

DS PRACTICAL SLIPS/slip1.1.c

```
1. #include<stdio.h>
2.
3. int binarySearch(int *arr, int n, int num){
4.     int low = 0;
5.     int high = n - 1;
6.     while(low <= high){
7.         int mid = (low + high)/2;
8.
9.         if(arr[mid] == num){
10.             return mid;
11.         }
12.
13.         else if(arr[mid] < num){
14.             low = mid + 1;
15.         }
16.         else{
17.             high = mid - 1;
18.         }
19.     }
20.     return -1;
21. }
22.
23. int main(){
24.     int n;
25.     int num;
26.     printf("Enter n: ");
27.     scanf("%d", &n);
28.     int arr[n];
29.     printf("Enter %d elements: ", n);
30.     for(int i = 0; i < n; i++){
31.         scanf("%d", &arr[i]);
32.     }
33.     printf("Enter number to search: ");
34.     scanf("%d", &num);
35.
36.     int res = binarySearch(arr, n, num);
37.
38.     if(res != -1){
39.         printf("%d found at [%d] position\n", num, res);
40.     }
41.     else{
42.         printf("Not in array\n");
43.     }
44.
45.     return 0;
46. }
47.
```

DS PRACTICAL SLIPS/slip1.2.c

```
1. #include<stdio.h>
2. #include<stdbool.h>
3. #define MAX 10
4.
5. int stack[MAX];
6. int top = -1;
7.
8. bool isEmpty(){
9.     return top == -1;
10. }
11.
```

```
12. void push(int data){
13.     if(top >= MAX - 1){
14.         printf("Stack Overflow |||\n");
15.     }
16.     else{
17.         stack[++top] = data;
18.         printf("PUSHED *thumbs up*\n");
19.     }
20. }
21.
22. int pop(){
23.     if(isEmpty()){
24.         printf("Stack underflow\\\\\\\\\\|\n");
25.         return -1;
26.     }
27.     else{
28.         int popped_value = stack[top--];
29.         return popped_value;
30.     }
31. }
32.
33. void display(){
34.     if(isEmpty()){
35.         printf("Rikama aahe stack\n");
36.         return;
37.     }
38.     else{
39.         printf("Printing stack from bottom\n");
40.         for(int i = top; i >= 0; i--){
41.             printf("[%d]\n", stack[i]);
42.         }
43.     }
44. }
45.
46. int main(){
47.     int choice;
48.     int value;
49.
50.     while(1){
51.         printf("1. Push\n");
52.         printf("2. Pop\n");
53.         printf("3. isEmpty ?\n");
54.         printf("4. Display\n");
55.         printf("Enter choice: ");
56.         scanf("%d", &choice);
57.         switch (choice)
58.         {
59.             case 1:
60.                 printf("Enter value to push: ");
61.                 scanf("%d", &value);
62.                 push(value);
63.                 break;
64.
65.             case 2:
66.                 value = pop();
67.                 if(value != -1){
68.                     printf("Popped value is: %d\n", value);
69.                 }
70.                 break;
71.
72.             case 3:
73.                 if(isEmpty()){
74.                     printf("EMPTY\n");
75.                 }
76.                 else{
77.                     printf("Not Empty\n");
78.                 }
79.         }
80.     }
81. }
```

```

79.         break;
80.
81.     case 4:
82.         display();
83.         break;
84.
85.     default:
86.         printf("Psch!! Disat nahi ka ?\n");
87.         break;
88.     }
89. }
90. }
91.

```

DS PRACTICAL SLIPS/slip2.1.c

```

1. #include<stdio.h>
2.
3. void display(int arr[],int n)
4. {
5.     for( int i=0 ;i<n;i++)
6.     {
7.         printf("%d\t", arr[i]);
8.     }
9.     printf("\n");
10. }
11.
12. void bubblySilk(int *arr, int n){
13.     for(int i = 0; i < n - 1; i++){
14.         for(int j = 0; j < n - i - 1; j++){
15.             if(arr[j] > arr[j + 1]){
16.                 int temp = arr[j];
17.                 arr[j] = arr[j + 1];
18.                 arr[j + 1] = arr[j];
19.             }
20.         }
21.     }
22. }
23.
24. int main()
25. {
26.     int n;
27.     printf("Enter the element ");
28.     scanf("%d",&n);
29.     int arr[n];
30.     printf("Enter %d elements: ", n);
31.     for(int i = 0; i < n; i++){
32.         scanf("%d", &arr[i]);
33.     }
34.     display(arr, n);
35.     bubblySilk(arr, n);
36.     display(arr, n);
37.
38.     return 0;
39. }
40.

```

DS PRACTICAL SLIPS/slip2.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;

```

```

6.     struct node *left;
7.     struct node *right;
8. };
9.
10. struct node *create(int data){
11.     struct node *newnode = (struct node *)malloc(sizeof(struct node));
12.     newnode->data = data;
13.     newnode->left = newnode->right = NULL;
14.
15.     return newnode;
16. }
17.
18. struct node *insert(struct node *root, int data){
19.     if(root == NULL){
20.         return create(data);
21.     }
22.
23.     if(data > root -> data){
24.         root->right = insert(root->right, data);
25.     }
26.     else if(data < root -> data){
27.         root->left = insert(root->left, data);
28.     }
29.
30.     return root;
31. }
32.
33. void inorder(struct node *root){
34.     if(root != NULL){
35.         inorder(root->left);
36.         printf("%d\t", root->data);
37.         inorder(root->right);
38.     }
39. }
40.
41. int main(){
42.     int n, c;
43.     struct node *root = NULL;
44.     printf("How many nodes: ");
45.     scanf("%d", &c);
46.
47.     for(int i = 0; i < c; i++){
48.         printf("Enter data for node %d: ", i + 1);
49.         scanf("%d", &n);
50.         root = insert(root, n);
51.     }
52.
53.     inorder(root);
54.
55.     return 0;
56. }
57.

```

DS PRACTICAL SLIPS/slip3.1.c

```

1. #include<stdio.h>
2.
3. void display(int *arr, int n){
4.     for(int i = 0; i < n; i++){
5.         printf("[%d]\t", arr[i]);
6.     }
7.     printf("\n");
8. }
9.
10. void insertionSort(int *arr, int n){

```

```

11.     int i, j, key;
12.     for(int i = 1; i < n; i++){
13.         key = arr[i];
14.         j = i - 1;
15.         while(j >= 0 && arr[j] >= key){
16.             arr[j + 1] = arr[j];
17.             j--;
18.         }
19.         arr[j + 1] = key;
20.     }
21. }
22.
23. int main(){
24.     int n;
25.     printf("Enter n: ");
26.     scanf("%d", &n);
27.     int arr[n];
28.     printf("Enter %d elements: ", n);
29.     for (int i = 0; i < n; i++)
30.     {
31.         scanf("%d", &arr[i]);
32.     }
33.
34.     printf("\n--BEFORE SORT--\n");
35.     display(arr, n);
36.     insertionSort(arr, n);
37.     printf("\n--AFTER SORT--\n");
38.     display(arr, n);
39.
40.     return 0;
41. }
42.

```

DS PRACTICAL SLIPS/slip3.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #include<ctype.h>
5. #define MAX 100
6.
7. int top = -1;
8. char stack[MAX];
9.
10. void push(char item){
11.     if(top >= MAX - 1){
12.         printf("Stack overflow\n");
13.     }
14.     stack[++top] = item;
15. }
16.
17. char pop(){
18.     if(top < 0){
19.         return '\0';
20.     }
21.     return stack[top--];
22. }
23.
24. char peek(){
25.     if(top < 0){
26.         return '\0';
27.     }
28.
29.     return stack[top];
30. }

```

```
31.
32.     int isEmpty(){
33.         return top == -1;
34.     }
35.
36.     int isOp(char ch){
37.         return (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '(' || ch == ')');
38.     }
39.
40.     int precedence(char op){
41.         if(op == '*' || op == '/')
42.             return 2;
43.         if(op == '+' || op == '-')
44.             return 1;
45.     }
46.
47.     void infixToPostfix(char infix[], char postfix[]){
48.         int i, j;
49.         char next_char;
50.         j = 0;
51.
52.         for(i = 0; infix[i] != '\0'; i++){
53.             next_char = infix[i];
54.
55.             if(isalnum(next_char)){
56.                 postfix[j++] = next_char;
57.             }
58.             else if(next_char == '('){
59.                 push(next_char);
60.             }
61.             else if(next_char == ')'){
62.                 while (!isEmpty() && peek() != '(')
63.                 {
64.                     postfix[j++] = pop();
65.                 }
66.                 if(peek() == '('){
67.                     pop();
68.                 }
69.             }
70.             else if(isOp(next_char)){
71.                 while (!isEmpty() && precedence(peek()) >= precedence(next_char)){
72.                     postfix[j++] = pop();
73.                 }
74.                 push(next_char);
75.             }
76.         }
77.
78.         while (!isEmpty())
79.         {
80.             postfix[j++] = pop();
81.         }
82.
83.
84.         postfix[j] = '\0';
85.     }
86.
87.
88.     int main(){
89.         char infix[MAX];
90.         char postfix[MAX];
91.
92.         printf("Enter Infix expression: ");
93.         if (fgets(infix, MAX, stdin) == NULL) {
94.             fprintf(stderr, "Error reading input.\n");
95.             return 1;
96.         }
97.     }
```

```

98. // Remove the newline character added by fgets(), if present
99. infix[strcspn(infix, "\n")] = 0;
100.
101. // Convert and display the result
102. infixToPostfix(infix, postfix);
103.
104.     infixToPostfix(infix, postfix);
105.     printf("Infix Equation: %s\n", infix);
106.     printf("Postfix Equation: %s\n\n", postfix);
107.
108.     return 0;
109. }
110.

