DATA STRUCTURE

DAY 2-25/07/2024

1.Single linked list

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    newNode->next = head;
    head = newNode;
    return head;
}
```

```
struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    if (head == NULL) {
        head = newNode;
    } else {
        struct Node* current = head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newNode;
    }
    return head;
}
struct Node* deleteAtBeginning(struct Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
    return head;
}
struct Node* deleteAtEnd(struct Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
```

```
    }
    if (head->next == NULL) {
        free(head);
        return NULL;
    }
    struct Node* current = head;
    struct Node* prev = NULL;
    while (current->next != NULL) {
        prev = current;
        current = current->next;
    }
    prev->next = NULL;
    free(current);
    return head;
}
 struct Node* insertAtPosition(struct Node* head, int data, int position) {
    if (position < 1) {
        printf("Invalid position.\n");
        return head;
    }
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    if (position == 1) {
        newNode->next = head;
        head = newNode;
        return head;
    }
    struct Node* current = head;
```

```
    int count = 1;
    while (current != NULL && count < position - 1) {
        current = current->next;
        count++;
    }
    if (current == NULL) {
        printf("Position is out of bounds.\n");
        free(newNode);
        return head;
    }
    newNode->next = current->next;
    current->next = newNode;
    return head;
}
 struct Node* deleteAtPosition(struct Node* head, int position) {
    if (position < 1 || head == NULL) {
        printf("Invalid position or empty list.\n");
        return head;
    }
    struct Node* temp = head;
    if (position == 1) {
        head = head->next;
        free(temp);
        return head;
    }
    struct Node* prev = NULL;
    int count = 1;
    while (temp != NULL && count < position) {
        prev = temp;
        temp = temp->next;
```

```
        count++;
    }
    if (temp == NULL) {
        printf("Position is out of bounds.\n");
        return head;
    }
    prev->next = temp->next;
    free(temp);
    return head;
}
void displayList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        printf("%d -> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
 void freeList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
}
int main() {
    struct Node* head = NULL;
     head = insertAtBeginning(head, 10);
    head = insertAtBeginning(head, 5);
```

```
    printf("Linked list after insertions at beginning: ");
    displayList(head);
    head = insertAtEnd(head, 20);
    head = insertAtEnd(head, 30);
    printf("Linked list after insertions at end: ");
    displayList(head);



