JADAVPUR UNIVERSITY

Department of Electronics and
TeleCommunication Engineering

Data Structures and Algorithms Lab

Assignment - 1 : Linked List

Programming Language : C (ANSI)

==

Sandip Dutta
2nd Year
Roll - 001910701017

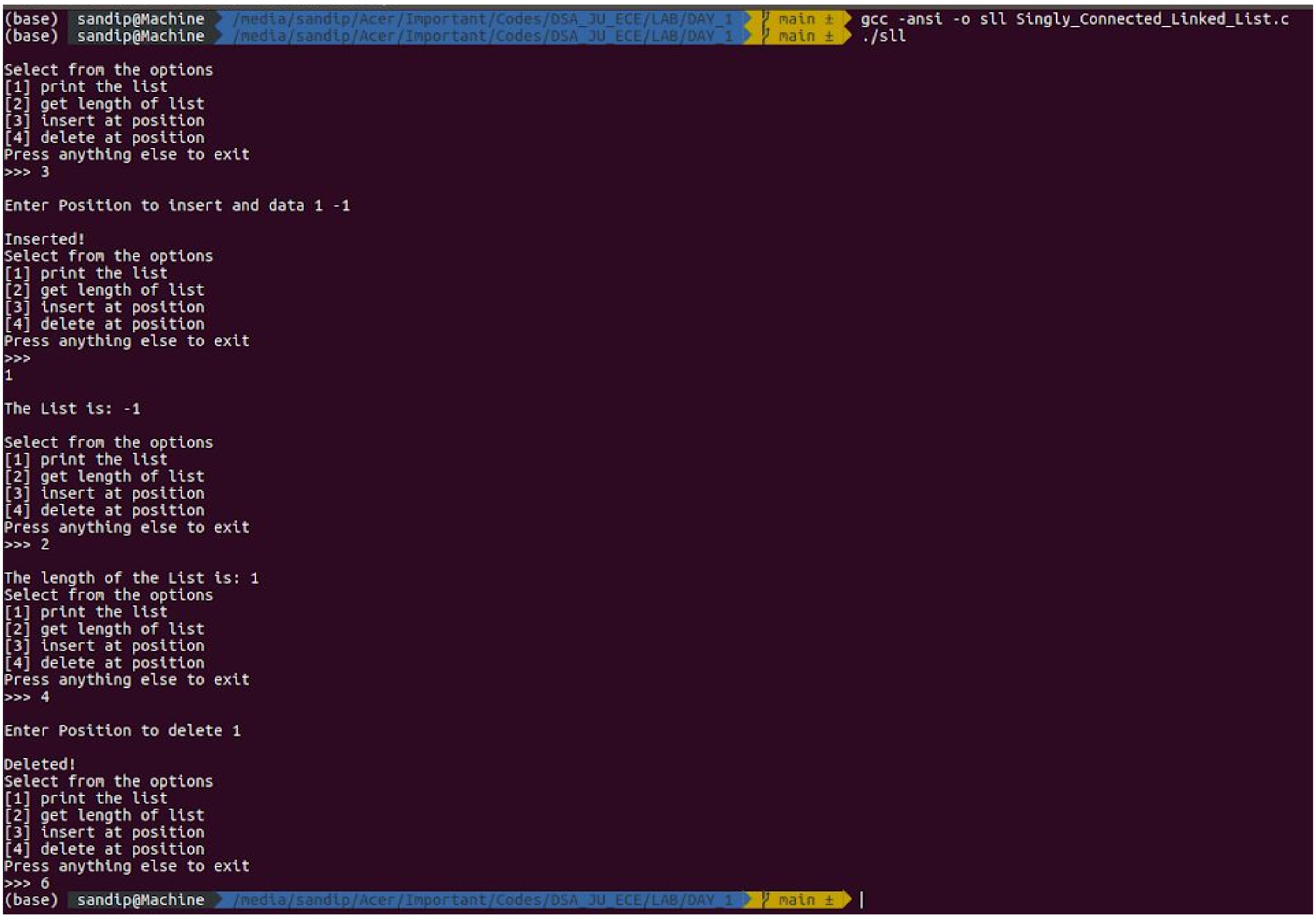
==

1. **Implement a singly connected linear linked list in C. Your program should typically implement insert and delete at all possible locations with proper check(s) as applicable. Include a display function as well and use it to show the content of your list after every operation. Include calls to insert and delete from the main.**

