1. Create a node in a linked list which will have the following details of student 1. Name, roll number, class, section, an array having marks of any three subjects Create a linked list for 5 students and print it.

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

typedef struct node
{
   char name[50];
   int roll_no;
   int class;
   char section;
   int marks[3];
   struct node *link;
}student;

int main()
{
   student *head = NULL;
   printf("Enter details of 5 students:\n");
   for(int i=0; i<5; i++)
   {
      student *new = (student *)malloc(sizeof(student));
      printf("\nStudent %d\n", i+1);
      printf("Enter the name of student: ");
      scanf(" %[^\n]", new->name);
      printf("Enter the roll no: ");
      scanf(" %d",&new->roll_no);
      printf("Enter the class and section: ");
      scanf("%d %c", &new->class, &new->section);
      printf("Enter the marks of 3 subjects: \n");
      for(int j=0; j<3; j++)
```

```
    {
        printf("Subject %d: ", j+1);
        scanf("%d",&new->marks[j]);
    }
    new->link = NULL;

    if(head == NULL)
    {
        head = new;
    }
    else
    {
        student *temp = head;
        while(temp->link != NULL)
        {
            temp = temp->link;
        }
        temp->link = new;
    }

}

//Display details

printf("\nStudent details\n");
student *temp = head;
int i=1;
while(temp != NULL)
{
    printf("\nStudent %d\n", i);
    printf("Name: %s\n", temp->name);
```

```
        printf("Roll no: %d\n", temp->roll_no);
        printf("Class %d, Sec: %c\n", temp->class, temp->section);
        printf("Marks: %d %d %d\n", temp->marks[0], temp->marks[1], temp->marks[2]);
        i++;
        temp = temp->link;
    }
    return 0;
}
/*****************************************************************************

insert at frist
insert at last
insert at middle
*****************************************************************************/
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int data;
    struct node *link;
}s_ll;

void insert_first(s_ll **, int);
void insert_last(s_ll **, int);
void insert_middle(s_ll **, int, int, int);
void print_list(s_ll *);

s_ll* create_node(int );

int main()
```

```
{
   int op = 0, data, b_data, a_data;
   s_ll *head = NULL;
   do{
      printf("\nSingle Linked List operations\n");
      printf("1. Insert at first\n");
      printf("2. Insert at last\n");
      printf("3. Insert at midlle\n");
      printf("4. Print list\n");
      printf("5. Exit\n");
      printf("Choose an option: ");
      scanf("%d", &op);

      switch(op)
      {
         case 1:
         {
            //Insert first
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insert_first(&head, data);
            printf("Inserted at first successfully!!!\n");
         }
         break;
         case 2:
         {
            //Insert last
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insert_last(&head, data);
            printf("Inserted at last successfully!!!\n");
```

```
    }
    break;
    case 3:
    {
       //Insert middle
       printf("Enter the data to be inserted: ");
       scanf("%d", &data);
       printf("Enter the data before new node: ");
       scanf("%d", &b_data);
       printf("Enter the data after new node: ");
       scanf("%d", &a_data);
       insert_middle(&head, data, b_data, a_data);

    }
    break;
    case 4:
    {
       //Print list
       print_list(head);
    }
    break;
    case 5:
    {
       printf("Exiting!!!");
    }
    break;
    default: printf("Invalid option !! Please try again\n");
  }

}while(op != 5);
```

```
    return 0;
}

void insert_first(s_ll **head, int data)
{
    s_ll *new = create_node(data);

    if(*head == NULL)
    {
        *head = new;
    }
    else
    {
        new->link = *head;
        *head = new;
    }

}
void insert_last(s_ll **head, int data)
{
     s_ll *new = create_node(data);
     s_ll *temp = *head;
     while(temp->link != NULL)
     {
         temp = temp->link;
     }

     temp->link = new;
}
void insert_middle(s_ll **head, int data, int b_data, int a_data)
{
```

