NAME – RAJDEEP JAISWAL	**DATE – 15 NOV 2021**
BRANCH – BTECH CSE	**SEC = 608 - A**
UID -20BCS2761	**Subject – DATA STRUCTURE Lab**

Question –

1. # C Program for Depth First Binary Tree Search using Recursion

```
2. #include <stdio.h> 3.
#include <stdlib.h>
4.
5.      struct node
6.      {
7.      int a;
8.      struct node *left;
9.      struct node *right;
10.     };
11.
12.     void generate(struct node **, int);
13.     void DFS(struct node *); 14. void delete(struct node
**);
15.
16.     int main()
17.     {
18.     struct node *head = NULL;
19.     int choice = 0, num, flag = 0, key;
20.
21.   do 22.
{
23.             printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
                Exit\nChoice: ");
24.             scanf("%d", &choice);
25.             switch(choice)
26.             {
27.             case 1:
28.             printf("Enter element to insert: ");
```

```
29.             scanf("%d", &num);
30.             generate(&head, num);
31.             break;
32.             case 2:
33.             DFS(head);
34.             break;
35.             case 3:
36.             delete(&head);
37.             printf("Memory Cleared\nPROGRAM TERMINATED\n");
38.             break;
39.             default:
40.             printf("Not a valid input, try again\n");
41.             }
42.             } while (choice != 3);
43.             return 0;
44.             }
45.
46.     void generate(struct node **head, int num)
47.     {
48.     struct node *temp = *head, *prev = *head;
49.
50.                     if (*head == NULL)
51.                     {
52.                     *head = (struct node *)malloc(sizeof(struct node));
53.                     (*head)->a = num;
54.                     (*head)->left = (*head)->right = NULL;
55.                     }
56.                     else
57.                     {
58.                     while (temp != NULL)
```

```
59.                 {
60.                 if (num > temp->a)
61.                 {
62.                 prev = temp;
63.                 temp = temp->right;
64.                 }
65.                 else
66.                 {
67.                 prev = temp;
68.                 temp = temp->left;
69.                 }
70.                 }
71.                 temp = (struct node *)malloc(sizeof(struct node));
72.                 temp->a = num;
73.                 if (num >= prev->a)
74.                 {
```

```
75.             prev->right = temp;
76.             }
77.             else
78.             {
79.             prev->left = temp;
80.             }
81.             }
82.             }
```

83.

```
84.             void DFS(struct node *head)
85.             {
86.             if (head)
87.             {
88.             if (head->left)
89.             {
90.             DFS(head->left);
91.             }
92.             if (head->right)
93.             {
94.             DFS(head->right);
95.             }
96.             printf("%d  ", head->a);
97.             }
98.             }
```

99.

```
100.              void delete(struct node **head)
101.              {
102.              if (*head != NULL)
```

```
103.            {
104.            if ((*head)->left)
105.            {
106.            delete(&(*head)->left);
107.            }
108.            if ((*head)->right)
109.            {
110.            delete(&(*head)->right);
111.            }
112.            free(*head);
113.            }
114.            }
```

OUTPUT

```
Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 5

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 3

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 4

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 2

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
```

```
Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 5

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 3

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 4

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 2

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
```

Question 2:

