

Output:

MENU

- =====
1. Append as element
 2. Delete last element
 3. Insert element at position
 4. Delete element at position
 5. Insert element at after value
 6. Insert element before value
 7. Display the list
 8. Display the list in reverse order
 9. Exit
- =====

1

Enter element to append: 1
(Prints menu again)

1

Enter element to append: 2
(Prints menu again)

2

(Prints menu again)
 $\leftarrow 1000\ldots 00 \mid 1 \mid 1000\ldots 00 \rightarrow$

(Prints menu again)

3

Enter element to insert : 1
Enter position to insert at : 1
(Prints menu again)

7

$\leftarrow 1000\ldots 00 \mid 1 \mid 681450 \mid \rightarrow \leftarrow \mid 681450 \mid 1 \mid 1000\ldots 00 \rightarrow$
(Prints menu again)

Lab Sheet 5

1. Write a menu-driven C program using structures to implement the following operations on a Doubly linked list.
- Insert an element at the rear end of the list.
 - Delete an element from the rear end of the list.
 - Insert an element at a given position in the list.
 - Delete an element from a given position in the list.
 - Insert an element after a node containing a specific value.
 - Insert an element before a node containing a specific value.
 - Traverse the list in forward direction.
 - Traverse the list in reverse direction.

Source Code :

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

```
struct Node {
    int data;
    struct Node *prev, *next;
```

6

Enter element to insert: 5

Enter element to insert before: 1

(Prints menu again)

8

$\leftarrow | 6 8 1 4 7 0 \right| | 1 | 0 0 0 \dots 0 0 | \rightarrow < | 6 8 1 7 8 0 \right| | 1 | 6 8 1 4 5 0 |$

$\rightarrow < | 0 0 0 0 \right| | 5 | 0 0 0 0 | 6 8 1 4 7 0 | \rightarrow$

(Prints menu again)

12

Invalid choice!

(Prints menu again)

9

```
} *head = NULL, *tail = NULL;
```

```
const char *memfail = "Failed to dynamically  
allocate memory!\n";  
const char *notfound = "Element not found!\n";  
const char *bar = "======"
```

```
void lastInsert(int data) {
```

```
    struct Node *newNode = (struct Node *) malloc  
(sizeof(struct Node));  
    if (!newNode) {  
        perror(memfail);  
        return;  
    }
```

```
    newNode->data = data;
```

```
    newNode->next = newNode->prev = NULL;
```

```
    if (!head) {
```

```
        head = tail = newNode;  
        return;
```

```
}
```

```
    tail->next = newNode;
```

```
    newNode->prev = tail;
```

```
    tail = newNode;
```

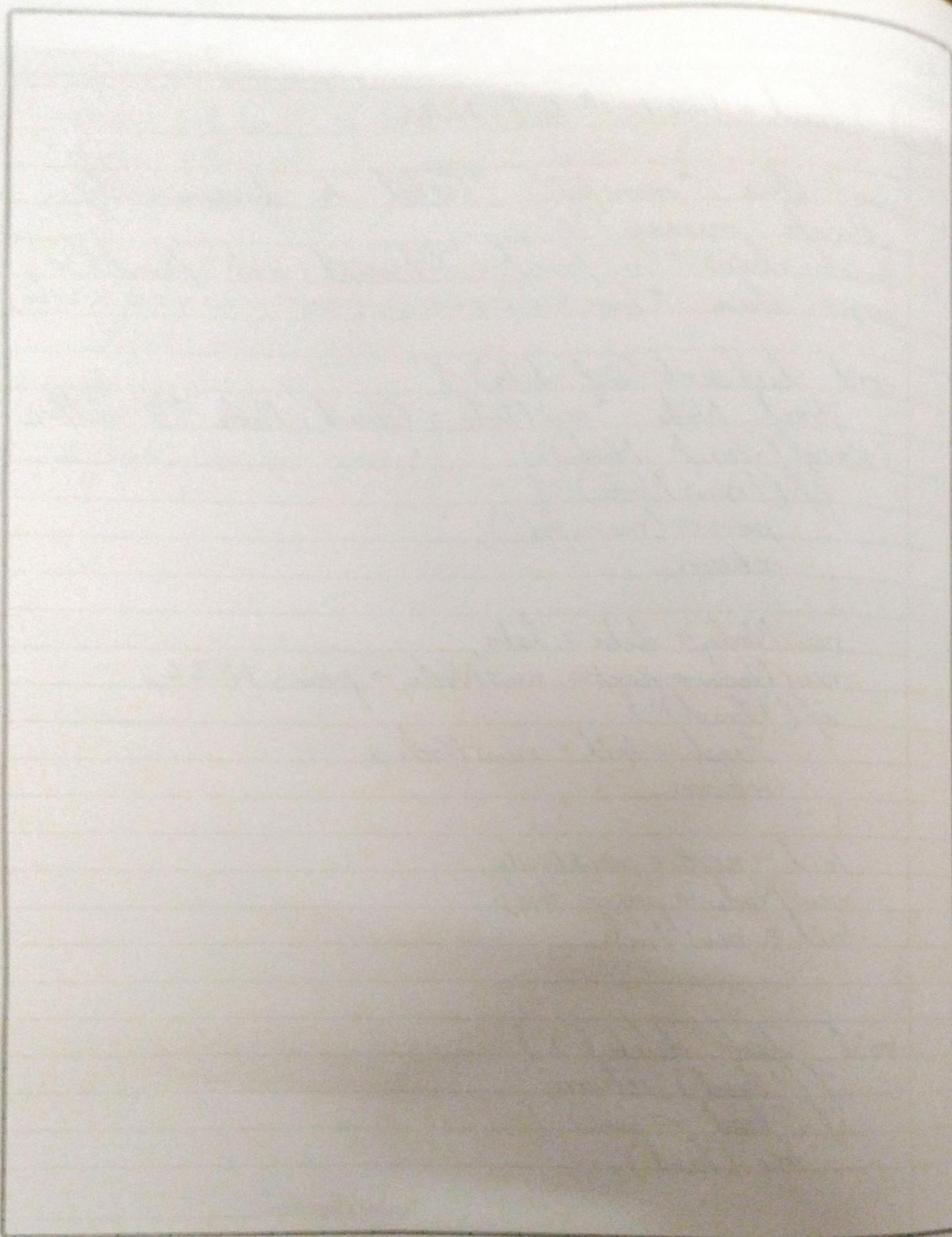
```
}
```

```
void lastDelete() {
```

```
    if (!head) return;
```

```
    if (head == tail) {
```

```
        free(head);
```



```
head = tail = NULL;
```

```
return;
```

```
}
```

```
tail -> prev -> next = NULL;
```

```
struct Node *temp = tail;
```

```
tail = tail -> prev;
```

```
free(temp);
```

```
}
```

```
void insert(int n, int data) {
```

```
int i;
```

```
if (n < 1) return;
```

```
struct Node *newNode = (struct Node *) malloc  
(sizeof (struct Node)), *temp;
```

```
if (!newNode) {
```

```
perror("mem fail");
```

```
return;
```

```
}
```

```
newNode -> data = data;
```

```
newNode -> prev = newNode -> next = NULL;
```

```
if (n == 1) {
```

```
newNode -> next = head;
```

```
head -> prev = newNode;
```

```
head = newNode;
```