```

DS PRACTICAL SLIPS/slip4.1.c

```

1. #include<stdio.h>
2.
3. void linearSearch(int *arr, int n){
4.     int key;
5.     int pos;
6.     int flag = 0;
7.     printf("Enter element to search: ");
8.     scanf("%d", &key);
9.
10.    for(int i = 0; i < n; i++){
11.        if(key == arr[i]){
12.            pos = i;
13.            flag = 1;
14.            break;
15.        }
16.    }
17.
18.    if(flag == 1){
19.        printf("%d basho de sore o mitsukematisha: [%d]\n", key, pos + 1);
20.    }
21.    else{
22.        printf("%d hairetsu-nai ni arimasen\n");
23.    }
24.
25. }
26.
27. int main(){
28.     int n;
29.     printf("Enter n: ");
30.     scanf("%d", &n);
31.
32.     int arr[n];
33.
34.     printf("Enter %d elements: ", n);
35.
36.     for(int i = 0; i < n; i++){
37.         scanf("%d", &arr[i]);
38.     }
39.
40.     linearSearch(arr, n);
41.
42.     return 0;
43. }
44.

```

DS PRACTICAL SLIPS/slip4.2.c

```

1. #include<stdio.h>

```

```
2. #include<stdlib.h>
3.
4. struct node
5. {
6.     int data;
7.     struct node *next;
8. };
9.
10.
11. struct node *create(struct node *head){
12.     int n, data;
13.     struct node *temp = head;
14.
15.     printf("How many nodes ");
16.     scanf("%d", &n);
17.
18.     for(int i = 0; i < n; i++){
19.         struct node *newnode = (struct node *)malloc(sizeof(struct node));
20.         printf("Enter data for node [%d]: ", i + 1);
21.         scanf("%d", &data);
22.
23.         newnode -> data = data;
24.         newnode -> next = NULL;
25.
26.         if(head == NULL){
27.             head = newnode;
28.             temp = newnode;
29.         }
30.         else{
31.             temp->next = newnode;
32.             temp = newnode;
33.             newnode->next = head;
34.         }
35.     }
36.     printf("Creation successful\n");
37.
38.     return head;
39. }
40.
41. struct node *insert(struct node *head){
42.     struct node *newnode = NULL;
43.     struct node *temp = head;
44.     newnode = (struct node *)malloc(sizeof(struct node));
45.     newnode -> next = NULL;
46.     printf("Enter data: ");
47.     scanf("%d", &newnode->data);
48.
49.     if(head == NULL){
50.         head = newnode;
51.         newnode->next = head;
52.     }
53.     else{
54.         while(temp -> next != head){
55.             temp = temp -> next;
56.         }
57.         newnode -> next = head;
58.         temp -> next = newnode;
59.         head = newnode;
60.     }
61.     printf("Insertion at the front successfull\n");
62.     return head;
63. }
64.
65. struct node *delete(struct node *head){
66.     struct node *temp = head;
67.     struct node *temp1 = head;
68.
```

```
69.     if(head->next == head){
70.         free(head);
71.         return NULL;
72.     }
73.     else{
74.         while(temp->next != head){
75.             temp = temp -> next;
76.         }
77.         head = head -> next;
78.         temp1->next = NULL;
79.         temp->next = head;
80.         free(temp1);
81.     }
82.     printf("Deletion successful\n");
83.
84.     return head;
85. }
86.
87. void display(struct node *head){
88.     struct node *temp = head;
89.     int i = 1;
90.     do{
91.         printf("Node %d -> Data %d\n", i, temp->data);
92.         temp = temp -> next;
93.         i++;
94.     }while(temp != head);
95. }
96.
97. int main(){
98.     int Case;
99.     struct node *head = NULL;
100.
101.    do{
102.        printf("1. Create List\n");
103.        printf("2. Insert a node\n");
104.        printf("3. Delete a node\n");
105.        printf("4. Display list\n");
106.        printf("5. Exit\n");
107.        printf("Select choice: ");
108.        scanf("%d", &Case);
109.
110.       switch (Case)
111.       {
112.           case 1:
113.               head = create(head);
114.               break;
115.
116.           case 2:
117.               head = insert(head);
118.               break;
119.
120.           case 3:
121.               head = delete(head);
122.               break;
123.
124.           case 4:
125.               display(head);
126.               break;
127.
128.           case 5:
129.               printf("Jaa rhe ho na chhad kar :(\nSab aesa hi karte hai :(\\nGood Bye :(\\n");
130.               break;
131.
132.           default:
133.               printf("Watashinonamaeha kirayoshikage. 33-Sai\\n");
134.               break;
135.     }
```

```
136.     }while(Case != 5);
137.
138.     return 0;
139. }
140.
141.
```

DS PRACTICAL SLIPS/slip5.1.c

```
1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *next;
7. };
8.
9. struct node *create(struct node *head){
10.    int n, data;
11.    struct node *temp = head;
12.    printf("How many node: ");
13.    scanf("%d", &n);
14.
15.    for(int i = 0; i < n; i++){
16.        struct node *newnode = (struct node *)malloc(sizeof(struct node));
17.        printf("Enter data for node [%d]: ", i + 1);
18.        scanf("%d", &data);
19.        newnode -> data = data;
20.        newnode -> next = NULL;
21.
22.        if(head == NULL){
23.            head = newnode;
24.            temp = newnode;
25.        }
26.        else{
27.            temp -> next = newnode;
28.            temp = newnode;
29.        }
30.    }
31.    return head;
32. }
33.
34. void display(struct node *head){
35.    int i = 1;
36.    struct node *tr = head;
37.    while(tr != NULL){
38.        printf("Node %d -> Data: %d\n", i, tr->data);
39.        tr = tr -> next;
40.        i++;
41.    }
42. }
43.
44. int main(){
45.    struct node *head = NULL;
46.    head = create(head);
47.    display(head);
48.    return 0;
49. }
50.
```

DS PRACTICAL SLIPS/slip5.2.c

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

```
3.
4. struct tree{
5.     int data;
6.     struct tree *left;
7.     struct tree *right;
8. };
9.
10. struct tree *create(int data){
11.     struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));
12.     newnode->data = data;
13.     newnode->left = newnode->right = NULL;
14.
15.     return newnode;
16. }
17.
18. struct tree *insert(struct tree *root, int data){
19.     if(root == NULL){
20.         return create(data);
21.     }
22.
23.     if(data < root -> data){
24.         root->left = insert(root->left, data);
25.     }
26.     else if(data > root -> data){
27.         root->right = insert(root->right, data);
28.     }
29.
30.     return root;
31. }
32.
33. void inorder(struct tree *root){
34.     if(root != NULL){
35.         inorder(root->left);
36.         printf("%d ", root->data);
37.         inorder(root->right);
38.     }
39. }
40.
41. void preorder(struct tree *root){
42.     if(root != NULL){
43.         printf("%d ", root->data);
44.         preorder(root->left);
45.         preorder(root->right);
46.     }
47. }
48.
49.
50. int main(){
51.     struct tree *root = NULL;
52.     int c, n;
53.     printf("How many nodes: ");
54.     scanf("%d", &c);
55.
56.     for(int i = 0; i < c; i++){
57.         printf("Enter element %d: ", i + 1);
58.         scanf("%d", &n);
59.         root = insert(root, n);
60.     }
61.
62.
63.     printf("Inorder: \n");
64.     inorder(root);
65.     printf("\n");
66.
67.     printf("Preorder: \n");
68.     preorder(root);
69.     printf("\n");
```

```
70.  
71.     return 0;  
72.  
73. }  
74.
```

DS PRACTICAL SLIPS/slip6.1.c

```
1. #include<stdio.h>  
2. #include<stdlib.h>  
3. #include<string.h>  
4. #define MAX 100  
5.  
6. struct stack{  
7.     int top;  
8.     char arr[MAX];  
9. };  
10.  
11. void init(struct stack *s){  
12.     s->top = -1;  
13. }  
14.  
15. int isFull(struct stack *s){  
16.     return s->top == MAX - 1;  
17. }  
18.  
19. int isEmpty(struct stack *s){  
20.     return s->top == -1;  
21. }  
22.  
23. void push(struct stack *s, char ch){  
24.     if(!isFull(s)){  
25.         s->arr[++(s->top)] = ch;  
26.     }else{  
27.         printf("Pani ki tanki bhar gayi hai, kripaya motor band karde\n");  
28.         exit(1);  
29.     }  
30. }  
31.  
32. char pop(struct stack *s){  
33.     if(!isEmpty(s)){  
34.         return s->arr[(s->top)--];  
35.     }else{  
36.         printf("Khali hai, khali hai\n");  
37.         exit(1);  
38.     }  
39. }  
40.  
41. void strrevers(char str[]){  
42.     struct stack s;  
43.     init(&s);  
44.  
45.     int n = strlen(str);  
46.  
47.     for(int i = 0; i < n; i++){  
48.         push(&s, str[i]);  
49.     }  
50.  
51.     for(int i = 0; i < n; i++){  
52.         str[i] = pop(&s);  
53.     }  
54. }  
55.  
56. int main(){  
57.     char str[MAX];
```

```

58.
59.     printf("Enter string: ");
60.     gets(str);
61.
62.     strrevs(str);
63.
64.     printf("Reversed string: \"%s\"\n", str);
65.
66.     return 0;
67. }
68.