    head = insertAtPosition(head, 15, 2);
    printf("Linked list after insertion at position 2: ");
    displayList(head);
     head = deleteAtBeginning(head);
    head = deleteAtEnd(head);
    printf("Linked list after deletion from beginning and end: ");
    displayList(head);
     head = deleteAtPosition(head, 2);
    printf("Linked list after deletion at position 2: ");
    displayList(head);
    freeList(head);
    head = NULL;
    return 0;
}
```

OUTPUT:

Linked list after insertions at beginning: 5 -> 10 -> NULL

Linked list after insertions at end: 5 -> 10 -> 20 -> 30 -> NULL

Linked list after insertion at position 2: 5 -> 15 -> 10 -> 20 -> 30 -> NULL

Linked list after deletion from beginning and end: 15 -> 10 -> 20 -> NULL

Linked list after deletion at position 2: 15 -> 20 -> NULL

2.Double linked list

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
 struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
struct Node* insertAtBeginning(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    if (head == NULL) {
        return newNode;
    }
    newNode->next = head;
```

```
    head->prev = newNode;
    return newNode;
}
 struct Node* insertAtEnd(struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    if (head == NULL) {
        return newNode;
    }
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    current->next = newNode;
    newNode->prev = current;
    return head;
}
 struct Node* insertAtPosition(struct Node* head, int data, int position) {
    if (position < 1) {
        printf("Invalid position.\n");
        return head;
    }
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return head;
    }
    if (position == 1) {
        if (head == NULL) {
```

```
            return newNode;
        }
        newNode->next = head;
        head->prev = newNode;
        return newNode;
    }
    struct Node* current = head;
    int count = 1;
    while (current != NULL && count < position - 1) {
        current = current->next;
        count++;
    }
    if (current == NULL) {
        printf("Position is out of bounds.\n");
        free(newNode);
        return head;
    }
    newNode->next = current->next;
    newNode->prev = current;
    if (current->next != NULL) {
        current->next->prev = newNode;
    }
    current->next = newNode;
    return head;
}
struct Node* deleteAtBeginning(struct Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }
```

```
    struct Node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    }
    free(temp);
    return head;
}
struct Node* deleteAtEnd(struct Node* head) {
    if (head == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }
    struct Node* current = head;
    while (current->next != NULL) {
        current = current->next;
    }
    if (current->prev != NULL) {
        current->prev->next = NULL;
    } else {
        head = NULL;
    }
    free(current);
    return head;
}
struct Node* deleteAtPosition(struct Node* head, int position) {
    if (position < 1 || head == NULL) {
        printf("Invalid position or empty list.\n");
        return head;
    }
```

```
    struct Node* temp = head;
    if (position == 1) {
        head = head->next;
        if (head != NULL) {
            head->prev = NULL;
        }
        free(temp);
        return head;
    }
    struct Node* prev = NULL;
    int count = 1;
    while (temp != NULL && count < position) {
        prev = temp;
        temp = temp->next;
        count++;
    }
    if (temp == NULL) {
        printf("Position is out of bounds.\n");
        return head;
    }
    prev->next = temp->next;
    if (temp->next != NULL) {
        temp->next->prev = prev;
    }
    free(temp);
    return head;
}
void displayList(struct Node* head) {
    struct Node* current = head;
    printf("NULL <-> ");
```

```
    while (current != NULL) {
        printf("%d <-> ", current->data);
        current = current->next;
    }
    printf("NULL\n");
}
 void freeList(struct Node* head) {
    struct Node* current = head;
    while (current != NULL) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
}
int main() {
    struct Node* head = NULL;
     head = insertAtBeginning(head, 10);
    head = insertAtBeginning(head, 5);
    printf("Linked list after insertions at beginning: ");
    displayList(head);
     head = insertAtEnd(head, 20);
    head = insertAtEnd(head, 30);
    printf("Linked list after insertions at end: ");
    displayList(head);
     head = insertAtPosition(head, 15, 2);
    printf("Linked list after insertion at position 2: ");
    displayList(head);
     head = deleteAtBeginning(head);
    head = deleteAtEnd(head);
    printf("Linked list after deletion from beginning and end: ");
```

```
    displayList(head);
     head = deleteAtPosition(head, 2);
  printf("Linked list after deletion at position 2: ");
    displayList(head);
     freeList(head);
    head = NULL;
    return 0;
}
```

OUTPUT:

Linked list after insertions at beginning: NULL <-> 5 <-> 10 <-> NULL

Linked list after insertions at end: NULL <-> 5 <-> 10 <-> 20 <-> 30 <-> NULL

Linked list after insertion at position 2: NULL <-> 5 <-> 15 <-> 10 <-> 20 <-> 30 <-> NULL

Linked list after deletion from beginning and end: NULL <-> 15 <-> 10 <-> 20 <-> NULL

Linked list after deletion at position 2: NULL <-> 15 <-> 20 <-> NULL

3.Circular linked list

PROGRAM:

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed.\n");
        return NULL;
    }
    newNode->data = data;
```

```
    newNode->next = NULL;
    return newNode;
}
 struct Node* insertAtBeginning(struct Node* last, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return last;
    }
    if (last == NULL) {
        newNode->next = newNode;
        return newNode;
    }
    newNode->next = last->next;
    last->next = newNode;
    return last;
}

struct Node* insertAtEnd(struct Node* last, int data) {
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return last;
    }
    if (last == NULL) {
        newNode->next = newNode;
        return newNode;
    }
    newNode->next = last->next;
    last->next = newNode;
    last = newNode;
    return last;
```

