

18-09-2020

Date _____
Page _____

DS Lab : Program 1 (Admission)

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

```
struct student {
```

```
{
```

```
char name[50];
```

```
int age, marks;
```

```
}
```

```
S[n];
```

```
int main()
```

```
{
```

```
int i, n;
```

```
printf("Enter the number of students %d");
```

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

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

```
    S[i];
```

```
    printf("Roll number: %d\n", i);
```

```
    printf("Enter age: %d\n");
```

```
    scanf("%d", &S[i].age);
```

```
    printf("Enter name: %s\n");
```

```
    scanf("%s", &S[i].name);
```

```
    printf("Enter marks: %d\n");
```

```
    scanf("%d", &S[i].marks);
```

```
    if (age < 20 || marks < 65) {
```

```
        printf("Not Eligible\n");
```

```
        break; }
```

Date _____
Page _____

(Program 1) Program 1

```
if(age >= 20 & marks >= 65) {
```

printf ("Selected");

printf (" %s ", SciJname);

```
if (age < 0 || marks < 0) {
```

printf ("Invalid");

break;

}

return 0;

}

(Program 2)

(Program 3)

(Program 4)

(Program 5)

(Program 6)

(Program 7)

(Program 8)

(Program 9)

(Program 10)

(Program 11)

(Program 12)

(Program 13)

(Program 14)

(Program 15)

(Program 16)

(Program 17)

(Program 18)

(Program 19)

(Program 20)

(Program 21)

(Program 22)

(Program 23)

(Program 24)

(Program 25)

(Program 26)

(Program 27)

(Program 28)

(Program 29)

(Program 30)

(Program 31)

(Program 32)

(Program 33)

(Program 34)

(Program 35)

(Program 36)

(Program 37)

(Program 38)

(Program 39)

(Program 40)

(Program 41)

(Program 42)

(Program 43)

(Program 44)

(Program 45)

(Program 46)

(Program 47)

(Program 48)

(Program 49)

(Program 50)

(Program 51)

(Program 52)

(Program 53)

(Program 54)

(Program 55)

(Program 56)

(Program 57)

(Program 58)

(Program 59)

(Program 60)

(Program 61)

(Program 62)

(Program 63)

(Program 64)

(Program 65)

(Program 66)

(Program 67)

(Program 68)

(Program 69)

(Program 70)

(Program 71)

(Program 72)

(Program 73)

(Program 74)

(Program 75)

(Program 76)

(Program 77)

(Program 78)

(Program 79)

(Program 80)

(Program 81)

(Program 82)

(Program 83)

(Program 84)

(Program 85)

(Program 86)

(Program 87)

(Program 88)

(Program 89)

(Program 90)

(Program 91)

(Program 92)

(Program 93)

(Program 94)

(Program 95)

(Program 96)

(Program 97)

(Program 98)

(Program 99)

(Program 100)

25-09-2020

Week 2: Stack Implementation (Program 2)

1. Soln: #include < stdio.h >

include < stdlib.h >

define max 5

int top = -1, stack[max];

void push();

void pop();

void display();

void main()

{

int ch;

while(1)

{

printf("In Stack Menu");

printf("1. Push 2. Pop 3. Display 4. Exit");

printf("Enter your choice ");

scanf("%d", &ch);

switch(ch)

{

case 1: push(); break;

case 2: pop(); break;

case 3: display(); break;

case 4: Exit(); break;

default: printf("Wrong Choice");

}

}

}

}

}

void push()

{

int val;

if ($top == max - 1$)

* printf ("Stack is Full \n");

else {

printf ("Enter Element to push \n");

scanf ("%d", &val);

~~top + 1~~;

stack [top] = val;

}

void pop()

if ($top == -1$)

* printf ("Stack is empty \n");

else {

* printf ("Enter element to pop \n");

scanf ("%d", &val);

~~top + 1~~;

printf ("Deleted element is %d", stack [top]);

~~top - 1~~;

}

void display()

{

int i; stack [i];

if ($top == -1$)

* printf ("Stack is empty \n");

else {

 printf ("Stack is: \n");

 for (i = top; i >= 0; i--) {

 printf ("%d ", stack[i]);

}

}

Week 2: Stack Using Pointer: (Program 3)

