
     printf("Enter the number of intergers: ");
     scanf("%d", &n);
     int* ptr = (int*)calloc(n, sizeof(int));
     if (ptr == NULL)
     {
          printf("Memory not available.");
          exit(1);
     }
     for (i = 0; i < n; i++)
     {
          printf("Enter an integer: ");
          scanf("%d", ptr + i);
     }
     for (i = 0; i < n; i++)
          printf("%d ", *(ptr + i));
     return 0;
}
```

```
Output:
Enter the number of intergers : 3
Enter an integer : 43
Enter an integer : 56
Enter an integer : 78
43 56 78
```

```
/* 3. realloc() */

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

int main()
{
	int i, n;
	int* ptr = (int*)malloc(2 * sizeof(int));
	if (ptr == NULL)
	{
		printf("Memory not available.");
		exit(1);
	}
	printf("Enter the two numbers: ");
	for (i = 0; i < 2; i++)
		scanf("%d", ptr + i);
	/*Memory Allocation for two more integer*/
	ptr = (int*)realloc(ptr, 4 * sizeof(int));
	if (ptr == NULL)
	{
		printf("Memory not available.");
		exit(1);
	}
	printf("Enter the two numbers: ");
	for (i = 2; i < 4; i++)
		scanf("%d", ptr + i);
	for (i = 0; i < 4; i++)
		printf("%d ", *(ptr + i));
	return 0;
}
```

Output:

```
Enter the two numbers :
45
78
Enter the two numbers :
79
90
45 78 79 90
```

```
/* 4. free() */

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

int* input()
{
     int* ptr, i;
     ptr = (int*)malloc(5 * sizeof(int));
     printf("Enter 5 numbers: ");
     for (i = 0; i < 5; i++)
          scanf("%d", ptr + i);
     return ptr;
}
int main()
{
     int i, sum = 0;
     int* ptr = input();
     for (i = 0; i < 5; i++)
          sum = sum + *(ptr + i);
     printf("Sum is: %d", sum);
     free(ptr);
     ptr = NULL;
     return 0;
}
```

```
Output:
Enter 5 numbers : 5 6 7 8 9
Sum is : 35
```

```c
/*5. Passing Structure members as argument*/
/*Just like variable we can pass structure members as argument to a
function*/

#include <stdio.h>

struct student {
	char name[50];
	int age;
	int roll;
	float marks;
};
void print(char name[], int age, int roll, float marks) {
	printf("%s %d %d %.2f\n", name, age, roll, marks);
}
int main() {
	struct student s1 = { "Nick", 20, 40, 87.6 };
	print(s1.name, s1.age, s1.roll, s1.marks);
}

Output: Nick 20 40 87.60
```

```
/* 6. Call by Reference Using Structure */
/*Instead of passing the copies of the structure members, we can pass
their addresses or references*/

#include <stdio.h>
struct charset {
      char c;
      int i;
};
void keyvalue(char* c, int* i) {
      scanf("%c %d", c, i);
}
int main() {
      struct charset cs;
      keyvalue(&cs.c, &cs.i);
      printf("%c %d", cs.c, cs.i);
      return 0;
}
```

Output:

```
A 1
A 1
```

```c
/* 7. Passing the structure variable as an Argument*/

#include <stdio.h>
struct point {
      int x;
      int y;
};
void print(struct point p) {
      printf("%d %d\n", p.x, p.y);
}
int main() {
      struct point p1 = { 23, 45 };
      struct point p2 = { 56, 90 };
      print(p1);
      print(p2);
      return 0;
}
```

```
Output:
23 45
56 90
```

```c
/* 8.Passing a pointer to structure as an argument*/

#include <stdio.h>
struct point {
      int x;
      int y;
};
void print(struct point* ptr) {
      printf("%d %d\n", ptr->x, ptr->y);
}
int main()
{
      struct point p1 = { 34, 45 };
      struct point p2 = { 56, 90 };
      print(&p1);
      print(&p2);
      return 0;
}
```

```
Output:
34 45
56 90
```

```c
/*9.Returning structure variable from the function*/

#include <stdio.h>

struct point {
	int x;
	int y;
};

struct point edit(struct point p) {
	(p.x)++;
	p.y = p.y + 5;
	return p;
}

void print(struct point p) {
	printf("%d %d\n", p.x, p.y);
}

int main()
{
	struct point p1 = { 12, 23 };
	struct point p2 = { 21, 78 };
	p1 = edit(p1);
	p2 = edit(p2);
	print(p1);
	print(p2);
	return 0;
}
```

```
Output:
13 28
22 83
```

```c
/*10.Returning a pointer to structure from the function*/

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

struct point {
	int x;
	int y;
};

struct point* fun(int a, int b) {
	struct point* ptr = (struct point*)malloc(sizeof(struct point));
	ptr->x = a;
	ptr->y = b + 5;
	return ptr;
}

void print(struct point* ptr) {
	printf("%d %d\n", ptr->x, ptr->y);
}

int main()
{
	struct point* ptr1, * ptr2;
	ptr1 = fun(2, 3);
	ptr2 = fun(6, 9);
	print(ptr1);
	print(ptr2);

	free(ptr1);
	free(ptr2);
	return 0;
}
```

```
Output:
2 8
6 14
```

```
/* free means I am deallocation the memory. When we allocate the
memory at heap it is our responsibility to deallocate the memory */
```

```c
/*11.Passing Array to Structure as an Argument*/

#include <stdio.h>

struct point {
	int x;
	int y;
};

void print(struct point arr[]) {
	int i;
	for (i = 0; i < 2; i++)
		printf("%d %d\n", arr[i].x, arr[i].y);
}

int main()
{
	struct point arr[2] = { {1, 2}, {3, 4} };
	print(arr);
	return 0;
}
```

```
Output:
1 2
3 4
```

```
/* 12.Self Referential Strucutre */
/* Self Referential Structure are those structure in which one
or more pointers points to the structure in the same type*/

#include <stdio.h>
struct code {
      int i;
      char c;
      struct code* ptr;
};
int main()
{
      struct code var1;
      struct code var2;
      var1.i = 65;
      var1.c = 'A';
      var1.ptr = NULL;
      var2.i = 65;
      var2.c = 'B';
      var2.ptr = NULL;
      var1.ptr = &var2;
      printf("%d %c", var1.ptr->i, var1.ptr->c);
      return 0;
}

output:
65 B
After use the memory we should release the memory.
After freeing the memory ptr become the dangling pointer.
```

```c
/* 13.Creatin a node in C */

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

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

int main()
{
	struct node *head = NULL;
	head = (struct node*)malloc(sizeof(struct node));

	head->data = 45;
	head->link = NULL;

	printf("%d", head->data);
	return 0;
}
```

```
Output: 45
```

```c
/* 14.Creating a single linked list in C (Two nods) */

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

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

int main()
{
	struct node *head = malloc(sizeof(struct node));
	head->data = 45;
	head->link = NULL;

	struct node *current = malloc(sizeof(struct node));
	current->data = 90;
	current->link = NULL;

	head->link = current;
	printf("%d %d\n", head->data, current->data);
	return 0;
}


Output: 45 90
```

```c
/* 15. Creating a single linked list in C(three nodes) */
#include <stdio.h>
#include <stdlib.h>

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

int main()
{
	struct node* head = NULL;
	head = malloc(sizeof(struct node));
	head->data = 45;
	head->link = NULL;

	struct node* current = NULL;
	current = malloc(sizeof(struct node));
	current->data = 90;
	current->link = NULL;

	head->link = current;

	struct node* third = NULL;
	third = malloc(sizeof(struct node));
	third->data = 98;
	third->link = NULL;

	head->link->link = third;

	printf("%d %d %d\n", head->data, current->data, third->data);
	return 0;
}

Output: 45 90 98
```

```c
/* 16.Traversing a single linked list.
   Program to count the number of nods */

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

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

void num_of_nods(struct node *head)
{
      int count = 0;
      struct node *ptr = NULL;
      ptr = head;
      while (ptr != NULL)
      {
            count = count + 1;
            ptr = ptr->link;
      }
      printf("Number of nods are: %d\n", count);
}
```