```
}
struct Node* insertAtPosition(struct Node* last, int data, int position) {
    if (position < 1) {
        printf("Invalid position.\n");
        return last;
    }
    struct Node* newNode = createNode(data);
    if (newNode == NULL) {
        return last;
    }
    if (position == 1) {
        return insertAtBeginning(last, data);
    }
    struct Node* current = last->next;
    int count = 1;
    while (current != last && count < position - 1) {
        current = current->next;
        count++;
    }
    if (current == last && count != position - 1) {
        printf("Position is out of bounds.\n");
        free(newNode);
        return last;
    }
    newNode->next = current->next;
    current->next = newNode;
    if (current == last) {
        last = newNode;
    }
    return last;
```

```
}
struct Node* deleteAtBeginning(struct Node* last) {
    if (last == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }
    if (last->next == last) {
        free(last);
        return NULL;
    }
    struct Node* temp = last->next;
    last->next = temp->next;
    free(temp);
    return last;
}
struct Node* deleteAtEnd(struct Node* last) {
    if (last == NULL) {
        printf("List is empty. Cannot delete.\n");
        return NULL;
    }
    if (last->next == last) {
        free(last);
        return NULL;
    }
    struct Node* current = last->next;
    while (current->next != last) {
        current = current->next;
    }
    current->next = last->next;
    free(last);
```

```
    last = current;
    return last;
}
struct Node* deleteAtPosition(struct Node* last, int position) {
    if (position < 1 || last == NULL) {
        printf("Invalid position or empty list.\n");
        return last;
    }
    if (position == 1) {
        return deleteAtBeginning(last);
    }
    struct Node* current = last->next;
    struct Node* prev = last;
    int count = 1;
    while (current != last && count < position) {
        prev = current;
        current = current->next;
        count++;
    }
    if (current == last && count != position) {
        printf("Position is out of bounds.\n");
        return last;
    }
    prev->next = current->next;
    free(current);
    return last;
}
 void displayList(struct Node* last) {
    if (last == NULL) {
        printf("List is empty.\n");
```

```
        return;
    }
    struct Node* current = last->next;
    printf("Circular Linked List: ");
    do {
        printf("%d -> ", current->data);
        current = current->next;
    } while (current != last->next);
    printf("\n");
}
 void freeList(struct Node* last) {
    if (last == NULL) {
        return;
    }
    struct Node* current = last->next;
    while (current != last) {
        struct Node* temp = current;
        current = current->next;
        free(temp);
    }
    free(last);
}
 int main() {
    struct Node* last = NULL;
     last = insertAtBeginning(last, 10);
    last = insertAtBeginning(last, 5);
    printf("Linked list after insertions at beginning: ");
    displayList(last);
     last = insertAtEnd(last, 20);
    last = insertAtEnd(last, 30);
```

```
    printf("Linked list after insertions at end: ");
    displayList(last);
     last = insertAtPosition(last, 15, 2);
    printf("Linked list after insertion at position 2: ");
    displayList(last);
     last = deleteAtBeginning(last);
    last = deleteAtEnd(last);
    printf("Linked list after deletion from beginning and end: ");
    displayList(last);
     last = deleteAtPosition(last, 2);
    printf("Linked list after deletion at position 2: ");
    displayList(last);
 list
    freeList(last);


    return 0;
}
```

OUTPUT:

Linked list after insertions at beginning: Circular Linked List: 5 -> 10 ->

Linked list after insertions at end: Circular Linked List: 5 -> 10 -> 20 -> 30 ->

Linked list after insertion at position 2: Circular Linked List: 5 -> 15 -> 10 -> 20 -> 30 ->

Linked list after deletion from beginning and end: Circular Linked List: 15 -> 10 -> 20 ->

Linked list after deletion at position 2: Circular Linked List: 15 -> 20 ->