Sol 2.1 #include <stdio.h>

void push (int stack[], int ele, int *top, int *size) {

 if (*top == *size - 1)

 printf ("Stack Overflow\n");

 else {

 (*top)++;

 stack[*top] = ele; }

}

int pop (int stack[], int *top, int *size)

 if (*top == -1) {

 printf ("Stack Underflow\n");

 return -1; }

 else {

 int n = stack[*top];

 (*top)--;

 return n; }

}

void display (int stack[], int *top, int size)

{

int i;

if (*top == -1)

printf ("Stack is empty\n");

else

for (i = *top; i >= 0; i--)

printf ("%d\n", stack[i]);

}

}

3. int main () { /* Main Function */ }

{

int size = 5;

int top = -1;

int stack [size];

int n = 1; ele;

while (n != 0)

{

printf ("1: Push\n 2: Pop\n 3: Display\n n: Exit\n");

printf ("Enter your choice\n");

scanf ("%d", &n);

if (n == 0)

break;

if (n == 1)

printf ("Enter Element:\n");

scanf ("%d", &ele);

push (stack, ele, &top, &size);

}

Date _____
Page _____

else if ($x == 3$) {

int popel = pop(stack, &top, &size);

printf ("The element popped out is: %d", popel);

else if ($x == 3$) {

printf ("The stack is: (%u");

display (stack, &top, &size);

}

return 0;

}

Week 3: Lab 3: Infix to Postfix (Program)

```
#include <stdio.h>
#define max 50
char stack[max];
int top = -1;
void push(char ch)
{
    if (top == max - 1)
        printf("Stack is full\n");
    else
        top++;
    stack[top] = ch;
}
```

```
char pop()
{
    char item;
```

```
    if (top == -1)
        printf("Stack is empty\n");
    else
        item = stack[top];
    top = top - 1;
    return item;
}
```

```
int empty()
{
    if (top == -1)
        return 1;
}
```

else return 0;

}

char top()

{

if (top == -1)

printf ("in stack is empty \n");

else

return stack[top];

int pri (char ch)

{

switch(ch)

{

case '+':

case '-': return (1);

case '*':

case '/': return (2);

case '^': return (3);

default: return(0);

}

int main (int argc, char **argv)

{

char infin[100];

int i, item;

printf ("Enter the infix expression \n");

scanf ("%s", infin);

printf ("Expression is: %s \n", infin);

```
printf("Postfix: %s", infix);
```

```
int i=0;
```

```
while (infix[i] != '\0')
```

```
l
```

```
switch (infix[i])
```

```
case '(': push(infix[i]);
```

break;

```
case ')': while ((item == pop()) != '(')
```

```
printf("%c", item);
```

```
break;
```

```
case '+':
```

```
case '-':
```

```
case '*':
```

```
case '/':
```

```
case '^': while (!empty() && pri(infix[i]) <
```

pri(stack[i]))

```
(item = pop());
```

```
printf("%c", item);
```

or

```
push(infix[i]);
```

```
break;
```

```
default: printf("%c", infix[i]);
```

```
break;
```

if

i + 1

j

1P-10-5050
Date _____
Page _____

```
while (!empty($)) {  
    $ = $->getNext();  
}
```

char item;

item = pop()

(non-printed ("T.C.", etc.) items)

```
printf("%d\n")
```

~~return 0;~~

3

introduction (~~not~~ right) to

© 2018 Brad R. Johnson

East side, 15' (approx.)

With *frank* ; *but* it *isn't* *possible* to *keep*

(N/3333333333, 3333333333) 3333333333

(جیسا کسی جو اپنے بھائی کو بھاگ دے دے

10. *Leucosia* *leucostoma* *leucostoma* *leucostoma*

Constitutive model of the soil

10. The following table gives the number of hours worked by 1000 workers.

($\sin \theta = -\frac{1}{2}$) *

Woolly Mammoth

(most) in right now) as well as by

16-10-2020

Week 4: Implement Queue (Program)