```c
if(head == NULL)
 printf("Linked list is empty\n");
```

```c
int main()
{
      /* Imazine that head is the pointer of the first node of the linked list */
      struct node *head = NULL;
      head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      struct node *current = NULL;
      current = malloc(sizeof(struct node));
      current->data = 90;
      current->link = NULL;

      head->link = current;
      struct node *third = NULL;
      third = malloc(sizeof(struct node));
      third->data = 98;
      third->link = NULL;

      head->link->link = third;

      num_of_nods(head);
      return 0;
}
```

```
Output: Number of nods are : 3
```

```c
/* 17. Traversing a single linked list. Program to print the data */
#include <stdio.h>
#include <stdlib.h>

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

void print_data(struct node *head)
{
      if (head == NULL)
            printf("Linked list is empty\n");
      int count = 0;
      struct node* ptr = NULL;
      ptr = head;
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
}

int main()
{
      /* Imazine that head is the pointer of the first node of the linked list */
      struct node *head = NULL;
      head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      struct node *current = NULL;
      current = malloc(sizeof(struct node));
      current->data = 90;
      current->link = NULL;

      head->link = current;
      struct node *third = NULL;
      third = malloc(sizeof(struct node));
      third->data = 98;
      third->link = NULL;

      head->link->link = third;

      print_data(head);
      return 0;
}

output: 45 90 98
```

```
/* 18. Traversing a single linked list.
   Printing the size Using Array */

#include <stdio.h>

int main()
{
     int a[] = { 45, 98, 3 };
     int n;
     n = sizeof(a) / sizeof(int);
     printf("%d", n);
     return 0;
}

Output: 3



/* 19.Traversing a single linked list.
   Print the data Using Array */

#include <stdio.h>

int main()
{
     int a[] = { 45, 98, 3 };
     int n, i;
     n = sizeof(a) / sizeof(int);
     for(i =0; i<n; i++)
          printf("%d ", a[i]);
     return 0;
}

Output: 45 98 3
```

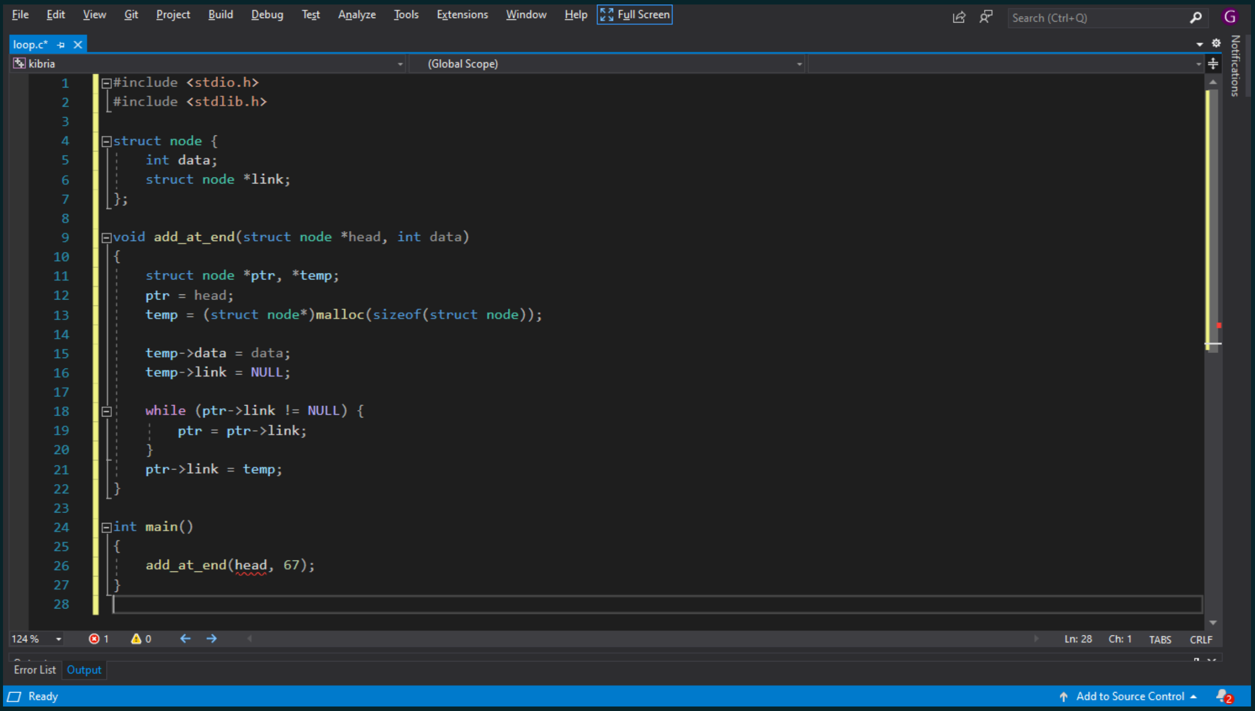
```
Time Complexity:

Linked List:
Counting the element: O(n)
printing the data: O(n)

Array:
Counting the element: O(1)
Printing the data: O(n)
```

22 নাম্বারটা করো। এটা দরকার নাই।

```c
/* 20.Single Linked List (Inserting a Node at the End) */

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

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

void add_at_end(struct node *head, int data)
{
	struct node *ptr, *temp;
	ptr = head;
	temp = (struct node*)malloc(sizeof(struct node));

	temp->data = data;
	temp->link = NULL;

	while (ptr->link != NULL) {
		ptr = ptr->link;
	}
	ptr->link = temp;
}

int main()
{
	add_at_end(head, 67);
}
```

22 নাম্বারটা করো। এটা দরকার নাই।

```
/* 21.Single Linked List (Inserting a Node at the End)
And find out time complexity */
```



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

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

void add_at_end(struct node *head, int data)
{
    struct node *ptr, *temp;
    ptr = head;
    temp = (struct node*)malloc(sizeof(struct node));

    temp->data = data;
    temp->link = NULL;

    while (ptr->link != NULL) {
        ptr = ptr->link;
    }
    ptr->link = temp;
}

int main()
{
    add_at_end(head, 67);
}
```

```
Time Complexity : O(n)


/*We are reaching each and every node of the list. If we have n nodes
this will takes n units of time. So, the time complexity is O(n)*/
```

```c
/*22.Singhle Linked List(Inserting a Node at the End)
And find out time complexity(Alternative Way)*/
#include <stdio.h>
#include <stdlib.h>

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

struct node *add_at_end(struct node *ptr, int data)
{
     struct node *temp = malloc(sizeof(struct node));

     temp->data = data;
     temp->link = NULL;

     ptr->link = temp;
     return temp;
}

int main()
{
     struct node *head = malloc(sizeof(struct node));
     head->data = 45;
     head->link = NULL;

     struct node *ptr = head;

     ptr = add_at_end(ptr, 90);
     ptr = add_at_end(ptr, 95);

     ptr = head;
     while (ptr != NULL)
     {
          printf("%d ", ptr->data);
          ptr = ptr->link;
     }
     return 0;
}
```

```
Output: 45 90 95
```

Time Complexity of add_at_end function is : O(1)

```
/* 23.Inserting a node at the end of a linked list
(When Array is Not Full) */

#include <stdio.h>

int add_at_end(int a[], int freepos, int data)
{
     a[freepos] = data;
     freepos++;
     return freepos;
}

int main()
{
     int a[10];
     int i, n, freepos;
     printf("Enter the number of element: ");
     scanf("%d", &n);
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);

     freepos = n;
     freepos = add_at_end(a, freepos, 65);

     for (i = 0; i < freepos; i++)
          printf("%d ", a[i]);
     return 0;
}

Output:
Enter the number of element : 3
1 2 3 65
1 2 3 65

Time complexity is : O(1)
```

```
/* 24.Inserting a node at the end of a linked list
(When Array is Full) */

#include <stdio.h>

int add_at_end(int a[], int b[], int n, int freepos, int data)
{
      /*We need to copy the element from a[] to b[].
      Because we know that a[] is already full*/
      int i;
      for (i = 0; i < n; i++)
            b[i] = a[i];
      b[freepos] = data;
      freepos++;
      return freepos;
}

int main()
{
      int a[3];
      int i, n, freepos;
      printf("Enter the number of element: ");
      scanf("%d", &n);
      for (i = 0; i < n; i++)
            scanf("%d", &a[i]);

      int size = sizeof(a) / sizeof(a[0]); /*int size=3*/
      freepos = n;

      if (n == size) /*n==size means array is full*/
      {
            int b[size + 2];
            freepos = add_at_end(a, b, size, freepos, 65);
            for (i = 0; i < freepos; i++)
                  printf("%d ", b[i]);
      }
      return 0;
}

Output:
Enter the number of element : 3
1 2 3
1 2 3 65
Time Complexity : O(n)
```

```c
/* 25.Adding the node at the beginning of the list */
#include <stdio.h>

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

struct node* add_beg(struct node* head, int d)
{
	struct node *ptr = malloc(sizeof(struct node));
	ptr->data = d;
	ptr->link = NULL;

	ptr->link = head;
	head = ptr;
	return head;
}

int main()
{
	struct node *head = malloc(sizeof(struct node));
	head->data = 45;
	head->link = NULL;

	struct node *ptr = malloc(sizeof(struct node));
	ptr->data = 90;
	ptr->link = NULL;

	head->link = ptr;

	int data = 3;

	head = add_beg(head, data);
	ptr = head;
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->link;
	}
	return 0;
}
```

```
Output: 3 45 90
```

