Objective: Explore linked lists, including types like singly and doubly linked lists, while using pointers, structures, and dynamic memory allocation. Demonstrate applications of linked lists.

Assignment Tasks

1. Singly Linked List Implementation:

• Create a structure for a singly linked list node with data and a next pointer.

• Implement functions for:

• Insertion at the beginning, end, and a specified position.

• Deletion from the beginning, end, and a specified position.

• Displaying the list.

CODE :

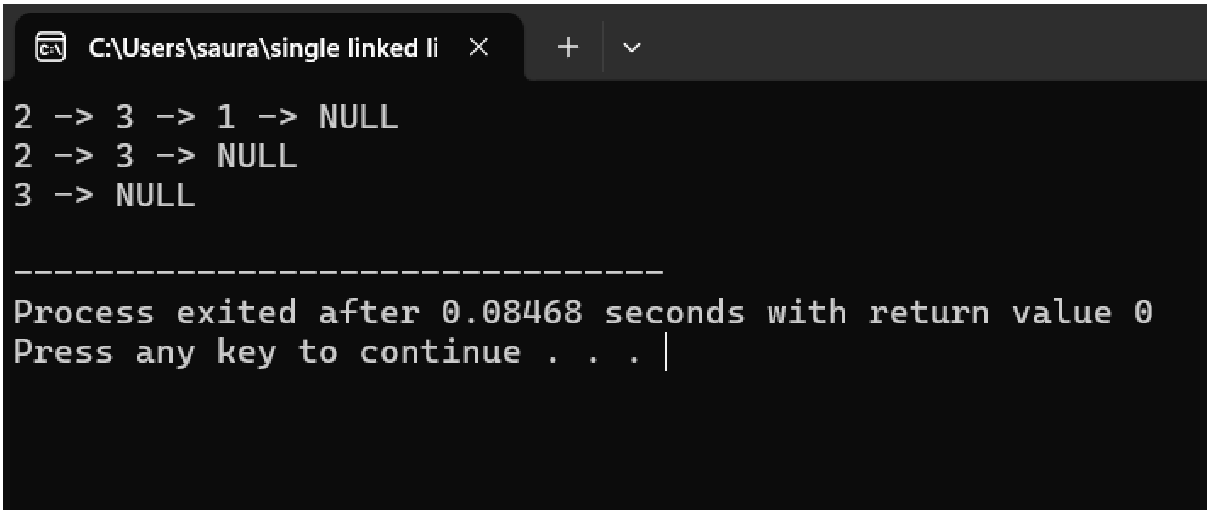
```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* next;
};
void insertAtBeginning(struct Node** head, int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = *head;
  *head = newNode;
}
void insertAtEnd(struct Node** head, int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
  if (*head == NULL) {
    *head = newNode;
    return;
  }
  struct Node* temp = *head;
  while (temp->next != NULL)
    temp = temp->next;
  temp->next = newNode;
}
void insertAtPosition(struct Node** head, int data, int position) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  if (position == 0) {
    newNode->next = *head;
    *head = newNode;
    return;
  }
```

```
    struct Node* temp = *head;
    for (int i = 0; i < position - 1 && temp != NULL; i++)
        temp = temp->next;
    if (temp == NULL) return;
    newNode->next = temp->next;
    temp->next = newNode;
}

// Function to delete from the beginning
void deleteFromBeginning(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    *head = (*head)->next;
    free(temp);
}
// Function to delete from the end
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next->next != NULL)
        temp = temp->next;
    free(temp->next);
    temp->next = NULL;
}
// Function to delete from a specified position
void deleteFromPosition(struct Node** head, int position) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    if (position == 0) {
        *head = temp->next;
        free(temp);
        return;
    }
    for (int i = 0; temp != NULL && i < position - 1; i++)
        temp = temp->next;
    if (temp == NULL || temp->next == NULL) return;
    struct Node* next = temp->next->next;
    free(temp->next);
    temp->next = next;
}
void display(struct Node* head) {
    struct Node* temp = head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 1);
    insertAtBeginning(&head, 2);
    insertAtPosition(&head, 3, 1);
    display(head);
    deleteFromEnd(&head);
    display(head);
    deleteFromBeginning(&head);
    display(head);
    return 0;
}
```

OUTPUT



```
2 -> 3 -> 1 -> NULL
2 -> 3 -> NULL
3 -> NULL

--------------------------------
Process exited after 0.08468 seconds with return value 0
Press any key to continue . . .
```

2. Doubly Linked List Implementation:

• Modify the singly linked list to a doubly linked list by adding a prev pointer.

