DS-lab → 22/01/2024

PAGE NO :
DATE :

Q1) WAP to implement singly linked list with following operations
a.) Create a linked list.
b.) Insertion of a node at first position at any position and at end of list
Display the contents of linked list.

Ans →

```
# include <stdio.h>
# include <stdlib.h>
struct Node {
        int data;
        struct Node * next ;
    };
~~void insert~~
struct Node* createNode(int data){
        struct Node * newNode = (struct Node *)
                malloc (sizeof(struct node));
    if (newNode == NULL){
        printf (" Memory allocation failed ");
        exit (1);
    }
    newNode → data = data ;
    newNode → next = NULL ;
    return newNode ;
}
struct Node* createLinkedList (int values[], int size)
{ struct Node * head = NULL ;
    struct Node * tail = NULL ;
    for( int i =0 ; i< size ; i++){
        struct Node * newnode = create Node
                (values[i]);
        if (head == NULL) {
```

```
    head = newNode;
    tail = newNode;
}
else {
    tail -> next = newNode;
    tail = newNode;
}
else { tail -> next = newNode; }
}
} return head;
}
void insertFirst (struct Node* head, int data) {
    struct Node* newNode = createNode(data);
    newNode -> next = *head;
    *head = newNode;
}
void insertAtPosition (struct Node** head, int data
    , int position) {
    if (position == 0) {
        insertFirst(head, data);
        return;
    }
    struct Node* new_node = createNode(data);
    struct Node* current = *head;
    for (int i = 0; i < position - 1; i++) {
        if (current == NULL) {
            printf ("Invalid position");
            return;
        }
        current = current -> next;
    }
    new_node -> next = current -> next;
    current -> next = new_node;
}
void insertend (struct Node **hed, int data) {
    struct Node * new_node = createNode(data);
    if (*head == NULL) {
        *head = new_node;
        return;
    }
    struct Node* current = *head;
    while (current -> next != NULL) {
        current = current -> next;
    }
    current -> next = new_node;
}
void display (struct Node * head) {
    while (head != NULL) {
        printf ("%d -> " head -> data);
        head = head -> next;
    }
    printf ("NULL \n");
}
int main()
    struct Node * linkedList = createLinkedList();
    int data;
    printf ("Enter data to insert at begining");
    scanf ("%d", &data);
    insertFirst (&linkedlist, &data);
    int position;
    printf ("Enter data to insert at a
            specific position :");
    scanf ("%d", &data);
    printf ("inter position:");
    scanf ("%d", &position);
    insertAtPosition (&linkedlist, data, position);
```

```
printf("Enter data to insert at end:");
scanf("%d", &data);
insertEnd(&linkedlist, data);
display(linkedlist);
return 0;
}
```

Output :-

```
Enter number of elements : 4
enter the elements :
1 2 3 4
Enter data to insert at the beginning : 5
Enter data to insert at a specific position : 34
Enter the position : 2
Enter data to insert at the end : 66
5 → 1 → 34 → 2 → 3 → 4 → 66 → NULL
```

Q2.) WAP to implement singly linked list with following operations.

a) Create a linked list.
b) Deletion of first element, specified element and last element in list.
Display the contents of linked list.

