

## LAB 6. (25/01/24)

2) Concatenation, sorting, reversing linked list

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
}
```

void

```
struct node * head = NULL; // head - head  
void main()
```

```
{
```

```
    int n, i, data; // data - data  
    printf ("Enter the no. of elements in first
```

```
linked list \n"); // list - list  
    scanf ("%d", &n);
```

```
    for (i=0; i<n; i++) // loop - loop  
    {
```

```
        struct node * last1 = head1; // head1 - head1
```

```
        struct node * new_node1; // new_node1 - new_node1
```

```
        new_node1 = (struct node*) malloc (sizeof  
        (struct node));
```

```
        scanf ("%d", &data); // data - data
```

```
        new_node1->data = data; // data - data
```

```
        new_node1->next = NULL; // next - next
```

```
        if (head1 == NULL)
```

```
            head1 = new_node1; // head1 - head1
```

```
        }
```

```
    }
```

else

{

    while (last1->next != NULL)

{

        last1 = last1->next;

}

    last1->next = new\_node;

}

printf("Elements after sorting\n");

void sort();

printf("Elements after reversing\n");

void reverse();

struct node \* head1 = NULL; struct node \* last2 = head1;

printf("Enter the no. of elements in second  
linked list to concatenate\n");

int n;

scanf("%d", &n);

printf("Enter the data elements\n");

for (i = 0; i < n; i++)

{

    struct node \* last2 = head1;

    struct node \* new\_node2;

    new\_node2 = (struct node \*) malloc(sizeof(struct node));

    scanf("%d", &data);

    new\_node2->data = data;

    new\_node2->next = NULL;

    if (head1 == NULL)

{

        head1 = new\_node2;

}

Page

Date

```
else  
{
```

```
    while (last2->next != NULL)  
    {
```

```
        last = last2->next; // last = head
```

```
}
```

```
    last2->next = newnode2; // new node 2
```

```
}
```

```
printf("Elements after concatenation are\n");
```

```
void concat();
```

```
}
```

```
void sort()  
{
```

```
    struct node *curr = head * curr->data;
```

```
    struct node *ptr = NULL;
```

```
    int temp;
```

```
    while (curr != NULL)
```

```
    {
```

```
        ptr = curr->next; // min value
```

```
        if (
```

```
            while (ptr != NULL)
```

```
            {
```

```
                if (curr->data > ptr->data)
```

```
                {
```

```
                    temp = curr->data;
```

```
                    curr->data = ptr->data;
```

```
                    ptr->data = temp;
```

```
                }
```

```
                if (ptr = ptr->next);
```

```
            }
```

```
            curr = curr->next;
```

```
        }
```

```
    }
```

```
    Page
```

```
Date
```

```

struct node *node = head1; // if list is empty
if (head == NULL)
{
    printf("List is empty\n");
}
else
{
    while (node != NULL)
    {
        printf("%d \n", node->data);
        node = node->next;
    }
}

```

```

void reverse()
{
    struct node *ptr, *prev;
    struct node *ptr = NULL; // head1 = NULL
    while (head1 != NULL)
    {
        ptr = head1->next;
        head1->next = prev;
        prev = head1;
        head1 = ptr;
    }
    head1 = prev;
}

```

```

struct node *node = head1;
if (head == NULL)
{
}

```

```
    } printf(" list is empty \n");  
}  
else  
{  
    while(node != NULL)  
{  
        printf("%d -> ", node->data);  
        node = node->next;  
    }  
    printf("\n");  
}  
}
```

```
void concat()
```

```
{ struct node *temp ;
```

```
if(head1 == NULL) // if first list is empty  
{ Create a new list.
```

```
struct node *node = head2;
```

```
if(head2 == NULL) // if second list is empty  
{
```

```
    printf(" list is empty \n");
```

```
else  
{
```

```
    while(node != NULL)
```

```
        printf("%d -> ", node->data);
```

```
        node = node->next;
```

```
}
```

```
    printf("\n");
}
else
{
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = head2;
}
```

$\therefore$  struct node \* node = head1;