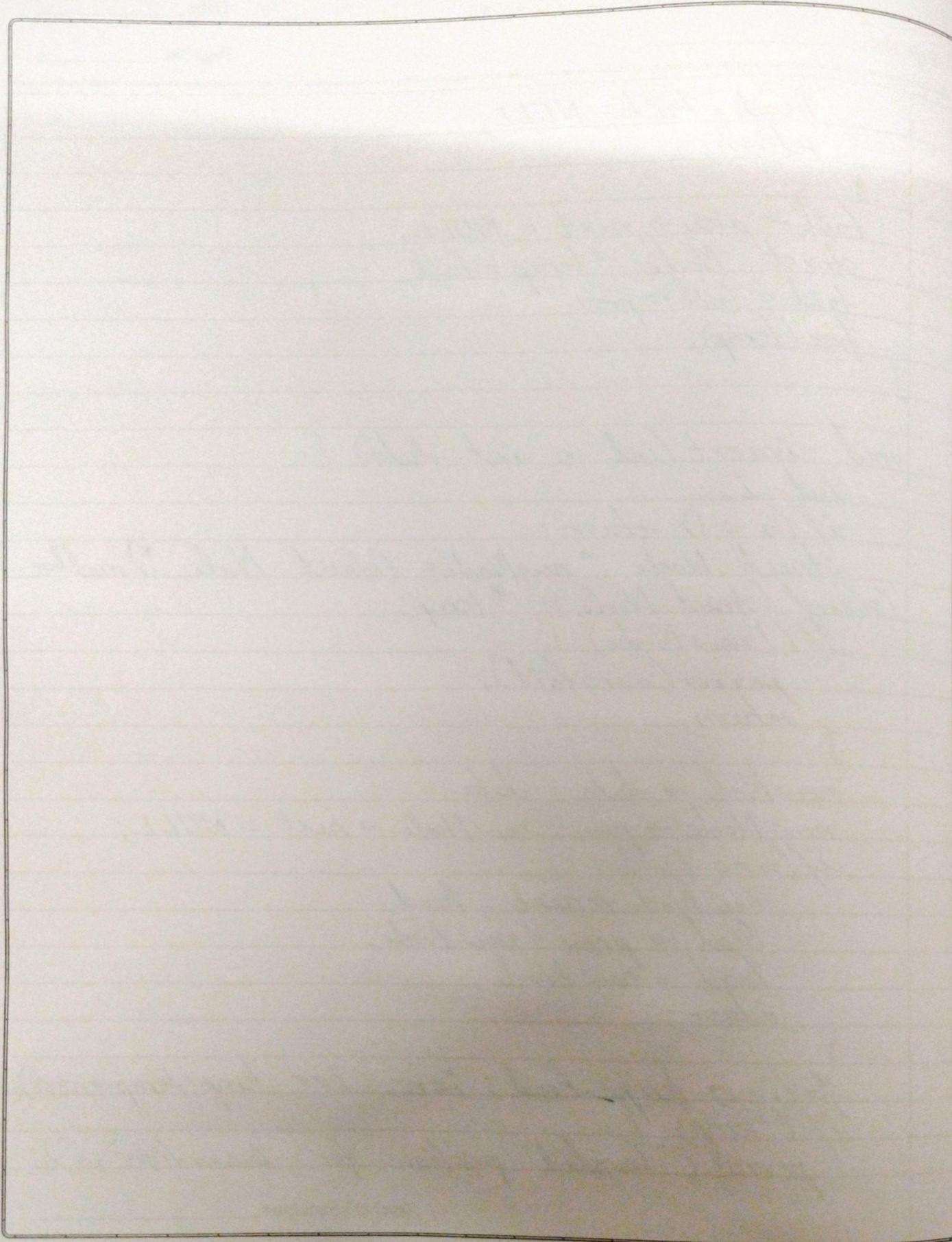
```
return;
```

```
}
```

```
for (i = 1; temp = head; i < n; i++, temp = temp -> next);
```

```
if (!temp) {
```

```
perror ("Invalid position for insertion \n");
```



```
free(newNode);
return;
```