এটা না করলেও হবে।

25 number টা করলেই হবে।

```
/*26.Adding the node at the beginning of the list (Alternative Way)*/

#include <stdio.h>

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

struct node* add_beg(struct node **head, int d)
{
	struct node *ptr = malloc(sizeof(struct node));
	ptr->data = d;
	ptr->link = NULL;

	ptr->link = *head;
	*head = ptr;
	return *head;
}

int main()
{
	struct node *head = malloc(sizeof(struct node));
	head->data = 45;
	head->link = NULL;

	struct node *ptr = malloc(sizeof(struct node));
	ptr->data = 90;
	ptr->link = NULL;

	head->link = ptr;

	int data = 3;

	add_beg(&head, data);
	ptr = head;
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->link;
	}
	return 0;
}
```

```
Output: 3 45 90
```

```c
/* 27.Adding the node at the beginning of the list
using array. (When Array is not full) */

#include <stdio.h>

int add_beg(int a[], int data, int n)
{
     int i;
     for (i = n - 1; i >= 0; i--)
     {
          a[i + 1] = a[i];
     }
     a[0] = data;
     return n + 1;
}

int main()
{
     int a[10], data = 20, i, n;
     printf("Enter the number of element? ");
     scanf("%d", &n);
     printf("Enter the element: ");
     for (i = 0; i < n; i++)
          scanf("%d", &a[i]);

     int x = add_beg(a, data, n);

     for (i = 0; i < x; i++)
          printf("%d ", a[i]);

     return 0;
}
```

```
Output:
Enter the number of element ? 4
Enter the element : 1 2 3 4
20 1 2 3 4
```

```
/* If there are n element in an array then n shifts are required
So, Time complexity: O(n) */
```

```c
/* 28.Inserting a node at a certain position */

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

struct node {
	int data;
	struct node *link;
};
void add_at_end(struct node* head, int data)
{
	struct node* ptr, * temp;
	ptr = head;
	temp = (struct node*)malloc(sizeof(struct node));

	temp->data = data;
	temp->link = NULL;

	while (ptr->link != NULL) {
		ptr = ptr->link;
	}
	ptr->link = temp;
}

void add_at_pos(struct node *head, int data, int pos)
{
	struct node *ptr = head;
	struct node *ptr2 = malloc(sizeof(struct node));
	ptr2->data = data;
	ptr2->link = NULL;

	pos--;
	while (pos != 1)
	{
		ptr = ptr->link;
		pos--;
	}
	ptr2->link = ptr->link;
	ptr->link = ptr2;
}
```

```
int main()
{
      struct node *head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      add_at_end(head, 98);
      add_at_end(head, 3);

      int data = 67, position = 3;

      add_at_pos(head, data, position);
      struct node *ptr = head;

      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
      return 0;
}
```

```
Output: 45 98 67 3
```

Time Complexity: Adding node at certain position using linked list

Time complexity is O(n)

Cause if there 2 nodes in the list and we have to add a new node at 3rd

postion then while loop run only 1 times.

So, if n nodes then the loop run (n-1) times.

We take the upper bound that is n so the time complexity is O(n).

```c
/* 29.Inserting node at certain position using array */

#include <stdio.h>

void add_at_pos(int arr[], int arr2[], int n, int data, int pos)
{
      int i;
      int index = pos - 1;
      for (i = 0; i <= index-1; i++)
            arr2[i] = arr[i];

      arr2[index] = data;
      int j;
      for (i = index + 1, j = index; i < n + 1, j < n; i++, j++)
            arr2[i] = arr[j];
}

int main()
{
      int arr[] = { 2, 34, 21, 6, 7, 43, 32, 67, 65, 54 };
      int pos = 5, data = 78, i;
      int size = sizeof(arr) / sizeof(arr[0]);

      int arr2[size + 1];
      add_at_pos(arr, arr2, size, data, pos);

      for (i = 0; i < size + 1; i++)
            printf("%d ", arr2[i]);
      return 0;
}
```

```
Output: 2 34 21 6 78 7 43 32 67 65 54
Time Complexity: O(n)
```

Remember:
>>We can not return an array from a function because an array created int the function in local to that function.
>>This is why arr2[] has been passed as a parameter to a function.
>>Passing arr2[](name of the array) is equivalent to passing the address of the first block of arr2. So the changes made in called function will be reflected in the caller function.

```c
/* 30.Deleting the first node */
#include <stdio.h>
#include <stdlib.h>

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

void print_data(struct node* head)
{
	if (head == NULL)
		printf("Linked list is empty\n");
	int count = 0;
	struct node* ptr = NULL;
	ptr = head;
	printf("Before deleting first node: ");
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->link;
	}
}

struct node* del_first(struct node* head)
{
	if (head == NULL)
		printf("List is already empty\n");
	else
	{
		struct node* temp = head;
		head = head->link;
		free(temp);
		temp = NULL;
	}
	return head;
}
```

```
int main()
{
      /* Imazine that head is the pointer of the first node of the linked list */
      struct node* head = NULL;
      head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      struct node* current = NULL;
      current = malloc(sizeof(struct node));
      current->data = 90;
      current->link = NULL;

      head->link = current;
      struct node* third = NULL;
      third = malloc(sizeof(struct node));
      third->data = 98;
      third->link = NULL;

      head->link->link = third;

      print_data(head);

      struct node* ptr;
      head = del_first(head);
      ptr = head;
      printf("\nAfter deleting first node: ");
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
      return 0;
}

Output:
Before deleting first node : 45 90 98
After deleting first node : 90 98
```

```
/* 31.Deleting the last node of the single linked list */

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

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

void print_data(struct node *head)
{
	if (head == NULL)
		printf("Linked list is empty\n");
	int count = 0;
	struct node *ptr = NULL;
	ptr = head;
	printf("Before deleting last node: ");
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->link;
	}
}

struct node *del_last(struct node *head)
{
	/* Checking if there is one node in the linked list */
	if (head == NULL)
		printf("List is already empty\n");
	else if (head->link == NULL)
	{
		free(head);
		head = NULL;
	}
	else
	{
		struct node *temp = head;
		struct node *temp2 = head;
		while (temp->link != NULL)
		{
			temp2 = temp;
			temp = temp->link;
		}
		temp2->link = NULL;
		free(temp);
		temp = NULL;
	}
	return head;
	/* There is no need return the head pointer as we are not updating it in the
funtion */
}
```

```
int main()
{
      /* Imazine that head is the pointer of the first node of the linked list */
      struct node *head = NULL;
      head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      struct node *current = NULL;
      current = malloc(sizeof(struct node));
      current->data = 90;
      current->link = NULL;

      head->link = current;
      struct node *third = NULL;
      third = malloc(sizeof(struct node));
      third->data = 98;
      third->link = NULL;

      head->link->link = third;

      print_data(head);

      struct node *ptr;
      head = del_last(head);
      ptr = head;
      printf("\nAfter deleting last node: ");
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
      return 0;
}
```