• Implement the same insertion, deletion, and display functions.

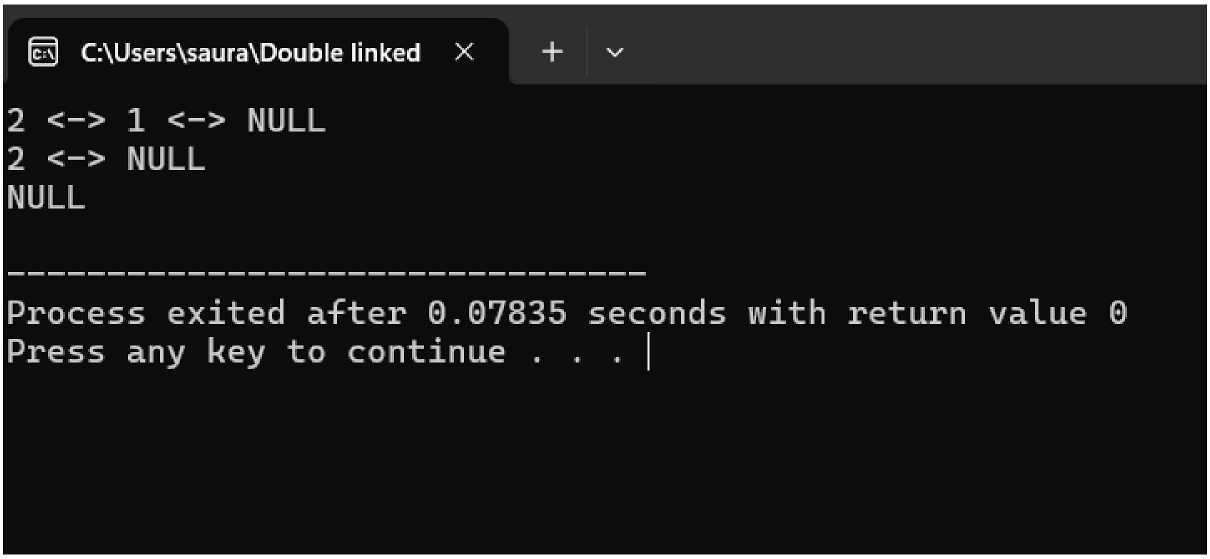
CODE:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
  int data;
  struct Node* next;
  struct Node* prev;
};
void insertAtBeginning(struct Node** head, int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = *head;
  newNode->prev = NULL;
  if (*head != NULL)
    (*head)->prev = newNode;
  *head = newNode;
}
void insertAtEnd(struct Node** head, int data) {
  struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = data;
  newNode->next = NULL;
  if (*head == NULL) {
    newNode->prev = NULL;
    *head = newNode;
    return;
  }
  struct Node* temp = *head;
  while (temp->next != NULL)
    temp = temp->next;
  temp->next = newNode;
  newNode->prev = temp;
}
void display(struct Node* head) {
  struct Node* temp = head;
  while (temp != NULL) {
    printf("%d <-> ", temp->data);
    temp = temp->next;
  }
  printf("NULL\n");
}
void deleteFromBeginning(struct Node** head) {
  if (*head == NULL) return;
  struct Node* temp = *head;
```

```
    *head = (*head)->next;
    if (*head != NULL)
        (*head)->prev = NULL;
    free(temp);
}
void deleteFromEnd(struct Node** head) {
    if (*head == NULL) return;
    struct Node* temp = *head;
    if (temp->next == NULL) {
        free(temp);
        *head = NULL;
        return;
    }
    while (temp->next != NULL)
        temp = temp->next;
    temp->prev->next = NULL;
    free(temp);
}

int main() {
    struct Node* head = NULL;
    insertAtEnd(&head, 1);
    insertAtBeginning(&head, 2);
    display(head);
    deleteFromEnd(&head);
    display(head);
    deleteFromBeginning(&head);
    display(head);
    return 0;
}
```

OUTPUT