===

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

/* Node*/
struct SLLNode{
    int data;
    struct SLLNode *next;
};

/*Length of Linked List*/
int get_length(struct SLLNode *head){
    struct SLLNode *curr = head;
    int length = 1;

    if (head == NULL) return 0;
    while(curr = curr -> next) ++length;
    return length;
}

/* Print the list */
void printList(struct SLLNode *head){
    if(head == NULL) return;
    struct SLLNode *curr = head;

    while(curr) {
        printf("%d ", curr -> data);
        curr = curr -> next;
    }
    printf("\n");
}

/* Insert in SLL*/
struct SLLNode* insert_at(int position, int data, struct SLLNode* head){
    int count = 1;
    struct SLLNode *p, *q, *newNode;

    newNode = (struct SLLNode*) malloc(sizeof(struct SLLNode));

    if(!newNode) return NULL;

    newNode -> data = data;

    p = head;

    if(position == 1){
        /*Insert at begin*/
        newNode -> next = p;
        return newNode;
    } else {
        while((p != NULL) && (count < position)) {
            ++count;
            q = p;
            p = p -> next;
        }
        q -> next = newNode;
        newNode -> next = p;
    }
    return head;
}

/* Delete */
struct SLLNode* delete_from(struct SLLNode *head, int position){
    int count = 1;
    struct SLLNode *p, *q;
    if(head == NULL) return head;

    p = head;

    if(position == 1){
        p = head;
        head = head -> next;
        free(p);
    } else {
        while ((p != NULL) && (count < position)){
            count++;
            q = p;
            p = p -> next;
        }

        if (p != NULL) {
            q -> next = p -> next;
            free(p);
        }
    }
    return head;
}

/*Show menu*/
void show_menu(){
    printf("\nSelect from the options\n");
    printf("[1] print the list\n");
    printf("[2] get length of list\n");
    printf("[3] insert at position\n");
    printf("[4] delete at position\n");
    printf("Press anything else to exit\n");
    printf(">>> ");
}

/* Driver Function */
int main(){
    struct SLLNode *head = NULL;
    int choice, position, data;

    while(1){
        show_menu();
        scanf("%d", &choice);

        switch (choice){
            case 1:
                printf("\nThe List is: ");
                printList(head);
                break;
            case 2:
                printf("\nThe length of the List is: %d", get_length(head));
                break;
            case 3:
                printf("\nEnter Position to insert and data ");
                scanf("%d %d", &position, &data);
                head = insert_at(position, data, head);
                printf("\nInserted!");
                break;
            case 4:
                printf("\nEnter Position to delete ");
                scanf("%d", &position);
                head = delete_from(head, position);
                printf("\nDeleted!");
                break;
            default:
                return 0;
        }
    }
    return 0;
}
================================================================================================
```

Output:



```
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  gcc -ansi -o sll Singly_Connected_Linked_List.c
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  ./sll

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 3

Enter Position to insert and data 1 -1

Inserted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>>
1

The List is: -1

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 2

The length of the List is: 1
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 4