Solⁿ:

```
#include <stdio.h>
#include <stdlib.h>
#define max 5;
int queue[max];
void insert (int val, int *front, int *rear)
{
    if (*rear == max - 1)
        printf ("The given queue is full \n");
    else {
        if (*front == -1)
            *front = 0;
        (*rear)++;
        queue[*rear] = val;
        printf ("Enqueue successful \n");
    }
}

void del (int *front, int *rear)
{
    if (*front == *rear)
        printf ("Queue empty condition \n");
    else {
        printf ("Deleted element is : %d \n", queue
*front + 1);
        *front += 1;
        if (*front == *rear)
            *front = *rear = -1;
    }
}

void display (int *front, int *rear)
{
}
```

(19)

```

if (*rear == -1)
    printf ("Queue empty condition \n");
else {
    int i;
    printf ("Queue elements are \n");
    for (i = *front; i <= *rear; i++)
        printf ("%d ", queue[i]);
}

```

2

int main ()

{

int val, ch;

int front = -1, rear = -1;

while (1)

{

printf ("Enter the choice \n");

printf ("1. Insert \n", 2. Delete \n", 3. Display \n");

scanf ("%d", &ch)

switch (ch) {

case 1: printf ("Enter the value to be inserted \n");

scanf ("%d", &val);

insert (val, &front, &rear);

break;

case 2: del (&front, &rear);

break;

case 3: display (&front, &rear);

break;

case 4: return 0;

default: printf("%u wrong choice")

else if (choice == 1) { /* Insertion */ }

return 0;

else if (choice == 2) { /* Deletion */ }

(choice == 3) { /* Exit */ }

else { /* Wrong choice */ }

(1) Insertion

(2) Deletion

(3) Exit

if (choice == 1) { /* Insertion */ }

insert(); /* Insertion & Deletion function */

else if (choice == 2) { /* Deletion */ }

deletion(); /* Deletion function */

else if (choice == 3) { /* Exit */ }

exit(); /* Exit function */

else { /* Wrong choice */ }

6-11-2020

~~Circular Queue~~ (Program 6)

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

```
#define size 5
```

```
int front = -1;
```

```
int rear = -1;
```

```
int queue[size];
```

```
void enqueue (int n) {
```

```
{
```

```
if (front == 0 & & rear == size - 1)
```

```
printf ("Queue is full");
```

```
else if (front == rear)
```

```
printf ("Queue is full");
```

```
else if (front == -1 & & rear == -1)
```

```
{
```

```
front = 0; rear = 0;
```

```
rear++;
```

```
queue[rear] = n;
```

```
}
```

```
else {
```

```
if (rear == size - 1)
```

```
queue[rear] = n;
```

```
}
```

```
}
```

```
int dequeue () {
```

```
{
```

```
if (front == -1 & & rear == -1)
```

```
return -1;
```

```
else {
```

```
int ele;
```

```

else
    ele = queue[front] // data
    if (front == rear) {
        front = -1;
        rear = -1;
    }
    else
        front = (front + 1) % size;
}

void display () {
    if (front == -1 & & rear == -1)
        printf ("Queue is empty");
    else
        printf ("Queue contains: ");
        if (front <= rear) {
            for (int i = front; i <= rear; i++)
                printf ("%d ", queue[i]);
        }
        else
            for (int i = front; i <= size - 1; i++)
                printf ("%d ", queue[i]);
            for (int i = 0; i <= rear; i++)
                printf ("%d ", queue[i]);
}

```

(Forward)
(Backward) Date _____
int main ()
{

int option;
int item;

do {

printf("In Circular Queue \n");

printf("1. Insert to Queue \n 2. Delete from Queue
3. Display Queue \n 4. Exit (\n)");

printf("Enter your choice (\n)");

scanf("%d", &option);

switch(option){

case 1: printf("Enter the element \n");

scanf("%d", &item);

enqueue(item);

break;

case 2: item = dequeue();

if (item == -1)

printf("Queue is empty \n");

else

printf("Removed element from Queue \n", item);

break;

case 3: display();

break;

case 4: exit(0);

}

while (option != 4);

return 0;

(Program 7)

20-11-2020 (Deletion from linked list)

Date _____
Page _____

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

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node * next;
```

```
} /* structure definition for linked list */
```

```
struct node * head = (NULL); /* initial value of head */
```

```
int length = 0; /* initial value of length */
```

```
void insertend (int ele) /* function to insert element at end */
```

```
{
```

```
    struct node * newnode, * temp;
```

```
    newnode = (struct node *) malloc (sizeof (struct node))
```

```
    (newnode -> data = ele); /* data of new node */
```

```
    newnode -> next = NULL;
```

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

