1. Implementation of Simple Linked List

Problem Statement:
Write a C program to create a simple linked list of three nodes and display the elements of the list.

Objective:
To understand the basic concept of linked lists, node creation, memory allocation, and pointer-based connections between nodes.

Program Code:
```c
#include <stdio.h>
#include <stdlib.h>

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

void printLinkedlist(struct node *p) {
  while (p != NULL) {
    printf("%d ", p->value);
    p = p->next;
  }
}

int main() {
  struct node *head;
  struct node *one = NULL;
  struct node *two = NULL;
  struct node *three = NULL;

  one = malloc(sizeof(struct node));
  two = malloc(sizeof(struct node));
  three = malloc(sizeof(struct node));

  one->value = 1;
  two->value = 2;
  three->value = 3;

  one->next = two;
  two->next = three;
  three->next = NULL;

  head = one;
  printLinkedlist(head);
}
```

Sample Output:

```
1 2 3
```

2. Linked List Operations

Problem Statement:

Write a C program to perform various linked list operations such as insertion, deletion, searching, and sorting.

Objective:

To understand different operations that can be performed on a linked list including insertion, deletion, searching, and sorting.

Program Code:

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

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

void insertAtBeginning(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}

void insertAfter(struct Node* prev_node, int new_data) {
  if (prev_node == NULL) {
    printf("the given previous node cannot be NULL");
    return;
  }
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = prev_node->next;
  prev_node->next = new_node;
}

void insertAtEnd(struct Node** head_ref, int new_data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  struct Node* last = *head_ref;

  new_node->data = new_data;
```

```c
  new_node->next = NULL;

  if (*head_ref == NULL) {
    *head_ref = new_node;
    return;
  }
  while (last->next != NULL) last = last->next;
  last->next = new_node;
}

void deleteNode(struct Node** head_ref, int key) {
  struct Node *temp = *head_ref, *prev;
  if (temp != NULL && temp->data == key) {
    *head_ref = temp->next;
    free(temp);
    return;
  }
  while (temp != NULL && temp->data != key) {
    prev = temp;
    temp = temp->next;
  }
  if (temp == NULL) return;
  prev->next = temp->next;
  free(temp);
}

int searchNode(struct Node** head_ref, int key) {
  struct Node* current = *head_ref;
  while (current != NULL) {
    if (current->data == key) return 1;
    current = current->next;
  }
  return 0;
}

void sortLinkedList(struct Node** head_ref) {
  struct Node *current = *head_ref, *index = NULL;
  int temp;
  if (head_ref == NULL) return;
  while (current != NULL) {
    index = current->next;
    while (index != NULL) {
      if (current->data > index->data) {
        temp = current->data;
        current->data = index->data;
        index->data = temp;
      }
      index = index->next;
    }
    current = current->next;
  }
}

void printList(struct Node* node) {
  while (node != NULL) {
    printf(" %d ", node->data);
    node = node->next;
  }
}

int main() {
  struct Node* head = NULL;
  insertAtEnd(&head, 1);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 3);
  insertAtEnd(&head, 4);
  insertAfter(head->next, 5);

  printf("Linked list: ");
  printList(head);

  printf("\nAfter deleting an element: ");
  deleteNode(&head, 3);
  printList(head);

  int item_to_find = 3;
  if (searchNode(&head, item_to_find)) {
    printf("\n%d is found", item_to_find);
  } else {
    printf("\n%d is not found", item_to_find);
  }

  sortLinkedList(&head);
  printf("\nSorted List: ");
  printList(head);
}
```

Sample Output:

```
Linked list: 3 2 5 1 4
After deleting an element: 2 5 1 4
3 is not found
Sorted List: 1 2 4 5
```