Output:

```
Before deleting last node : 45 90 98
After deleting last node : 45 90
```

```
/* 32. Deleting the last node of the single linked list using pointer
*/

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

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

/* void *del_last(struct node *head) */
struct node *del_last(struct node *head)
{
	/* Checking if there is one node in the linked list */
	if (head == NULL)
		printf("List is already empty\n");
	else if (head->link == NULL)
	{
		free(head);
		head = NULL;
	}
	else
	{
		struct node *temp = head;
		/* Time complexity of this while loop is O(n)*/
		while (temp->link->link != NULL)
		{
			temp = temp->link;
		}
		free(temp->link);
		temp->link = NULL;
	}
	return head;
	/* There is no need return the head pointer as we are not
updating it in the funtion */
}
```

```c
int main()
{
      struct node *head;
      struct node *ptr;
      head = malloc(sizeof(struct node));

      head = del_last(head);
      ptr = head;
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
      return 0;
}
```

```c
/* 33. Deleting at the end (Link list VS Array) */

#include <stdio.h>

int main()
{
	int a[] = { 12, 23, 34, 45, 56, 54, 43, 32 };
	int size = sizeof(a) / sizeof(a[0]);
	int i;

	for (i = 0; i < size - 1; i++)
		printf("%d ", a[i]);
	return 0;
}
```

```
Output: 12 23 34 45 56 54 43
Time complexity is O(1)
```

```c
/* 34.Deleting the first node using array and find out time complexity */
#include <stdio.h>

int del_first(int a[], int n)
{
	int i;
	if (n == 0)
	{
		printf("Array is empty\n");
		return n;
	}
	for (i = 0; i < n - 1; i++)
		a[i] = a[i + 1];
	return n - 1;
}

int main()
{
	int a[10];
	int n, i;
	printf("Enter the number of elements: ");
	scanf("%d", &n);
	for (i = 0; i < n; i++)
		scanf("%d", &a[i]);
	n = del_first(a, n);
	printf("After deleting first node: ");
	for (i = 0; i < n; i++)
		printf("%d ", a[i]);
	return 0;
}
```

Output:

```
Enter the number of elements : 5
12 23 34 45 56
After deleting first node : 23 34 45 56
```

Time Complexity:

Deleting the first node using linked list = O(1)
Deleting the first node using array = O(n)

```c
/* 35.Deleting the node at particular position */
#include <stdio.h>
#include <stdlib.h>

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

void del_pos(struct node** head, int position)
{
	struct node* current = *head;
	struct node* previous = *head;
	if (*head == NULL)
		printf("List is already empty\n");
	else if (position == 1)
	{
		*head = current->link;
		free(current);
		current = NULL;
	}
	else
	{
		while (position != 1)
		{
			previous = current;
			current = current->link;
			position--;
		}
		previous->link = current->link;
		free(current);
		current = NULL;
	}
}
```

```
int main()
{
      struct node* head = NULL;
      head = malloc(sizeof(struct node));
      head->data = 45;
      head->link = NULL;

      struct node* current = NULL;
      current = malloc(sizeof(struct node));
      current->data = 90;
      current->link = NULL;

      head->link = current;

      struct node* third = NULL;
      third = malloc(sizeof(struct node));
      third->data = 98;
      third->link = NULL;

      head->link->link = third;
      printf("Before Deleting: ");
      printf("%d %d %d\n", head->data, current->data, third->data);
      struct node *ptr;
      int position = 2;
      del_pos(&head, position);
      ptr = head;
      printf("After deleting: ");
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->link;
      }
      return 0;
}
```

```
Output:
Before Deleting : 45 90 98
After deleting : 45 98
```

```c
/* 36.Deleting a whole single linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* del_list(struct node* head)
{
	struct node* temp = head;

	while (temp != NULL)
	{
		temp = temp->link;
		free(head);
		head = temp;
	}
	return head;
}

int main()
{
	struct node* head = NULL;
	head = malloc(sizeof(struct node));
	head->data = 45;
	head->link = NULL;

	struct node* current = NULL;
	current = malloc(sizeof(struct node));
	current->data = 90;
	current->link = NULL;

	head->link = current;

	struct node* third = NULL;
	third = malloc(sizeof(struct node));
	third->data = 98;
	third->link = NULL;

	head->link->link = third;
	printf("Before Deleting: ");
	printf("%d %d %d\n", head->data, current->data, third->data);

	head = del_list(head);
	if (head == NULL)
		printf("Linked is deleted successfully\n");
	return 0;
}
```

```
Output:
Before Deleting : 45 90 98
Linked is deleted successfully
```

```c
/* 37.Reverse a single linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *reverse(struct node *head)
{
       struct node *prev = NULL;
       struct node *next = NULL;
       while (head != NULL)
       {
              next = head->link;
              head->link = prev;
              prev = head;
              head = next;
       }
       head = prev;
       return head;
}

int main()
{
       struct node *head = NULL;
       head = malloc(sizeof(struct node));
       head->data = 45;
       head->link = NULL;

       struct node *current = NULL;
       current = malloc(sizeof(struct node));
       current->data = 90;
       current->link = NULL;

       head->link = current;

       struct node *third = NULL;
       third = malloc(sizeof(struct node));
       third->data = 98;
       third->link = NULL;

       head->link->link = third;
       printf("Before Reversing: ");
       printf("%d %d %d\n", head->data, current->data, third->data);

       printf("After Reversing: ");
       struct node* ptr;
       head = reverse(head);
       ptr = head;
       while (ptr != NULL)
       {
              printf("%d ", ptr->data);
              ptr = ptr->link;
       }
       return 0;
}
```

```
Output:
Before Reversing : 45 90 98
After Reversing : 98 90 45
```

Time complexity:
- Insertion at the front: o(1); Constant amount of time.
- Insertion at the end: o(1)
- Deleting at the front node: o(1)
- Interchanging the first two element of the linked list: O(1)

- Deleting at the last node: o(n)

Because, in order to delete the last node of a single linked list we need to address of the second last node. We only use a pointer to the last node. Therefore, we need to traverse the list in order to get the address of the second last node of the list.

➢ Let p be a singly linked list. Let q be a pointer to an intermediate node x in the list. What is the worst = case time complexity of the best-known algorithm to delete the node x from the list?

Answer: O(1)

