Name Priyadarshi Prabhakar SAP ID 590029237

Experiment 10: Dynamic Memory Allocation – Linked List Algorithms, Pseudocode & C Programs

1. Create a simple linked list in C using pointer and structure

Algorithm:

1. Start
2. Define a structure NODE with data and next pointer
3. Initialize head = NULL
4. Read number of nodes (n)
5. Repeat n times:
    a. Allocate memory for new node
    b. Read data from user
    c. Set new->next = NULL
    d. If head is NULL, set head = new and temp = new
       Else set temp->next = new and update temp = new
6. Traverse list from head and display each node's data
7. Stop

Pseudocode:

```
STRUCT NODE
   INTEGER data
   POINTER to NODE next
END STRUCT
```

```
BEGIN
  head = NULL
  READ n
  FOR i = 1 TO n
    ALLOCATE new NODE
    READ new.data
    new.next = NULL
    IF head == NULL THEN
      head = new
      temp = new
    ELSE
      temp.next = new
      temp = new
    ENDIF
  ENDFOR

  temp = head
  WHILE temp != NULL
    PRINT temp.data
    temp = temp.next
  ENDWHILE
END
```

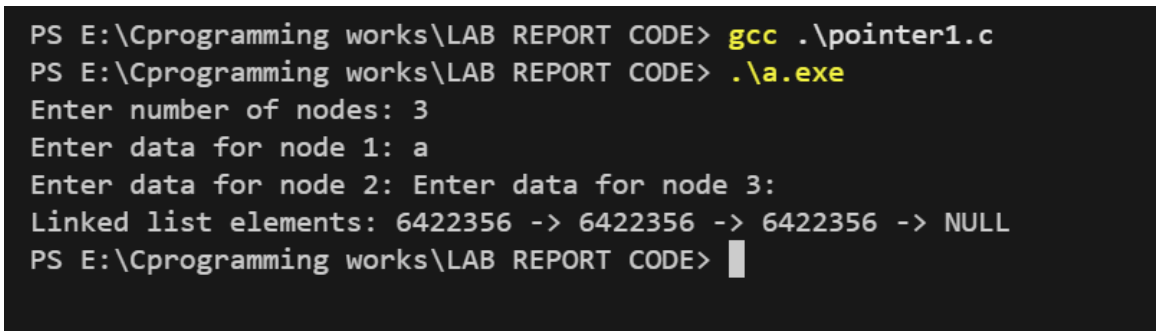
C Program:

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

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

int main() {
   struct Node *head = NULL, *temp = NULL, *newNode;
   int n, i, value;

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

   for (i = 0; i < n; i++) {
      newNode = (struct Node *)malloc(sizeof(struct Node));
      if (newNode == NULL) {
         printf("Memory allocation failed\n");
         return 1;
      }

      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &value);

      newNode->data = value;
      newNode->next = NULL;
```

```
        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }

    printf("\nLinked list elements: ");
    temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");

    return 0;
}
```

OUTPUT



2. Insert an item in the middle of the linked list

Algorithm:

1. Start
2. Create a linked list as in Program 1
3. Count total number of nodes (say count)
4. Compute middle position as mid = count / 2

   (for even count, insert after count/2 nodes)
5. Allocate memory for new node and read its value
6. Traverse list up to node just before middle position
7. Adjust pointers:

   new->next = current->next

   current->next = new
8. Display updated list
9. Stop

Pseudocode:

```
BEGIN
  // Assume head points to an existing linked list


  // Step 1: Count nodes
  count = 0
  temp = head
  WHILE temp != NULL
     count = count + 1
     temp = temp.next
```

```
ENDWHILE

mid = count / 2         // integer division

// Step 2: Create new node
ALLOCATE new NODE
READ new.data

// Step 3: Traverse to (mid-1)th node
temp = head
i = 0
WHILE i < mid - 1
   temp = temp.next
   i = i + 1
ENDWHILE

// Step 4: Insert new node
new.next = temp.next
temp.next = new

// Step 5: Display new list
temp = head
WHILE temp != NULL
   PRINT temp.data
   temp = temp.next
ENDWHILE
```

END

C Program:

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

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

struct Node* createList(int n) {
   struct Node *head = NULL, *temp = NULL, *newNode;
   int i, value;

   for (i = 0; i < n; i++) {
      newNode = (struct Node *)malloc(sizeof(struct Node));
      if (newNode == NULL) {
         printf("Memory allocation failed\n");
         exit(1);
      }

      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &value);

      newNode->data = value;
```

```
        newNode->next = NULL;

        if (head == NULL) {
            head = newNode;
            temp = newNode;
        } else {
            temp->next = newNode;
            temp = newNode;
        }
    }
    return head;
}

void displayList(struct Node *head) {
    struct Node *temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

struct Node* insertMiddle(struct Node *head, int value) {
    if (head == NULL) {
        struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
        newNode->data = value;
```

```
    newNode->next = NULL;
    return newNode;
}

int count = 0, mid, i;
struct Node *temp = head;

// Count nodes
while (temp != NULL) {
    count++;
    temp = temp->next;
}

mid = count / 2;  // middle position (0-based index)

struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
newNode->data = value;
newNode->next = NULL;

// If insert after head when mid == 0
if (mid == 0) {
    newNode->next = head->next;
    head->next = newNode;
    return head;
}
```

```
    temp = head;
    for (i = 0; i < mid - 1; i++) {
        temp = temp->next;
    }

    newNode->next = temp->next;
    temp->next = newNode;

    return head;
}

int main() {
    int n, value;
    struct Node *head = NULL;

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

    head = createList(n);

    printf("\nOriginal list:\n");
    displayList(head);

    printf("\nEnter value to insert in the middle: ");
    scanf("%d", &value);
```

```
    head = insertMiddle(head, value);

    printf("\nList after inserting in the middle:\n");
    displayList(head);

    return 0;
}
```

OUTPUT