Ans →

```
#include <stdio.h>
#include <stdio.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");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}
struct Node* createLinkedList(){
    struct Node* head = NULL;
    struct Node* tail = NULL;
    int size, data;
    printf("Enter no. of elements:");
    scanf("%d", &size);
    printf("Enter the elements:\n");
    for(int i=0; i<size; i++){
        scanf("%d", &data);
        struct Node* newNod = createNode(data);
        if(Nod == NULL){
            head = newNode;
            tail = newNode;
        }
        else
        { tail->next = newNode;
          tail = newNode;
        }
    }
    return head;
}
```

```
void deleteFirst(struct Node** head) {
    if(*head == NULL) {
        printf("List is empty :");
        return;
    }
    struct Node* temp = *head;
    *head = (*head) -> next;
    free(temp);
}
void deleteElement(struct Node** head, int key){
    if (*head == NULL){
        printf("List is empty");
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current!=NULL && current -> data!=key){
        prev = current;
        current = current -> next;
    }
    if (current == NULL){
        printf("Element not found");
        return;
    }
    if(prev == NULL){
        *head = current -> next;
    }
    else{ prev -> next = current -> next; }
    free(current);
}
```

```
void deleteLast(struct Node** head){
    if (*head == NULL){
        printf("List is empty");
        return;
    }
    if ((*head) -> next == NULL){
        free(*head);
        *head = NULL;
        return;
    }
    struct Node* current = *head;
    struct Node* prev = NULL;
    while (current -> next != NULL){
        prev = current;
        current = current -> next;
    }
    prev -> next = NULL;
    free(current);
}
void display(struct Node* head){
    while(head != NULL){
        printf("%d -> ", head -> data);
        head = head -> next;
    }
    printf("NULL\n");
}
int main(){
    struct Node* LinkedList = createLinkedList();
    printf("Linked List before deletion:\n");
    display(LinkedList);
    int key;
    printf("Enter element to delete:");
```

```
scanf("%d", &key);
deleteElement(&linkedlist, key);
printf("linked list after deleting");
display(linked list);
delete last(& linked list);
printf("linked list after deleting last element");
display(linked list);
return 0;
}
```

O/P:-

Enter number of elements : 4
enter the elements:
1 2 3 4
linked list before deletion :
1 → 2 → 3 → 4 → NULL

linked list after deleting first element
2 → 3 → 4 → NULL

Enter element to delete : 4
Linked list after deleting specified element :
2 → 3 → NULL

linked list after deleting last element.
2 → NULL

Sat 5/2/24

Sat → 29/1/24

Q.) Single linked list :-
sort;
Reverse;
concatination;

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

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

void insertAtBeg(struct Node** headRef, int data) {
    struct Node* newNode = (struct Node*)
        malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->next = *headRef;
    *headRef = newNode;
}

void printList(struct Node* head) {
    while (head != NULL) {
        printf("%d", head->data);
        head = head->next;
    }
    printf("\n");
}

void sortList(struct Node** head) {
    struct Node *current, *nextNode;
```

```
    int temp;
    current = *head;
    while (current != NULL){
        nextNode = current -> next;
        while ( nextNode != NULL) {
            if ( current -> data > nextNode -> data){
                temp = current -> data;
                current -> data = nextNode -> data;
                nextNode -> data = temp;
            }
            nextNode = nextNode -> next;
        }
        current = current -> next;
    }
}
void reverseList (struct Node ** headRef) {
    struct Node *prev, * current, * nextNode;
    prev = NULL;
    current = *headRef;
    while (current != NULL) {
        nextNode = current -> next;
        prev = current;
        current = nextNode;
    }
    *headRef = prev;
}

void concat(struct Node **list1, struct Node* list2){
    if ( *list1 == NULL) {
        *list1 = list2;
        return;
    }
    struct Node* temp = *list1;
    while ( temp -> next != NULL) {
        temp = temp -> next;
    }
    temp -> next = list2;
}

int main() {
    struct Node * list1 = NULL;
    struct Node* list2 = NULL;
    int choice;
    int data;
    while (1)
    {
        printf("\n1. Insert into List1\n");
        printf("2. Insert into list 2 \n");
        printf("3- Sort list 1 \n");
        printf("4. Reverse List\n");
        printf("5 Concatination of lists");
        printf("6 Print list \n");
        printf("0. Exit\n");
        printf("Enter choice");
        scanf("%d", &choice);
        switch (choice) {
            case 1: printf("Enter data");
                    scanf("%d", &data);
                    insertAtBegin(&list1, data);
                    break;
            case 2: printf("Enter data to insert");
                    scanf("%d", &data);
                    insertAtBegin(&list2, data);
                    break;
```