➢ Moving the last element to the front of the list.

```
q->next = NULL;
p->next = head;
head = p;
```

```c
/* 38. Creating a Node */

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

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

int main()
{
	struct node *head = malloc(sizeof(struct node));
	head->prev = NULL;
	head->data = 10;
	head->next = NULL;

	printf("%d", head->data);
	return 0;
}

Output: 10



/*
A doubly linked list is different from a singly linked list in a way
that each node has a extra pointer that pointes to the previous node,
together with the next pointer and data similar to singly linked list
*/
```

```c
/* 39. Insertion in an Empty List */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

int main()
{
	struct node* head = NULL;
	head = addToEmpty(head, 45);

	printf("%d", head->data);
	return 0;
}
```

```
Output: 45
```

```
/* 40. Insertion at the beginning of a doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtBeg(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;
	temp->next = head;
	head->prev = temp;

	head = temp;
	return head;
}

int main()
{
	struct node* head = NULL;
	struct node* ptr;
	head = addToEmpty(head, 45);
	head = addAtBeg(head, 25);

	ptr = head;
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->next;
	}
	return 0;
}

Output: 25 45
```

```c
/* 41. Insertion at the end of a doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtBeg(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;
	temp->next = head;
	head->prev = temp;

	head = temp;
	return head;
}

struct node* addAtEnd(struct node* head, int data)
{
	struct node *temp, *tp;
	temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	tp = head;
	while (tp->next != NULL)
	{
		tp = tp->next;
	}
	tp->next = temp;
	temp->prev = tp;
	return head;
}
```

```
int main()
{
      struct node* head = NULL;
      struct node* ptr;
      head = addToEmpty(head, 45);
      head = addAtBeg(head, 25);
      head = addAtEnd(head, 90);

      ptr = head;
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->next;
      }
      return 0;
}

Ouptut: 25 45 90
```

```c
/* 42. Insertion in between the nodes of a doubly linked list(part-1) */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtBeg(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;
	temp->next = head;
	head->prev = temp;

	head = temp;
	return head;
}

struct node* addAtEnd(struct node* head, int data)
{
	struct node *temp, *tp;
	temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	tp = head;
	while (tp->next != NULL)
	{
		tp = tp->next;
	}
	tp->next = temp;
	temp->prev = tp;
	return head;
}
```

```c
struct node *addAfterPos(struct node *head, int data, int position)
{
	struct node* newP = NULL;
	struct node* temp = head;
	struct node* temp2 = NULL;
	newP = addToEmpty(newP, data);

	while (position != 1)
	{
		temp = temp->next;
		position--;
	}

	if (temp->next == NULL)
	{
		temp->next = newP;
		newP->prev = temp;
	}
	else
	{
		temp2 = temp->next;
		temp->next = newP;
		temp2->prev = newP;
		newP->next = temp2;
		newP->prev = temp;
	}
	return head;
};

int main()
{
	struct node* head = NULL;
	struct node* ptr;
	int position = 2;
	head = addToEmpty(head, 45);
	head = addAtBeg(head, 25);
	head = addAtEnd(head, 90);
	head = addAfterPos(head, 30, position);

	ptr = head;
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->next;
	}
	return 0;
}
```

```
Output: 25 45 30 90
```

```
/* 43. Insertion in between the nodes of a doubly linked list(part-2) */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtBeg(struct node* head, int data)
{
	struct node* temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;
	temp->next = head;
	head->prev = temp;

	head = temp;
	return head;
}

struct node* addAtEnd(struct node* head, int data)
{
	struct node *temp, *tp;
	temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	tp = head;
	while (tp->next != NULL)
	{
		tp = tp->next;
	}
	tp->next = temp;
	temp->prev = tp;
	return head;
}
```

```
struct node *addBeforePos(struct node *head, int data, int position)
{
	struct node* newP = NULL;
	struct node* temp = head;
	struct node* temp2 = NULL;
	newP = addToEmpty(newP, data);

	int pos = position;
	while (pos > 2)
	{
		temp = temp->next;
		pos--;
	}

	if (position == 1)
		head = addAtBeg(head, data);

	else
	{
		temp2 = temp->next;
		temp->next = newP;
		temp2->prev = newP;
		newP->next = temp2;
		newP->prev = temp;
	}
	return head;
};

int main()
{
	struct node* head = NULL;
	struct node* ptr;
	int position = 2;
	head = addToEmpty(head, 45);
	head = addAtBeg(head, 25);
	head = addAtEnd(head, 90);
	head = addBeforePos(head, 30, position);

	ptr = head;
	while (ptr != NULL)
	{
		printf("%d ", ptr->data);
		ptr = ptr->next;
	}
	return 0;
}
```

```
Output: 25 30 45 90
```

```c
/* 44. Creating an entire list in double linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addToEmpty(struct node *head, int data)
{
      struct node *temp = malloc(sizeof(struct node));

      temp->prev = NULL;
      temp->data = data;
      temp->next = NULL;

      head = temp;
      return head;
}

struct node* addAtEnd(struct node* head, int data)
{
      struct node *temp, *tp;
      temp = malloc(sizeof(struct node));

      temp->prev = NULL;
      temp->data = data;
      temp->next = NULL;

      tp = head;
      while (tp->next != NULL)
      {
            tp = tp->next;
      }
      tp->next = temp;
      temp->prev = tp;
      return head;
}
```

```
struct node *createList(struct node *head)
{
      int n, data, i;
      printf("Enter the number of nodes: ");
      scanf("%d", &n);

      if (n == 0)
            return head;
      printf("Enter the element for node 1: ");
      scanf("%d", &data);
      head = addToEmpty(head, data);

      for (i = 1; i < n; i++)
      {
            printf("Enter the element for node %d: ", i + 1);
            scanf("%d", &data);
            head = addAtEnd(head, data);
      }
      return head;
}

int main()
{
      struct node *head = NULL;
      struct node *ptr;
      head = createList(head);

      ptr = head;
      while (ptr != NULL)
      {
            printf("%d ", ptr->data);
            ptr = ptr->next;
      }
      return 0;
}
```

Output:

```
Enter the number of nodes : 3
Enter the element for node 1 : 23
Enter the element for node 2 : 78
Enter the element for node 3 : 98
23 78 98
```

```c
/* 45. Deleting the first node of the list using double linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addToEmpty(struct node *head, int data)
{
      struct node *temp = malloc(sizeof(struct node));

      temp->prev = NULL;
      temp->data = data;
      temp->next = NULL;

      head = temp;
      return head;
}

struct node* addAtEnd(struct node* head, int data)
{
      struct node *temp, *tp;
      temp = malloc(sizeof(struct node));

      temp->prev = NULL;
      temp->data = data;
      temp->next = NULL;

      tp = head;
      while (tp->next != NULL)
      {
            tp = tp->next;
      }
      tp->next = temp;
      temp->prev = tp;
      return head;
}
```

```
struct node* delFirst(struct node* head)
{
      struct node* temp = head;
      head = head->next;
      free(temp);
      temp = NULL;
      head->prev = NULL;
      return head;
}

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

int main()
{
      struct node *head = NULL;
      struct node *ptr;
      head = addToEmpty(head, 24);
      head = addAtEnd(head, 45);
      head = addAtEnd(head, 9);

      printf("Before Deletion: ");
      print(head);

      head = delFirst(head);
      printf("After Deletion: ");
      print(head);

      return 0;
}
```

```
Output:
Before Deletion : 24 45 9
After Deletion : 45 9
```

```c
/* 46. Deleting the last node of the list using double linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addToEmpty(struct node *head, int data)
{
	struct node *temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtEnd(struct node* head, int data)
{
	struct node *temp, *tp;
	temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	tp = head;
	while (tp->next != NULL)
	{
		tp = tp->next;
	}
	tp->next = temp;
	temp->prev = tp;
	return head;
}
```

```c
struct node* delLast(struct node* head)
{
     struct node* temp = head;
     struct node* temp2;
     while (temp->next != NULL)
          temp = temp->next;
     temp2 = temp->prev;
     temp2->next = NULL;
     free(temp);
     return head;
}

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

int main()
{
     struct node *head = NULL;
     struct node *ptr;
     head = addToEmpty(head, 24);
     head = addAtEnd(head, 45);
     head = addAtEnd(head, 9);

     printf("Before Deletion: ");
     print(head);

     head = delLast(head);
     printf("After Deletion: ");
     print(head);

     return 0;
}
```

```
Output:
Before Deletion : 24 45 9
After Deletion : 24 45
```

```c
/* 47. Deleting the middle/intermediate node of the list using doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(struct node* head, int data)
{
    struct node* temp = malloc(sizeof(struct node));

    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;

    head = temp;
    return head;
}

struct node* addAtEnd(struct node* head, int data)
{
    struct node* temp, * tp;
    temp = malloc(sizeof(struct node));

    temp->prev = NULL;
    temp->data = data;
    temp->next = NULL;

    tp = head;
    while (tp->next != NULL)
    {
        tp = tp->next;
    }
    tp->next = temp;
    temp->prev = tp;
    return head;
}

struct node* delFirst(struct node* head)
{
    struct node* temp = head;
    head = head->next;
    free(temp);
    return head;
}
```

```c
struct node* delLast(struct node* head)
{
    struct node* temp = head;
    struct node* temp2;
    while (temp->next != NULL)
        temp = temp->next;
    temp2 = temp->prev;
    temp2->next = NULL;
    free(temp);
    return head;
}

struct node* delInter(struct node* head, int position)
{
    struct node* temp = head;
    struct node* temp2 = NULL;
    if (position == 1)
    {
        head = delFirst(head);
        return head;
    }

    while (position > 1)
    {
        temp = temp->next;
        position--;
    }

    if (temp->next == NULL)
        head = delLast(head);
    else
    {
        temp2 = temp->prev;
        temp2->next = temp->next;
        temp->next->prev = temp2;
        free(temp);
        temp = NULL;
    }
    return head;
}

void print(struct node* head)
{
    struct node* ptr = head;
    while (ptr != NULL)
    {
        printf("%d ", ptr->data);
        ptr = ptr->next;
    }
    printf("\n");
}
```

```c
int main()
{
    struct node* head = NULL;
    struct node* ptr;
    head = addToEmpty(head, 24);
    head = addAtEnd(head, 45);
    head = addAtEnd(head, 9);

    printf("Before Deletion: ");
    print(head);

    head = delInter(head, 2);
    printf("After Deletion: ");
    print(head);

    return 0;
}
```

```
Output:
Before Deletion : 24 45 9
After Deletion : 24 9
```

```c
/* 48. Reversing the doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addToEmpty(struct node *head, int data)
{
	struct node *temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	head = temp;
	return head;
}

struct node* addAtEnd(struct node* head, int data)
{
	struct node *temp, *tp;
	temp = malloc(sizeof(struct node));

	temp->prev = NULL;
	temp->data = data;
	temp->next = NULL;

	tp = head;
	while (tp->next != NULL)
	{
		tp = tp->next;
	}
	tp->next = temp;
	temp->prev = tp;
	return head;
}
```

```
struct node* reverse(struct node* head)
{
      struct node* ptr1 = head;
      struct node* ptr2 = ptr1->next;

      ptr1->next = NULL;
      ptr1->prev = ptr2;

      while (ptr2 != NULL)
      {
            ptr2->prev = ptr2->next;
            ptr2->next = ptr1;
            ptr1 = ptr2;
            ptr2 = ptr2->prev;
      }
      head = ptr1;
      return head;
}

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

int main()
{
      struct node *head = NULL;
      struct node *ptr;
      head = addToEmpty(head, 24);
      head = addAtEnd(head, 45);
      head = addAtEnd(head, 9);

      printf("Before Reversing the list: ");
      print(head);

      head = reverse(head);
      printf("After Reversing the list: ");
      print(head);

      return 0;
}

Output:
Before Reversing the list : 24 45 9
After Reversing the list : 9 45 24
```

