Practice 3

Practice 3.1

Aim: Write a C program for Single linked list:which perform different operations in single linked list.

Program:

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

// Node structure
struct node
{
   int data;
   struct node* next;
};

// Global pointers
struct node *first = NULL;
struct node *last = NULL;
int count = 0,i;

// Function prototypes
void create();
void display();
void insert();
void deleteNode();
void search();
void replace();

int main()
{
   int ch;
  while (1)
  {
      printf("\n1. Create\n2. Display\n3. Insert\n4. Delete\n5. Search\n6.
Replace\n7. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch)
 {
         case 1: create(); break;
         case 2: display(); break;
         case 3: insert(); break;
         case 4: deleteNode(); break;
```

```
            case 5: search(); break;
            case 6: replace(); break;
            case 7: exit(0);
            default: printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}

// Function to create a new node at the end
void create() {
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter data: ");
    scanf("%d", &newnode->data);

    newnode->next = NULL;

    if (first == NULL) {
        first = last = newnode;
    } else {
        last->next = newnode;
        last = newnode;
    }
    count++;
}

// Function to display the linked list
void display() {
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct node* temp = first;
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to insert a node at a given position
void insert() {
    int pos, x;
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter position to insert (1-%d): ", count + 1);
    scanf("%d", &pos);
    printf("Enter data: ");
    scanf("%d", &x);

    newnode->data = x;
    newnode->next = NULL;

    if (pos < 1 || pos > count + 1) {
        printf("Invalid position!\n");
        free(newnode);
        return;
    }

    if (pos == 1) {
        newnode->next = first;
        first = newnode;
    } else {
        struct node* temp = first;
        int i;
        for (i = 1; i < pos - 1; i++) {
            temp = temp->next;
        }
        newnode->next = temp->next;
        temp->next = newnode;
        if (pos == count + 1) {
            last = newnode;
        }
    }
    count++;
```

```
}

// Function to delete a node at a given position
void deleteNode() {
    int pos;
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter position to delete (1-%d): ", count);
    scanf("%d", &pos);

    if (pos < 1 || pos > count) {
        printf("Invalid position!\n");
        return;
    }

    struct node* temp = first;
    if (pos == 1) {
        first = first->next;
        free(temp);
    } else {
        struct node* prev = NULL;
        for (i = 1; i < pos; i++) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = temp->next;
        if (pos == count) {
            last = prev;
        }
        free(temp);
    }
    count--;
}

// Function to search for an element in the linked list
void search()
{
    int key, found = 0;
   if (first == NULL)
   {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to search: ");
    scanf("%d", &key);

    struct node* temp = first;
    int i = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Value %d found at position %d\n", key, i);
            found = 1;
        }
        temp = temp->next;
        i++;
    }

    if (!found) {
        printf("Value %d not found in the list!\n", key);
    }
}

// Function to replace a node's data with a new value
void replace()
{
    int key, new_value, found = 0;
   if (first == NULL)
   {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to replace: ");
    scanf("%d", &key);
    printf("Enter new value: ");
    scanf("%d", &new_value);

    struct node* temp = first;
   while (temp != NULL)
   {
        if (temp->data == key)
        {
            temp->data = new_value;
            printf("Replaced %d with %d\n", key, new_value);
            found = 1;
        }
        temp = temp->next;
    }

    if (!found)
{
        printf("Value %d not found in the list!\n", key);
    }
}
```

OUTPUT:

```
1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 10

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 20

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
```

```
Linked List: 10 -> 20 -> NULL

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 3
Enter position to insert (1-3): 2
Enter data: 15

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Linked List: 10 -> 15 -> 20 -> NULL

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 4
Enter position to delete (1-3): 1

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Linked List: 15 -> 20 -> NULL
```

```
1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 5
Enter value to search: 20
Value 20 found at position 2

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 6
Enter value to replace: 15
Enter new value: 5

