CS102 - Lab 8 - 29/02/2024

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 8
//PROGRAM 1
//Date : 29/02/2024

/*
1.Write a C program to check whether the two doubly linked lists are
equivalent or not.
Note: Two doubly linked lists are said to be equivalent when the number of
nodes, elements in
nodes and the sequence of elements are all the same in both the linked
lists.
*/

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

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

node* create_node(int item) {
   node *new_node = (node*)malloc(sizeof(node));
   if (new_node == NULL) {
       printf("Memory allocation failed\n");
       exit(EXIT_FAILURE);
   }
   new_node->data = item;
   new_node->prev = NULL;
   new_node->next = NULL;
   return new_node;
}

void insert_end(node **head, int item) {
```

```
    node *new_node = create_node(item);

    if (*head == NULL) {
        *head = new_node;
        return;
    }

    node *p = *head;
    while (p->next != NULL) {
        p = p->next;
    }

    p->next = new_node;
    new_node->prev = p;
}

void display(node *head) {
    printf("Doubly Linked List:\n");
    while (head != NULL) {
        printf("%d -> ", head->data);
        head = head->next;
    }
    printf("NULL\n");
}

int check_equivalent(node *head1, node *head2) {
    while (head1 != NULL && head2 != NULL) {
        if (head1->data != head2->data) {
            return 0;
        }

        head1 = head1->next;
        head2 = head2->next;
    }

    return (head1 == NULL && head2 == NULL);
}

int main() {
    node *dll1 = NULL;
```

```c
    node *dll2 = NULL;

    insert_end(&dll1, 10);
    insert_end(&dll1, 20);
    insert_end(&dll1, 30);

    insert_end(&dll2, 10);
    insert_end(&dll2, 20);
    insert_end(&dll2, 30);
    insert_end(&dll2, 30);

    display(dll1);
    display(dll2);

    if (check_equivalent(dll1, dll2)) {
        printf("Doubly linked lists are equivalent.\n");
    } else {
        printf("Doubly linked lists are not equivalent.\n");
    }

    return 0;
}
```

OUTPUT1:

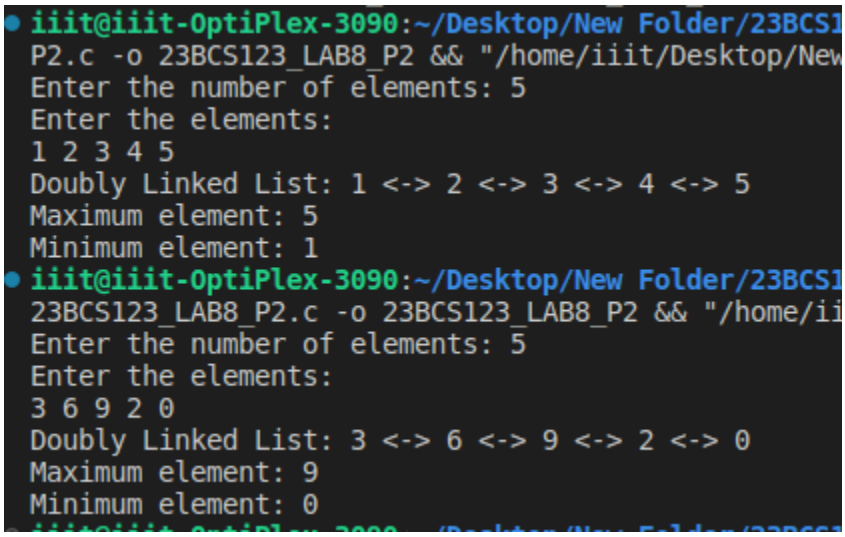
```
 23BCS123_LAB8_P1.c -o 23BCS123_LAB8_P1 &&
Doubly Linked List:
10 -> 20 -> 30 -> NULL
Doubly Linked List:
10 -> 20 -> 30 -> NULL
Doubly linked lists are equivalent.
iiit@iiit-OptiPlex-3090:~/Desktop/New Folde
23BCS123_LAB8_P1.c -o 23BCS123_LAB8_P1 &&
Doubly Linked List:
10 -> 20 -> 30 -> NULL
Doubly Linked List:
10 -> 20 -> 30 -> 30 -> NULL
Doubly linked lists are not equivalent.
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 8
//PROGRAM 2
//Date : 29/02/2024

/*
2.Write a C program to find the maximum and minimum elements in a doubly linked list.
*/


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

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

node* create_node(int data) {
   node* new_node = (node*)malloc(sizeof(node));
   if (new_node == NULL) {
       printf("Memory allocation failed\n");
       exit(EXIT_FAILURE);
   }
   new_node->data = data;
   new_node->prev = NULL;
   new_node->next = NULL;
   return new_node;
}

void insert_end(node** head, int data) {
   node* new_node = create_node(data);

   if (*head == NULL) {
       *head = new_node;
       return;
```

```
    }

    node* p = *head;
    while (p->next != NULL) {
        p = p->next;
    }

    p->next = new_node;
    new_node->prev = p;
}

void find_max_min(node* head, int* max, int* min) {
    if (head == NULL) {
        printf("Doubly linked list is empty\n");
        return;
    }