Time Complexity:

- Insertion at the front (both singly and doubly) = O(1)
- Insertion at the end (only singly) = O(1)
- Insertion at the end (only doubly) = O(n)

- Deletion at the front (both singly and double) = O(1)
- Deletion at the last (only singly) = O(1)
- Deletion at the last (only doubly) = O(n)

Middle node deletion (only doubly) = O(n)
Doubly linked list traversal can perfume both sides.

Circular Linked List:

- Circular **Singly** Linked List

It is similar to the singly linked list except that the last node of the circular singly linked list points to the first node.

- Circular **Doubly** Linked List

It is similar to the doubly linked list except that the last node of the circular doubly linked list points to the first node and the first node of the circular doubly linked list points to the last node.

```
/* 49. Creating a node of a circular singly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *circularSingly(int data)
{
      struct node *temp = malloc(sizeof(struct node));
      temp->data = data;
      temp->next = temp;
      return temp;
}

int main()
{
      int data = 34;
      struct node *tail; // *tail = malloc(sizeof(struct node));
      tail = circularSingly(data);

      printf("%d ", tail->data);
      return 0;
}

Output: 34
```

```c
/* 50. Creating a node of a circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *circularDoubly(int data)
{
	struct node *temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	temp->prev = temp;
	return temp;
}

int main()
{
	int data = 34;
	struct node *tail;
	tail = circularDoubly(data);

	printf("%d ", tail->data);
	return 0;
}
```

```
Output: 34
```

```c
/* 51. Inserting at the beginning of a circular singly linked list */
/* 52. Traversal of a circular singly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addtoEmpty(int data)
{
	struct node *temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node *addatBeg(struct node *tail, int data)
{
	struct node *newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = tail->next;
	tail->next = newP;
	return tail;
}

void print(struct node *tail)
{
	struct node *p; // = tail->next;
	do {
		printf("%d ", p->data);
		p = p->next;
	} while (p != tail->next);
}

int main()
{
	struct node *tail;
	tail = addtoEmpty(45);
	tail = addatBeg(tail, 35);
	/* tail = addatBeg(tail, 25); */

	print(tail);
	return 0;
}
```

```
Outuput: 35 45
Output: 25 35 45
```

```c
/* 53. Insertion at the end of a circular singly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node *addtoEmpty(int data)
{
	struct node *temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node *addatBeg(struct node *tail, int data)
{
	struct node *newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = tail->next;
	tail->next = newP;
	return tail;
}

struct node* addatEnd(struct node* tail, int data)
{
	struct node *newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = NULL;

	newP->next = tail->next;
	tail->next = newP;
	tail = tail->next;
	return tail;
}

void print(struct node *tail)
{
	struct node *p = tail->next;
	do {
		printf("%d ", p->data);
		p = p->next;
	} while (p != tail->next);
}

int main()
{
	struct node *tail; //tail = malloc(sizeof(struct node));
	tail = addtoEmpty(45);
	tail = addatBeg(tail, 35);
	tail = addatEnd(tail, 55);
	tail = addatEnd(tail, 65);

	print(tail);
	return 0;
}
```

```
Output: 35 45 55 65
```

```c
/* 54. Insertion in between the nods. Circular singly linked list */
#include <stdio.h>
#include <malloc.h>

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

struct node *addToEmpty(int data)
{
       struct node *temp = malloc(sizeof(struct node));
       temp->data = data;
       temp->next = temp;
       return temp;
}

struct node *addAtEnd(struct node *tail, int data)
{
       struct node *newP = malloc(sizeof(struct node));
       newP->data = data;
       newP->next = NULL;

       newP->next = tail->next;
       tail->next = newP;
       tail = tail->next;
       return tail;
}

struct node *addAfterPos(struct node* tail, int data, int pos)
{
       struct node *p = tail->next;
       struct node *newP = malloc(sizeof(struct node));
       newP->data = data;
       newP->next = NULL;
       while (pos > 1)
       {
              p = p->next;
              pos--;
       }
       newP->next = p->next;
       p->next = newP;
       if (p == tail)
              tail = tail->next;

       return tail;
}

void print(struct node* tail)
{
       struct node *p = tail->next;
       do {
              printf("%d ", p->data);
              p = p->next;
       } while (p != tail->next);
       printf("\n");
}
```

```
int main()
{
       struct node *tail;
       tail = addToEmpty(34);
       tail = addAtEnd(tail, 45);
       tail = addAtEnd(tail, 6);
       print(tail);
       tail = addAfterPos(tail, 66, 2);
       print(tail);
       return 0;
}
```

```
Output:
34 45 6
34 45 66 6
```

```c
/* 55. Creating a circular singly linked list */
#include <stdio.h>
#include <malloc.h>

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

struct node *addToEmpty(int data)
{
       struct node *temp = malloc(sizeof(struct node));
       temp->data = data;
       temp->next = temp;
       return temp;
}

struct node *addAtEnd(struct node *tail, int data)
{
       struct node *newP = malloc(sizeof(struct node));
       newP->data = data;
       newP->next = NULL;

       newP->next = tail->next;
       tail->next = newP;
       tail = tail->next;
       return tail;
}

struct node* createList(struct node* tail)
{
       int i, n, data;
       printf("Enter the number of nodes of the linked list: ");
       scanf("%d", &n);

       if (n == 0)
              return tail;
       printf("Enter the element 1: ");
       scanf("%d", &data);
       tail = addToEmpty(data);

       for (i = 1; i < n; i++)
       {
              printf("Enter the element %d: ", i + 1);
              scanf("%d", &data);
              tail = addAtEnd(tail, data);
       }
       return tail;
}
```

```
void print(struct node* tail)
{
	if (tail == NULL)
		printf("No node in the list.");
	else
	{
		struct node* p = tail->next;
		do {
			printf("%d ", p->data);
			p = p->next;
		} while (p != tail->next);
	}
	printf("\n");
}

int main()
{
	struct node *tail;
	tail = createList(tail);
	print(tail);
	return 0;
}
```

```
Output:
Enter the number of nodes of the linked list : 4
Enter the element 1 : 12
Enter the element 2 : 23
Enter the element 3 : 34
Enter the element 4 : 45
12 23 34 45
```

```c
/* 56. Deleting the first node. Circular singly linked list */
#include <stdio.h>
#include <malloc.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node* newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = NULL;

	newP->next = tail->next;
	tail->next = newP;
	tail = tail->next;
	return tail;
}

struct node* createList(struct node* tail)
{
	int i, n, data;
	printf("Enter the number of nodes of the linked list: ");
	scanf("%d", &n);

	if (n == 0)
		return tail;
	printf("Enter the element 1: ");
	scanf("%d", &data);
	tail = addToEmpty(data);

	for (i = 1; i < n; i++)
	{
		printf("Enter the element %d: ", i + 1);
		scanf("%d", &data);
		tail = addAtEnd(tail, data);
	}
	return tail;
}
```

```
struct node* delFirst(struct node* tail)
{
       if (tail == NULL)
              return tail;
       if (tail->next == tail)
       {
              free(tail);
              tail = NULL;
              return tail;
       }
       struct node *temp = tail->next;
       tail->next = temp->next;
       free(temp);
       temp = NULL;
       return tail;
}

void print(struct node* tail)
{
       if (tail == NULL)
              printf("No node in the list.");
       else
       {
              struct node* p = tail->next;
              do {
                     printf("%d ", p->data);
                     p = p->next;
              } while (p != tail->next);
       }
       printf("\n");
}

int main()
{
       struct node *tail = NULL;
       tail = createList(tail);
       printf("List before deletion: ");
       print(tail);

       tail = delFirst(tail);
       printf("List afte deletion: ");
       print(tail);
       return 0;
}
```