```
    if (*head == NULL)
    {
        printf("Error: List is empty!\n");
        return;
    }

    s_ll *temp = *head;
    while (temp != NULL)
    {
        if (temp->link != NULL && temp->data == b_data)
        {
            s_ll *new_node = create_node(data);
            new_node->link = temp->link;
            temp->link = new_node;
            printf("Inserted at middle successfully!\n");
            return;
        }
        else if (temp->link == NULL && temp->data == b_data)
        {
            printf("Error: The node with %d is the last node.\n", b_data);
        }

        temp = temp->link;
    }

    printf("Error: Node with data %d not found in the list.\n", b_data);
}

void print_list(s_ll *head)
{
    if(head == NULL)
```

```
    printf("List is empty!!\n");
    s_ll *temp = head;
    while(temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->link;
    }
    printf("NULL\n");
}

s_ll* create_node(int data)
{
    s_ll *new = (s_ll*)malloc(sizeof(s_ll));
    new->data = data;
    new->link = NULL;
}
```

Problem 1: Reverse a Linked List

Write a C program to reverse a singly linked list. The program should traverse the list, reverse the pointers between the nodes, and display the reversed list.

Requirements:

1. Define a function to reverse the linked list iteratively.
2. Update the head pointer to the new first node.
3. Display the reversed list.

Example Input:

rust

Copy code

Initial list: 10 -> 20 -> 30 -> 40

Example Output:

rust

Copy code

Reversed list: 40 -> 30 -> 20 -> 10

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

typedef struct node
{
   int data;
   struct node *link;
}s_ll;

void insert_first(s_ll **, int);
void insert_last(s_ll **, int);
void reverse_ll(s_ll **);
void print_list(s_ll *);

s_ll* create_node(int );

int main()
{
   int op = 0, data, b_data, a_data;
   s_ll *head = NULL;
   do{
      printf("\nSingle Linked List operations\n");
      printf("1. Insert at first\n");
      printf("2. Insert at last\n");
      printf("3. Reverse list\n");
      printf("4. Print list\n");
      printf("5. Exit\n");
      printf("Choose an option: ");
      scanf("%d", &op);
```

```
switch(op)
{
   case 1:
   {
      //Insert first
      printf("Enter the data to be inserted: ");
      scanf("%d", &data);
      insert_first(&head, data);
      printf("Inserted at first successfully!!!\n");
   }
   break;
   case 2:
   {
      //Insert last
      printf("Enter the data to be inserted: ");
      scanf("%d", &data);
      insert_last(&head, data);
      printf("Inserted at last successfully!!!\n");
   }
   break;
   case 3:
   {
      //Reverse
      reverse_ll(&head);

   }
   break;
   case 4:
   {
      //Print list
```

```
            print_list(head);
        }
        break;
        case 5:
        {
            printf("Exiting!!!");
        }
        break;
        default: printf("Invalid option !! Please try again\n");
      }

   }while(op != 5);

   return 0;
}

void insert_first(s_ll **head, int data)
{
   s_ll *new = create_node(data);

   if(*head == NULL)
   {
      *head = new;
   }
   else
   {
      new->link = *head;
      *head = new;
   }

}
```

```
void insert_last(s_ll **head, int data)
{
    s_ll *new = create_node(data);
    s_ll *temp = *head;
    while(temp->link != NULL)
    {
       temp = temp->link;
    }

    temp->link = new;
}
void reverse_ll (s_ll **head)
{
   if (*head == NULL)
   {
      printf("Error: List is empty!\n");
      return;
   }

   s_ll *next = NULL;
   s_ll *current = *head;
   s_ll *prev = NULL;

   while(current != NULL)
   {
      next = current->link;
      current->link = prev;
      prev = current;
      current = next;
   }
```

```
    *head = prev;

}

void print_list(s_ll *head)
{
    if(head == NULL)
    printf("List is empty!!\n");
    s_ll *temp = head;
    while(temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->link;
    }
    printf("NULL\n");
}

s_ll* create_node(int data)
{
    s_ll *new = (s_ll*)malloc(sizeof(s_ll));
    new->data = data;
    new->link = NULL;
}
```

Problem 2: Find the Middle Node

Write a C program to find and display the middle node of a singly linked list. If the list has an even number of nodes, display the first middle node.

Requirements:

1. Use two pointers: one moving one step and the other moving two steps.
2. When the faster pointer reaches the end, the slower pointer will point to the middle node.