Enter Position to delete 1

Deleted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 6
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  |
```

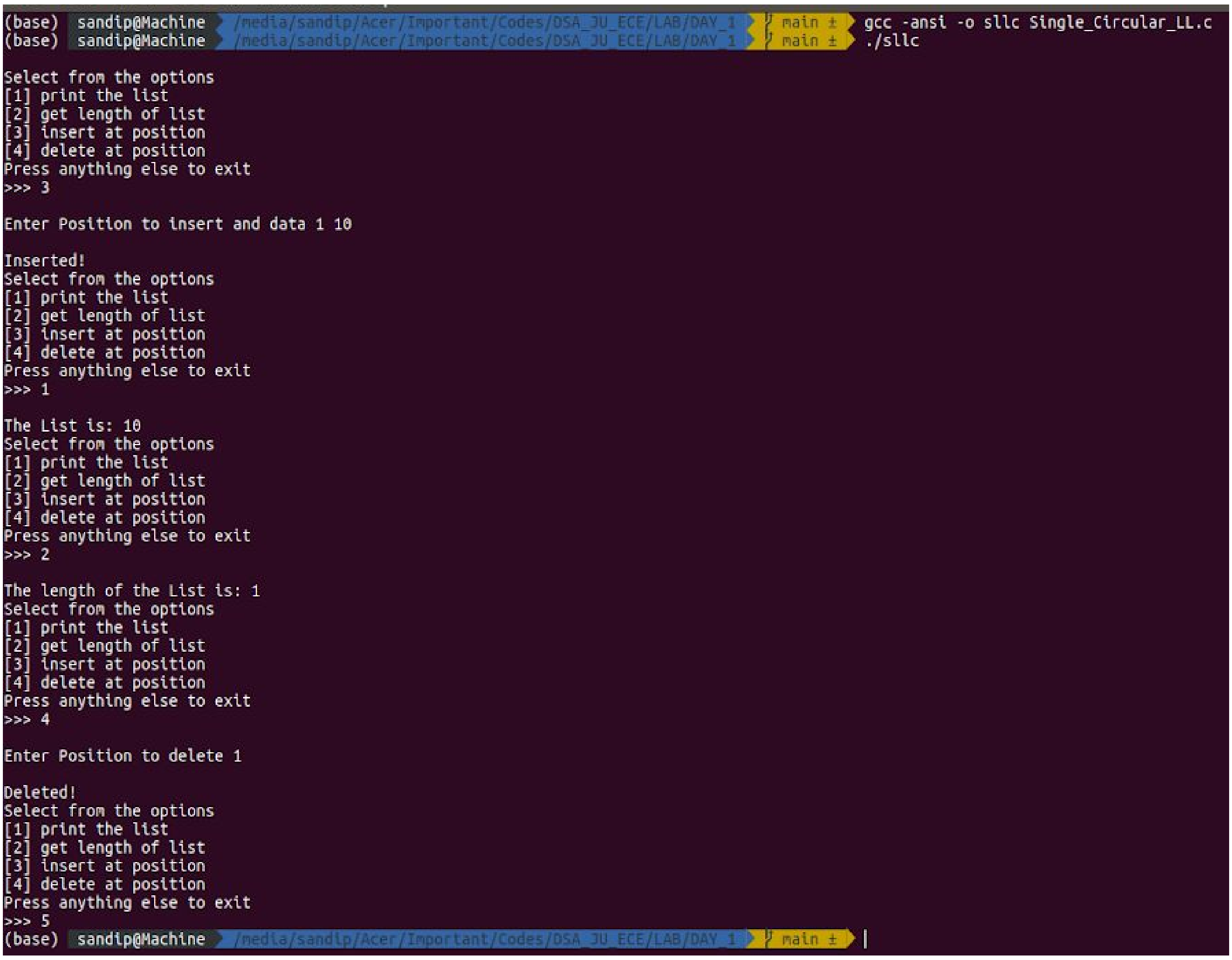
2. **Implement a singly connected circular linked list in C. Your program should typically implement insert and delete at all possible locations with proper check(s) as applicable. Include a display function as well and use it to show the content of your list after every operation. Include calls to insert and delete from the main.**

===

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

/*
NOTE:
1. In this implementation, we use 2 pointers.
One for the head and one for the tail.
This can be used to implement a circular queue
2. We declare head and tail, two pointers globally to try
out this variation
*/

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

node* head = NULL, *tail = NULL;

void insertAtBeginning(int data)
{
    /*Insert data at the beginning of the list*/
    node *newNode = (node *)malloc(sizeof(node));

    newNode->data = data;
    newNode->next = newNode;

    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        newNode->next = head;
        tail->next = newNode;
        head = newNode;
    }
}

/*Insert a node at tail of a circular singly linked list*/
void insertAtEnd(int data)
{
    node *newNode = (node *)malloc(sizeof(node));

    newNode->data = data;
    newNode->next = newNode;

    if (head == NULL)
    {
        head = newNode;
        tail = newNode;
    }
    else
    {
        tail->next = newNode;
        newNode->next = head;
        tail = newNode;
    }
}

/* Insert a node at position of a circular singly linked list */
void insertAtPosition(int data, int position)
{
    if (position == 1)
    {
        insertAtBeginning(data);
        return;
    }
    else if (position > 1 && head != NULL)
    {
        node *current = head;
        node *temp = (node *)malloc(sizeof(node));
        int count = 0;

        do
        {
            count++;
            temp = current;
            current = current->next;
        } while (current->next != head && count < position - 1);

        if (count == position - 1)
        {
            if (temp == tail)
                insertAtEnd(data);
            else
            {
                node *newNode = (node *)malloc(sizeof(node));
                newNode->data = data;

                temp->next = newNode;
                newNode->next = current;
            }
            return;
        }
    }
}

/* Delete HEAD node of a circular SLL */
void deleteAtBeginning()
{
    if (head == NULL)
        return;

    node *temp = head;

    tail->next = head->next;
    head = head->next;

    free(temp);
}

/* Delete TAIL node of a circular SLL */
void deleteAtEnd()
{