Output:

```
Enter the number of nodes of the linked list : 3
Enter the element 1 : 78
Enter the element 2 : 90
Enter the element 3 : 98
List before deletion : 78 90 98
List afte deletion : 90 98
```

```c
/* 57. Deleting the last node. Circular singly linked list */
#include <stdio.h>
#include <malloc.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node* newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = NULL;

	newP->next = tail->next;
	tail->next = newP;
	tail = tail->next;
	return tail;
}

struct node* createList(struct node* tail)
{
	int i, n, data;
	printf("Enter the number of nodes of the linked list: ");
	scanf("%d", &n);

	if (n == 0)
		return tail;
	printf("Enter the element 1: ");
	scanf("%d", &data);
	tail = addToEmpty(data);

	for (i = 1; i < n; i++)
	{
		printf("Enter the element %d: ", i + 1);
		scanf("%d", &data);
		tail = addAtEnd(tail, data);
	}
	return tail;
}
```

```
struct node* delLast(struct node* tail)
{
       if (tail == NULL)
              return tail;
       if (tail->next == tail)
       {
              free(tail);
              tail = NULL;
              return tail;
       }

       struct node *temp = tail->next;
       while (temp->next != tail)
              temp = temp->next;

       temp->next = tail->next;
       free(tail);
       tail = temp;
       return tail;
}

void print(struct node* tail)
{
       if (tail == NULL)
              printf("No node in the list.");
       else
       {
              struct node* p = tail->next;
              do {
                     printf("%d ", p->data);
                     p = p->next;
              } while (p != tail->next);
       }
       printf("\n");
}

int main()
{
       struct node *tail = NULL;
       tail = createList(tail);
       printf("List before deletion: ");
       print(tail);

       tail = delLast(tail);
       printf("List afte deletion: ");
       print(tail);
       return 0;
}
```

```
Output:
Enter the number of nodes of the linked list : 3
Enter the element 1 : 45
Enter the element 2 : 56
Enter the element 3 : 67
List before deletion : 45 56 67
List afte deletion : 45 56
```

```c
/* 58. Deleting the intermediate node. Circular singly linked list */
#include <stdio.h>
#include <malloc.h>

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

struct node* addToEmpty(int data)
{
       struct node* temp = malloc(sizeof(struct node));
       temp->data = data;
       temp->next = temp;
       return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
       struct node* newP = malloc(sizeof(struct node));
       newP->data = data;
       newP->next = NULL;

       newP->next = tail->next;
       tail->next = newP;
       tail = tail->next;
       return tail;
}

struct node* createList(struct node* tail)
{
       int i, n, data;
       printf("Enter the number of nodes of the linked list: ");
       scanf("%d", &n);

       if (n == 0)
              return tail;
       printf("Enter the element 1: ");
       scanf("%d", &data);
       tail = addToEmpty(data);

       for (i = 1; i < n; i++)
       {
              printf("Enter the element %d: ", i + 1);
              scanf("%d", &data);
              tail = addAtEnd(tail, data);
       }
       return tail;
}
```

```c
struct node* delInter(struct node* tail, int pos)
{
	if (tail == NULL)
		return tail;
	struct node *temp = tail->next;
	if (tail->next == tail)
	{
		free(tail);
		tail = NULL;
		return tail;
	}
	while (pos > 2)
	{
		temp = temp->next;
		pos--;
	}
	struct node *temp2 = temp->next;
	temp->next = temp2->next;
	/* Handling the case of deleting the last node */
	if (temp2 == tail)
		tail = temp;
	free(temp);
	temp = NULL;
	return tail;
}

void print(struct node* tail)
{
	if (tail == NULL)
		printf("No node in the list.");
	else
	{
		struct node* p = tail->next;
		do {
			printf("%d ", p->data);
			p = p->next;
		} while (p != tail->next);
	}
	printf("\n");
}

int main()
{
	struct node *tail = NULL;
	tail = createList(tail);
	printf("List before deletion: ");
	print(tail);

	/* We can also check (tail, 2) */
	tail = delInter(tail, 3);
	printf("List afte deletion: ");
	print(tail);
	return 0;
}
```