1. C Program to Traverse the Tree Recursively

Code:

```
1. #include <stdio.h> 2.
#include <stdlib.h>
3.
4.      struct node
5.      {
6.      int a;
7.      struct node *left;
8.      struct node *right;
9.      };
10.
11.      void generate(struct node **, int);
12.      void infix(struct node *);
13.      void postfix(struct node *); 14. void prefix(struct node *); 15. void delete(struct
node **);
16.
17.       int main()
18.       {
19.       struct node *head = NULL;
20.       int choice = 0, num, flag = 0, key;
21.
22.      do
23.      {
24.      printf("\nEnter your choice:\n1. Insert\n2. Traverse via infix\n3.Traverse via
prefix\n4. Traverse via postfix\n5. Exit\nChoice: "); 25.       scanf("%d", &choice);
26.                switch(choice)
27.                {
28.                case 1:
29.                printf("Enter element to insert: ");
30.                scanf("%d", &num);
31.                generate(&head, num);
```

```
32.             break;
33.             case 2:
34.             infix(head);
35.             break;
36.             case 3:
37.             prefix(head);
38.             break;
```

```
39.             case 4:
40.             postfix(head);
41.             break;
42.             case 5:
43.             delete(&head);
44.             printf("Memory Cleared\nPROGRAM TERMINATED\n");
45.             break;
46.             default: printf("Not a valid input, try again\n");
47.             }
48.             } while (choice != 5);
49.             return 0;
50.             }
51.
52.     void generate(struct node **head, int num)
53.     {
54.     struct node *temp = *head, *prev = *head;
55.
56.                     if (*head == NULL)
57.                     {
58.                     *head = (struct node *)malloc(sizeof(struct node));
59.                     (*head)->a = num;
60.                     (*head)->left = (*head)->right = NULL;
61.                     }
62.                     else
63.                     {
64.                     while (temp != NULL)
65.                     {
66.                     if (num > temp->a)
67.                     {
68.                     prev = temp;
```

```
69.                 temp = temp->right;
70.                 }
71.                 else
72.                 {
73.                 prev = temp;
74.                 temp = temp->left;
75.                 }
76.                 }
77.                 temp = (struct node *)malloc(sizeof(struct node));
78.                 temp->a = num;
79.                 if (num >= prev->a)
80.                 {
81.                 prev->right = temp;
82.                 }
83.                 else
84.                 {
```

```
85.                 prev->left = temp;
86.                 }
87.                 }
88.                 }
89.
90.           void infix(struct node *head)
91.           {
92.           if (head)
93.           {
94.           infix(head->left);
95.           printf("%d   ", head->a);
96.           infix(head->right);
97.           }
98.           }
99.
100.    void prefix(struct node *head)
101.    {
102.    if (head)
103.    {
104.    printf("%d   ", head->a);
105.    prefix(head->left);
106.    prefix(head->right);
107.    }
108.    } 109.
110.    void postfix(struct node *head)
111.    {
112.    if (head)
113.    {
114.    postfix(head->left);
115.    postfix(head->right);
```

```
116.    printf("%d   ", head->a);
117.    }
118.    } 119.
120.                        void delete(struct node **head)
121.                        {
122.                        if (*head != NULL)
123.                        {
124.                        if ((*head)->left)
125.                        {
126.                        delete(&(*head)->left);
127.                        }
128.                        if ((*head)->right)
129.                        {
130.                        delete(&(*head)->right);
```

```
131.                }
132.                free(*head);
133.            }
134.        }
```

OUTPUT:

```
Enter element to insert: 2

Enter your choice:
1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit
Choice: 2
2   3   4   5   6
Enter your choice:
1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit
Choice: 3
5   3   2   4   6
Enter your choice:
1. Insert
2. Traverse via infix
3. Traverse via prefix
4. Traverse via postfix
5. Exit
Choice: 4
2   4   3   6   5
Enter your choice:
1. Insert
2. Traverse via infix
```

Question3:

1. C Program to Search an Element in a Tree Recursively **Code:**

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

```
3.      struct node
4.      {
5.      int info;
6.      struct node *left, *right;
7.      };
```

```
8.      struct node *createnode(int key)
9.      {
10.     struct node *newnode = (struct node*)malloc(sizeof(struct node));
11.     newnode->info = key;
12.     newnode->left = NULL;
13.     newnode->right = NULL;
14.     return(newnode);
15.     }
16.     int search(struct node *head, int key)
17.     {
18.     while (head != NULL)
19.     {
20.     if (key > head->info)
21.     {
22.     return search(head->right, key);
23.     }
24.     else if (key < head->info)
25.     {
26.     return search(head->left, key);
27.     }
28.     else
29.     {
30.     return 1;
31.     }
32.     }
33.     return 0;
34.     }
35.     /*
36.     * Main Function
37.     */
```

```
38.     int main()
39.     {
40.     int flag = 0;
41.     /* Creating first Tree. */
42.     struct node *newnode = createnode(25);
43.     newnode->left = createnode(17);
44.     newnode->right = createnode(35);
45.     newnode->left->left = createnode(13);
46.     newnode->left->right = createnode(19);
47.     newnode->right->left = createnode(27); 48.    newnode->right->right =
createnode(55);
49.      /* Sample Tree 1:
50.      *         25
51.      *       /  \
52.      *      17   35
53.      *     /\   /\
```

```
54.         *      13  19   27 55
55.         */
56.         flag = search(newnode,15);
57.         if (flag)
58.         {
59.         printf("Key %d found in tree 1 \n", 15);
60.         }
61.         else
62.         {
63.         printf("Key %d not found in tree 1\n", 15);
64.         }
65.
66.         /* Creating second Tree. */
67.         struct node *node = createnode(1);
68.         node->right = createnode(2);
69.         node->right->right = createnode(3);
70.         node->right->right->right = createnode(4);
71.         node->right->right->right->right = createnode(5);
72.         /* Sample Tree 2:   Right Skewed Tree (Unbalanced).
73.         *         1
74.         *          \
75.         *           2
76.         *            \
77.         *             3
78.         *              \
79.         *               4
80.         *                \
81.         *                 5
82.         */
83.         flag = search(node,4);
```

```
84.         if (flag)
85.         {
86.         printf("Key %d found in tree 2\n", 4);
87.         }
88.         else
89.         {
90.         printf("Key %d not found in tree 2\n", 4);
91.         }
92.
93.     /* Creating third Tree. */
94.     struct node *root = createnode(15);
95.     /* Sample Tree 3- Tree having just one root node.
96.     *         15
97.     */
98.     flag = search(root,15);
99.     if (flag)
```

```
100.        {
101.            printf("Key %d found in tree 3 \n", 15);
102.        }
103.        else
104.        {
105.                printf("Key %d not found in tree 3\n", 15);
106.        }
107.        return 0;
108.    }
```

OUTPUT:

```
Key 15 not found in tree 1
Key 4 found in tree 2
Key 15 found in tree 3
```

Question 4:

1. C Program to Search an Element in a Tree Non-Recursively **Code:**

```
1. #include <stdio.h> 2.
#include <stdlib.h>
3.
4.      struct node
5.      {
6.      int a;
7.      struct node *left;
8.      struct node *right;
9.      };
10.
11. void generate(struct node **, int);
12. int search(struct node *, int);
13. void delete(struct node **);
14.
15.      int main()
16.      {
17.      struct node *head = NULL;
18.      int choice = 0, num, flag = 0, key;
```

```
19.
20.                 do
21.                 {
22.                 printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
23.                 scanf("%d", &choice);
24.                 switch(choice)
25.                 {
26.                 case 1:
27.                 printf("Enter element to insert: ");
28.                 scanf("%d", &num);
29.                 generate(&head, num);
30.                 break;
31.                 case 2:
32.                 printf("Enter key to search: ");
33.                 scanf("%d", &key);
34.                 flag = search(head, key);
35.                 if (flag)
36.                 {
37.                 printf("Key found in tree\n");
38.                 }
39.                 else
40.                 {
41.                 printf("Key not found\n");
42.                 }
43.                 break;
44.                 case 3:
45.                 delete(&head);
46.                 printf("Memory Cleared\nPROGRAM TERMINATED\n");
47.                 break;
48.                 default: printf("Not a valid input, try again\n");
```

```
49.                     }
50.                     } while (choice != 3);
51.                     return 0;
52.                     }
53.
54.     void generate(struct node **head, int num)
55.     {
56.     struct node *temp = *head, *prev = *head;
57.
58.          if (*head == NULL)
59.          {
60.          *head = (struct node *)malloc(sizeof(struct node));
61.          (*head)->a = num;
62.          (*head)->left = (*head)->right = NULL;
63.          }
64.          else
```

```
65.                 {
66.                 while (temp != NULL)
67.                 {
68.                 if (num > temp->a)
69.                 {
70.                 prev = temp;
71.                 temp = temp->right;
72.                 }
73.                 else
74.                 {
75.                 prev = temp;
76.                 temp = temp->left;
77.                 }
78.                 }
79.                 temp = (struct node *)malloc(sizeof(struct node));
80.                 temp->a = num;
81.                 if (num >= prev->a)
82.                 {
83.                 prev->right = temp;
84.                 }
85.                 else
86.                 {
87.                 prev->left = temp;
88.                 }
89.                 }
90.                 }
91.
92.                     int search(struct node *head, int key)
93.                     {
94.                     while (head != NULL)
```

```
95.                 {
96.                 if (key > head->a)
97.                 {
98.                 head = head->right;
99.                 }
100.                else if (key < head->a)
101.                {
102.                head = head->left;
103.                }
104.                else
105.                {
106.                return 1;
107.                }
108.                }
109.                return 0;
110.                }
```

```
111.
112.        void delete(struct node **head)
113.        {
114.           if (*head != NULL)
115.           {
116.              if ((*head)->left)
117.              {
118.                 delete(&(*head)->left);
119.              }
120.              if ((*head)->right)
121.              {
122.                 delete(&(*head)->right);
123.              }
124.              free(*head);
125.           }
126.        }
```