```

DS PRACTICAL SLIPS/slip6.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct tree{
5.     int data;
6.     struct tree *left;
7.     struct tree *right;
8. };
9.
10. struct tree *create(int data){
11.     struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));
12.     newnode->data = data;
13.     newnode->left = newnode->right = NULL;
14.
15.     return newnode;
16. }
17.
18. struct tree *insert(struct tree *root, int data){
19.     if(root == NULL){
20.         return create(data);
21.     }
22.
23.     if(data < root->data){
24.         root->left = insert(root->left, data);
25.     }
26.     else if(data > root->data){
27.         root->right = insert(root->right, data);
28.     }
29.
30.     return root;
31. }
32.
33. int countNonLeaf(struct tree *root){
34.     if(root == NULL){
35.         printf("Tree is cutted down to logs\nBurned at fireplaces\nThe Sacrifices will be
remembered by the Generation\n");
36.         return 0;
37.     }
38.     if(root->left == NULL && root->right == NULL){
39.         return 0;
40.     }
41.     else{
42.         return (1 + countNonLeaf(root->left) + countNonLeaf(root->right));
43.     }
44. }
45.
46. int countLeaf(struct tree *root){
47.     if(root == NULL){
48.         return 0;
49.     }
50.     else if((root->left == NULL) && (root->right == NULL)){

```

```

51.         return 1;
52.     }
53.     else{
54.         return(countLeaf(root->left) + countLeaf(root->right));
55.     }
56. }
57.
58. int main(){
59.     struct tree *root = NULL;
60.     int ch;
61.     int c, n;
62.     int gg;
63.     int j;
64.     int k;
65.     printf("How many nodes: ");
66.     scanf("%d", &c);
67.
68.     for(int i = 0; i < c; i++){
69.         printf("Enter element for node %d: ", i + 1);
70.         scanf("%d", &n);
71.         root = insert(root, n);
72.     }
73.
74.     do{
75.         printf("1. Insert new element in tree\n");
76.         printf("2. Count total non-leaf nodes\n");
77.         printf("3. count total leaf nodes\n");
78.         printf("0. Exit\n");
79.         printf("Select choice: ");
80.         scanf("%d", &ch);
81.
82.         switch (ch)
83.         {
84.             case 1:
85.                 printf("Enter element: ");
86.                 scanf("%d", &gg);
87.                 root = insert(root, gg);
88.                 break;
89.
90.             case 2:
91.                 j = countNonLeaf(root);
92.                 printf("Total non-leaf node: %d\n", j);
93.                 break;
94.
95.             case 3:
96.                 k = countLeaf(root);
97.                 printf("Total leaf node: %d\n", k);
98.                 break;
99.
100.            case 0:
101.                printf("Kaisi teri khud garzi naa dhoop chune na chava\n");
102.                break;
103.
104.            default:
105.                printf("Re kabiraaaa.... maaan jaa.... thik se dekh k button daba.....\n");
106.                break;
107.         }
108.
109.     }while(ch != 0);
110.
111.     return 0;
112. }
113.

```

```
1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *next;
7. };
8.
9. struct node *ins(struct node *head){
10.    struct node *newnode = (struct node *)malloc(sizeof(struct node));
11.    newnode->next = NULL;
12.
13.    printf("Enter data: ");
14.    scanf("%d", &newnode->data);
15.
16.    if(head == NULL){
17.        newnode->next = NULL;
18.        head = newnode;
19.    }
20.    else{
21.        newnode->next = head;
22.        head = newnode;
23.    }
24.    printf("Insertion successful\n");
25.    return head;
26. }
27.
28. void disp(struct node *head){
29.     struct node *tr = head;
30.     int i = 1;
31.     while(tr!=NULL){
32.         printf("Node: %d -> Data: %d\n", i, tr->data);
33.         tr = tr -> next;
34.         i++;
35.     }
36. }
37.
38. int lent(struct node *head){
39.     struct node *tr = head;
40.     int lens = 1;
41.     while(tr != NULL){
42.         lens++;
43.         tr = tr -> next;
44.     }
45.
46.     return lens;
47. }
48.
49. int main(){
50.     struct node *head = NULL;
51.     int ch;
52.     int len;
53.     do{
54.         printf("1. Create/Insert node\n");
55.         printf("2. Display list\n");
56.         printf("3. Length of list\n");
57.         printf("0. Muze kaise bhi karke nikal mere bhai me fass gaya hun yaha...\n");
58.         printf("Enter choice: ");
59.         scanf("%d", &ch);
60.
61.         switch(ch){
62.             case 1:
63.                 head = ins(head);
64.                 break;
65.
66.             case 2:
67.                 disp(head);
```

```

68.         break;
69.
70.     case 3:
71.         len = lent(head);
72.         printf("Length of list: [%d]\n", len);
73.         break;
74.
75.     case 0:
76.         printf("Sayonara\n");
77.         break;
78.
79.     default:
80.         printf("Bhaisahab ye kis line me aa gaye aap? \n");
81.         break;
82.     }
83.
84. }while(ch!=0);
85.
86. return 0;
87. }
88.

```

DS PRACTICAL SLIPS/slip7.2.c

```

1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. struct Stack {
5.     int *arr;
6.     int top;
7.     int capacity;
8. };
9.
10. struct Stack* createStack(int capacity) {
11.     struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
12.     stack->capacity = capacity;
13.     stack->top = -1;
14.     stack->arr = (int *)malloc(stack->capacity * sizeof(int));
15.     return stack;
16. }
17.
18. int isEmpty(struct Stack *stack) {
19.     return stack->top == -1;
20. }
21.
22. int isFull(struct Stack *stack) {
23.     return stack->top == stack->capacity - 1;
24. }
25.
26. void push(struct Stack *stack, int data) {
27.     if (isFull(stack)) {
28.         printf("Stack overflow! Cannot push %d\n", data);
29.         return;
30.     }
31.     stack->top++;
32.     stack->arr[stack->top] = data;
33.     printf("%d pushed to stack\n", data);
34. }
35.
36. int pop(struct Stack *stack) {
37.     if (isEmpty(stack)) {
38.         printf("Stack underflow! Nothing to pop\n");
39.         return -1;
40.     }
41.     int popped = stack->arr[stack->top];

```

```
42.     stack->top--;
43.     return popped;
44. }
45.
46. void display(struct Stack *stack) {
47.     if (isEmpty(stack)) {
48.         printf("Stack is empty\n");
49.         return;
50.     }
51.     printf("Stack elements (top -> bottom): ");
52.     for (int i = stack->top; i >= 0; i--) {
53.         printf("%d ", stack->arr[i]);
54.     }
55.     printf("\n");
56. }
57.
58. int main() {
59.     int capacity;
60.     printf("Enter stack capacity: ");
61.     scanf("%d", &capacity);
62.
63.     struct Stack *stack = createStack(capacity);
64.
65.     int choice, value;
66.
67.     do {
68.         printf("\n--- Stack Menu ---\n");
69.         printf("1. Push\n");
70.         printf("2. Pop\n");
71.         printf("3. Display\n");
72.         printf("4. IsEmpty\n");
73.         printf("5. IsFull\n");
74.         printf("0. Exit\n");
75.         printf("Enter your choice: ");
76.         scanf("%d", &choice);
77.
78.         switch(choice) {
79.             case 1:
80.                 printf("Enter value to push: ");
81.                 scanf("%d", &value);
82.                 push(stack, value);
83.                 break;
84.             case 2:
85.                 value = pop(stack);
86.                 if (value != -1)
87.                     printf("Popped element: %d\n", value);
88.                 break;
89.             case 3:
90.                 display(stack);
91.                 break;
92.             case 4:
93.                 if (isEmpty(stack))
94.                     printf("Stack is empty\n");
95.                 else
96.                     printf("Stack is not empty\n");
97.                 break;
98.             case 5:
99.                 if (isFull(stack))
100.                     printf("Stack is full\n");
101.                 else
102.                     printf("Stack is not full\n");
103.                 break;
104.             case 0:
105.                 printf("Jaa raha hun mein....\n");
106.                 break;
107.             default:
108.                 printf("Chalo galti insaan se hi hoti hai...\n");
```

```
109.     }
110. } while(choice != 0);
112.
113. return 0;
114. }
115.
```

```
1. DS PRACTICAL SLIPS/slip8.1.c
2. #include<stdio.h>
3. #include<stdlib.h>
4.
5. struct node{
6.     int data;
7.     struct node *next;
8.     struct node *prev;
9. };
10.
11. struct node *create(struct node *head){
12.     int data;
13.     int val;
14.     struct node *temp = head;
15.
16.     printf("How many nodes: ");
17.     scanf("%d", &val);
18.
19.     for(int i = 0; i < val; i++){
20.         struct node *newnode = (struct node *)malloc(sizeof(struct node));
21.         printf("Enter data for node %d: ", i + 1);
22.         scanf("%d", &data);
23.
24.         newnode->data = data;
25.         newnode->prev = newnode->next = NULL;
26.
27.         if(head == NULL){
28.             head = newnode;
29.             temp = newnode;
30.         }
31.         else{
32.             temp->next = newnode;
33.             newnode->prev = temp;
34.             temp = newnode;
35.         }
36.     }
37.     printf("Creation successfull\n");
38.
39.     return head;
40. }
41.
42. struct node *display(struct node *head){
43.     struct node *tr = head;
44.     int i = 1;
45.     while(tr != NULL){
46.         printf("Node %d -> data %d\n", i, tr->data);
47.         tr = tr ->next;
48.         i++;
49.     }
50. }
51.
52. int main(){
53.     struct node *head = NULL;
54.
55.     head = create(head);
```

```
56.     display(head);
57.
58.     return 0;
59. }
60.
```

DS PRACTICAL SLIPS/slip8.2.c