```
while (node != NULL)
{
```

```
    printf("%d -> ", node->data);
    node = node->next;
}
```

}

Output: Enter the number of elements in linked list

3

Enter the elements to be inserted for first linked list

3 2 1

Implementation of sort

1 → 2 → 3 →

Implementation of reverse.

3 → 2 → 1 →

Enter the number of element in second list

3

Enter the elements to be inserted for second

1 2 3

Elements after concatenation are

3 → 2 → 1 → 1 → 2 → 2 → 3 →

All operation are done

LAB 6. (25/01/24)

## 2) Concatenation, sorting, reversing linked list

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

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next;
```

```
};
```

Code

```
struct node * head = NULL; // main function
```

```
void main()
```

```
{
```

```
    int n, i, data; // head = first node
```

```
    printf ("Enter the no. of elements in first  
linked list (n);
```

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

```
    for (i=0; i<n; i++)
```

```
        struct node * last1 = head1;
```

```
        struct node * new_node1;
```

```
        new_node1 = (struct node *) malloc (sizeof  
        (struct node));
```

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

```
        new_node1->data = data;
```

```
        new_node1->next = NULL;
```

```
        if (head1 == NULL)
```

```
            head1 = new_node1;
```

```
}
```

```
Page _____
```

```

else
{
    while (last1->next != NULL)
    {
        last2 = last1->next;
        last1->next = new_node;
    }
}

printf("Elements after sorting\n");
void sort();

printf("Elements after reversing\n");
void reverse();
struct node * head1 = NULL; struct node * last2 = head1;
printf("Enter the no. of elements in second
linked list to concatenate\n");
int n;
scanf("%d", &n);
printf("Enter the data elements\n");
for (i=0; i<n; i++)
{
    struct node * last2 = head1;
    struct node * new_node2;
    new_node2 = (struct node *) malloc(sizeof(struct node));
    scanf("%d", &data);
    new_node2->data = data;
    new_node2->next = NULL;
    if (head2 == NULL)
    {
        head2 = new_node2;
    }
}

```

```

else
{
    while (last2->next != NULL)
    {
        last = last2->next;
    }
    last2->next = newnode2;
}

printf("Elements after concatenation are \n");
void concat();
void sort();
struct node *curr = head;
struct node *ptr = NULL;
int temp;
while (curr != NULL)
{
    ptr = curr->next;
    if (
        while (ptr != NULL)
        {
            if (curr->data > ptr->data)
            {
                temp = curr->data;
                curr->data = ptr->data;
                ptr->data = temp;
            }
            ptr = ptr->next;
        }
        curr = curr->next;
    }
}

```

```

struct node *node = head1;
if (head == NULL)
{
    printf("list is empty\n");
}
else
{
    while (node != NULL)
    {
        printf("%d - ", node->data);
        node = node->next;
    }
    printf("\n");
}

```

```

void reverse()
{
    struct node *ptr, *prev = NULL;
    struct node *ptr = NULL;
    while (head1 != NULL)
    {
        ptr = head1->next;
        head1->next = prev;
        prev = head1;
        head1 = ptr;
    }
    head1 = prev;
}

```

```

struct node *node = head1;
if (head == NULL)
{

```

```
    } printf("List is empty\n");
}
else
{
    while(node != NULL)
    {
        printf("%d ->", node->data);
        node = node->next;
    }
    printf("\n");
}
```

```
void concat()
```

```
{
    struct node *temp;
    if(head1 == NULL)
    {
        struct node *node = head2;
        if(head2 == NULL)
        {
            printf("List is empty\n");
        }
        else
        {
            while(node != NULL)
            {
                printf("%d ->", node->data);
                node = node->next;
            }
        }
    }
}
```

```
    printf("\n");
}
else
{
    while (temp->next != NULL)
    {
        temp = temp->next;
    }
    temp->next = head2;
}
```

```
∴ xx struct node * node = head1;
while (node != NULL)
{
```

```
    printf("%d → ", node->data);
    node = node->next;
}
```

}

Output: Enter the number of elements in linked list

3

Enter the elements to be inserted for first linked list

3 2 1

Implementation of sort

1 → 2 → 3 →

Implementation of reverse

3 → 2 → 1 →

Enter the number of element in second list

3

Enter the elements to be inserted for sort

1 2 3

Elements after concatenation are

3 → 2 → 1 → 1 → 2 → 2 → 3 →

All operation are done

# LAB 6. (25/01/24)

## Stack. Implementation

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

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

struct node *head = NULL;
```

```
void main()
{
    int ch;
    printf (" Enter the choice.\n");
    scanf ("%d", &ch);
    printf (" Enter 1 : pop \n 2 : push );
    switch (while (ch != 3))
    {
        case 1 : pop();
                    break;
        case 2 : push();
                    break;
        case 3 : exit(); display();
                    break;
        case 4 : exit();
                    break;
    }
}
```