    *max = *min = head->data;
    node* p = head;

    while (p != NULL) {
        if (p->data > *max) {
            *max = p->data;
        }

        if (p->data < *min) {
            *min = p->data;
        }

        p = p->next;
    }
}

void free_list(node* head) {
    while (head != NULL) {
        node* temp = head;
        head = head->next;
        free(temp);
    }
}
```

```c
int main() {
    node* head = NULL;
    int max, min, n, data;

    printf("Enter the number of elements: ");
    scanf("%d", &n);

    if (n <= 0) {
        printf("Invalid input. Please enter a positive integer.\n");
        return 1;
    }

    printf("Enter the elements:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &data);
        insert_end(&head, data);
    }

    find_max_min(head, &max, &min);

    printf("Doubly Linked List: ");
    node* p = head;
    while (p != NULL) {
        printf("%d", p->data);
        if (p->next != NULL) {
            printf(" <-> ");
        }
        p = p->next;
    }
    printf("\n");

    printf("Maximum element: %d\n", max);
    printf("Minimum element: %d\n", min);

    free_list(head);

    return 0;
}
```

OUTPUT2:


iiit@iiit-OptiPlex-3090:~/Desktop/New Folder/23BCS1
P2.c -o 23BCS123_LAB8_P2 && "/home/iiit/Desktop/New
Enter the number of elements: 5
Enter the elements:
1 2 3 4 5
Doubly Linked List: 1 <-> 2 <-> 3 <-> 4 <-> 5
Maximum element: 5
Minimum element: 1
iiit@iiit-OptiPlex-3090:~/Desktop/New Folder/23BCS1
23BCS123_LAB8_P2.c -o 23BCS123_LAB8_P2 && "/home/ii
Enter the number of elements: 5
Enter the elements:
3 6 9 2 0
Doubly Linked List: 3 <-> 6 <-> 9 <-> 2 <-> 0
Maximum element: 9
Minimum element: 0

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 8
//PROGRAM 3
//Date : 29/02/2024

/*
3.Write a C program to implement the operations of a double ended queue
using a doubly linked list.
*/

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

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

typedef struct Deque {
   node *front;
   node *rear;
} Deque;

node* create_node(int data) {
   node* new_node = (node*)malloc(sizeof(node));
   if (new_node == NULL) {
       printf("Memory allocation failed\n");
       exit(EXIT_FAILURE);
   }
   new_node->data = data;
   new_node->prev = NULL;
   new_node->next = NULL;
   return new_node;
}

Deque* init_Deque() {
   Deque* deque = (Deque*)malloc(sizeof(Deque));
```

```
    if (deque == NULL) {
        printf("Memory allocation failed\n");
        exit(EXIT_FAILURE);
    }
    deque->front = NULL;
    deque->rear = NULL;
    return deque;
}

int is_empty(Deque* deque) {
    return (deque->front == NULL);
}

void insert_front(Deque* deque, int data) {
    node* new_node = create_node(data);
    if (is_empty(deque)) {
        deque->front = deque->rear = new_node;
    } else {
        new_node->next = deque->front;
        deque->front->prev = new_node;
        deque->front = new_node;
    }
    printf("Inserted %d at the front of the deque\n", data);
}

void insert_rear(Deque* deque, int data) {
    node* new_node = create_node(data);
    if (is_empty(deque)) {
        deque->front = deque->rear = new_node;
    } else {
        new_node->prev = deque->rear;
        deque->rear->next = new_node;
        deque->rear = new_node;
    }
    printf("Inserted %d at the rear of the deque\n", data);
}

void delete_front(Deque* deque) {
    if (is_empty(deque)) {
        printf("Deque Underflow\n");
```

```
        return;
    }

    node* temp = deque->front;
    if (deque->front == deque->rear) {
        deque->front = deque->rear = NULL;
    } else {
        deque->front = deque->front->next;
        deque->front->prev = NULL;
    }
    free(temp);
    printf("Deleted element from the front of the deque\n");
}

void delete_rear(Deque* deque) {
    if (is_empty(deque)) {
        printf("Deque Underflow\n");
        return;
    }

    node* temp = deque->rear;
    if (deque->front == deque->rear) {
        deque->front = deque->rear = NULL;
    } else {
        deque->rear = deque->rear->prev;
        deque->rear->next = NULL;
    }
    free(temp);
    printf("Deleted element from the rear of the deque\n");
}

void display(Deque* deque) {
    if (is_empty(deque)) {
        printf("Deque is empty\n");
        return;
    }

    node* p = deque->front;
    printf("Deque elements: ");
    while (p != NULL) {
```

```
        printf("%d ", p->data);
        p = p->next;
    }
    printf("\n");
}

void free_Deque(Deque* deque) {
    node* p = deque->front;
    node* next;

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

    free(deque);
}

int main() {
    Deque* deque = init_Deque();
    int choice, data;

    while (1) {
        printf("\n1. Insert at front 2. Insert at rear 3. Delete from front
4. Delete from rear 5. Display 6. Quit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter the element to insert at the front: ");
                scanf("%d", &data);
                insert_front(deque, data);
                break;
            case 2:
                printf("Enter the element to insert at the rear: ");
                scanf("%d", &data);
                insert_rear(deque, data);
                break;
```

```
            case 3:
                delete_front(deque);
                break;
            case 4:
                delete_rear(deque);
                break;
            case 5:
                display(deque);
                break;
            case 6:
                free_Deque(deque);
                exit(0);
            default:
                printf("Invalid choice\n");
        }
    }

    return 0;
}
```