```
2 <-> 1 <-> NULL
2 <-> NULL
NULL

--------------------------------
Process exited after 0.07835 seconds with return value 0
Press any key to continue . . .
```

3. Application Example:

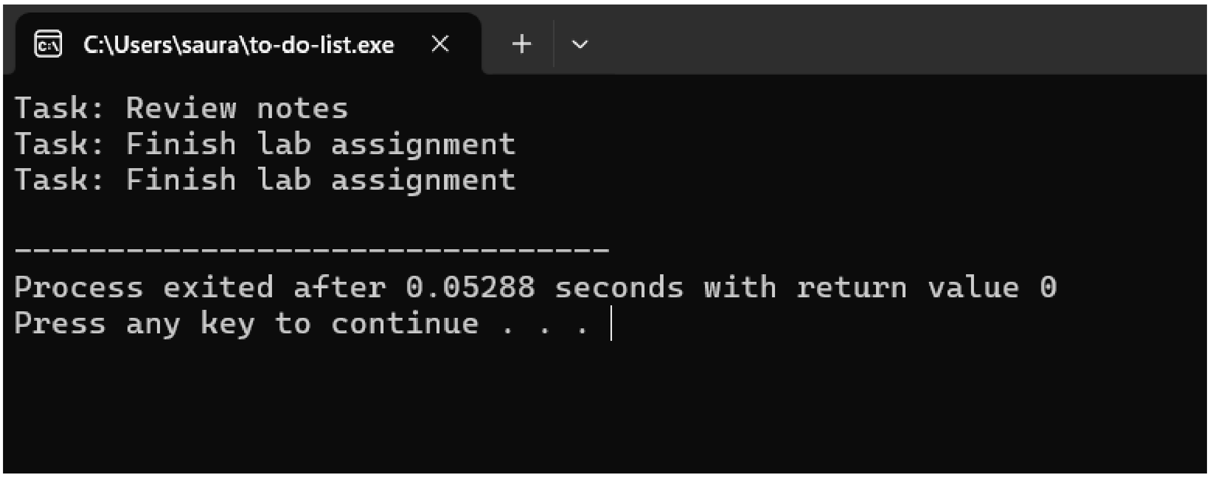
- Demonstrate a practical use of linked lists, such as a to-do list manager or a basic stack/queue.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Task {
  char name[50];
  struct Task* next;
};
void addTask(struct Task** head, char* name) {
  struct Task* newTask = (struct Task*)malloc(sizeof(struct Task));
  strcpy(newTask->name, name);
  newTask->next = *head;
  *head = newTask;
}
void displayTasks(struct Task* head) {
  struct Task* temp = head;
  while (temp != NULL) {
    printf("Task: %s\n", temp->name);
    temp = temp->next;
  }
}
void completeTask(struct Task** head) {
  if (*head == NULL) return;
  struct Task* temp = *head;
  *head = (*head)->next;
  free(temp);
}

int main() {
  struct Task* toDoList = NULL;
  addTask(&toDoList, "Finish lab assignment");
  addTask(&toDoList, "Review notes");
  displayTasks(toDoList);
  completeTask(&toDoList);
  displayTasks(toDoList);
  return 0;
}
```

OUTPUT



```
Task: Review notes
Task: Finish lab assignment
Task: Finish lab assignment

--------------------------------
Process exited after 0.05288 seconds with return value 0
Press any key to continue . . .
```

4. Memory Usage and Dynamic Allocation:

- Use malloc and free to dynamically allocate and deallocate memory.