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Linked List: 5 -> 20 -> NULL
```

Practice 3.2
Aim: Write a C program for Reversing a Single linked list.

Program:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
struct node *first = NULL;
struct node *last = NULL;
// Function prototypes
void create();
void reverse();
void display();
int main()
{
   int ch;
   while (1)
         {
      printf("\n1. Create\n2. Display\n3. Reverse\n4. Exit\n");  // Fixed
formatting issue
      printf("Enter your choice: ");
      scanf("%d", &ch);
      switch (ch)
                {
         case 1: create(); break;
         case 2: display(); break;
         case 3: reverse(); break;
         case 4: exit(0);  // Exit condition
         default: printf("Invalid choice\n"); break;
       }
   }
   return 0;
}

// Function to create a new node and append it to the list
void create()
 {
   int d;
   struct node *newnode = (struct node*)malloc(sizeof(struct node));
   printf("Enter data: ");
   scanf("%d", &d);
   newnode->data = d;
```

```
    newnode->next = NULL;
    if (first == NULL)
        first = last = newnode;
    else
          {
        last->next = newnode;
        last = newnode;
    }
}

// Function to display the linked list
void display() {
    struct node *temp = first;
    printf("Linked List:\n ");
    while (temp != NULL)
          {
        printf("%d->", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Function to reverse the linked list
void reverse()
{
    struct node *current = first, *next = NULL, *prev = NULL;
    while (current != NULL)
          {
        next = current->next;  // Save the next node
        current->next = prev;  // Reverse the link
        prev = current;        // Move prev forward
        current = next;        // Move current forward
    }
    first = prev;  // Update head of the reversed list
    printf("List reversed successfully.\n");
}
```

OUTPUT:

```
1. Create
2. Display
3. Reverse
4. Exit
Enter your choice: 1
Enter data: 10

1. Create
2. Display
3. Reverse
4. Exit
Enter your choice: 1
Enter data: 20

1. Create
2. Display
3. Reverse
4. Exit
Enter your choice: 1
Enter data: 30

1. Create
2. Display
3. Reverse
4. Exit
Enter your choice: 2
Linked List:
 10->20->30->NULL

1. Create
2. Display
3. Reverse
4. Exit
Enter your choice: 2
Linked List:
 10->20->30->NULL

1. Create
2. Display
3. Reverse
4. Exit
Enter your choice:4
```

Practice 3.3

Aim: Write a C program which perform different operations in double linked list.