```
case 3: sortlist (&list1);
        printf ("list 1 sorted\n");
        break
case 4: reverse list (&list1);
        printf ("list 1 reversed");
        break;
case 5: concat (&list1, list2);
        printf ("lists concat");
        break.
case 6: print ("list1");
        printlist(list1);
        print ("list2");
        printlist(list2); break;
case 7: exit(0);
        break;
default: printf ("Invalid choice");
    }
}
return 0;
}
```

O/P

5/2/24

Lab → 2

1.) WAP to Implement doubly link list with primitive operations
a) Create a doubly linked list
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value.

a.) →
```
struct node {
    int data
    struct node * prev;
    struct node * next;
};
struct node * head, *tail;
head = 0;

void create_dll(){
    struct node *newnode;
    newnode = (struct node *) malloc (sizeof(struct node));
    printf("Enter data:");
    scanf("%d", newnode → data);
    newnode → next = 0;
    newnode → prev = 0;

    if (head == 0) {
        head = tail = newnode;
    }
    else {
        newnode → prev = tail;
        tail → next = newnode;
        tail = newnode;
    }
}
```

b.) →

```
void insatleft () {
    struct node *newnode, *ptr ;
    ptr = head;
    newnode = (struct node*) malloc (sizeof (struct node));
    printf ("Enter data:");
    scanf ("%d", &newnode → data);
    int count = 0, pos;
    printf ("Enter position : ");
    scanf ("%d", &pos );
    while (count < pos) {
        ptr = ptr → next ;
        count ++ ;
    }
    newnode → prev = ptr → prev ;
    newnode → next = ptr ;
    ptr → prev → next = newnode ;
    ptr → prev = newnode ;
}
```

o/p :-

```
Enter data : 5
Enter position : 2
```

a.) Output :-

```
Enter data : 5
Enter data : 6
Enter data : 1
```

5 → 6 → 1

c.) →

```
void del_pos() {
    struct node *ptr ; ptr = head ;
    int key , val , flag = 0 ;
    printf ("Enter val ");
    scanf ("%d", &key) ;
    while (ptr != NULL) {
        if (ptr → data == key){
            ptr → prev → next = ptr → next ;
            ptr → next → prev = ptr → prev ;
            flag = 1 ;
            free (ptr) ;
            break ; }
        else {
            ptr = ptr → next ;
        }
    }
    if (flag == 1) {
        printf ("Element deleted");
    }
    else {
        printf ("Elemt not fount ");
    }
}
```

o/p :-

```
Enter value to be deleted : 5
Element deleted

2 3 5 1
2 3 1
```

PAGE NO :
DATE :

Q3) WAP to
a) construct Binary search tree
b) Traverse the tree using inorder, postorder, preorder
c) Display the elements in the tree.

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

Struct btnode
{
    int value;
    struct btnode *left;
    Sbtnode *right;
} *root = NULL, *temp = NULL, *t2, *t1;

void insert();
void create();
void inorder(struct btnode *t);
void preorder(struct btnode *t);
void postorder(struct btnode *t);
struct btnode *
create(){
  int data;
  printf("Enter data of node ");
  scanf("%d", &data);
  temp -> value = data;
  temp -> l = temp -> r = NULL; return temp;
}
```

```
struct btnode* insert() {
    t = create();
    if (t -> l == NULL && t -> r == NULL)
        root = t;
    else {
        if (t -> value > root)
```

```
Struct btnode * insert(struct btnode *root, int val){
    if (root == NULL) {
        return createNode(val);
    }
    if (val < root -> val) {
        root -> left = insert(root -> left, val);
    } else if (val > root -> val) {
        root -> right = insert(root -> right, val);
    } return root;
}

void inorder(struct btnode * root){
    if (root == NULL){
        return;
    }
    inorder(root -> left);
    printf("%d", root -> value); inorder(root->right);
}

void preorder(struct btnode * root){
    if (root == NULL){
        return;
    } printf("%d", root -> val);
    preorder(root -> left); preorder(root -> right);
}
```

```
void display (struct tNode *root) {
    printf ("Elements ");
    inorder (root);
    printf ("\n");

void post_order (struct tNode * root ) {
    if (root == NULL) {
        return;
    }
    post order (root -> left);
    post order (root -> right);
    printf ("%d", root -> data);
}
```