{

```
temp->prev->next = newNode;
newNode->prev = temp->prev;
newNode->next = temp;
temp->prev = newNode;
```

{

```
void delete (int n) {
    if(n < 1 || !head) return;
    int i;
    struct Node *temp = head;
    if(n == 1) {
        if(head->next) head->next->prev = NULL;
        head = head->next;
        free(temp);
        return;
    }
```

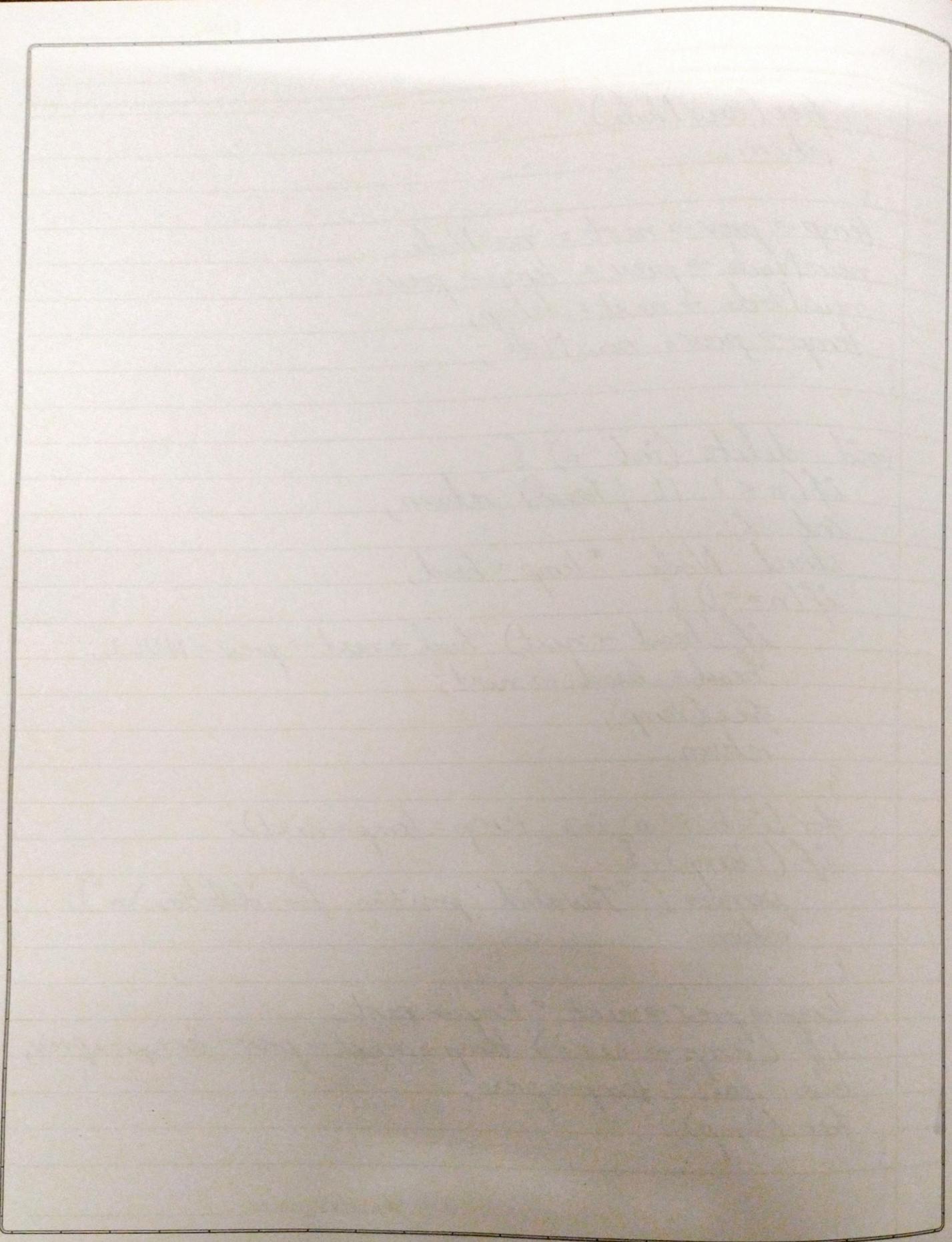
```
for(i=1; i<n; i++, temp = temp->next);
```

```
if (!temp) {
    perror("Invalid position for deletion \n");
    return;
}
```

```
temp->prev->next = temp->next;
```

```
if (temp->next) temp->next->prev = temp->prev;
else tail = temp->prev;
free(temp);
```