Program:

```
#include <stdio.h>
#include <stdlib.h>
// Node structure for doubly linked list
struct node {
    int data;
    struct node* prev;
    struct node* next;
};
// Global pointers
struct node *head = NULL;
struct node *tail = NULL;
int count = 0;
// Function prototypes
void create();
void display();
void insert();
void deleteNode();
void search();
void replace();
int main() {
    int ch;
    while (1) {
        printf("\n1. Create\n2. Display\n3. Insert\n4. Delete\n5. Search\n6. 
Replace\n7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: create(); break;
            case 2: display(); break;
            case 3: insert(); break;
            case 4: deleteNode(); break;
            case 5: search(); break;
            case 6: replace(); break;
            case 7: exit(0);
            default: printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
```

```c
}

// Function to create a new node at the end
void create() {
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter data: ");
    scanf("%d", &newnode->data);

    newnode->prev = tail;
    newnode->next = NULL;

    if (head == NULL) {
        head = tail = newnode;
    } else {
        tail->next = newnode;
        tail = newnode;
    }
    count++;
}

// Function to display the doubly linked list
void display() {
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

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

// Function to insert a node at a given position
void insert() {
```

```
int pos, x;
struct node* newnode = (struct node*)malloc(sizeof(struct node));
if (newnode == NULL) {
    printf("Memory allocation failed!\n");
    return;
}

printf("Enter position to insert (1-%d): ", count + 1);
scanf("%d", &pos);
printf("Enter data: ");
scanf("%d", &x);

newnode->data = x;

if (pos < 1 || pos > count + 1) {
    printf("Invalid position!\n");
    free(newnode);
    return;
}

if (pos == 1) {
    newnode->prev = NULL;
    newnode->next = head;
    if (head != NULL) {
        head->prev = newnode;
    }
    head = newnode;
    if (tail == NULL) {
        tail = newnode;
    }
} else if (pos == count + 1) {
    newnode->prev = tail;
    newnode->next = NULL;
    tail->next = newnode;
    tail = newnode;
} else {
    struct node* temp = head;
    for (int i = 1; i < pos - 1; i++) {
        temp = temp->next;
    }
    newnode->prev = temp;
    newnode->next = temp->next;
    temp->next->prev = newnode;
```

```
        temp->next = newnode;
    }
    count++;
}

// Function to delete a node at a given position
void deleteNode() {
    int pos;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter position to delete (1-%d): ", count);
    scanf("%d", &pos);

    if (pos < 1 || pos > count) {
        printf("Invalid position!\n");
        return;
    }

    struct node* temp = head;
    if (pos == 1) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        } else {
            tail = NULL;
        }
        free(temp);
    } else if (pos == count) {
        temp = tail;
        tail = tail->prev;
        tail->next = NULL;
        free(temp);
    } else {
        for (int i = 1; i < pos; i++) {
            temp = temp->next;
        }
        temp->prev->next = temp->next;
        temp->next->prev = temp->prev;
        free(temp);
    }
    count--;
}

// Function to search for an element in the doubly linked list
void search() {
    int key, found = 0;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to search: ");
    scanf("%d", &key);

    struct node* temp = head;
    int i = 1;
    while (temp != NULL) {
        if (temp->data == key) {
            printf("Value %d found at position %d\n", key, i);
            found = 1;
        }
        temp = temp->next;
        i++;
    }

    if (!found) {
        printf("Value %d not found in the list!\n", key);
    }
}

// Function to replace a node's data with a new value
void replace() {
    int key, new_value, found = 0;
    if (head == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to replace: ");
    scanf("%d", &key);
    printf("Enter new value: ");
    scanf("%d", &new_value);
    struct node* temp = head;
    while (temp != NULL) {
        if (temp->data == key) {
            temp->data = new_value;
            printf("Replaced %d with %d\n", key, new_value);
            found = 1;
        }
        temp = temp->next;
    }

   if (!found)
   {
        printf("Value %d not found in the list!\n", key);
    }
}
```

OUTPUT:

```
1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 10

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 20

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
```

```
7. Exit
Enter your choice: 2
Doubly Linked List: 10 <-> 20 <-> NULL

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 3
Enter position to insert (1-3): 2
Enter data: 15

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Doubly Linked List: 10 <-> 15 <-> 20 <-> NULL

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 4
Enter position to delete (1-3): 1

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
```

```
Doubly Linked List: 15 <-> 20 <-> NULL

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 5
Enter value to search: 20
Value 20 found at position 2

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 6
Enter value to replace: 15
Enter new value: 5

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Doubly Linked List: 5 <-> 20 <-> NULL
```