    if (head == NULL)
        return;

    node *temp = head;
    node *current = head;

    while (current->next != head)
    {
        temp = current;
        current = current->next;
    }

    temp->next = head;
    tail = temp;
    free(current);
}

/* Delete a node at position in the circular singly linked list */
void deleteAtPosition(int position)
{
    if (head == NULL)
        return;
```

```c
    if (position == 1)
    {
        deleteAtBeginning();
        return;
    }

    node *current = head;
    node *temp;
    int count = 0;

    do
    {
        count++;
        temp = current;
        current = current->next;
    } while (current->next != head && count < position - 1);

    if (count == position - 1)
    {
        if (current == tail)
        {
            deleteAtEnd();
            return;
        }

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

/* Print all node's data of a circular singly linked list */
void printList()
{
    if (head == NULL)
        return;

    node *current = head;

    do
    {
        printf("%d ", current->data);
        current = current->next;
    } while (current != head);

}

/* Determine the data of nodes in circular singly linked list */
int listLength()
{
    if (head == NULL)
        return 0;

    int count = 0;
    node *current = head;

    do
    {
        count++;
        current = current->next;
    } while (current != head);

    return count;
}

/*Show menu*/
void show_menu(){
    printf("\nSelect from the options\n");
    printf("[1] print the list\n");
    printf("[2] get length of list\n");
    printf("[3] insert at position\n");
    printf("[4] delete at position\n");
    printf("Press anything else to exit\n");
    printf(">>> ");
}

/* Driver Function */
int main(){
    int choice, position, data;

    do{
        show_menu();
        scanf("%d", &choice);

        switch (choice){
            case 1:
                printf("\nThe List is: ");
                printList();
                break;
            case 2:
                printf("\nThe length of the List is: %d", listLength());
                break;
            case 3:
                printf("\nEnter Position to insert and data ");
                scanf("%d %d", &position, &data);
                insertAtPosition(data, position);
                printf("\nInserted!");
                break;
            case 4:
                printf("\nEnter Position to delete ");
                scanf("%d", &position);
                deleteAtPosition(position);
                printf("\nDeleted!");
                break;
            default:
                exit(0);
                break;
        }
    }while(1);
    return 0;
}
================================================================================================
```