```
void push()
{
    int x, i, data;
    printf("Enter the data\n");
    struct node *last = head;
    struct node *new_node;
    new_node = (struct node*)malloc(sizeof(struct node));
    scanf("%d", &data);
    new_node->data = data;
    new_node->next = NULL;
    if (head == NULL)
        head = new_node;
    else
        {
            while (last->next != NULL)
                last = last->next;
            last->next = new_node;
        }
}
```

```
void pop()
{
    struct node *ptr;
    struct node *ptr1;
    if (head == NULL)
        printf("List is empty\n");
    else
        {
            ptr = head;
            head = head->next;
            free(ptr);
        }
}
```

```
printf("List is empty\n");
```

else if (head  $\rightarrow$  next == NULL)

} free(head);

{ else

ptr = head;

while ((ptr  $\rightarrow$  next) == NULL)

ptr = ptr;

ptr = ptr  $\rightarrow$  next;

};

free(ptr);

ptr  $\rightarrow$  next = NULL;

printf("Element at %d is popped\n");

}.

void display()

{

struct node \* node = head;

if (head == NULL)

}; printf("List is empty\n");

else

while (node != NULL)

printf("%d \rightarrow ", node  $\rightarrow$  data);

node = node  $\rightarrow$  next;

printf("\n");

## Queue Implementation

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

struct node * head, * front, * rear, * temp;
int data;
struct node * next; int ch;
struct node * head = NULL; int i, j, k;
```

void main()

```
{
```

printf("Enter 1: enqueue\n 2: dequeue\n 3: display")

```
int ch;
```

printf("Enter the choice\n");

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

```
while(ch != 4)
```

```
{
```

switch(ch):

```
case 1: enqueue(); front = tail;
```

```
break;
```

```
case 2: dequeue();
```

```
break;
```

```
case 3: display();
```

```
break;
```

```
case 4: exit();
```

```
}
```

```
}
```

```
void enqueue()
```

```
{
```

```
    int data;
```

```
    printf("Enter the data to be inserted \n");
```

```
    struct node * last = head;
```

```
    struct node * new_node;
```

```
    new_node = (struct node *) malloc(sizeof(struct node));
```

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

```
    new_node->data = data;
```

```
    new_node->next = NULL;
```

```
    if (head == NULL)
```

```
{
```

```
    head = new_node;
```

```
}
```

```
else
```

```
{
```

```
i. (Previously add node) Then,
```

```
while (last->next != NULL)
```

```
{
```

```
    last = last->next;
```

```
}
```

```
last->next = new_node;
```

```
}
```

```
void dequeue()
```

```
struct node * ptr;
```

```
if (head == NULL)
```

```
{
```

```
    printf("List is empty \n");
```

```
}
```

```
else
```

```

ptr = head;
head = head->next;
free(ptr);
ptr->printf ("First element is deleted\n");
}

void display()
{
    struct node * node = head;
    if (head == NULL)
    {
        printf ("List is empty\n");
    }
    else
    {
        while (node != NULL)
        {
            printf ("%d", node->data);
            node = node->next;
        }
        printf ("\n");
    }
}

```

Output: Enter

1 : enqueue

2 : dequeue

3 : display

4 : exit

Enter the choice

1

Enter the data to be inserted

1  
Enter the choice

2  
Enter the data to be inserted

3  
Enter the choice

1

2  
Enter the data to be inserted

3

4  
Enter the choice

5  
Enter the choice

3

6  
1 → 2 → 3 →

7  
Enter the choice

8  
Enter & 2

9  
First element is deleted

10  
Enter the choice

11  
3

12  
2 → 3 →

13  
Enter the choice

14  
4

15  
output for stack implementation:

16  
Enter 1 : pop

17  
2 : push

18  
3 : display

19  
4 : exit

20  
Enter the choice

21  
2

22  
Enter the data to be inserted

23  
1

24  
Enter the choice

25  
2

1 Enter the data to be inserted

2

Enter the choice

2

Enter the data to be inserted

3

Enter the choice

1

Element at the end is deleted

Enter the choice

4.