```
        head = newnode;
```

```
    length++; /* increment length */
```

```
} /* end of insertend */
```

```
else /* if list is not empty */
```

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

```
    temp = head;
```

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

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

```
    temp->next = newnode;
```

```
    length++; /* increment length */
```

```
}
```

```
void deletefront ()
```

```

if (length == 0)
    printf ("List is empty \n");
else {
    struct node * temp;
    temp = (struct node *) malloc (sizeof (struct node));
    temp = head;
    head = head->next;
    temp->next = NULL;
    length--;
    printf ("The element deleted is : %d", temp->data);
}

```

9

```

void deleteend () {
    if (length == 0)
        printf ("List is empty \n");
    else {
        struct node * temp;
        temp = (struct node *) malloc (sizeof (struct node));
        temp = head;
        while (temp->next->next != NULL)
            temp = temp->next;
        struct node * del;
        del = (struct node *) malloc (sizeof (struct node));
        del = temp->next;
        temp->next = NULL;
        length--;
    }
}

```

2) \rightarrow `printf ("Element deleted is %u\n", del->data);`

3)

`void deleterandom (int pos)`

{

`if (length == 0)`

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

`elseif (pos == 1)`

`deletefront ();`

`else if (pos >= length + 1)`

`deleteend ();`

`else {`

`struct node *del,`

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

`struct node *temp;`

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

`temp = head;`

`for (int i=1; i< pos-1; i++)`

`temp = temp->next;`

`del = temp->next;`

`temp->next = del->next;`

`del->next = NULL;`

`length--;`

`printf ("The element deleted is : %u\n", del->data);`

}

`void display ()`

{

struct node * temp;

temp = (struct node *) malloc (sizeof(struct node));

temp = head;

if (temp == NULL)

 printf ("List is empty \n");

else {

 printf ("Contents of the list are: \n");

 while (temp != NULL)

 {

 printf ("%d \n", temp -> data);

 temp = temp -> next;

}

}

}

int main()

{

 int choice, ele, pos;

 char ch;

 do {

 printf ("\n 1. Insert at end \n 2. Display \n 3. Delete from front \n 4. Delete from rear \n 5. Delete from random position \n 6. Exit ");

 printf ("Enter your choice ");

 scanf ("%d", &choice);

 switch (choice)

 {

 case 1: printf ("Enter the element to be inserted ");

 scanf ("%d", &ele);

inverted (ell);

break;

case 2: cliff bay (J)

wreck;

case 3: `deletefront()`

break

case 4: `deletemd()`

break

Case 5: ~~differentiable~~();

```
printf ("Enter the position in list: ");
```

near 9 ("100% d.", & bes);

deletorius (yes);

break;

3 while (choice != 6)

return Di

۷

(Program 8)

20-11-2020 (Insertion to Linked List)

Date _____
Page _____

```
#include <stdio.h> // for cout, cout::endl  
// for new, delete
```

```
struct node
```

```
{
```

```
    int data;
```

```
    struct node *next; // next = next element
```

```
}
```

```
struct node *head=NULL;
```

```
int length=0;
```

```
void insertend (int ele) {
```

```
{
```

```
    struct node *newnode, *temp; // new node
```

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

```
newnode->data=ele; // data = element to be inserted
```

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

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

```
{
```

```
    head=newnode;
```

```
length=1; // initial length of list
```

```
}
```

```
else {
```

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

```
    newnode->data=ele;
```

```
    newnode->next=NULL;
```

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

```
        head=newnode; // insertion
```

```
((objnode->next)=newnode); // length + 1, head = new node
```

```
length++; // length + 1
```

```
((objnode->next)=NULL); // new node = NULL
```

(8 unsorted)

Date _____
Page _____

```
temp = (struct node*) malloc (sizeof (struct node));  
temp = head;  
while (temp->next != NULL)  
    temp = temp->next;  
temp->next = newnode;  
length++;
```

void insertfront (int ele)

{