{



```

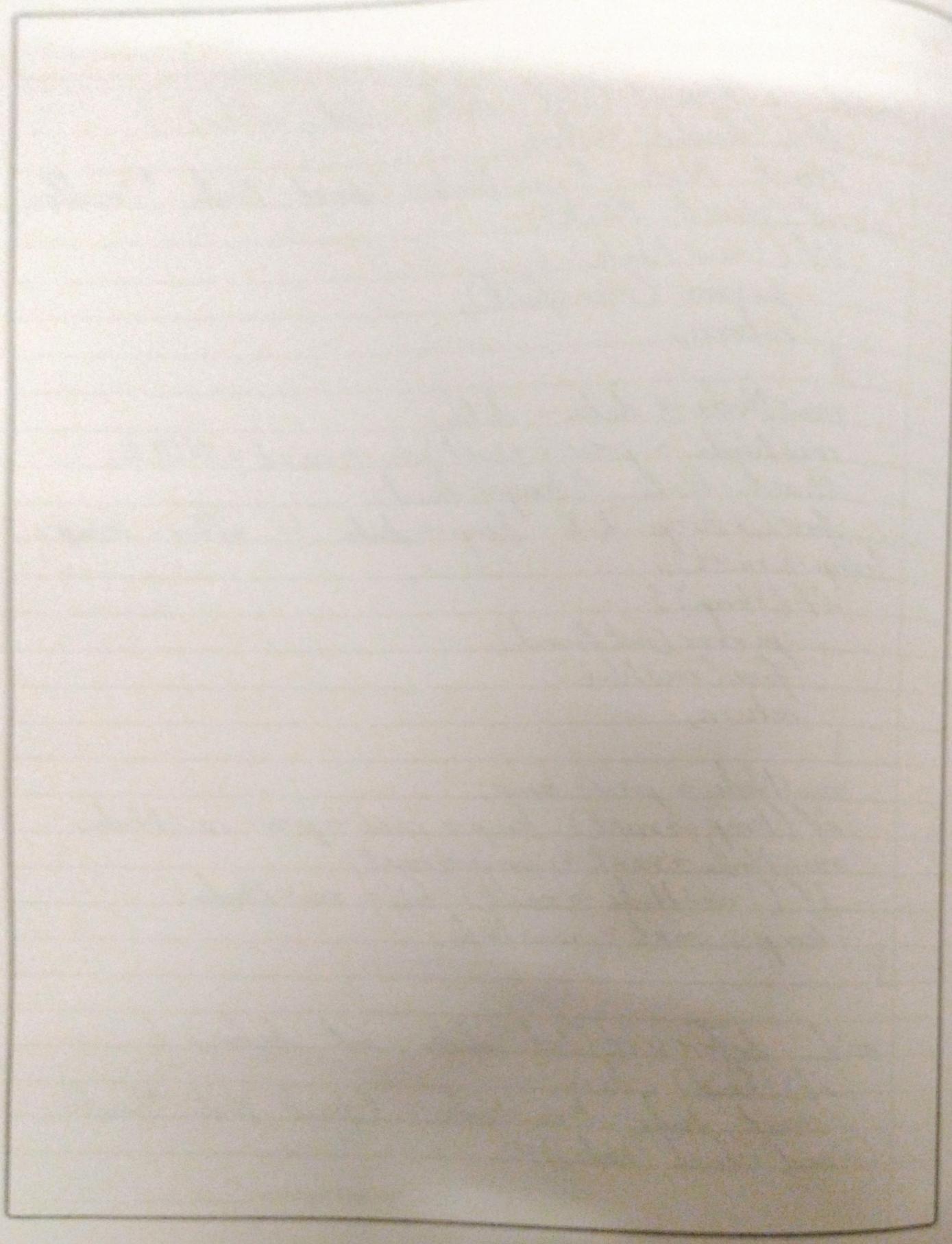
void afterInsert(int data, int after) {
    if (!head) return;
    struct Node *newNode = (struct Node *)malloc(
        sizeof(struct Node));
    if (!newNode) {
        perror("memfail");
        return;
    }
    newNode->data = data;
    newNode->prev = newNode->next = NULL;
    struct Node *temp = head;
    for (; temp && temp->data != after; temp =
        temp->next);
    if (!temp) {
        perror("not found");
        free(newNode);
        return;
    }
    newNode->prev = temp;
    if (temp->next) temp->next->prev = newNode;
    newNode->next = temp->next;
    if (!newNode->next) tail = newNode;
    temp->next = newNode;
}

```

```

void beforeInsert(int data, int before) {
    if (!head) return;
    struct Node *newNode = (struct Node *)malloc(
        sizeof(struct Node));

```



```

if(!newNode) {
    perror(memfail);
    return;
}

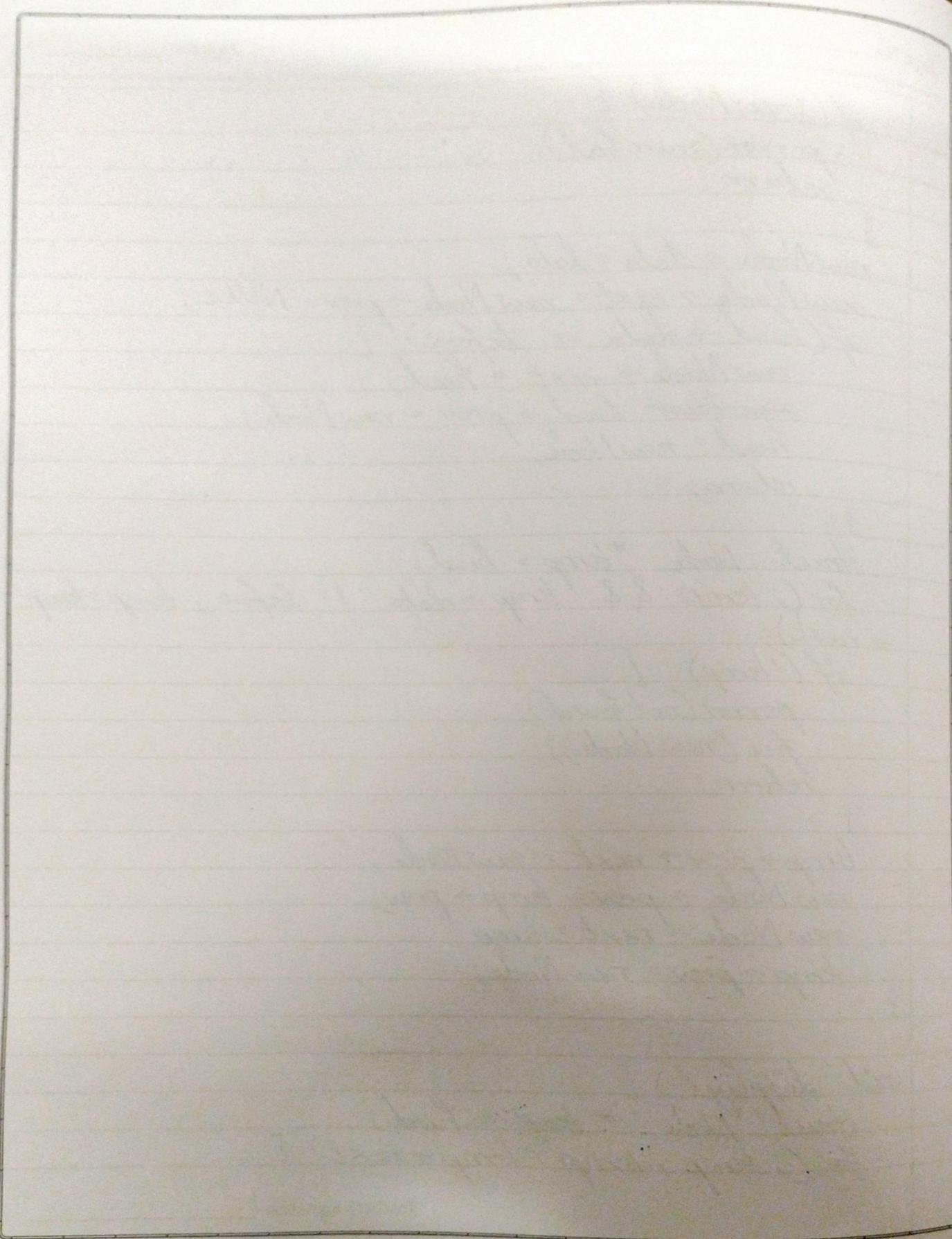
newNode->data = data;
newNode->next = newNode->prev = NULL;
if(head->data == before) {
    newNode->next = head;
    newNode head->prev = newNode;
    head = newNode;
    return;
}

struct Node *temp = head;
for(; temp && temp->data != before; temp = temp->next);
if(!temp) {
    perror(notfound);
    free(newNode);
    return;
}

temp->prev->next = newNode;
newNode->prev = temp->prev;
newNode->next = temp;
temp->prev = newNode;
}

void display() {
    struct Node *temp = head;
    for(; temp; temp = temp->next) {
}