Output



```
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  gcc -ansi -o sllc Single_Circular_LL.c
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  ./sllc

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 3

Enter Position to insert and data 1 10

Inserted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 1

The List is: 10
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 2

The length of the List is: 1
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 4

Enter Position to delete 1

Deleted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 5
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  |
```

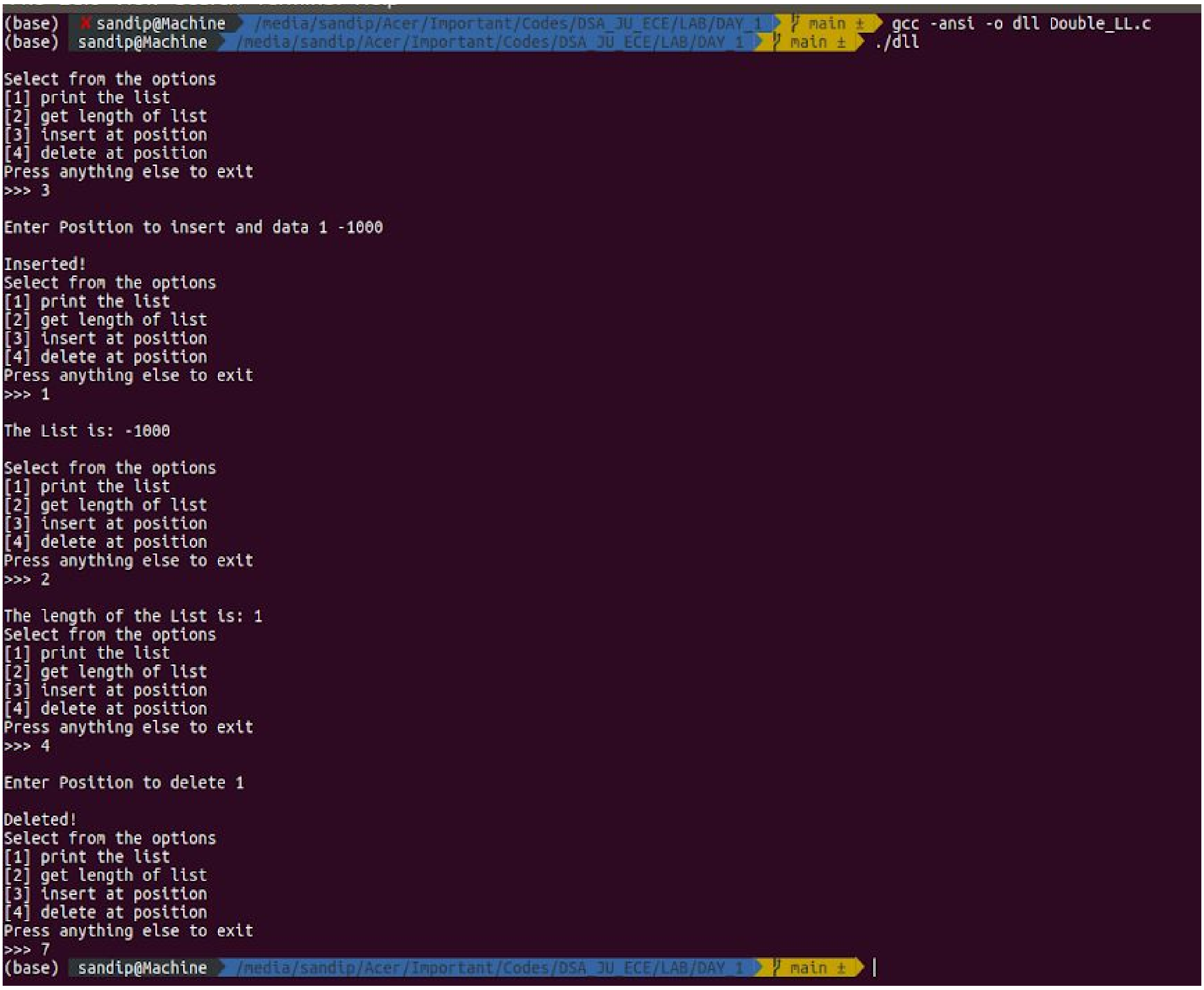
3. **Implement a doubly connected linear linked list in C. Your program should typically implement insert and delete at all possible locations with proper check(s) as applicable. Include a display function as well and use it to show the content of your list after every operation. Include calls to insert and delete from the main.**

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

struct DLLNode{
    int data;
    struct DLLNode *next, *prev;
};