Output:

```
Enter the number of nodes of the linked list : 4
Enter the element 1 : 12
Enter the element 2 : 23
Enter the element 3 : 34
Enter the element 4 : 45
List before deletion : 12 23 34 45
List afte deletion : 12 23 34
```

```
/* 59. Counting the number of elements. Cercular singly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
       struct node *temp = malloc(sizeof(struct node));
       temp->data = data;
       temp->next = temp;
       return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
       struct node *newP = malloc(sizeof(struct node));
       newP->data = data;
       newP->next = NULL;

       newP->next = tail->next;
       tail->next = newP;
       tail = tail->next;
       return tail;

}

void countElements(struct node* tail)
{
       struct node* temp = tail->next;
       int count = 0;
       while (temp != tail)
       {
              temp = temp->next;
              count++;
       }
       count++;
       printf("There are %d elements in the list. \n", count);
}

int main()
{
       struct node *tail = NULL;
       tail = addToEmpty(34);
       tail = addAtEnd(tail, 45);
       tail = addAtEnd(tail, 66);
       tail = addAtEnd(tail, 6);

       countElements(tail);
       return 0;
}

Output: There are 4 elements in the list.
```

```
printf("The elements are: ");
print(tail);
void print(struct node* tail)
{
       if (tail == NULL)
              printf("No node in the list");
       else {
              struct node* p = tail->next;
              do {
                     printf("%d ", p->data);
                     p = p->next;
              } while (p != tail->next);
       }
       printf("\n");
}
```

```c
/* 60. Searching an elements. Cercular singly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
	struct node *temp = malloc(sizeof(struct node));
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node *newP = malloc(sizeof(struct node));
	newP->data = data;
	newP->next = NULL;

	newP->next = tail->next;
	tail->next = newP;
	tail = tail->next;
	return tail;

}

int searchElement(struct node* tail, int element)
{
	struct node *temp;
	int index = 0;

	if (tail == NULL)
		return -2;

	temp = tail->next;
	do {
		if (temp->data == element)
			return index;
		temp = temp->next;
		index++;
	} while (temp != tail->next);
	return -1;
}
```

```
void print(struct node* tail)
{
	if (tail == NULL)
		printf("No node in the list.");
	else
	{
		struct node* p = tail->next;
		do {
			printf("%d ", p->data);
			p = p->next;
		} while (p != tail->next);
	}
	printf("\n");
}


int main()
{
	struct node *tail = NULL;
	int element;
	tail = addToEmpty(34);
	tail = addAtEnd(tail, 45);
	tail = addAtEnd(tail, 66);
	tail = addAtEnd(tail, 6);

	printf("The elements are: ");
	print(tail);
	printf("\nEnter the element u want to search: ");
	scanf("%d", &element);

	int index = searchElement(tail, element);
	if (index == -1)
		printf("Element not found\n");
	else if (index == -2)
		printf("Linked list is empty\n");
	else
		printf("Element %d is at index %d\n", element, index);
	return 0;
}
```

```
Output:
The elements are : 34 45 66 6

Enter the element u want to search : 66
Element 66 is at index 2
```

```c
/* 61. Insertion at the beginning. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
      struct node* temp = malloc(sizeof(struct node));
      temp->prev = temp;
      temp->data = data;
      temp->next = temp;
      return temp;
}

struct node* addAtBeg(struct node* tail, int data)
{
      struct node *newP = addToEmpty(data);
      if (tail == NULL)
            return newP;
      else
      {
            struct node *temp = tail->next;

            newP->prev = tail;
            newP->next = temp;
            temp->prev = newP;
            tail->next = newP;
            return tail;
      }
}

void print(struct node* tail)
{
      if (tail == NULL)
            printf("No element in the list\n");
      else
      {
            struct node *temp = tail->next;
            do {
                  printf("%d ", temp->data);
                  temp = temp->next;
            } while (temp != tail->next);
      }
      printf("\n");
}
```

```
int main()
{
      struct node* tail = NULL;
      tail = addToEmpty(45);
      tail = addAtBeg(tail, 34);
      print(tail);
      return 0;
}

Output: 34 45
```

```c
/* 62. Insertion at the end. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->prev = temp;
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtBeg(struct node* tail, int data)
{
	struct node *temp = tail->next;
	struct node *newP = malloc(sizeof(struct node));
	newP->prev = NULL;
	newP->data = data;
	newP->next = NULL;

	newP->prev = tail;
	newP->next = temp;
	temp->prev = newP;
	tail->next = newP;
	return tail;

}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node *newP = addToEmpty(data);
	if (tail == NULL)
		return newP;
	else
	{
		struct node *temp = tail->next;
		newP->next = temp;
		newP->prev = tail;
		tail->next = newP;
		temp->prev = newP;
		tail = newP;
		return tail;
	}
}
```

```
void print(struct node* tail)
{
	if (tail == NULL)
		printf("No element in the list\n");
	else
	{
		struct node *temp = tail->next;
		do {
			printf("%d ", temp->data);
			temp = temp->next;
		} while (temp != tail->next);
	}
	printf("\n");
}

int main()
{
	struct node *tail = NULL;
	tail = addToEmpty(45);
	tail = addAtBeg(tail, 34);
	tail = addAtEnd(tail, 56);
	print(tail);
	return 0;
}

Output: 34 45 56
```

```c
/* 63. Insertion between the nodes. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->prev = temp;
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtBeg(struct node* tail, int data)
{
	struct node *temp = tail->next;
	struct node *newP = malloc(sizeof(struct node));
	newP->prev = NULL;
	newP->data = data;
	newP->next = NULL;

	newP->prev = tail;
	newP->next = temp;
	temp->prev = newP;
	tail->next = newP;
	return tail;

}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node *newP = addToEmpty(data);
	if (tail == NULL)
		return newP;
	else
	{
		struct node *temp = tail->next;
		newP->next = temp;
		newP->prev = tail;
		tail->next = newP;
		temp->prev = newP;
		tail = newP;
		return tail;
	}
}
```

```c
struct node *addAfterPos(struct node* tail, int data, int pos)
{
	struct node *newP = addToEmpty(data);
	if (tail == NULL)
		return newP;
	struct node *temp = tail->next;
	while (pos > 1)
	{
		temp = temp->next;
		pos--;
	}
	newP->prev = temp;
	newP->next = temp->next;
	temp->next->prev = newP;
	temp->next = newP;
	if (temp == tail)
		tail = tail->next;
	return tail;
}

void print(struct node* tail)
{
	if (tail == NULL)
		printf("No element in the list\n");
	else
	{
		struct node *temp = tail->next;
		do {
			printf("%d ", temp->data);
			temp = temp->next;
		} while (temp != tail->next);
	}
	printf("\n");
}

int main()
{
	struct node *tail = NULL;
	tail = addToEmpty(45);
	/* tail = addAtBeg(tail, 34); */
	tail = addAtEnd(tail, 56);
	tail = addAtEnd(tail, 60);
	tail = addAfterPos(tail, 90, 2);
	print(tail);
	return 0;
}

Output: 45 56 90 60
```

```
/* 64.Deleting the first node. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->prev = temp;
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node* newP = addToEmpty(data);
	if (tail == NULL)
		return newP;
	else
	{
		struct node* temp = tail->next;
		newP->next = temp;
		newP->prev = tail;
		tail->next = newP;
		temp->prev = newP;
		tail = newP;
		return tail;
	}
}

struct node* delFirst(struct node* tail)
{
	if (tail == NULL)
		return tail;
	struct node *temp = tail->next;
	if (temp == tail)
	{
		free(tail);
		tail = NULL;
		return tail;
	}
	tail->next = temp->next;
	temp->next->prev = tail;
	free(temp);
	return tail;
}
```

```c
void print(struct node* tail)
{
	if (tail == NULL)
		printf("No element in the list\n");
	else
	{
		struct node* temp = tail->next;
		do {
			printf("%d ", temp->data);
			temp = temp->next;
		} while (temp != tail->next);
	}
	printf("\n");
}

int main()
{
	struct node* tail = NULL;
	tail = addToEmpty(45);
	tail = addAtEnd(tail, 56);
	tail = addAtEnd(tail, 60);

	printf("List before deletion: ");
	print(tail);

	tail = delFirst(tail);

	printf("List after deletion: ");
	print(tail);
	return 0;
}
```

```
Output:
List before deletion : 45 56 60
List after deletion : 56 60
We can also check the output by testing this method:
//tail = addToEmpty(45);
//tail = addAtEnd(tail, 56);
//tail = addAtEnd(tail, 60);
```

```c
/* 65.Deleting the last node. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
      struct node* temp = malloc(sizeof(struct node));
      temp->prev = temp;
      temp->data = data;
      temp->next = temp;
      return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
      struct node* newP = addToEmpty(data);
      if (tail == NULL)
            return newP;
      else
      {
            struct node* temp = tail->next;
            newP->next = temp;
            newP->prev = tail;
            tail->next = newP;
            temp->prev = newP;
            tail = newP;
            return tail;
      }
}
```

```c
struct node* delLast(struct node* tail)
{
      if (tail == NULL)
            return tail;
      struct node *temp = tail->next;
      if (temp == tail)
      {
            free(tail);
            tail = NULL;
            return tail;
      }
      temp = tail->prev;
      temp->next = tail->next;
      tail->next->prev = temp;
      free(tail);
      tail = temp;
      return tail;
}

void print(struct node* tail)
{
      if (tail == NULL)
            printf("No element in the list\n");
      else
      {
            struct node* temp = tail->next;
            do {
                  printf("%d ", temp->data);
                  temp = temp->next;
            } while (temp != tail->next);
      }
      printf("\n");
}
int main()
{
      struct node* tail = NULL;
      tail = addToEmpty(45);
      tail = addAtEnd(tail, 56);
      tail = addAtEnd(tail, 60);

      printf("List before deletion: ");
      print(tail);

      tail = delLast(tail);

      printf("List after deletion: ");
      print(tail);
      return 0;
}
```

```
Output:
List before deletion : 45 56 60
List after deletion : 45 56
```

```c
/* 66.Deleting the intermediate node. Circular doubly linked list */
#include <stdio.h>
#include <stdlib.h>

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

struct node* addToEmpty(int data)
{
	struct node* temp = malloc(sizeof(struct node));
	temp->prev = temp;
	temp->data = data;
	temp->next = temp;
	return temp;
}

struct node* addAtEnd(struct node* tail, int data)
{
	struct node* newP = addToEmpty(data);
	if (tail == NULL)
		return newP;
	else
	{
		struct node* temp = tail->next;
		newP->next = temp;
		newP->prev = tail;
		tail->next = newP;
		temp->prev = newP;
		tail = newP;
		return tail;
	}
}

struct node* delInter(struct node* tail, int pos)
{
	struct node *temp = tail->next;
	while (pos > 1)
	{
		temp = temp->next;
		pos--;
	}
	struct node *temp2 = temp->prev;
	temp2->next = temp->next;
	temp->next->prev = temp2;
	free(temp);
	if (temp == tail)
		tail = temp2;
	return tail;
}
```

```
void print(struct node* tail)
{
      if (tail == NULL)
            printf("No element in the list\n");
      else
      {
            struct node* temp = tail->next;
            do {
                  printf("%d ", temp->data);
                  temp = temp->next;
            } while (temp != tail->next);
      }
      printf("\n");
}

int main()
{
      struct node* tail = NULL;
      tail = addToEmpty(45);
      tail = addAtEnd(tail, 56);
      tail = addAtEnd(tail, 60);

      printf("List before deletion: ");
      print(tail);

      tail = delInter(tail, 2);

      printf("List after deletion: ");
      print(tail);
      return 0;
}
```

Output:

```
List before deletion : 45 56 60
List after deletion : 45 60
```