```

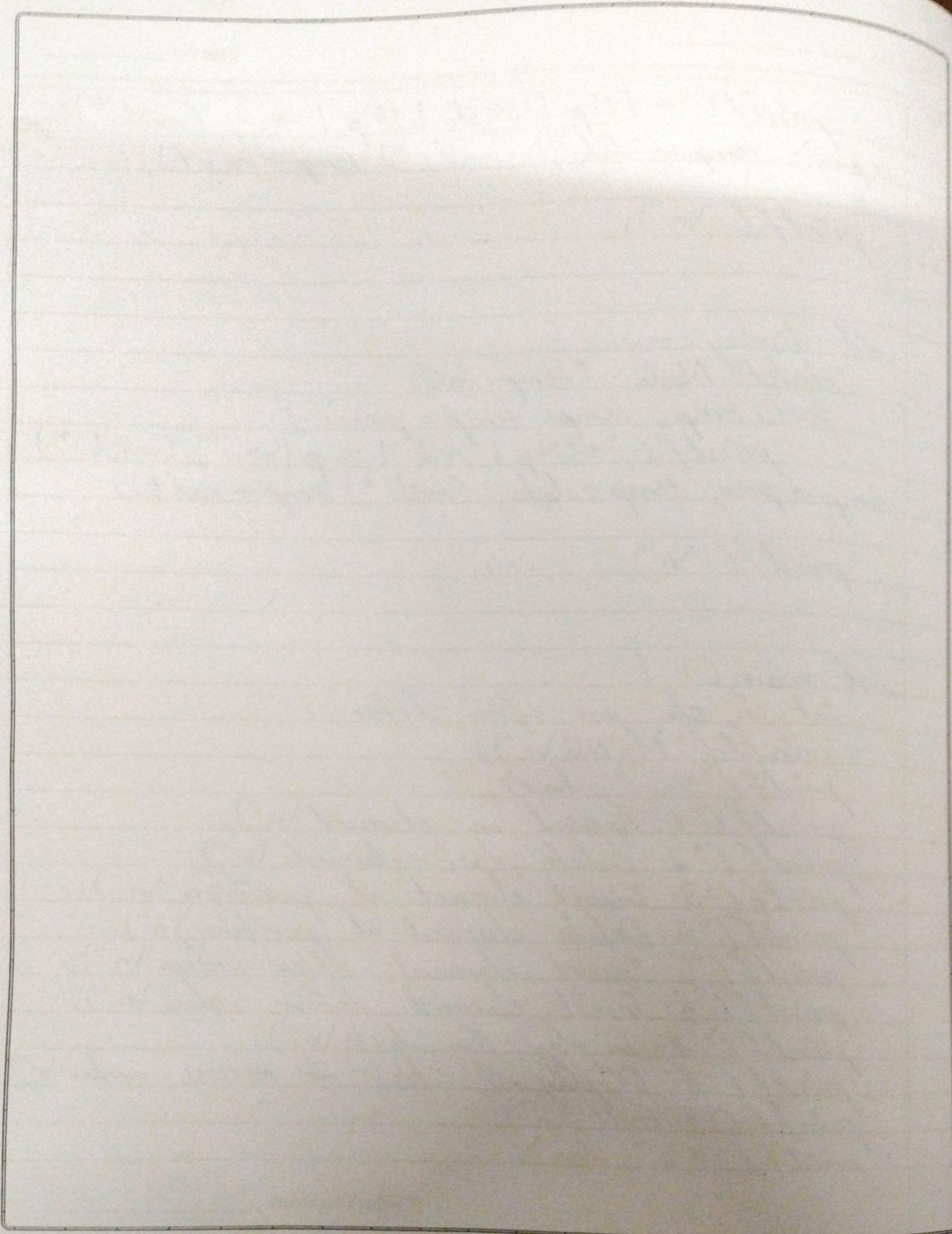


```
printf("<|%p | %d |%p|→", (void *)temp
      → prev, temp → data, (void *)temp → next);
      }
```

```
printf("\n");
      }
```

```
void display_rev() {
    struct Node *temp = tail;
    for ( ; temp; temp = temp → prev) {
        printf("<|%p | %d |%p|→", (void *)temp
              → prev, temp → data, (void *)temp → next);
    }
    printf("\n");
}
```

```
int main() {
    int ch, ele, pos, after, before;
    printf("MENU\n");
    printf("%s", bar);
    printf("1. Append an element\n");
    printf("2. Delete last element\n");
    printf("3. Insert element at position\n");
    printf("4. Delete element at position\n");
    printf("5. Insert element after value\n");
    printf("6. Insert element before value\n");
    printf("7. Display the list\n");
    printf("8. Display the list in reverse order\n");
    printf("9. Exit\n");
    printf("%s", bar);
```



```
scanf("%d", &ch);
```

```
switch(ch) {
```

```
case 1:
```

```
    printf("Enter element to append: ");
```

```
    scanf("%d", &ele);
```

```
    lastinsert(ele);
```

```
    break;
```

```
case 2:
```

```
    lastDelete();
```

```
    break;
```

```
case 3:
```

```
    printf("Enter element to insert: ");
```

```
    scanf("%d", &ele);
```

```
    printf("Enter position to insert at: ");
```

```
    scanf("%d", &pos);
```

```
    insert(pos, ele);
```

```
    break;
```

```
case 4:
```

```
    printf("Enter position of element to delete: ");
```

```
    scanf("%d", &pos);
```

```
    delete(pos);
```

```
    break;
```

```
case 5:
```

```
    printf("Enter element to insert: ");
```

```
    scanf("%d", &ele);
```

```
    printf("Enter element to insert after: ");
```

```
    scanf("%d", &after);
```

```
    afterinsert(ele, after);
```

```
    break;
```

```
case 6:  
    printf("Enter element to insert: ");  
    scanf("%d", &ele);  
    printf("Enter element to insert before: ");  
    scanf("%d", &before);  
    beforeinsert(ele, before);  
    break;  
case 7:  
    display();  
    break;  
case 8:  
    display_rev();  
    break;  
case 9:  
    while(head) last-delete();  
    return 0;  
default:  
    printf("Invalid choice!\\n");  
}  
main();  
}
```