```
1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *next;
7. };
8.
9. struct node *create(struct node *head){
10.    int data;
11.    int val;
12.    struct node *temp = head;
13.    printf("How many nodes: ");
14.    scanf("%d", &val);
15.    for(int i = 0; i < val; i++){
16.        struct node *newnode = (struct node *)malloc(sizeof(struct node));
17.        printf("Insert data for node [%d]: ", i + 1);
18.        scanf("%d", &data);
19.
20.        newnode->data = data;
21.        newnode->next = NULL;
22.
23.        if(head == NULL){
24.            head = newnode;
25.            temp = newnode;
26.        }
27.        else{
28.            temp->next = newnode;
29.            temp = newnode;
30.        }
31.    }
32.    return head;
33. }
34.
35. struct node *concat(struct node *head1, struct node *head2){
36.    struct node *temp = head1;
37.    while(temp->next != NULL){
38.        temp = temp -> next;
39.    }
40.    temp -> next = head2;
41.
42.    return head1;
43. }
44.
45. void display(struct node *head){
46.    int i = 1;
47.    struct node *tr = head;
48.    while (tr != NULL)
49.    {
50.        printf("Node %d -> Data: [%d]\n", i, tr->data);
51.        tr = tr -> next;
52.        i++;
53.    }
54. }
55.
56. int main(){
57.     struct node *head1 = NULL;
```

```

58.     struct node *head2 = NULL;
59.
60.     printf("Initializing process for list 1\n");
61.     head1 = create(head1);
62.     printf("Displaying list 1: \n");
63.     display(head1);
64.
65.     printf("Initializing process for list 2\n");
66.     head2 = create(head2);
67.     printf("Displaying list 2: \n");
68.     display(head2);
69.
70.     printf("Concatinating both lists....\n");
71.     printf("Taal se Taal milaaa\n");
72.     concat(head1, head2);
73.     printf("Concatination successful\n");
74.     printf("Displaying concatenated list\n");
75.     display(head1);
76.
77.     return 0;
78.
79. }
80.

```

DS PRACTICAL SLIPS/slip9.1.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *next;
7. };
8.
9. struct node *create(struct node *head){
10.    int data;
11.    int val;
12.    struct node *temp = head;
13.    printf("How many nodes: ");
14.    scanf("%d", &val);
15.    for(int i = 0; i < val; i++){
16.        struct node *newnode = (struct node *)malloc(sizeof(struct node));
17.        printf("Insert data for node [%d]: ", i + 1);
18.        scanf("%d", &data);
19.
20.        newnode->data = data;
21.        newnode->next = NULL;
22.
23.        if(head == NULL){
24.            head = newnode;
25.            temp = newnode;
26.        }
27.        else{
28.            temp->next = newnode;
29.            temp = newnode;
30.        }
31.    }
32.    return head;
33. }
34.
35. void display(struct node *head){
36.    int i = 1;
37.    struct node *tr = head;
38.    while (tr != NULL)
39.    {

```

```

40.         printf("Node %d -> Data: [%d]\n", i, tr->data);
41.         tr = tr -> next;
42.         i++;
43.     }
44. }
45.
46. struct node *noderev(struct node *head){
47.     struct node *prev = NULL;
48.     struct node *curr = head;
49.     struct node *next = NULL;
50.
51.     while(curr != NULL){
52.         next = curr -> next;
53.         curr->next = prev;
54.         prev = curr;
55.         curr = next;
56.     }
57.
58.     return prev;
59. }
60.
61. int main(){
62.     struct node *head = NULL;
63.     head = create(head);
64.     display(head);
65.
66.     printf("In Reverse order: \n");
67.     head = noderev(head);
68.     display(head);
69.
70.     return 0;
71. }
72.

```

DS PRACTICAL SLIPS/slip9.2.c

```

1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. struct node
5. {
6.     int data;
7.     struct node *next;
8. };
9.
10. struct node *create(struct node *head)
11. {
12.     int data;
13.     int val;
14.     struct node *temp = head;
15.     printf("How many nodes: ");
16.     scanf("%d", &val);
17.     for (int i = 0; i < val; i++)
18.     {
19.         struct node *newnode = (struct node *)malloc(sizeof(struct node));
20.         printf("Insert data for node [%d]: ", i + 1);
21.         scanf("%d", &data);
22.
23.         newnode->data = data;
24.         newnode->next = NULL;
25.
26.         if (head == NULL)
27.         {
28.             head = newnode;
29.             temp = newnode;

```

```
30.     }
31.     else
32.     {
33.         temp->next = newnode;
34.         temp = newnode;
35.     }
36. }
37. return head;
38. }
39.
40. void display(struct node *head)
41. {
42.     int i = 1;
43.     struct node *tr = head;
44.     while (tr != NULL)
45.     {
46.         printf("Node %d -> Data: [%d]\n", i, tr->data);
47.         tr = tr->next;
48.         i++;
49.     }
50. }
51.
52. void nodeser(struct node *head)
53. {
54.     struct node *temp = head;
55.     int n;
56.     int flag = 0;
57.     int pos = 1;
58.     printf("Enter element to search: ");
59.     scanf("%d", &n);
60.
61.     while (temp != NULL)
62.     {
63.         if (temp->data == n)
64.         {
65.             flag = 1;
66.         }
67.         temp = temp->next;
68.         pos++;
69.     }
70.     if (flag == 1)
71.     {
72.         printf("%d is in %d node\n", n, pos);
73.     }
74.     else
75.     {
76.         printf("%d is not in list\n");
77.     }
78. }
79.
80. struct node* deleteNode(struct node *head) {
81.     int pos;
82.     struct node *temp = head;
83.     printf("Enter node number to delete: ");
84.     scanf("%d", &pos);
85.
86.     if (pos == 1) {
87.         head = head->next;
88.         free(temp);
89.         printf("Node %d deleted.\n", pos);
90.         return head;
91.     }
92.
93.     struct node *prev = NULL;
94.     int i = 1;
95.     while (temp != NULL && i < pos) {
96.         prev = temp;
```

```

97.         temp = temp->next;
98.         i++;
99.     }
100.
101.    prev->next = temp->next;
102.    free(temp);
103.    printf("Node %d deleted.\n", pos);
104.
105.    return head;
106. }
107.
108.
109. int main()
110. {
111.     struct node *head = NULL;
112.     int sel;
113.
114.     while (1)
115.     {
116.         printf("\nMenu:\n");
117.         printf("1. Create Linked List\n");
118.         printf("2. Display Linked List\n");
119.         printf("3. Search Node\n");
120.         printf("4. Delete from specific pos\n");
121.         printf("5. Exit\n");
122.         printf("Enter your choice: ");
123.         scanf("%d", &sel);
124.
125.         switch (sel)
126.         {
127.             case 1:
128.                 head = create(head);
129.                 break;
130.             case 2:
131.                 display(head);
132.                 break;
133.             case 3:
134.                 nodeser(head);
135.                 break;
136.             case 4:
137.                 head = deleteNode(head);
138.                 break;
139.
140.             case 5:
141.                 exit(0);
142.                 break;
143.
144.             default:
145.                 printf("Galat baat!\n");
146.         }
147.     }
148.
149.     return 0;
150. }
151.

```

DS PRACTICAL SLIPS/slip10.1.c

```

1. #include<stdio.h>
2.
3. int main(){
4.     int n;
5.     printf("Enter number of nodes: ");
6.     scanf("%d", &n);
7.

```

```
8.     int graph[n][n];
9.     printf("Enter adjacency matrix: \n");
10.    for(int i = 0; i < n; i++){
11.        for(int j = 0; j < n; j++){
12.            scanf("%d", &graph[i][j]);
13.        }
14.    }
15.
16.    for(int i = 0; i < n; i++){
17.        int indeg = 0; int outdeg = 0;
18.        for(int j = 0; j < n; j++){
19.            outdeg = outdeg + graph[i][j];
20.            indeg = indeg + graph[j][i];
21.        }
22.        printf("Node %d -> Indegree: [%d] || Outdegree: [%d]\n", i, indeg, outdeg);
23.    }
24.    return 0;
25. }
26.
27. DS PRACTICAL SLIPS/slip10.2.c
28. #include<stdio.h>
29. #include<stdlib.h>
30.
31. struct node{
32.     int data;
33.     struct node *left;
34.     struct node *right;
35. };
36.
37. struct node *queue[100];
38. int fr = -1, rear = -1;
39.
40. struct node *create(int data){
41.     struct node *newnode = (struct node *)malloc(sizeof(struct node));
42.     newnode->data = data;
43.     newnode->left = newnode->right = NULL;
44.
45.     return newnode;
46. }
47.
48. struct node *insert(struct node *root, int data){
49.     if(root == NULL){
50.         return create(data);
51.     }
52.
53.     if(data < root->data){
54.         root->left = insert(root->left, data);
55.     }
56.     else if(data > root->data){
57.         root->right = insert(root->right, data);
58.     }
59.
60.     return root;
61. }
62.
63. void enqueue(struct node *nodule){
64.     if(nodule != NULL){
65.         if(rear == 99) return;
66.         queue[++rear] = nodule;
67.     }
68. }
69.
70. struct node *dequeue(){
71.     if(fr == rear){
72.         return NULL;
73.     }
74.     return queue[++fr];
```

```

75. }
76.
77. void levelOrder(struct node *root){
78.     if(root == NULL){
79.         return;
80.     }
81.     enqueue(root);
82.     while(front != rear){
83.         struct node *curr = dequeue();
84.         printf("[%d]\t", curr->data);
85.         enqueue(curr->left);
86.         enqueue(curr->right);
87.     }
88. }
89.
90. int main(){
91.     int n, c;
92.     struct node *root = NULL;
93.     printf("How many nodes: ");
94.     scanf("%d", &c);
95.     for(int i = 0; i < c; i++){
96.         printf("Enter data for node %d: ", i + 1);
97.         scanf("%d", &n);
98.         root = insert(root, n);
99.     }
100.    printf("Displaying tree: \n");
101.    levelOrder(root);
102.
103.    return 0;
104. }
105.