OUTPUT3:

```
1. Insert at front 2. Insert at rear 3. Delete from front 4. Delete from rear 5. Display 6. Quit
Enter your choice: 2
Enter the element to insert at the rear: 5
Inserted 5 at the rear of the deque

1. Insert at front 2. Insert at rear 3. Delete from front 4. Delete from rear 5. Display 6. Quit
Enter your choice: 5
Deque elements: 1 2 4 5

1. Insert at front 2. Insert at rear 3. Delete from front 4. Delete from rear 5. Display 6. Quit
Enter your choice: 3
Deleted element from the front of the deque

1. Insert at front 2. Insert at rear 3. Delete from front 4. Delete from rear 5. Display 6. Quit
Enter your choice: 4
Deleted element from the rear of the deque

1. Insert at front 2. Insert at rear 3. Delete from front 4. Delete from rear 5. Display 6. Quit
Enter your choice: 5
Deque elements: 2 4
```

```c
//SHIVAMBU DEV PANDEY
//23BCS123
//CSE B
//LAB 8
//PROGRAM 4
//Date : 29/02/2024

/*
4.Write a C program to delete the odd numbered nodes from a singly linked
list.
*/

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

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

void traverse(node *head) {
   if (head == NULL) {
       printf("\nLinked List is empty.\n");
       return;
   }

   int count = 1;
   node *p = head;
   while (p != NULL) {
       printf("\nNode_%d _data: %d", count, p->data);
       count++;
       p = p->next;
   }
}

void insert_end(node **head, int item) {
   node *new_node = (node*) malloc(sizeof(node));
   new_node->data = item;
```

```
    new_node->next = NULL;
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    node *p = *head;
    while(p->next != NULL) {
        p = p->next;
    }
    p->next = new_node;
}

void delete_odd_nodes(node **head) {
    if (*head == NULL) {
        printf("\nLinked List is empty. No nodes to delete.\n");
        return;
    }

    node *p = *head, *q = NULL;
    int count = 1;

    while (p != NULL) {
        if (count % 2 != 0) {
            if (q != NULL) {
                q->next = p->next;
                free(p);
                p = q->next;
            } else {
                *head = p->next;
                free(p);
                p = *head;
            }
        } else {
            q = p;
            p = p->next;
        }
        count++;
    }

    printf("\nOdd-numbered nodes deleted successfully.\n");
```

```c
}

int main() {
   node *head = NULL;
   int choice, item;

   while(1) {
       printf("\nSingly Linked List\n");
       printf("1. Insert at end   ");
       printf("2. Delete odd nodes ");
       printf("3. Traverse ");
       printf("4. Exit\n");
       printf("Enter your choice: ");
       scanf("%d", &choice);

       switch(choice) {
           case 1:
               printf("Enter the item to be inserted at the end: ");
               scanf("%d", &item);
               insert_end(&head, item);
               break;
           case 2:
               delete_odd_nodes(&head);
               traverse(head);
               break;
           case 3:
               traverse(head);
               break;
           case 4:
               exit(0);
           default:
               printf("Invalid choice\n");
       }
   }

   return 0;
}
```

OUTPUT4:

```
Singly Linked List
1. Insert at end   2. Delete odd nodes 3. Traverse 4. Exit
Enter your choice: 3

Node_1 _data: 1
Node_2 _data: 2
Node_3 _data: 3
Node_4 _data: 4
Node_5 _data: 5
Singly Linked List
1. Insert at end   2. Delete odd nodes 3. Traverse 4. Exit
Enter your choice: 2

Odd-numbered nodes deleted successfully.

Node_1 _data: 2
Node_2 _data: 4
Singly Linked List
1. Insert at end   2. Delete odd nodes 3. Traverse 4. Exit
Enter your choice: 2

Odd-numbered nodes deleted successfully.

Node_1 _data: 4
```