Output:

Enter number of elements in list A:

2

Enter array elements: 1 2

Enter number of elements in list B:

2

Enter array elements: 3 4

Elements in concatenated list:

$\leftarrow |000000| 1 |661490 | \rightarrow \leftarrow |661470 | 2 |6614B0|$
 $\rightarrow \leftarrow |661490| 3 |6614D0| \rightarrow \leftarrow |6614B0| 4 |000000|$
 \rightarrow

Lab Sheet 5

Continued

2. Write a program to concatenate two doubly linked lists X1 and X2. After concatenation, X1 should point to the first node of the resulting list.

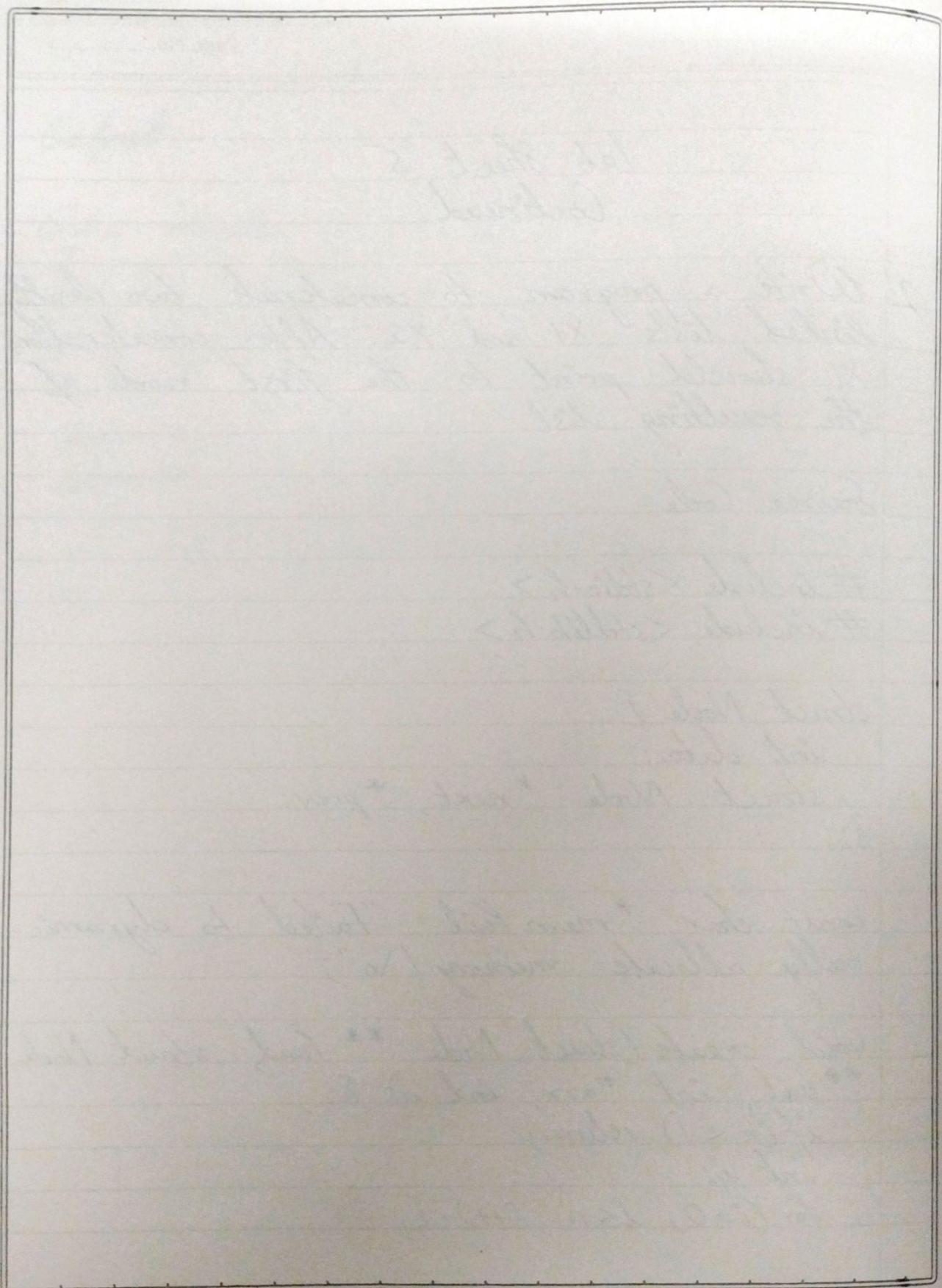
Source Code:

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

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

```
const char *memfail = "Failed to dynamically allocate memory! \n";
```

```
void create(struct Node ** head, struct Node
** tail, int *arr, int n) {
    if(n < 1) return;
    int i;
    for(i=0; i<n; i++) {
```



```

struct Node * newNode = (struct Node *)
malloc(sizeof(struct Node));
if (!newNode) {
    perror("memfail");
    return;
}

```

```

newNode->data = arr[i];
newNode->prev = newNode->next = NULL;
if (*head == NULL) {
    *head = *tail = newNode;
    continue;
}

```

```

(*tail)->next = newNode;
newNode->prev = *tail;
*tail = newNode;
}

```

```

void concat(struct Node **tail1, struct Node
**head2) {

```

```

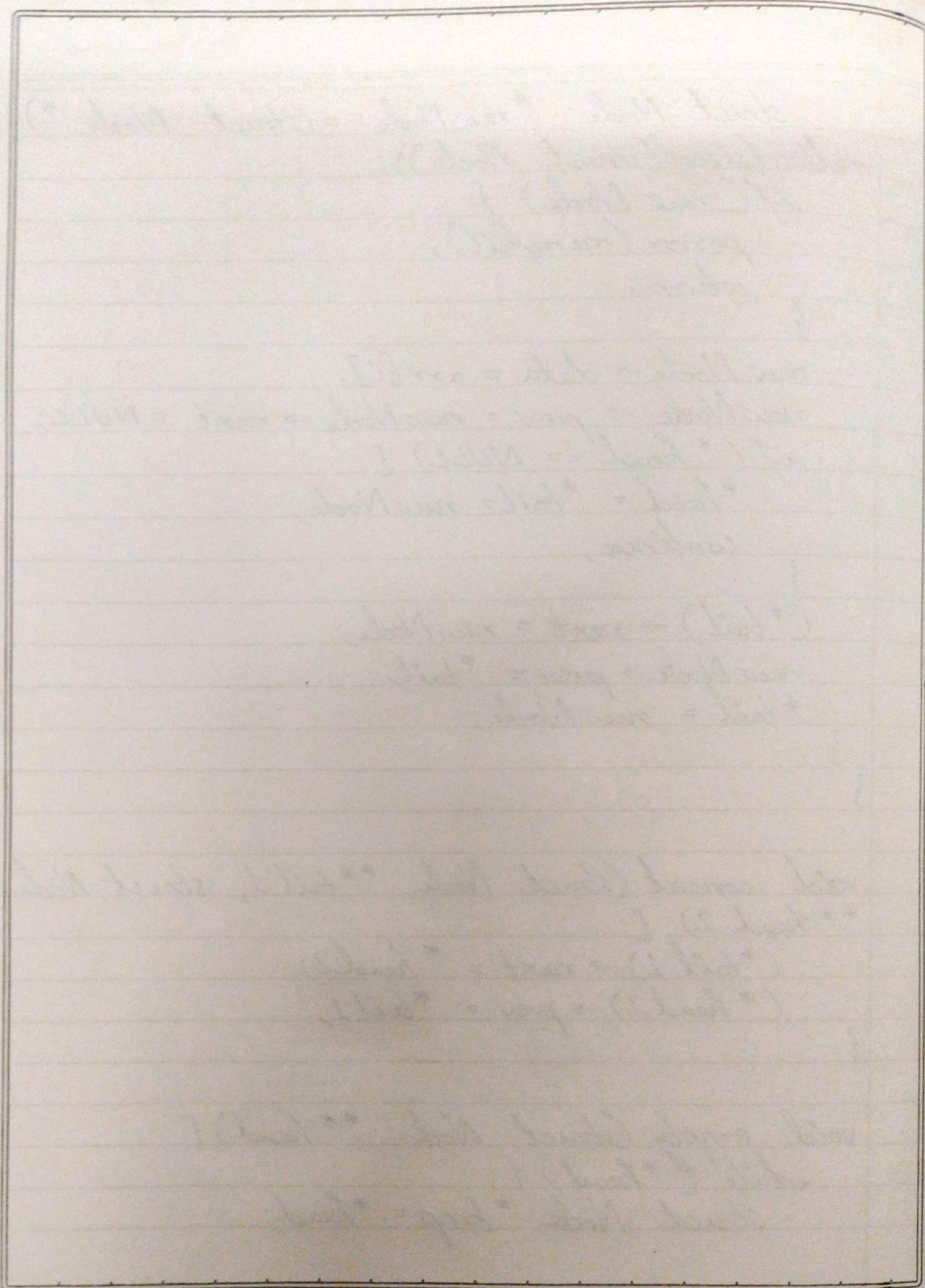
    (*tail1)->next = *head2;
    (*head2)->prev = *tail1;
}