Example Input:

rust

Copy code

List: 10 -> 20 -> 30 -> 40 -> 50

Example Output:

scss

Copy code

Middle node: 30

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

typedef struct node
{
   int data;
   struct node *link;
}s_ll;

void insert_first(s_ll **, int);
void insert_last(s_ll **, int);
void find_middle(s_ll **);
void print_list(s_ll *);

s_ll* create_node(int );

int main()
```

```
{
   int op = 0, data;
   s_ll *head = NULL;
   do{
      printf("\nSingle Linked List operations\n");
      printf("1. Insert at first\n");
      printf("2. Insert at last\n");
      printf("3. Find midlle\n");
      printf("4. Print list\n");
      printf("5. Exit\n");
      printf("Choose an option: ");
      scanf("%d", &op);

      switch(op)
      {
         case 1:
         {
            //Insert first
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insert_first(&head, data);
            printf("Inserted at first successfully!!!\n");
         }
         break;
         case 2:
         {
            //Insert last
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insert_last(&head, data);
            printf("Inserted at last successfully!!!\n");
```

```
        }
        break;
        case 3:
        {
           //Insert middle
           find_middle(&head);

        }
        break;
        case 4:
        {
           //Print list
           print_list(head);
        }
        break;
        case 5:
        {
           printf("Exiting!!!");
        }
        break;
        default: printf("Invalid option !! Please try again\n");
      }

   }while(op != 5);

   return 0;
}

void insert_first(s_ll **head, int data)
{
   s_ll *new = create_node(data);
```

```
    if(*head == NULL)
    {
        *head = new;
    }
    else
    {
        new->link = *head;
        *head = new;
    }

}
void insert_last(s_ll **head, int data)
{
     s_ll *new = create_node(data);
     s_ll *temp = *head;
     while(temp->link != NULL)
     {
         temp = temp->link;
     }

     temp->link = new;
}
void find_middle(s_ll **head)
{
    if (*head == NULL)
    {
        printf("Error: List is empty!\n");
        return;
    }
```

```
    s_ll *fast = *head;
    s_ll *slow = *head;

    while(fast != NULL && fast->link !=NULL)
    {
        fast=fast->link->link;
        slow=slow->link;
    }

    printf("The middle node is having data %d\n", slow->data);
}

void print_list(s_ll *head)
{
    if(head == NULL)
    printf("List is empty!!\n");
    s_ll *temp = head;
    while(temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->link;
    }
    printf("NULL\n");
}