```
struct node * temp;  
temp = (struct node*) malloc (sizeof (struct node));  
temp-> data = ele;  
temp-> next = head;  
head = temp;  
length++;
```

}

void insertrandom (int ele, int pos)

{

if (pos == 1)

insertfront (ele);

else if (pos >= length + 1)

insertend (ele);

else {

struct node * inst;

inst = (struct node*) malloc (sizeof (struct node));

struct node * temp;

temp = (struct node*) malloc (sizeof (struct node));

temp = head;
for (int i=1; i<pos-1; i++)
 temp = temp->next;
inst->data = ele;
inst->next = temp->next;
temp->next = inst;
length++;

void display()
{
 struct node * temp;
 temp = (struct node *) malloc(sizeof(struct node));
 temp = head;
 if (temp == NULL)
 printf("The list is empty\n");
 else
 printf("The contents of the list are:\n");
 while (temp != NULL)
 printf("%d\n", temp->data);
 temp = temp->next;

y
y

int main()

{

 int choice, ele, pos;

```
char ch;
do {
```

1: Insert at end 2: Insert at front
 3: Insert at random position 4: Display
 5: Exit (w) & break

```
printf("Enter your choice ('n'): ");
scanf("%d", &choice);
switch (choice)
{
```

case 1: printf("Enter element to be inserted ('n'): ");
 scanf("%d", &ele);
 insertend(ele);

((break)) break;

case 2: printf("Enter element to be inserted ('n'): ");
 scanf("%d", &ele);
 insertfront(ele);
 break;

case 3: printf("Enter element to be inserted ('n'): ");
 scanf("%d", &ele);
 printf("Enter the position ('n'): ");
 scanf("%d", &pos);
 insertrandom(ele, pos);
 break;

case 4: display();
 break;

I ~~case~~

I while (choice != 5);

return 0;

(Program 9)

11-12-2020 Doubly Linked List

Date _____
Page _____

```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node *prev;
    struct Node *next;
} node;
int length(node *head) {
    int count = 0;
    while (head != NULL) {
        head = head->next;
        count++;
    }
    return count;
}
void insertAtEnd(node **head, int d) {
    node *n, *temp = *head;
    if (*head == NULL) {
        *head = (node *)malloc(sizeof(node));
        (*head)->prev = NULL;
        (*head)->data = d;
        (*head)->next = NULL;
    } else {
        while (temp->next != NULL)
            temp = temp->next;
        n = (node *)malloc(sizeof(node));
        n->data = d;
    }
}
```

(P program)

Date _____

Paper _____

→ next = NULL; → head > return it
n → prev = temp; → null > return it
temp → next = n; → else do the insertion
} → insert it

printf ("6). d was inserted in the list (%d, %d);

9

void insertleft (node ** h, int d, int ele) { }

node * head = * h; → head + sizeof (head) = head. tail

if (head → data == ele) { } → if true

node * temp1 = NULL; → return it

temp1 = (node *) malloc (sizeof (node));

temp1 → prev = NULL; → head

temp1 → data = d; → head

temp1 → next = * h; → return it

(* h) → prev = temp1;

* (h) = temp1; → return it

printf ("6). d was inserted at the start (%d, %d);

return; → if true : head = temp1;

((about) 6) and (* about) = head *

node * temp; → head = (head *); → head

while (head != NULL) { } →

if (head → data == ele) { }

head = head → prev; →

temp = (node *) malloc (sizeof (node));

(temp → data) = d; →

(* temp → prev = head;

temp → next = head → next;

temp → next → prev = temp;

head \rightarrow next = temp;

printf("'%c' was inserted to the left of '%c'. d = %d, old = %d);\nbreak;

} else

head = head \rightarrow next;

}

printf("Given element is not present in the list\n");

void deleteNode(node **head, int d) {
 node *temp = *head;
 if (*head == NULL) {
 printf("No elements in the list to delete\n");
 return;
 }

}

while (temp != NULL) {

if (temp \rightarrow data == d) {

if (temp == *head) {

*head = (*head) \rightarrow next

(*head) \rightarrow prev = NULL;

}

else if (temp \rightarrow next == NULL) {

temp \rightarrow next \rightarrow prev = NULL;

free(temp);

else {

temp \rightarrow prev \rightarrow next = temp \rightarrow next;

temp \rightarrow next \rightarrow prev = temp \rightarrow prev;