/* Insert at a position in the DLL.
Double pointer as send reference to the list */
void insert(struct DLLNode **head, int data, int position){
    int count = 1;
    struct DLLNode *temp, *newNode;
    newNode = (struct DLLNode*) malloc(sizeof(struct DLLNode));

    if(!newNode){
        printf("MEMORY ERROR!\n");
    }

    /* init node */
    newNode->data = data;
    newNode->next = NULL;
    newNode->prev = NULL;

    if(position == 1){
        /* Value of head */
        newNode->next = *head;
        if(*head) (*head) -> prev = newNode;
        *head = newNode;
        return;
    }

    temp = *head;

    while(count < position && temp->next != NULL) {
        temp = temp -> next;
        count++;
    }

    if(count < position - 1){
        printf("INVALID!");
        return;
    }

    newNode ->next = temp->next;
    newNode -> prev = temp;
    if (temp ->next) temp ->next ->prev = newNode;
    temp ->next = newNode;
}

/* Delete at given position of the doubly connected linked list */

void delete(struct DLLNode **head, int position){
    struct DLLNode *temp2, *temp = *head;

    int count = 1;
    if(*head==NULL){
        printf("EMPTY LIST!\n");
        return;
    }

    if(position == 1){
        *head = (*head) -> next;
        if(*head!=NULL)
            (*head) -> prev = NULL;
        free(temp);
        return;
```

```c
    }

    while(count < position && temp->next != NULL) {
        temp = temp -> next;
        count++;
    }

    if(count < position - 1){
        printf("INVALID!");
        return;
    }

    temp2 = temp -> prev;
    temp2 -> next = temp ->next;
    if(temp -> next) temp->next->prev = temp2;
    free(temp);
}

/*Length of Linked List*/
int get_length(struct DLLNode *head){
    struct DLLNode *curr = head;
    int length = 1;

    if (head == NULL) return 0;
    while(curr = curr -> next) ++length;
    return length;
}

/* Print the list */
void printList(struct DLLNode *head){
    if(head == NULL) return;
    struct DLLNode *curr = head;

    while(curr) {
        printf("%d ", curr -> data);
        curr = curr -> next;
    }
    printf("\n");
}

/*Show menu*/
void show_menu(){
    printf("\nSelect from the options\n");
    printf("[1] print the list\n");
    printf("[2] get length of list\n");
    printf("[3] insert at position\n");
    printf("[4] delete at position\n");
    printf("Press anything else to exit\n");
    printf(">>> ");
}

/* Driver Function */
int main(){
    struct DLLNode *head = NULL;
    int choice, position, data;

    do{
        show_menu();
        scanf("%d", &choice);

