Output:

```
Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice: 8
Enter value to insert in sorted list: 1

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete Alternate
8. Insert sorted
9. Exit
Enter choice: 1
Enter value to and element to insert before: 2 1
```

(P.T.O.)

Date

Expt. No. Page No.

Lab Sheet 4

1. Write a menu-driven C program using structures to implement the following operations on a singly linked list:

1. Insert an element before a specified element
2. Insert an element after a specified element
3. Delete a specified element
4. Traverse the list & display all the elements
5. Reverse the linked list
6. Sort the list in ascending order
7. Delete every alternate node from the list
8. Insert an element into a sorted linked list while maintaining the ~~sto~~ sorted order

Requirements: Use dynamic memory allocation (malloc / free) for node creation & deletion.

Source Code:

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

struct Node {
    int data;
    struct Node* next;
};
```

Teacher's Signature

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice : 4
Linked list: 2 → 1 → NULL

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice : 5

(P.T.O.)

Date _____

Expt. No. _____ Page No. _____

```
struct Node *head = NULL;

struct Node * createNode(int data) {
    struct Node * newNode = (struct Node *) malloc
(sizeof(struct Node));
  newNode -> data = data;
  newNode -> next = NULL;
  return newNode;
}

void beforesert(int val, int before) {
  struct Node * newNode = createNode(val);
  if(head == NULL || head -> data == before) {
      newNode -> next = head;
      head = newNode;
      return;
  }
  struct Node *curr = head;
  while(curr -> next != NULL && curr -> next -> data
!= before) curr = curr -> next;

  if(curr -> next != NULL) {
    newNode -> next = curr -> next;
    curr -> next = newNode;
  }
}

void aftersert(int val, int after) {
    struct Node *curr = head;
```

Teacher's Signature _____

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice: 4
Linked list: 1 → 2 → NULL

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice: 9

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice: 4
Linked list: 1 → 2 → NULL

Menu:
1. Insert before
2. Insert after
3. Delete
4. Display
5. Reverse
6. Sort
7. Delete alternate
8. Insert sorted
9. Exit
Enter choice: 9

Date _____

Expt. No. _____ Page No. _____

```
    while (curr != NULL && curr -> data != after)
    curr = curr -> next;

    if (curr != NULL) {
        struct Node *newNode = createNode(val);
        newNode -> next = curr -> next;
        curr -> next = newNode;
    }
}

void delete (int val) {
    if (head == NULL) return;
    if (head -> data == val) {
        struct Node *temp = head;
        head = head -> next;
        free (temp);
        return;
    }
    struct Node *curr = head;
    while (curr -> next != NULL && curr -> next ->
data != val) curr = curr -> next;

    if (curr -> next != NULL) {
        struct Node *temp = curr -> next;
        curr -> next = temp -> next;
        free (temp);
    }
}
```

Teacher's Signature _____

Date _____

Expt. No. _____ Page No. _____

```
void display () {
    if (head == NULL) {
        printf ("List is empty \n");
        return;
    }
    struct Node *curr = head;
    printf ("Linked list: ");
    while (curr != NULL) {
        printf ("%d -> ", curr -> data);
        curr = curr -> next;
    }
    printf ("NULL\n");
}

void reverse () {
    struct Node *prev = NULL, *curr = head, *next
= NULL;
    while (curr != NULL) {
        next = curr -> next;
        curr -> next = prev;
        prev = curr;
        curr = next;
    }
    head = prev;
}

void sort() {
    if (head == NULL) return;
    int swapped, temp;
```

Teacher's Signature _____

```
struct Node *ptr, *lptr = NULL;
do {
  swapped = 0;
  ptr = head;
  while (ptr -> next != lptr) {
    if (ptr -> data > ptr -> next -> data) {
      temp = ptr -> data;
      ptr -> data = ptr -> next -> data;
      ptr -> next -> data = temp;
      swapped = 1;
    }
    ptr = ptr -> next;
  }
  lptr = ptr;
} while (swapped);
}

void delete_alternate() {
  if(head == NULL) return;
  struct Node *curr = head -> next, *prev = head;
  while (curr != NULL) {
    prev -> next = curr -> next;
    free (curr);
    prev = prev -> next;
    if (prev == NULL) break;
    curr = prev -> next;
  }
}
```

Teacher's Signature _____

```
void sortsert(int val) {
    struct Node *newNode = createNode(val);
    if(head == NULL || head -> data >= val) {
        newNode -> next = head;
        head = newNode;
        return;
    }
    struct Node *curr = head;
    while(curr -> next != NULL && curr -> next
-> data < val)
        curr = curr -> next;
    newNode -> next = curr -> next;
    curr -> next = newNode;
}

int main() {
    int choice, val, key, i;
    while(1) {
        printf("\n Menu: \n");
        printf("1. Insert before \n 2. Insert after \n
3. Delete \n 4. Display \n 5. Reverse \n 6. Sort \n 7. De-
-lete alternate \n 8. Insert sorted \n 9. Exit \n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter value & element to insert
before: ");
                scanf("%d %d", &val, &key);
```

Teacher's Signature _____

```
            beforesert(val, key);
            break;
        case 2:
            printf("Enter value and element to insert
after: ");
            scanf("%d %d", &val, &key);
            aftersert(val, key);
            break;
        case 3:
            printf("Enter value to delete: ");
            scanf("%d", &val);
            delete(val);
            break;
        case 4:
            display();
            break;
        case 5:
            reverse();
            break;
        case 6:
            sort();
            break;
        case 7:
            delete_alternate();
            break;
        case 8:
            printf("Enter value to insert in sorted
list: ");
            scanf("%d", &val);
```

Teacher's Signature _____

```
            sortsort(val);
            break;
        case 9:
            exit(0);
        default:
            printf("Invalid choice\n");
    }
}
return 0;
}
```

Teacher's Signature _____