OUTPUT:

```
2. Search
3. Exit
Choice: 1
Enter element to insert: 3

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 1
Enter element to insert:
4

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 2
Enter key to search: 1
Key found in tree

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 2
Enter key to search: 6
Key not found
```

Question 5:

1. C Program to Traverse the Tree Non-Recursively **Code:**

```
1. #include <stdio.h> 2.
#include <stdlib.h>
3.
4.      struct node
5.      {
6.      int a;
7.      struct node *left;
8.      struct node *right;
9.      };
10.
11. void generate(struct node **, int);
12. int search(struct node *, int);
13. void delete(struct node **);
14.
15.      int main()
16.      {
17.      struct node *head = NULL;
18.      int choice = 0, num, flag = 0, key;
19.
20.                  do
21.                  {
22.                  printf("\nEnter your choice:\n1. Insert\n2. Search\n3. Exit\nChoice: ");
23.                  scanf("%d", &choice);
24.                  switch(choice)
25.                  {
26.                  case 1:
27.                  printf("Enter element to insert: ");
28.                  scanf("%d", &num);
29.                  generate(&head, num);
30.                  break;
```

```
31.             case 2:
32.             printf("Enter key to search: ");
33.             scanf("%d", &key);
34.             flag = search(head, key);
35.             if (flag)
36.             {
37.             printf("Key found in tree\n");
38.             }
39.             else
40.             {
41.             printf("Key not found\n");
```

```
42.             }
43.             break;
44.             case 3:
45.             delete(&head);
46.             printf("Memory Cleared\nPROGRAM TERMINATED\n");
47.             break;
48.             default: printf("Not a valid input, try again\n");
49.             }
50.             } while (choice != 3);
51.             return 0;
52.             }
53.
54.     void generate(struct node **head, int num)
55.     {
56.     struct node *temp = *head, *prev = *head;
57.
58.                     if (*head == NULL)
59.                     {
60.                     *head = (struct node *)malloc(sizeof(struct node));
61.                     (*head)->a = num;
62.                     (*head)->left = (*head)->right = NULL;
63.                     }
64.                     else
65.                     {
66.                     while (temp != NULL)
67.                     {
68.                     if (num > temp->a)
69.                     {
70.                     prev = temp;
71.                     temp = temp->right;
```

```
72.                 }
73.                 else
74.                 {
75.                 prev = temp;
76.                 temp = temp->left;
77.                 }
78.                 }
79.                 temp = (struct node *)malloc(sizeof(struct node));
80.                 temp->a = num;
81.                 if (num >= prev->a)
82.                 {
83.                 prev->right = temp;
84.                 }
85.                 else
86.                 {
87.                 prev->left = temp;
```

```
88.          }
89.          }
90.          }
91.
92.     int search(struct node *head, int key)
93.     {
94.     while (head != NULL)
95.     {
96.     if (key > head->a)
97.     {
98.     head = head->right;
99.     }
100.    else if (key < head->a)
101.    {
102.    head = head->left;
103.    }
104.    else
105.    {
106.    return 1;
107.    }
108.    }
109.    return 0;
110.    } 111.
112.                    void delete(struct node **head)
113.                    {
114.                    if (*head != NULL)
115.                    {
116.                    if ((*head)->left)
117.                    {
118.                    delete(&(*head)->left);
```

```
119.                    }
120.                    if ((*head)->right)
121.                    {
122.                    delete(&(*head)->right);
123.                    }
124.                    free(*head);
125.                    }
126.                    }
```

