LAB 6. (25/01/24)

2) Concatenation, sorting, reversing Linked list

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

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

struct node * head = NULL;
void main()
{
    int n, i, data;
    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_node -> next = NULL;
        if(head1 == NULL)
        {
            head1 = new_node1;
        }
```

```
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 = head;
    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 = new node2;
}
}
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(head2 == 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;

        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;
};

void
struct node* head = NULL;
void main()
{
    int n, i, data;
    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_node -> next = NULL;
        if (head1 == NULL)
        {
            head1= new_node1};
        }
```

```
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 = head;
  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 = new_node2;
}
}
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;

    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)

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");
    while(ch != 4)
    {
    switch(ch)
    {
        case 1: pop();
                break;
        case 2: push();
                break;
        case 3: display();
                break;
        case 4: exit();
    }
}
```

```
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;
    }
}
```

15/2/24

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

```
    else if (head -> next == NULL)
    {
        free(head);
    }
    else
    {
        ptr = head;
        while (ptr -> next != NULL)
        {
            ptr1 = ptr;
            ptr = ptr -> next;
        }
        free(ptr);
        ptr1 -> next = NULL;
        printf(" element %d is poped \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("X\n");
    }
}
```

Queue Implementation

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

struct node **
{
    int data;
    struct node * next;
};
struct node * head = NULL;

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

        case 2: deque();
                break;

        case 3: display();
                break;
        case 4: exit();
    }
}
}
```

```
void do_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
    {
        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);
        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

1

Enter the data to be inserted

2

Enter the choice

1

Enter the data to be inserted

3

Enter the choice

3

1 → 2 → 3 →

Enter the choice

Enter 2

First element is deleted

Enter the choice

3

2 → 3 →

Enter the choice

4

output for stack implementation:

Enter 1: pop
2: push
3: display
4: exit
Enter the choice
2
Enter the data to be inserted
1
Enter the choice
2

Enter the data to be inserted
2
Enter the choice
2
Enter the data to be inserted
3
Enter the choice
1
Elemenent at the end is deleted
Enter the choice
4.