```

DS PRACTICAL SLIPS/slip11.1.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<time.h>
4.
5. void disp(int *arr, int n){
6.     for(int i = 0; i < n; i++){
7.         printf("[%d]\t", arr[i]);
8.     }
9.     printf("\n");
10. }
11.
12. void swap(int *a, int *b){
13.     int t = *a;
14.     *a = *b;
15.     *b = t;
16. }
17.
18. int party(int *arr, int low, int high){
19.     int piv = arr[low];
20.     int i = low + 1;
21.     int j = high;
22.
23.     while(1){
24.         while(i <= high && arr[i] <= piv){
25.             i++;
26.         }
27.
28.         while (arr[j] > piv){
29.             j--;
30.         }
31.

```

```

32.         if(i < j){
33.             swap(&arr[i], &arr[j]);
34.         }else{
35.             break;
36.         }
37.     }
38. }
39.
40. swap(&arr[low], &arr[j]);
41. return j;
42.
43. }
44.
45. void quick(int *arr, int low, int high){
46.     if(low < high){
47.         int pi = party(arr, low, high);
48.         quick(arr, low, pi - 1);
49.         quick(arr, pi + 1, high);
50.     }
51. }
52.
53. int main(){
54.     int n;
55.     printf("Enter n: ");
56.     scanf("%d", &n);
57.     int arr[n];
58.     srand(time(NULL));
59.     printf("Entering %d elements: \n", n);
60.     for(int i = 0; i < n; i++){
61.         arr[i] = rand();
62.     }
63.     printf("Insertion successful\n");
64.     printf("Displaying: \n");
65.     disp(arr, n);
66.     quick(arr, 0, n - 1);
67.     printf("After sort: \n");
68.     disp(arr, n);
69.
70.     return 0;
71. }
72.

```

DS PRACTICAL SLIPS/slip11.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct tree{
5.     int data;
6.     struct tree *left;
7.     struct tree *right;
8. };
9.
10. struct tree *create(int data){
11.     struct tree *newnode = (struct tree *)malloc(sizeof(struct tree));
12.
13.     newnode->data = data;
14.     newnode->left = newnode->right = NULL;
15.
16.     return newnode;
17. }
18.
19. struct tree *insert(struct tree *root, int data){

```

```
20.     if(root == NULL){
21.         return create(data);
22.     }
23.
24.     if(data < root -> data){
25.         root -> left = insert(root->left, data);
26.     }
27.
28.     else if(data > root -> data){
29.         root -> right = insert(root->right, data);
30.     }
31.
32.     return root;
33. }
34.
35. int leafNode(struct tree *root){
36.     if(root == NULL){
37.         return 0;
38.     }
39.     else if(root->left == NULL && root->right == NULL){
40.         return 1;
41.     }
42.     else{
43.         return (leafNode(root->left) + leafNode(root->right));
44.     }
45. }
46.
47. int nonLeafNode(struct tree *root){
48.     if(root == NULL){
49.         return 0;
50.     }
51.     else if(root->left == NULL && root->right == NULL){
52.         return 0 ;
53.     }
54.     else{
55.         return (1 + nonLeafNode(root->left) + nonLeafNode(root->right));
56.     }
57. }
58.
59. void inorder(struct tree *root){
60.     if(root != NULL){
61.         inorder(root->left);
62.         printf("[%d] ", root->data);
63.         inorder(root->right);
64.     }
65.
66. }
67.
68. int main(){
69.     int n, c;
70.     struct tree *root = NULL;
71.     printf("Enter n: ");
72.     scanf("%d", &c);
73.
74.     for(int i = 0; i < c; i++){
75.         printf("Enter data for node %d: ", i + 1);
76.         scanf("%d", &n);
77.         root = insert(root, n);
78.     }
79.
80.     printf("Displaying tree: \n");
81.     inorder(root);
82.
83.     int i = leafNode(root);
84.     int j = nonLeafNode(root);
85.
86.     printf("\nTotal leaf node: [%d]\n", i);
```

```

87.     printf("\nTotal non leaf node: [%d]\n", j);
88.
89.     return 0;
90. }
91.
92. DS PRACTICAL SLIPS/slip12.1.c
93. #include <stdio.h>
94.
95. int main() {
96.     int deg1, deg2, maxDeg;
97.
98.     // Input first polynomial
99.     printf("Enter degree of first polynomial: ");
100.    scanf("%d", &deg1);
101.    int poly1[deg1 + 1];
102.    printf("Enter coefficients of first polynomial (from highest degree to constant):\n");
103.    for (int i = deg1; i >= 0; i--) {
104.        printf("Coefficient of x^%d: ", i);
105.        scanf("%d", &poly1[i]);
106.    }
107.
108.    // Input second polynomial
109.    printf("\nEnter degree of second polynomial: ");
110.    scanf("%d", &deg2);
111.    int poly2[deg2 + 1];
112.    printf("Enter coefficients of second polynomial (from highest degree to constant):\n");
113.    for (int i = deg2; i >= 0; i--) {
114.        printf("Coefficient of x^%d: ", i);
115.        scanf("%d", &poly2[i]);
116.    }
117.
118.    // Determine maximum degree
119.    maxDeg = (deg1 > deg2) ? deg1 : deg2;
120.
121.    // Arrays to hold result
122.    int result[maxDeg + 1];
123.
124.    // Initialize result
125.    for (int i = 0; i <= maxDeg; i++) {
126.        int a = (i <= deg1) ? poly1[i] : 0;
127.        int b = (i <= deg2) ? poly2[i] : 0;
128.        result[i] = a - b;
129.    }
130.
131.    // Display result polynomial
132.    printf("\nResult (P1 - P2) = ");
133.    for (int i = maxDeg; i >= 0; i--) {
134.        if (result[i] != 0) {
135.            if (i != maxDeg && result[i] > 0) printf("+"); // add + for positive terms
136.            if (i == 0) printf("%d ", result[i]); // constant
137.            else if (i == 1) printf("%dx ", result[i]); // x term
138.            else printf("%dx^%d ", result[i], i); // x^i term
139.        }
140.    }
141.    printf("\n");
142.
143.    return 0;
144. }
145.

```

DS PRACTICAL SLIPS/slip12.2.c

```

1. #include<stdio.h>
2. #define Size 10
3.

```

```
4. int queue[Size];
5. int fr = -1, rr = -1;
6.
7. void insert(int value){
8.     if(fr == -1){
9.         fr = rr = 0;
10.    }else{
11.        rr = (rr + 1) % Size;
12.    }
13.
14.    queue[rr] = value;
15.    printf("INSERTION SUCESSFULL\n");
16. }
17.
18. void delete(){
19.     int deleted = queue[fr];
20.     if(fr == rr){
21.         fr = rr = -1;
22.     }else{
23.         fr = (fr + 1) % Size;
24.     }
25.     printf("DELETION SUCCESSFULL\n");
26. }
27.
28. void disp(){
29.     int i = fr;
30.     while(1){
31.         printf("%d ", queue[i]);
32.         if(i == rr)
33.             break;
34.
35.         i = (i + 1) % Size;
36.     }
37.     printf("\n");
38. }
39.
40. int main(){
41.     int val, ch;
42.
43.     while(1){
44.         printf("1. Insert element\n");
45.         printf("2. Delete element\n");
46.         printf("3. Display Queue\n");
47.         printf("4. Exit\n");
48.         printf("Your choice: ");
49.         scanf("%d", &ch);
50.
51.         switch(ch){
52.             case 1:
53.                 printf("Enter value to insert: ");
54.                 scanf("%d", &val);
55.                 insert(val);
56.                 break;
57.
58.             case 2:
59.                 delete();
60.                 break;
61.
62.             case 3:
63.                 disp();
64.                 break;
65.
66.             case 4:
67.                 return 0;
68.
69.             default:
70.                 printf("Galat jawab, ye aapke bank mese 7 crores mere bank mein\n");
```

```
71.         break;
72.     }
73. }
74.
75. return 0;
76. }
77.
```

DS PRACTICAL SLIPS/slip13.1.c

```
1. #include<stdio.h>
2.
3. void disp(int *arr, int n){
4.     for(int i = 0; i < n; i++){
5.         printf("%d\t", arr[i]);
6.     }
7.     printf("\n");
8. }
9.
10. void mergerSort(int *arr, int low, int mid, int high){
11.     int copArr[100];
12.     int i = low;
13.     int j = mid + 1;
14.     int k = low;
15.
16.     while(i <= mid && j <= high){
17.         if(arr[i] < arr[j]){
18.             copArr[k] = arr[i];
19.             i++;
20.         }
21.         else{
22.             copArr[k] = arr[j];
23.             j++;
24.         }
25.         k++;
26.     }
27.     while(i <= mid){
28.         copArr[k] = arr[i];
29.         k++;
30.         i++;
31.     }
32.     while(j <= high){
33.         copArr[k] = arr[j];
34.         k++;
35.         j++;
36.     }
37.     for(int l = 0; l <= high; l++){
38.         arr[l] = copArr[l];
39.     }
40. }
41.
42. void merger(int *arr, int low, int high){
43.     int mid;
44.     if(low < high){
45.         mid = (low + high) / 2;
46.         merger(arr, low, mid);
47.         merger(arr, mid + 1, high);
48.         mergerSort(arr, low, mid, high);
49.     }
50. }
51.
52. int main(){
53.     int n;
54.     printf("Enter n: ");
55.     scanf("%d", &n);
```

```

56.     int arr[n];
57.     printf("Enter %d elements: ", n);
58.
59.     for(int i = 0; i < n; i++){
60.         scanf("%d", &arr[i]);
61.     }
62.
63.     disp(arr, n);
64.     printf("After Sorting: \n");
65.     merger(arr, 0, n - 1);
66.     disp(arr, n);
67.
68.     return 0;
69. }
70.