s_ll* create_node(int data)
{
    s_ll *new = (s_ll*)malloc(sizeof(s_ll));
    new->data = data;
    new->link = NULL;
}
```

Problem 3: Detect and Remove a Cycle in a Linked List

Write a C program to detect if a cycle (loop) exists in a singly linked list and remove it if present. Use Floyd's Cycle Detection Algorithm (slow and fast pointers) to detect the cycle.

Requirements:

1. Detect the cycle in the list.
2. If a cycle exists, find the starting node of the cycle and break the loop.
3. Display the updated list.

Example Input:

rust

Copy code

List: 10 -> 20 -> 30 -> 40 -> 50 -> (points back to 30)

Example Output:

rust

Copy code

Cycle detected and removed.

Updated list: 10 -> 20 -> 30 -> 40 -> 50

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

typedef struct node
{
   int data;
   struct node *link;
} s_ll;

void insert_first(s_ll **, int);
void insert_last(s_ll **, int);
void find_cyclic(s_ll **);
void remove_cycle(s_ll **);
```

```
void print_list(s_ll *);

s_ll* create_node(int );
void create_cycle(s_ll **, int); // To create a cycle at a specific node index

int main()
{
   int op = 0, data, cycle_index;
   s_ll *head = NULL;
   do{
      printf("\nSingle Linked List operations\n");
      printf("1. Insert at first\n");
      printf("2. Insert at last\n");
      printf("3. Find cyclic linked list\n");
      printf("4. Remove cycle\n");
      printf("5. Print list\n");
      printf("6. Create cycle (for testing)\n");
      printf("7. Exit\n");
      printf("Choose an option: ");
      scanf("%d", &op);

      switch(op)
      {
         case 1:
         {
            // Insert first
            printf("Enter the data to be inserted: ");
            scanf("%d", &data);
            insert_first(&head, data);
            printf("Inserted at first successfully!!!\n");
         }
```

```
break;
case 2:
{
   // Insert last
   printf("Enter the data to be inserted: ");
   scanf("%d", &data);
   insert_last(&head, data);
   printf("Inserted at last successfully!!!\n");
}
break;
case 3:
{
   // Detect cycle
   find_cyclic(&head);
}
break;
case 4:
{
   // Remove cycle
   remove_cycle(&head);
}
break;
case 5:
{
   // Print list
   print_list(head);
}
break;
case 6:
{
   // Create a cycle (for testing)
```

```
            printf("Enter the index where you want to create a cycle (1-based): ");
            scanf("%d", &cycle_index);
            create_cycle(&head, cycle_index);
        }
        break;
        case 7:
        {
            printf("Exiting!!!\n");
        }
        break;
        default:
            printf("Invalid option !! Please try again\n");
      }

   } while(op != 7);

   return 0;
}

void insert_first(s_ll **head, int data)
{
   s_ll *new = create_node(data);

   if(*head == NULL)
   {
      *head = new;
   }
   else
   {
      new->link = *head;
      *head = new;
```

```
    }
}

void insert_last(s_ll **head, int data)
{
    s_ll *new = create_node(data);
    s_ll *temp = *head;

    if (*head == NULL) {
        *head = new;
    } else {
        while(temp->link != NULL)
        {
            temp = temp->link;
        }
        temp->link = new;
    }
}

void find_cyclic(s_ll **head)
{
    if (*head == NULL)
    {
        printf("Error: List is empty!\n");
        return;
    }

    s_ll *fast = *head;
    s_ll *slow = *head;

    while(fast != NULL && fast->link != NULL)
```

```
    {
        fast = fast->link->link;
        slow = slow->link;

        if(slow == fast)  // Cycle detected
        {
            printf("Cycle detected in the list.\n");
            return;
        }
    }
    printf("No cycle found in the list.\n");
}

void remove_cycle(s_ll **head)
{
    if (*head == NULL)
    {
        printf("Error: List is empty!\n");
        return;
    }

    s_ll *fast = *head;
    s_ll *slow = *head;

    while(fast != NULL && fast->link != NULL)
    {
        fast = fast->link->link;
        slow = slow->link;

        if(slow == fast)  // Cycle detected
        {
```

```
            // Find the start of the cycle
            slow = *head;
            while(slow != fast)
            {
               slow = slow->link;
               fast = fast->link;
            }

            // Remove the cycle
            s_ll *temp = fast;
            while(temp->link != fast)
            {
               temp = temp->link;
            }
            temp->link = NULL;
            printf("Cycle removed successfully.\n");
            return;
         }
      }
      printf("No cycle to remove.\n");
}

void print_list(s_ll *head)
{
   if(head == NULL)
   {
      printf("List is empty!!\n");
      return;
   }

   s_ll *temp = head;
```

```
    while(temp != NULL)
    {
        printf("%d -> ", temp->data);
        temp = temp->link;
    }
    printf("NULL\n");
}

s_ll* create_node(int data)
{
    s_ll *new = (s_ll*)malloc(sizeof(s_ll));
    new->data = data;
    new->link = NULL;
    return new;
}

void create_cycle(s_ll **head, int cycle_start_index)
{
    if (*head == NULL)
    {
        printf("List is empty, cannot create a cycle.\n");
        return;
    }

    s_ll *temp = *head;
    s_ll *cycle_start_node = NULL;
    int index = 1;

    while (temp->link != NULL)
    {
        if (index == cycle_start_index)
```

```
        {
            cycle_start_node = temp;
        }
        temp = temp->link;
        index++;
    }

    if (cycle_start_node != NULL)
    {
        temp->link = cycle_start_node;  // Create cycle
        printf("Cycle created at index %d\n", cycle_start_index);
    }
    else
    {
        printf("Invalid index. No cycle created.\n");
    }
}
```