        switch (choice){
            case 1:
                printf("\nThe List is: ");
                printList(head);
                break;
            case 2:
                printf("\nThe length of the List is: %d", get_length(head));
                break;
            case 3:
                printf("\nEnter Position to insert and data ");
                scanf("%d %d", &position, &data);
                insert(&head, data, position);
                printf("\nInserted!");
                break;
            case 4:
                printf("\nEnter Position to delete ");
                scanf("%d", &position);
                delete(&head, position);
                printf("\nDeleted!");
                break;
            default:
                exit(0);
                break;
        }
    }while(1);
    return 0;
}
```

===

Output



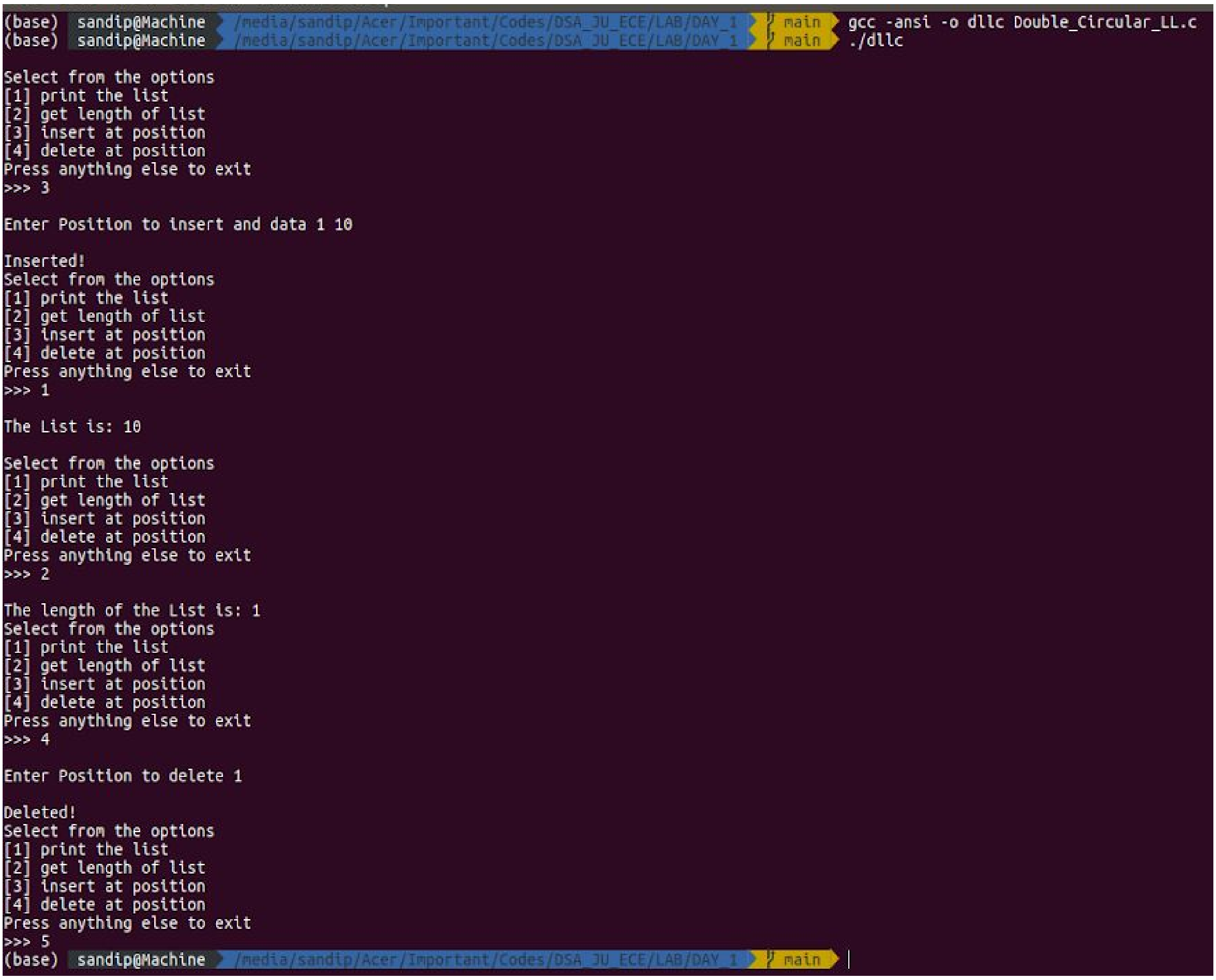
```
(base)  ✘ sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  gcc -ansi -o dll Double_LL.c
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  ./dll

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 3

Enter Position to insert and data 1 -1000

Inserted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 1

The List is: -1000

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 2

The length of the List is: 1
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 4