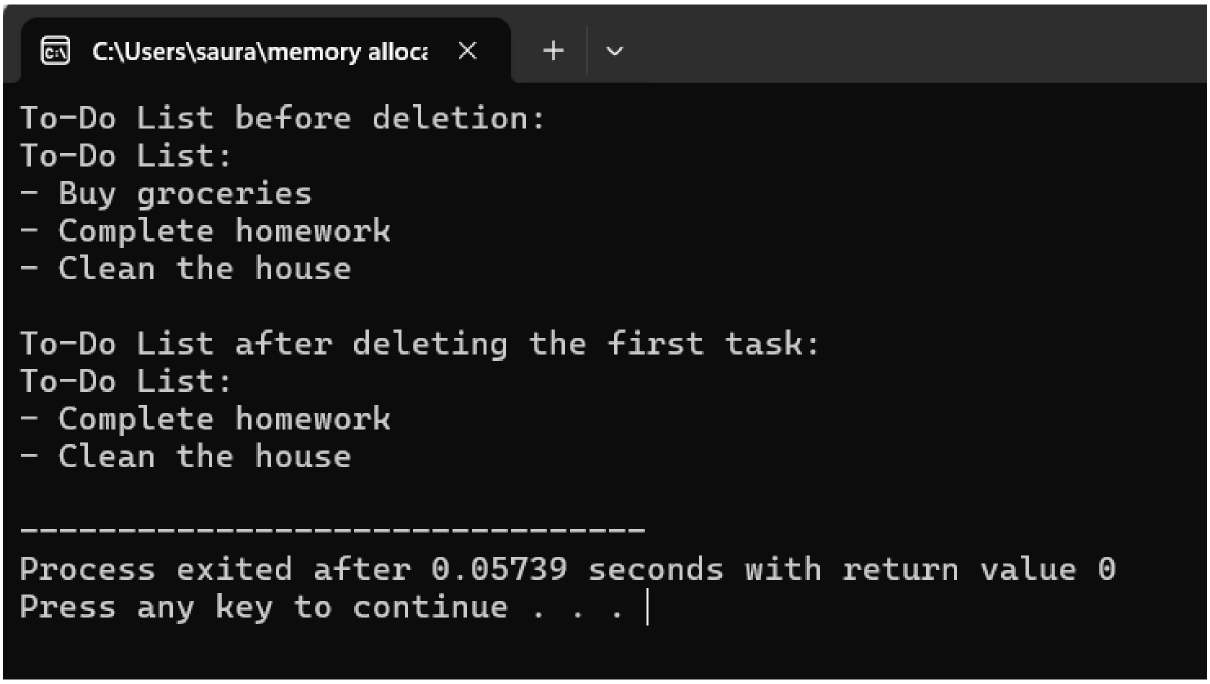
- Ensure memory is correctly freed after operations to prevent memory leaks.

CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct DNode {
  char task[100];
  struct DNode* next;
  struct DNode* prev;
};
struct DNode* createDNode(char* task) {
  struct DNode* newNode = (struct DNode*)malloc(sizeof(struct DNode));
  if (newNode == NULL) {
    printf("Memory allocation failed.\n");
    exit(1);
  }
  strcpy(newNode->task, task);
  newNode->next = NULL;
  newNode->prev = NULL;
  return newNode;
}
void insertAtEnd(struct DNode** head, char* task) {
  struct DNode* newNode = createDNode(task);
  if (*head == NULL) {
    *head = newNode;
  } else {
    struct DNode* temp = *head;
    while (temp->next != NULL) {
      temp = temp->next;
    }
    temp->next = newNode;
    newNode->prev = temp;
  }
}
void deleteFromBeginning(struct DNode** head) {
  if (*head == NULL) {
    printf("No tasks left.\n");
    return;
  }
  struct DNode* temp = *head;
  *head = (*head)->next;
  if (*head != NULL) {
    (*head)->prev = NULL;
  }
```

```
    free(temp);
}
void displayList(struct DNode* head) {
    if (head == NULL) {
        printf("The to-do list is empty.\n");
        return;
    }
    printf("To-Do List:\n");
    struct DNode* temp = head;
    while (temp != NULL) {
        printf("- %s\n", temp->task);
        temp = temp->next;
    }
}
int main() {
    struct DNode* head = NULL;
    insertAtEnd(&head, "Buy groceries");
    insertAtEnd(&head, "Complete homework");
    insertAtEnd(&head, "Clean the house");
    printf("To-Do List before deletion:\n");
    displayList(head);
    deleteFromBeginning(&head);
    printf("\nTo-Do List after deleting the first task:\n");
    displayList(head);
    return 0;
}
```

OUTPUT



```
To-Do List before deletion:
To-Do List:
- Buy groceries
- Complete homework
- Clean the house

To-Do List after deleting the first task:
To-Do List:
- Complete homework
- Clean the house

--------------------------------
Process exited after 0.05739 seconds with return value 0
Press any key to continue . . .
```