```

DS PRACTICAL SLIPS/slip13.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *next;
7. };
8.
9. struct node *create(struct node *head){
10.    int data;
11.    int n;
12.    struct node *temp = head;
13.    printf("How many nodes: ");
14.    scanf("%d", &n);
15.    for(int i = 0; i < n; i++){
16.        struct node *newnode = (struct node *)malloc(sizeof(struct node));
17.        printf("Enter data for node %d: ", i + 1);
18.        scanf("%d", &data);
19.
20.        newnode->data = data;
21.        newnode->next = NULL;
22.
23.        if(head == NULL){
24.            head = newnode;
25.            temp = newnode;
26.        }
27.        else{
28.            temp->next = newnode;
29.            temp = newnode;
30.        }
31.    }
32.    printf("Creation Successful\n");
33.    return head;
34. }
35.
36. void display(struct node *head){
37.    int i = 1;
38.    struct node *tr = head;
39.    while(tr != NULL){
40.        printf("Node -> %d : Data -> %d\n", i, tr->data);
41.        tr = tr ->next;
42.        i++;
43.    }
44. }
45.
46. int len(struct node *head){
47.    int i = 0;

```

```

48.     struct node *tr = head;
49.     while(tr != NULL){
50.         tr = tr ->next;
51.         i++;
52.     }
53.     return i;
54. }
55.
56. void sum(struct node *head){
57.     struct node *temp = head;
58.     int sum = 0;
59.     while(temp != NULL){
60.         sum = sum + temp->data;
61.         temp = temp -> next;
62.     }
63.
64.     printf("The Sum of total elements in list is: [%d]\n", sum);
65. }
66.
67. int main(){
68.     int n;
69.     struct node *head = NULL;
70.
71.     while(1){
72.         printf("1. Create List\n");
73.         printf("2. Length of list\n");
74.         printf("3. Sum of data in list\n");
75.         printf("4. Display\n");
76.         printf("5. Exit\n");
77.         printf("Enter choice: ");
78.         scanf("%d", &n);
79.
80.         switch (n)
81.         {
82.             case 1:
83.                 head = create(head);
84.                 break;
85.
86.             case 2:
87.                 printf("The total length: %d\n", len(head));
88.                 break;
89.
90.             case 3:
91.                 sum(head);
92.                 break;
93.
94.             case 4:
95.                 display(head);
96.                 break;
97.
98.             case 5:
99.                 return 0;
100.
101.            default:
102.                printf("Wrong choice\nJust like ur ex\n");
103.                break;
104.        }
105.
106.    }
107.
108.    return 0;
109. }
110.

```

```

1. #include<stdio.h>
2.
3. int binSearch(int *arr, int low, int high, int key){
4.
5.     if (low > high){
6.         return -1;
7.     }
8.
9.     int mid = (low + high) / 2;
10.
11.    if(arr[mid] == key){
12.        return mid;
13.    }
14.    else if(arr[mid] < key){
15.        return binSearch(arr, mid + 1, high, key);
16.    }
17.    else{
18.        return binSearch(arr, low, mid - 1, key);
19.    }
20. }
21.
22. int main(){
23.     int n;
24.     int key;
25.     printf("Enter n: ");
26.     scanf("%d", &n);
27.     int arr[n];
28.     printf("Enter elements in sorted order: ");
29.     for(int i = 0; i < n; i++)
30.         scanf("%d", &arr[i]);
31.
32.     printf("Enter element to search: ");
33.     scanf("%d", &key);
34.
35.     int res = binSearch(arr, 0, n - 1, key);
36.
37.     if(res != -1){
38.         printf("[%d] found at [%d] location\n", key, res);
39.     }
40.     else{
41.         printf("Not found / array is not in sorted order");
42.     }
43.
44.     return 0;
45. }
46. }
47.

```

DS PRACTICAL SLIPS/slip14.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct tree{
5.     int data;
6.     struct tree *left;
7.     struct tree *right;
8. };
9.
10. struct tree *create(int data){
11.     struct tree *newtree = (struct tree *)malloc(sizeof(struct tree));
12.
13.     newtree->data = data;
14.     newtree->left = newtree->right = NULL;
15.

```

```
16.     return newtree;
17. }
18.
19. struct tree *insert(struct tree *root, int data){
20.     if(root == NULL){
21.         return create(data);
22.     }
23.
24.     if(data < root -> data){
25.         root->left = insert(root->left, data);
26.     }
27.
28.     else if(data > root -> data){
29.         root->right = insert(root->right, data);
30.     }
31.
32.     return root;
33. }
34.
35. void inorder(struct tree *root){
36.     if(root != NULL){
37.         inorder(root->left);
38.         printf("%d ", root->data);
39.         inorder(root->right);
40.     }
41. }
42.
43. int countNode(struct tree *root){
44.     static int count;
45.     if(root == NULL){
46.         return 0;
47.     }
48.     else{
49.         count = 1 + countNode(root->left) + countNode(root->right);
50.     }
51.
52.     return count;
53. }
54.
55. int countOdd(struct tree *root){
56.     if(root == NULL){
57.         return 0;
58.     }
59.
60.
61.     int isOdd = ((root->data % 2 != 0) ? 1 : 0);
62.
63.     return isOdd + countOdd(root->left) + countOdd(root->right);
64. }
65.
66. int main(){
67.     int ch;
68.     int n, c;
69.     struct tree *root = NULL;
70.
71.     printf("How many nodes: ");
72.     scanf("%d", &n);
73.     for(int i = 0; i < n; i++){
74.         printf("Enter data for node %d: ", i + 1);
75.         scanf("%d", &c);
76.         root = insert(root, c);
77.     }
78.     printf("\n");
79.
80.     printf("Displaying: \n");
81.     inorder(root);
82.     printf("\n");
```

```
83.     printf("Total no. of nodes: %d\n", countNode(root));
84.     printf("Total odd numbers: %d\n", countOdd(root));
85.
86.     return 0;
87. }
88.
```

DS PRACTICAL SLIPS/slip15.1.c

```
1. #include<stdio.h>
2.
3. void disp(int *arr, int n){
4.     for(int i = 0; i < n; i++){
5.         printf("[%d]\t", arr[i]);
6.     }
7.     printf("\n");
8. }
9.
10. void swap(int *a, int *b){
11.     int t = *a;
12.     *a = *b;
13.     *b = t;
14. }
15.
16. int part(int *arr, int low, int high){
17.     int piv = arr[low];
18.     int i = low + 1;
19.     int j = high;
20.
21.     while (1)
22.     {
23.         while(i <= high && arr[i] <= piv){
24.             i++;
25.         }
26.
27.         while (arr[j] >= piv)
28.         {
29.             j--;
30.         }
31.
32.         if(i < j){
33.             swap(&arr[i], &arr[j]);
34.         }else{
35.             break;
36.         }
37.     }
38.
39.     swap(&arr[low], &arr[j]);
40.     return j;
41. }
42. }
43.
44. void qckSort(int *arr, int low, int high){
45.     int pi;
46.     if(low < high){
47.         pi = part(arr, low, high);
48.         qckSort(arr, low, pi - 1);
49.         qckSort(arr, pi + 1, high);
50.     }
51. }
52.
53. int main(){
54.     int n;
55.     printf("Enter n: ");
56.     scanf("%d", &n);
```

```

57.     int arr[n];
58.     printf("Enter %d elements: ", n);
59.     for(int i = 0; i < n; i++){
60.         scanf("%d", &arr[i]);
61.     }
62.
63.     disp(arr, n);
64.     printf("After Sort\n");
65.     qckSort(arr, 0, n - 1);
66.     disp(arr, n);
67.
68.     return 0;
69. }
70.

```

DS PRACTICAL SLIPS/slip15.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3. #include<string.h>
4. #define MAX 100
5.
6. struct stack{
7.     int top;
8.     int arr[MAX];
9. };
10.
11. void init(struct stack *s){
12.     s->top = -1;
13. }
14.
15. int isFull(struct stack *s){
16.     return (s->top == MAX - 1);
17. }
18.
19. int isEmpty(struct stack *s){
20.     return (s->top == -1);
21. }
22.
23. void push(struct stack *s, char ch){
24.     if(!isFull(s)){
25.         s->arr[++(s->top)] = ch;
26.     }
27.     else{
28.         printf("Stack Overflow\n");
29.         exit(1);
30.     }
31. }
32.
33. char pop(struct stack *s){
34.     if(!isEmpty(s)){
35.         return s->arr[(s->top)--];
36.     }
37.     else{
38.         printf("Stack underflow \n");
39.         exit(1);
40.     }
41. }
42.
43. void stringIsReversedByStackWOW(char str[]){
44.     struct stack s;
45.     init(&s);
46.

```

```

47.     int n = strlen(str);
48.
49.     for(int i = 0; i < n; i++){
50.         push(&s, str[i]);
51.     }
52.
53.     for(int i = 0; i < n; i++){
54.         str[i] = pop(&s);
55.     }
56. }
57.
58. int main(){
59.     char str[MAX];
60.     printf("Enter string: ");
61.     gets(str);
62.
63.     stringIsReversedByStackWOW(str);
64.
65.
66.     printf("Reversed String: %s\n", str);
67.
68.     return 0;
69. }
70.

```

DS PRACTICAL SLIPS/slip16.1.c