Enter Position to delete 1

Deleted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 7
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1  main ±  |
```

4. **Implement a doubly connected circular linked list in C. Your program should typically implement insert and delete at all possible locations with proper check(s) as applicable. Include a display function as well and use it to show the content of your list after every operation. Include calls to insert and delete from the main. (For Bonus)**

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

/* Here we have implemented Circular Doubly Linked List
using only a single pointer head. Some implementations use
a tail pointer for speeding up some routines.
We have tried all variations in this assignment */

/* Doubly linked list node */
struct Node{
      int data;
      struct Node* next;
      struct Node* prev;
};

/* Global Head Node */
struct Node* head = NULL;

/* For printing list */
void printList(){
    struct Node* curr = head;
    if(head!=NULL && head -> next != NULL && head -> prev != NULL) {
        do{
            printf("%d ", curr->data);
            curr = curr->next;
        }while(curr != head);
    }
    printf("\n");
}

/* Returns the length of the linked list */
int getLength(){
    struct Node* curr = head;
    int count=0;
    if(head!=NULL&& head -> next != NULL && head -> prev != NULL){
        do{
            count++;
            curr = curr->next;
        }while(curr!=head);
    }
    return count;
}

/* For Deleting a node from the list */
void deleteAtPosition(int position){
    int length = getLength();
    int j;
    struct Node *temp = head, *temp2 = NULL;

    if(length == 0 || position > length){
        printf("INVALID POSITION\n");
        return;
    }

      if(position == 1){
        /* Delete head. Get the pointer pointing
        to head and swap */
        temp = temp -> prev;
        temp -> next = head -> next;
        head->next->prev = temp;

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

    /* Middle or end deletion */
    for(j = 0;j < length - 2;++j) temp = temp -> next;

    temp2 = temp->next;
    temp -> next = temp2 -> next;
    temp2 -> next -> prev = temp;
    free(temp2);
}

/* Insertion at beginning of node.
Since the insert function becomes very large
We have broken it up into 3 parts */
void insertAtBeginning(int data){
    struct Node *newNode, *prevNode;

    /* Allocate a new Node */
    newNode = (struct Node*)malloc(sizeof(struct Node*));
    newNode -> data = data;
    newNode -> next = newNode;
    newNode -> prev = newNode;

    /* No node */
    if(head == NULL){
        head = newNode;
        newNode -> next = head;
        return;
    }

    prevNode = head -> prev;
    newNode->next = head;
    prevNode->next = newNode;
    head = newNode;
}

/* End insertion */
void insertAtEnd(int data){
    struct Node *newNode, *prevNode;

    /* Allocate a new Node */
    newNode = (struct Node*)malloc(sizeof(struct Node*));
    newNode -> data = data;
    newNode -> next = newNode;
    newNode -> prev = newNode;

    if(head == NULL){
        head = newNode;
        newNode -> next = head;
        return;
    }

    prevNode = head -> prev;
    newNode->next = prevNode->next;
    prevNode->next = newNode;
    newNode->prev = prevNode;
    newNode->next->prev = newNode;
    return;
}

/* Insert at any general position */
void insertAtPosition(int position, int data){
    int length = getLength();
      struct Node *newNode, *prevNode;
    int j;

      if(position >= length + 2){
           printf("\nINVALID POSITION\n");
           return;
      }

      if(position == 1){
        insertAtBeginning(data);
        return;
    }
    else if(position == length){
```

```c
        insertAtEnd(data);
        return;
    }

    /* Allocate a new Node */
    newNode = (struct Node*)malloc(sizeof(struct Node*));
    newNode -> data = data;
    newNode -> next = newNode;
    newNode -> prev = newNode;

    for(j = 0;j < length - 2; j++) prevNode = prevNode->next;

    newNode->next = prevNode->next;
    prevNode->next = newNode;
    newNode->prev = prevNode;
    return;
}

/*Show menu*/
void show_menu(){
    printf("\nSelect from the options\n");
    printf("[1] print the list\n");
    printf("[2] get length of list\n");
    printf("[3] insert at position\n");
    printf("[4] delete at position\n");
    printf("Press anything else to exit\n");
    printf(">>> ");
}

/* Driver Function */
int main(){
    int choice, position, data;

    do{
        show_menu();
        scanf("%d", &choice);

        switch (choice){
            case 1:
                printf("\nThe List is: ");
                printList(head);
                break;
            case 2:
                printf("\nThe length of the List is: %d", getLength(head));
                break;
            case 3:
                printf("\nEnter Position to insert and data ");
                scanf("%d %d", &position, &data);
                insertAtPosition(position, data);
                printf("\nInserted!");
                break;
            case 4:
                printf("\nEnter Position to delete ");
                scanf("%d", &position);
                deleteAtPosition(position);
                printf("\nDeleted!");
                break;
            default:
                exit(0);
                break;
        }
    }while(1);
    return 0;
}
================================================================================================
```

Outputs



```
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1   main  gcc -ansi -o dllc Double_Circular_LL.c
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1   main  ./dllc

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 3

Enter Position to insert and data 1 10

Inserted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 1

The List is: 10

Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 2

The length of the List is: 1
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 4

Enter Position to delete 1

Deleted!
Select from the options
[1] print the list
[2] get length of list
[3] insert at position
[4] delete at position
Press anything else to exit
>>> 5
(base)  sandip@Machine  /media/sandip/Acer/Important/Codes/DSA_JU_ECE/LAB/DAY_1   main 
```