```
free(temp); free(d);
```

```
printf("%d was deleted\n", d);
return;
```

3

```
temp = temp->next; head = head
```

3

```
printf("%d is not present in the list\n", d);
3
```

```
void display(node *head) {
    if(head == NULL) {
        printf("Empty list\n");
        return;
    }
3
```

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

```
    printf("- %d ->", head->data);
    head = head->next;
3
```

```
    printf("\n");
3
```

```
int main() {
```

```
    node *head = NULL;
```

```
    int data, pos, opt;
```

```
    printf("Insert few elements in the list (Press-1 to
stop):\n");
```

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

```
    while(data != -1) {
```

```
        insertAtEnd(&head, &data);
    }
3
```

Ques. No. 2 (Any 2 marks)

Printf ("Operations of Doubly linked list in ");
 printf ("1: Insert at left\n 2: Delete specified node
 \n 3: Display\n 4: Insert at end\n
 \n 5: Exit\n");

scanf ("%d", &opt);

while (opt != 5) {

switch (opt) {

case 1: printf ("Enter element to be inserted\n");

scanf ("%d", &data);

printf ("Enter the node: ");

scanf ("%d", &pos);

insertLeft (&head, data, pos);

break;

case 2: printf ("Enter element to be deleted\n");

scanf ("%d", &data);

deleteNode (&head, data);

break;

case 3: display (head);

break;

case 4: printf ("Enter data to be inserted at end\n");

scanf ("%d", &data);

insertAtEnd (&head, data);

break;

printf ("while (opt!=5);

if (head == NULL) break;

(head->prev = head) & (head->next = head);

18-12-2020

Date _____

Page _____

Program 10 : Binary Search Tree

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

```
#include < stdbool.h >
```

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

```
typedef struct binary_node {
```

```
    int data;
```

```
    struct binary_node * left;
```

```
    struct binary_node * right;
```

```
} node;
```

```
void insert (node* &root, int d) {
```

```
    if (*root == NULL) {
```

```
        (*root) = (node*) malloc (sizeof (node));
```

```
        (*root) -> left = NULL;
```

```
        (*root) -> data = d;
```

```
        (*root) -> right = NULL;
```

```
    } else {
```

```
        if (d < ((*root) -> data)) {
```

```
            insert (&((*root) -> left), d);
```

```
        } else {
```

```
            insert (&((*root) -> right), d);
```

```
}
```

```
void inorder (node* &root) {
```

```
    if (root == NULL)
```

```
        return;
```

```
    inorder (root -> left);
```

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

```
    inorder (root -> right);
```

```

3 void preorder ( node *root ) {
    if ( root == NULL )
        return;
    printf ("%d", root->data);
    preorder ( root->left );
    preorder ( root->right );
}

4 void postorder ( node *root ) {
    if ( root == NULL )
        return;
    postorder ( root->left );
    postorder ( root->right );
    printf ("%d", root->data);
}

5 bool search ( node *root, int key ) {
    if ( root == NULL )
        return false;
    if ( root->data == key )
        return true;
    else if ( key < root->data )
        return search ( root->left, key );
    else
        return search ( root->right, key );
}

6 int main () {
    node *root = NULL;
    int choice;
}

```

exit d;

~~and do l d~~

printf ("1: Insert in BST\n 2: Pre Order\n 3: In Order
4: Post Order\n 5: Search\n 6: Exit\n");

scanf ("%d", &choice);

switch (choice) {

case 1: insert (&root, d);

break; /* End of case 1 */

case 2: preorder (root);

break; /* End of case 2 */

case 3: inorder (root);

break; /* End of case 3 */

case 4: postorder (root);

break; /* End of case 4 */

case 5: search (&root, d);

break; /* End of case 5 */

case 6: printf ("Enter element to be searched: ");

scanf ("%d", &d);

if (d == -1)

printf ("Element not found\n");

else if (d == root->data)

printf ("Element found\n");

Date _____
Page _____

```
if (search (root, d)) {  
    printf ("Element found! \n");  
} else
```

~~printf~~ Element not found! \n";

} while (choice != 6);

2

return 0;

3

— →