```

```

void empty(struct Node **head) {
    while (*head) {
        struct Node *tmp = *head;

```



```

*head = (*head) → next;
free(temp);
}
}

```

```

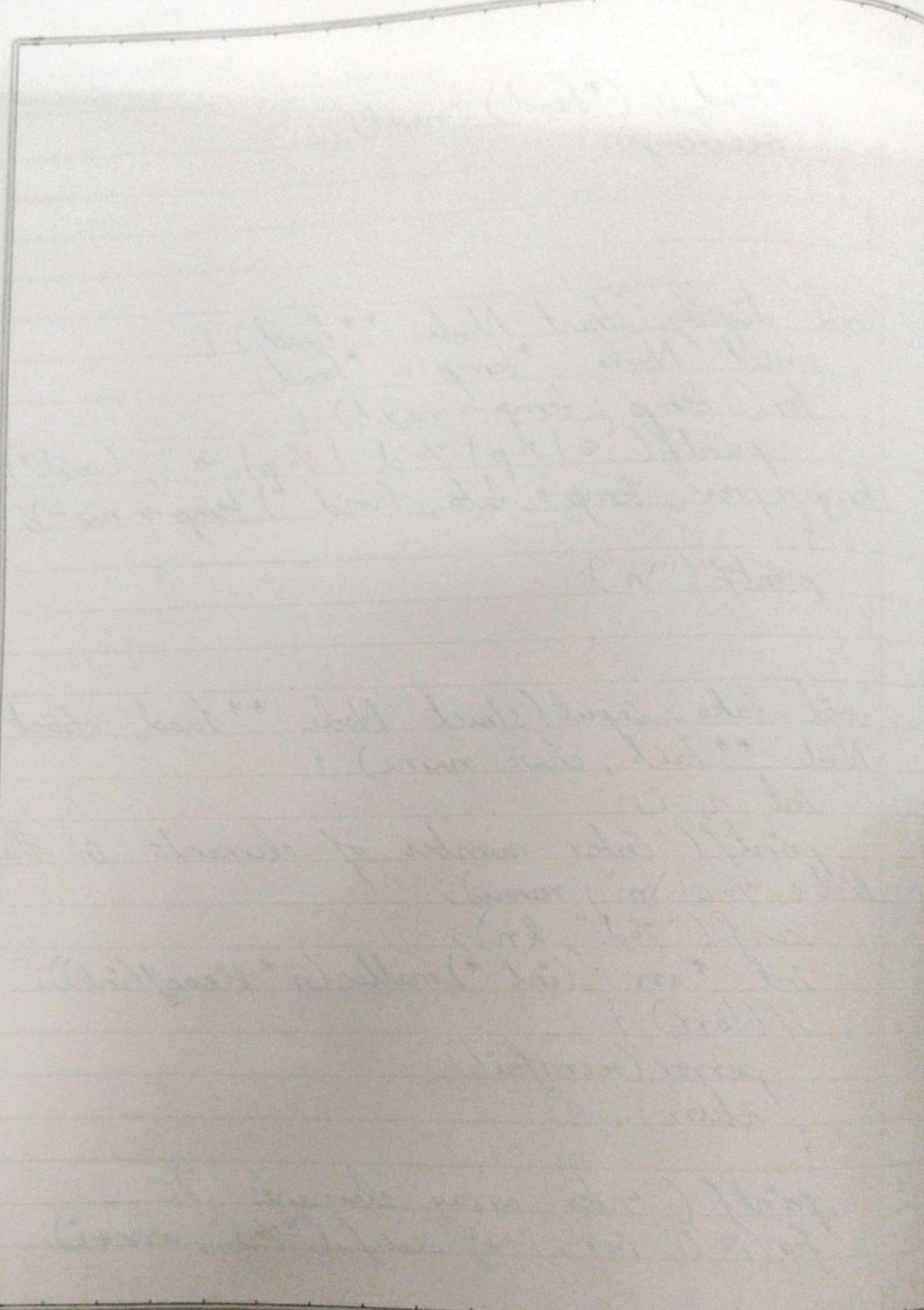
void display(struct Node **head) {
    struct Node *temp = *head;
    for(; temp; temp → next) {
        printf("<| %p |%d| %p|>", (void *)
temp → prev, temp → data, (void *) temp → next);
    }
    printf("\n");
}

```

```

void take_input(struct Node **head, struct
Node **tail, char name) {
    int n, i;
    printf("Enter number of elements in the
list: %c:\n", name);
    scanf("%d", &n);
    int *arr = (int *) malloc(n * sizeof(int));
    if(!arr) {
        perror(memfail);
        return;
    }
    printf("Enter array elements: ");
    for(i=0; i<n; i++) scanf("%d", arr+i);
}

```



create(head, tail, arr, n);
free(arr);
}

int main() {
 struct Node *head1, *tail1, *head2, *tail2;
 head1 = tail1 = head2 = tail2 = NULL;
 take_input(&head1, &tail1, 'A');
 take_input(&head2, &tail2, 'B');
 concat(&tail1, &head2);
 printf("Elements in the concatenated list:
\\n");
 display(&head1);
 empty(&head1);
 return 0;

{