Practice 3.4
Aim: Write a C program for Circular linked list.
Program:
```
#include <stdio.h>
#include <stdlib.h>

// Node structure
struct node {
    int data;
    struct node* next;
};

// Global pointers
struct node *first = NULL;
struct node *last = NULL;
int count = 0;

// Function prototypes
void create();
void display();
void insert();
void deleteNode();
void search();
void replace();

int main() {
    int ch;
    while (1) {
        printf("\n1. Create\n2. Display\n3. Insert\n4. Delete\n5. Search\n6.
Replace\n7. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &ch);

        switch (ch) {
            case 1: create(); break;
            case 2: display(); break;
            case 3: insert(); break;
            case 4: deleteNode(); break;
            case 5: search(); break;
            case 6: replace(); break;
            case 7: exit(0);
            default: printf("Invalid choice! Try again.\n");
        }
    }
    return 0;
}

// Function to create a new node at the end
void create() {
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter data: ");
    scanf("%d", &newnode->data);

    newnode->next = NULL;

    if (first == NULL) {
        first = last = newnode;
        last->next = first; // Making it circular
    } else {
        last->next = newnode;
        last = newnode;
        last->next = first; // Maintain circular link
    }
    count++;
}

// Function to display the circular linked list
void display() {
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    struct node* temp = first;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", temp->data);
        temp = temp->next;
    } while (temp != first);
    printf("(Back to %d)\n", first->data);
}
```

```c
// Function to insert a node at a given position
void insert() {
    int pos, x;
    struct node* newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }

    printf("Enter position to insert (1-%d): ", count + 1);
    scanf("%d", &pos);
    printf("Enter data: ");
    scanf("%d", &x);

    newnode->data = x;

    if (pos < 1 || pos > count + 1) {
        printf("Invalid position!\n");
        free(newnode);
        return;
    }

    if (pos == 1) {  // Insert at beginning
        newnode->next = first;
        first = newnode;
        last->next = first; // Update last node's next pointer
    } else {
        struct node* temp = first;
        for (int i = 1; i < pos - 1; i++) {
            temp = temp->next;
        }
        newnode->next = temp->next;
        temp->next = newnode;

        if (pos == count + 1) {  // Insert at end
            last = newnode;
            last->next = first;
        }
    }
    count++;
}
```

```
// Function to delete a node at a given position
void deleteNode() {
    int pos;
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter position to delete (1-%d): ", count);
    scanf("%d", &pos);

    if (pos < 1 || pos > count) {
        printf("Invalid position!\n");
        return;
    }

    struct node* temp = first;
    if (pos == 1) { // Delete first node
        if (first == last) { // Only one node
            free(first);
            first = last = NULL;
        } else {
            first = first->next;
            last->next = first;
            free(temp);
        }
    } else {
        struct node* prev = NULL;
        for (int i = 1; i < pos; i++) {
            prev = temp;
            temp = temp->next;
        }
        prev->next = temp->next;
        if (pos == count) { // If deleting last node
            last = prev;
            last->next = first;
        }
        free(temp);
    }
    count--;
}

// Function to search for an element in the circular linked list
```

```
void search() {
    int key, found = 0;
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to search: ");
    scanf("%d", &key);

    struct node* temp = first;
    int i = 1;
    do {
        if (temp->data == key) {
            printf("Value %d found at position %d\n", key, i);
            found = 1;
        }
        temp = temp->next;
        i++;
    } while (temp != first);

    if (!found) {
        printf("Value %d not found in the list!\n", key);
    }
}

// Function to replace a node's data with a new value
void replace()
{
    int key, new_value, found = 0;
    if (first == NULL) {
        printf("List is empty!\n");
        return;
    }

    printf("Enter value to replace: ");
    scanf("%d", &key);
    printf("Enter new value: ");
    scanf("%d", &new_value);

    struct node* temp = first;
   do
   {
        if (temp->data == key)
        {
            temp->data = new_value;
            printf("Replaced %d with %d\n", key, new_value);
            found = 1;
        }
        temp = temp->next;
    } while (temp != first);

   if (!found)
   {
        printf("Value %d not found in the list!\n", key);
    }
}
```

OUTPUT:

```
1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 10

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 1
Enter data: 20

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
```

```
Enter your choice: 2
Circular Linked List: 10 -> 20 -> (Back to 10)

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 3
Enter position to insert (1-3): 2
Enter data: 15

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Circular Linked List: 10 -> 15 -> 20 -> (Back to 10)

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 4
Enter position to delete (1-3): 1

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Circular Linked List: 15 -> 20 -> (Back to 15)
```

```
1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 5
Enter value to search: 20
Value 20 found at position 2

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 6
Enter value to replace: 15
Enter new value: 5

1. Create
2. Display
3. Insert
4. Delete
5. Search
6. Replace
7. Exit
Enter your choice: 2
Circular Linked List: 5 -> 20 -> (Back to 5)
```