```

1. #include <stdio.h>
2. #include <string.h>
3. #include <stdlib.h>
4.
5. void swap(char *a, char *b){
6.     char t = *a;
7.     *a = *b;
8.     *b = t;
9. }
10.
11. void sorter(char str[], int n){
12.     int i, j;
13.     for(i = 0; i < n - 1; i++){
14.         for(j = 0; j < n - i - 1; j++){
15.             if(str[j] > str[j + 1]){
16.                 swap(&str[j], &str[j + 1]);
17.             }
18.         }
19.     }
20. }
21.
22. void disp(char str[], int n){
23.     int i;
24.     for(i = 0; i < n; i++){
25.         printf("[%c]", str[i]);
26.     }
27.     printf("\n");
28. }
29.
30. int main(){
31.     char str[100];
32.     int n;
33.
34.     printf("Enter string: ");
35.     gets(str);
36.

```

```

37.     n = strlen(str);
38.     sorter(str, n);
39.
40.     printf("After Sort:\n");
41.     disp(str, n);
42.
43.     return 0;
44. }
45.

```

DS PRACTICAL SLIPS/slip16.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct queue{
5.     int data;
6.     struct queue *next;
7. };
8.
9. struct queue *fr = NULL;
10. struct queue *rr = NULL;
11.
12. struct queue *create(int val){
13.     struct queue *newnode = (struct queue *)malloc(sizeof(struct queue));
14.     newnode->data = val;
15.     newnode->next = NULL;
16.
17.     return newnode;
18. }
19.
20. void insert(int val){
21.     struct queue *newnode = create(val);
22.     if(fr == NULL){
23.         fr = rr = newnode;
24.     }
25.     else{
26.         rr->next = newnode;
27.         rr = newnode;
28.     }
29.
30.     rr->next = fr;
31.     printf("INSERTION SUCCESSFUL\n");
32. }
33.
34. void delete(){
35.     struct queue *temp;
36.     if(fr == NULL){
37.         printf("QUEUE EMPTY\n");
38.         return;
39.     }
40.
41.     temp = fr;
42.     if(fr == rr){
43.         fr = NULL;
44.         rr = NULL;
45.     }else{
46.         fr = fr->next;
47.         rr->next = fr;
48.     }
49.     free(temp);
50. }
51.
52. void display(){
53.     struct queue *temp;

```

```

54.
55.
56.     if(fr == NULL){
57.         printf("Jane kaha mera jiya kidhar gaya ji...? Abhi abhi yahi tha kidhar gaya ji...?");
58.         return;
59.     }
60.     temp = fr;
61.     do{
62.         printf("[%d]->", temp->data);
63.         temp = temp->next;
64.     }while(temp != fr);
65.
66.     printf("\n");
67. }
68.
69. int main(){
70.     int ch, val;
71.
72.     while(1){
73.         printf("1. Insert element\n");
74.         printf("2. Delete element\n");
75.         printf("3. Display\n");
76.         printf("4. Exit\n");
77.         printf("Enter choice: ");
78.         scanf("%d", &ch);
79.
80.         switch (ch)
81.         {
82.             case 1:
83.                 printf("Enter number: ");
84.                 scanf("%d", &val);
85.                 insert(val);
86.                 break;
87.
88.             case 2:
89.                 delete();
90.                 break;
91.
92.             case 3:
93.                 display();
94.                 break;
95.
96.             case 4:
97.                 return 0;
98.
99.             default:
100.                 printf("WRONG CHOICE\n");
101.                 break;
102.             }
103.         }
104.
105.     return 0;
106. }
107.

```

DS PRACTICAL SLIPS/slip17.1.c

```

1. #include<stdio.h>
2.
3. struct cust{
4.     int cust_id;
5.     char name[50];
6.     char address[100];
7. };
8.

```

```

9. void disp(struct cust arr[], int n){
10.    for(int i = 0; i < n; i++){
11.        printf("\n");
12.        printf("Cust: %d\t", i + 1);
13.        printf("-----\n");
14.        printf("ID:\t%d\n", arr[i].cust_id);
15.        printf("Name:\t%s\n", arr[i].name);
16.        printf("Address:\t%s\n", arr[i].address);
17.    }
18.    printf("-----\n");
19.    printf("\n\n");
20. }
21.
22. void sorterOP(struct cust arr[], int n){
23.    for(int i = 0; i < n - 1; i++){
24.        for(int j = 0; j < n - i - 1; j++){
25.            if(arr[j].cust_id > arr[j + 1].cust_id){
26.                struct cust temp = arr[j];
27.                arr[j] = arr[j + 1];
28.                arr[j + 1] = temp;
29.            }
30.        }
31.    }
32. }
33.
34. int main(){
35.    int n;
36.    printf("How many customer: ");
37.    scanf("%d", &n);
38.    struct cust custumber[n];
39.
40.    for(int i = 0; i < n; i++){
41.        printf("Enter details for cust %d\n", i + 1);
42.        printf("Enter customer id: ");
43.        scanf("%d", &custumber[i].cust_id);
44.
45.        printf("Enter Name: ");
46.        scanf("%s", &custumber[i].name);
47.
48.        printf("Enter address: ");
49.        scanf("%s", &custumber[i].address);
50.    }
51.
52.    printf("Displaying list: \n");
53.    disp(custumber, n);
54.    printf("AFTER SORTING\n");
55.    sorterOP(custumber, n);
56.    disp(custumber, n);
57.
58.    return 0;
59. }
60.

```

DS PRACTICAL SLIPS/slip17.2.c

```

1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *left;
7.     struct node *right;
8. };
9.
10. struct node *create(int data){

```

```
11.     struct node *newnode = (struct node *)malloc(sizeof(struct node));
12.
13.     newnode->data = data;
14.     newnode->left = newnode->right = NULL;
15.
16.     return newnode;
17. }
18.
19. struct node *insert(struct node *root, int data){
20.     if(root == NULL){
21.         return create(data);
22.     }
23.
24.     if(data < root->data){
25.         root->left = insert(root->left, data);
26.     }
27.     else if(data > root->data){
28.         root->right = insert(root->right, data);
29.     }
30.
31.     return root;
32. }
33.
34. void inorder(struct node *root){
35.     if(root != NULL){
36.         inorder(root->left);
37.         printf("[%d]\t", root->data);
38.         inorder(root->right);
39.     }
40. }
41.
42. int calcHeight(struct node *root){
43.     if(root == NULL){
44.         return -1;
45.     }
46.     else{
47.         int lheight = calcHeight(root->left);
48.         int rheight = calcHeight(root->right);
49.
50.         return(lheight > rheight ? lheight : rheight) + 1;
51.     }
52. }
53.
54. int main(){
55.     int n, c;
56.     struct node *root = NULL;
57.
58.     printf("How many nodes: ");
59.     scanf("%d", &n);
60.
61.     for(int i = 0; i < n; i++){
62.         printf("Enter data for node %d: ", i + 1);
63.         scanf("%d", &c);
64.
65.         root = insert(root, c);
66.     }
67.
68.     printf("Displaying tree: \n");
69.     inorder(root);
70.
71.     printf("\nHeight of tree: %d\n", calcHeight(root));
72.
73.     return 0;
74. }
75.
```

DS PRACTICAL SLIPS/slip18.1.c

```
1. #include<stdio.h>
2. #include<ctype.h>
3. #include<string.h>
4.
5. struct city{
6.     int city_code;
7.     char name[30];
8. };
9.
10. int main(){
11.     int n;
12.     char str[30];
13.     int flag = 0;
14.     printf("Enter n: ");
15.     scanf("%d", &n);
16.     struct city c[n];
17.
18.     printf("Enter city code and name\n");
19.     for(int i = 0; i < n; i++){
20.         printf("Enter data for city %d: \n", i + 1);
21.         printf("\n-----\n");
22.         printf("Enter city code: ");
23.         scanf("%d", &c[i].city_code);
24.
25.         printf("Enter city name: ");
26.         scanf("%s", &c[i].name);
27.         printf("\n-----\n");
28.         printf("\n\n");
29.     }
30.
31.     printf("Enter city to search: ");
32.     scanf("%s", str);
33.
34.     for(int i = 0; i < n; i++){
35.         if(strcmp(c[i].name, str) == 0){
36.             printf("Code: [%d] : City [%s]", c[i].city_code, c[i].name);
37.             flag = 1;
38.         }
39.     }
40.     if(flag == 0){
41.         printf("%s is not present\n", str);
42.     }
43.
44.     return 0;
45. }
46.
```

DS PRACTICAL SLIPS/slip18.2.c

```
1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct queue{
5.     int data;
6.     struct queue *next;
7. };
8.
9. struct queue *fr = NULL;
10. struct queue *rr = NULL;
11.
12. struct queue *create(int val){
13.     struct queue *newnode = (struct queue *)malloc(sizeof(struct queue));
14.     newnode->data = val;
```

```
15.     newnode->next = NULL;
16.
17.     return newnode;
18. }
19.
20. void insert(int val){
21.     struct queue *newnode = create(val);
22.     if(fr == NULL){
23.         fr = rr = newnode;
24.     }
25.     else{
26.         rr->next = newnode;
27.         rr = newnode;
28.     }
29.
30.     printf("INSERTION SUCCSSFUL\n");
31. }
32.
33. void delete(){
34.     struct queue *temp;
35.     if(fr == NULL){
36.         printf("Queue is empty\n");
37.         return;
38.     }
39.     temp = fr;
40.     if(fr == rr){
41.         fr = rr = NULL;
42.     }
43.     else{
44.         fr = fr -> next;
45.     }
46.     free(temp);
47.     printf("DELETION SUCCESSFUL\n");
48. }
49.
50. void display(){
51.     struct queue *temp;
52.
53.     if(fr == NULL){
54.         printf("Accha ji me hara chalo maan jaao na...\\n");
55.         return;
56.     }
57.     temp = fr;
58.
59.     while(temp != NULL){
60.         printf("[%d]->", temp->data);
61.         temp = temp->next;
62.     }
63.     printf("\\n\\a");
64. }
65.
66. void length(){
67.     struct queue *temp = fr;
68.     int i = 1;
69.     while(temp != NULL){
70.         i++;
71.         temp = temp -> next;
72.     }
73.     printf("Length of queue: [%d]\\n", i - 1);
74. }
75.
76. void search(){
77.     struct queue *temp = fr;
78.     int n;
79.     int pos = 1;
80.     int flag = 0;
81.     printf("Enter element to search: ");
```

```
82.     scanf("%d", &n);
83.
84.     while (temp != NULL)
85.     {
86.         if(n == temp->data){
87.             flag = 1;
88.             pos++;
89.         }
90.         temp = temp -> next;
91.     }
92.
93.     if(flag == 1){
94.         printf("%d found at %d location\n", n, pos + 1);
95.     }
96.     else{
97.         printf("%d is not in queue\n");
98.     }
99. }
100.
101. int main(){
102.     int val;
103.     int n;
104.
105.     while(1){
106.         printf("1. Insert element\n");
107.         printf("2. Delete element\n");
108.         printf("3. Display queue\n");
109.         printf("4. Length of queue\n");
110.         printf("5. Search elemnent in queue\n");
111.         printf("6. Exit\n");
112.         printf("Your choice: ");
113.         scanf("%d", &n);
114.         switch (n)
115.         {
116.             case 1:
117.                 printf("Enter value: ");
118.                 scanf("%d", &val);
119.                 insert(val);
120.                 break;
121.
122.             case 2:
123.                 delete();
124.                 break;
125.
126.             case 3:
127.                 display();
128.                 break;
129.
130.             case 4:
131.                 length();
132.                 break;
133.
134.             case 5:
135.                 search();
136.                 break;
137.
138.             case 6:
139.                 return 0;
140.
141.             default:
142.                 printf("galat kadam\nEk kadam swatchata ki ore\nnswatch bharat abhiyaan me apna
sahyong de\n");
143.                 break;
144.         }
145.     }
146.     return 0;
147. }
```

DS PRACTICAL SLIPS/slip19.1.c

```
1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. struct Node {
5.     char data;
6.     struct Node* next;
7. };
8.
9. void insertAtEnd(struct Node** head, char newData) {
10.    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
11.    if (newNode == NULL) {
12.        printf("Memory allocation failed!\n");
13.        return;
14.    }
15.    newNode->data = newData;
16.    newNode->next = NULL;
17.
18.    if (*head == NULL) {
19.        *head = newNode;
20.        return;
21.    }
22.
23.    struct Node* current = *head;
24.    while (current->next != NULL) {
25.        current = current->next;
26.    }
27.
28.    current->next = newNode;
29. }
30.
31. void displayList(struct Node* head) {
32.     printf("Vowels Linked List: ");
33.     struct Node* current = head;
34.
35.     while (current != NULL) {
36.         printf("%c -> ", current->data);
37.         current = current->next;
38.     }
39.     printf("NULL\n");
40. }
41.
42. void freeList(struct Node* head) {
43.     struct Node* current = head;
44.     struct Node* nextNode;
45.
46.     while (current != NULL) {
47.         nextNode = current->next;
48.         free(current);
49.         current = nextNode;
50.     }
51. }
52.
53. int main() {
54.     struct Node* head = NULL;
55.
56.     printf("--- Creating the Linked List of Vowels ---\n");
57.
58.     insertAtEnd(&head, 'a');
59.     insertAtEnd(&head, 'e');
60.     insertAtEnd(&head, 'i');
61.     insertAtEnd(&head, 'o');
```

```

62.     insertAtEnd(&head, 'u');
63.
64.     printf("\n--- Displaying the Linked List ---\n");
65.     displayList(head);
66.
67.     freeList(head);
68.
69.     return 0;
70. }
71.

```

DS PRACTICAL SLIPS/slip19.2.c

```

1. #include <stdio.h>
2. #include <stdlib.h>
3.
4. struct Node {
5.     int data;
6.     struct Node* next;
7. };
8.
9. struct Node* createNode(int data) {
10.     struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
11.
12.     newNode->data = data;
13.     newNode->next = NULL;
14.     return newNode;
15. }
16.
17. void push(struct Node** top, int data) {
18.     struct Node* newNode = createNode(data);
19.     newNode->next = *top;
20.     *top = newNode;
21.     printf("Pushed %d onto the stack.\n", data);
22. }
23.
24. int pop(struct Node** top) {
25.     if (*top == NULL) {
26.         printf("Stack Underflow: Stack is empty.\n");
27.         return -1;
28.     }
29.     struct Node* temp = *top;
30.     int data = temp->data;
31.     *top = (*top)->next;
32.     free(temp);
33.     return data;
34. }
35.
36. void displayStack(struct Node* top) {
37.     if (top == NULL) {
38.         printf("Stack is empty.\n");
39.         return;
40.     }
41.     printf("Stack (TOP -> BOTTOM): ");
42.     struct Node* current = top;
43.     while (current != NULL) {
44.         printf("%d ", current->data);
45.         current = current->next;
46.     }
47.     printf("\n");
48. }
49.
50. void reverseDisplay(struct Node* top) {
51.     if (top == NULL) {
52.         return;

```

```
53.    }
54.    reverseDisplay(top->next);
55.    printf("%d ", top->data);
56. }
57.
58. int main() {
59.     struct Node* top = NULL;
60.     int choice, data;
61.
62.     while (1) {
63.         printf("\n--- Menu ---\n");
64.         printf("1. Push\n");
65.         printf("2. Pop\n");
66.         printf("3. Display (Top to Bottom)\n");
67.         printf("4. Reverse Display (Bottom to Top)\n");
68.         printf("5. Exit\n");
69.         printf("Enter your choice: ");
70.
71.         if (scanf("%d", &choice) != 1) {
72.             while (getchar() != '\n');
73.             printf("Invalid input. Please enter a number.\n");
74.             continue;
75.         }
76.
77.         switch (choice) {
78.             case 1:
79.                 printf("Enter data to push: ");
80.                 scanf("%d", &data);
81.                 push(&top, data);
82.                 break;
83.
84.             case 2:
85.                 data = pop(&top);
86.                 if (data != -1) {
87.                     printf("Popped element: %d\n", data);
88.                 }
89.                 break;
90.
91.             case 3:
92.                 displayStack(top);
93.                 break;
94.
95.             case 4:
96.                 printf("Stack in Reverse Order (Insertion Order): ");
97.                 reverseDisplay(top);
98.                 printf("\n");
99.                 break;
100.
101.            case 5:
102.                printf("Exiting program.\n");
103.                freeStack(top);
104.                return 0;
105.
106.            default:
107.                printf("Invalid choice. Please try again.\n");
108.        }
109.    }
110. }
```

DS PRACTICAL SLIPS/slip20.1.c

```
1. #include<stdio.h>
2.
3. void display(int *arr, int n){
4.     for(int i = 0; i < n; i++){
5.         printf("[%d]\t", arr[i]);
6.     }
7.     printf("\n");
8. }
9.
10. void insertionSort(int *arr, int n){
11.     int i, j, key;
12.     for(int i = 1; i < n; i++){
13.         key = arr[i];
14.         j = i - 1;
15.         while(j >= 0 && arr[j] >= key){
16.             arr[j + 1] = arr[j];
17.             j--;
18.         }
19.         arr[j + 1] = key;
20.     }
21. }
22.
23. int main(){
24.     int n;
25.     printf("Enter n: ");
26.     scanf("%d", &n);
27.     int arr[n];
28.     printf("Enter %d elements: ", n);
29.     for (int i = 0; i < n; i++)
30.     {
31.         scanf("%d", &arr[i]);
32.     }
33.
34.     printf("\n--BEFORE SORT--\n");
35.     display(arr, n);
36.     insertionSort(arr, n);
37.     printf("\n--AFTER SORT--\n");
38.     display(arr, n);
39.
40.     return 0;
41. }
42.
```

DS PRACTICAL SLIPS/slip20.2.c

```
1. #include<stdio.h>
2. #include<stdlib.h>
3.
4. struct node{
5.     int data;
6.     struct node *left;
7.     struct node *right;
8. };
9.
10. struct node *create(int data){
11.     struct node *newnode = (struct node *)malloc(sizeof(struct node));
12.     newnode->data = data;
13.     newnode->left = newnode->right = NULL;
14.
15.     return newnode;
16. }
17.
18. struct node *insert(struct node *root, int data){
```

```
19.     if(root == NULL){
20.         return create(data);
21.     }
22.
23.     if(data > root -> data){
24.         root->right = insert(root->right, data);
25.     }
26.     else if(data < root -> data){
27.         root->left = insert(root->left, data);
28.     }
29.
30.     return root;
31. }
32.
33. void inorder(struct node *root){
34.     if(root != NULL){
35.         inorder(root->left);
36.         printf("%d\n", root->data);
37.         inorder(root->right);
38.     }
39. }
40.
41. int main(){
42.     int n, c;
43.     struct node *root = NULL;
44.     printf("How many nodes: ");
45.     scanf("%d", &c);
46.
47.     for(int i = 0; i < c; i++){
48.         printf("Enter data for node %d: ", i + 1);
49.         scanf("%d", &n);
50.         root = insert(root, n);
51.     }
52.
53.     inorder(root);
54.
55.     return 0;
56. }
```

<END>