Output

```
1. Insert
2. Search
3. Exit
Choice: 1
Enter element to insert: 3

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 1
Enter element to insert:
4

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 2
Enter key to search: 1
Key found in tree

Enter your choice:
1. Insert
2. Search
3. Exit
Choice: 2
Enter key to search: 6
Key not found

Enter your choice:
```

Question:

6.C Program for Depth First Binary Tree Search without using Recursion

Code:

```
1. #include <stdio.h> 2.
#include <stdlib.h>
3.
4.      struct node
5.      {
6.      int a;
7.      struct node *left;
8.      struct node *right;
9.      int visited;
```

```
10. };
11.
12.     void generate(struct node **, int);
13.     void DFS(struct node *); 14. void delete(struct node
**);
15.
16.     int main()
17.     {
18.     struct node *head = NULL;
19.     int choice = 0, num, flag = 0, key;
20.
21.  do 22.
{
23.             printf("\nEnter your choice:\n1. Insert\n2. Perform DFS Traversal\n3.
                Exit\nChoice: ");
24.             scanf("%d", &choice);
25.             switch(choice)
26.             {
27.             case 1:
28.             printf("Enter element to insert: ");
29.             scanf("%d", &num);
30.             generate(&head, num);
31.             break;
32.             case 2:
33.             DFS(head);
34.             break;
35.             case 3:
36.             delete(&head);
37.             printf("Memory Cleared\nPROGRAM TERMINATED\n");
38.             break;
```

```
39.             default:
40.             printf("Not a valid input, try again\n");
41.             }
42.             } while (choice != 3);
43.
44.     return 0;
45.     }
46.
47.     void generate(struct node **head, int num)
48.     {
49.     struct node *temp = *head, *prev = *head;
50.
51.         if (*head == NULL)
52.         {
53.         *head = (struct node *)malloc(sizeof(struct node));
54.         (*head)->a = num;
```

```
55.                 (*head)->visited = 0;
56.                 (*head)->left = (*head)->right = NULL;
57.                 }
58.                 else
59.                 {
60.                 while (temp != NULL)
61.                 {
62.                 if (num > temp->a)
63.                 {
64.                 prev = temp;
65.                 temp = temp->right;
66.                 }
67.                 else
68.                 {
69.                 prev = temp;
70.                 temp = temp->left;
71.                 }
72.                 }
73.                 temp = (struct node *)malloc(sizeof(struct node));
74.                 temp->a = num;
75.                 temp->visited = 0;
76.                 if (temp->a >= prev->a)
77.                 {
78.                 prev->right = temp;
79.                 }
80.                 else
81.                 {
82.                 prev->left = temp;
83.                 }
84.                 }
```

```
85.                 }
86.
87.     void DFS(struct node *head)
88.     {
89.     struct node *temp = head, *prev;
90.
91.                         printf("On DFS traversal we get:\n");
92.                         while (temp && !temp->visited)
93.                         {
94.                         if (temp->left && !temp->left->visited)
95.                         {
96.                         temp = temp->left;
97.                         }
98.                         else if (temp->right && !temp->right->visited)
99.                         {
100.                        temp = temp->right;
```

```
101.            }
102.            else
103.            {
104.               printf("%d  ", temp->a);
105.               temp->visited = 1;
106.               temp = head;
107.            }
108.         }
109.      }
110.
111.      void delete(struct node **head)
112.      {
113.         if (*head != NULL)
114.         {
115.            if ((*head)->left)
116.            {
117.               delete(&(*head)->left);
118.            }
119.            if ((*head)->right)
120.            {
121.               delete(&(*head)->right);
122.            }
123.            free(*head);
124.         }
125.      }
```

Output:

```
Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 7

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 6

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 1
Enter element to insert: 8

Enter your choice:
1. Insert
2. Perform DFS Traversal
3. Exit
Choice: 2
On DFS traversal we get:
1  2  4  3  6  8  7  5

Enter your choice:
```

Question:

7.C Program to Find Nth Node in the Inorder Traversal of a Tree

Code:

```
1.  typedef struct node
2.  {
3.  int value;
4.  struct node *left;
```

```
5.      struct node *right;
6.      }newnode;
7.
8. newnode *root;
9. static ctr;
10.
11. void nthnode(newnode *root, int n, newnode **nthnode);
12. int main()
13. {
```

```
14.     newnode *temp;
15.     root=0;
16.
17.     // Construct the tree
18.     add(19);
19.     add(20);
20.     add(11);
21.     inorder(root);
22.     // Get the pointer to the nth Inorder node
23.     nthinorder(root, 6, &temp);
24.     printf("\n[%d]\n", temp->value);
25.     return(0);
26.     }
27.
28.    // Get the pointer to the nth inorder node in "nthnode"
29.    void nthinorder(newnode *root, int n, newnode
**nthnode) 30. {
31.     static whichnode;
32.     static found;
33.
34.                 if (!found)
35.                 {
36.                 if (root)
37.                 {
38.                 nthinorder(root->left, n , nthnode);
39.                 if (++whichnode == n)
40.                 {
41.                 printf("\n Found %dth node\n", n);
42.                 found = 1;
43.                 *nthnode = root;
```

```
44.                     }
45.                     nthinorder(root->right, n , nthnode);
46.                     }
47.                     }
48.                     }
49.
50.     inorder(newnode *root)
51.     {
52.     }
53.     // Add value to a Binary Search Tree
54.     add(int value)
55.     {
56.     newnode *temp, *prev, *cur;
57.
58.     temp = malloc(sizeof(newnode));
59.     temp->value = value;
```

```
60.             temp->left  = 0;
61.             temp->right = 0;
62.             if (root == 0)
63.             {
64.             root = temp;
65.             }
66.             else
67.             {
68.             prev = 0;
69.             cur = root;
70.             while(cur)
71.             {
72.             prev = cur;
73.             cur =(value < cur->value)? cur->left : cur->right;
74.             }
75.             if (value > prev->value)
76.             prev->right = temp;
77.             else
78.             prev->left  = temp;
79.             }
80.             }
```

Output:

```
$ cc pgm63.c
$ a.out

[1416572]
```

Question:

8.C Program to Find the Largest value in a Tree using Inorder Traversal

Code:

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
int info;
struct node *left, *right;
};
struct node *createnode(int key)
```

```
9.      {
10.     struct node *newnode = (struct node*)malloc(sizeof(struct node));
11.     newnode->info = key;
12.     newnode->left = NULL;
13.     newnode->right = NULL;
14.     return(newnode);
15.     }
16.     void inorder(struct node *root)
17.     {
18.     if(root != NULL)
19.     {
20.     inorder(root->left);
21.     printf(" %d ",root->info);
22.     inorder(root->right);
23.     }
24.     }
25.     void largest(struct node *root)
26.     {
27.     while (root != NULL && root->right != NULL)
28.     {
29.     root = root->right;
30.     }
31.     printf("\nLargest value is %d\n", root->info);
32.     }
33.     /*
34.     * Main Function
35.     */
36.     int main()
37.     {
38.     /* Creating first Tree. */
```

```
39.     struct node *newnode = createnode(25);
40.     newnode->left = createnode(17);
41.     newnode->right = createnode(35);
42.     newnode->left->left = createnode(13);
43.     newnode->left->right = createnode(19);
44.     newnode->right->left = createnode(27); 45.    newnode->right->right =
createnode(55);
46.      /* Sample Tree 1:
47.      *         25
48.      *        /  \
49.      *       17   35
50.      *      /\   /\
51.      *     13 19  27 55
52.      */
53.      printf("Inorder traversal of tree 1 :");
54.      inorder(newnode);
```

```
55.    largest(newnode);
56.
57.        /* Creating second Tree. */
58.        struct node *node = createnode(1);
59.        node->right = createnode(2);
60.        node->right->right = createnode(3);
61.        node->right->right->right = createnode(4);
62.        node->right->right->right->right = createnode(5);
63.        /* Sample Tree 2:   Right Skewed Tree (Unbalanced).
64.        *         1
65.        *          \
66.        *           2
67.        *            \
68.        *             3
69.        *              \
70.        *               4
71.        *                \
72.        *                 5
73.        */
74.        printf("\nInorder traversal of tree 2 :");
75.        inorder(node);
76.        largest(node);
77.
78.        /* Creating third Tree. */
79.        struct node *root = createnode(15);
80.        /* Sample Tree 3- Tree having just one root node.
81.        *         15
82.        */
```

```
83.     printf("\nInorder traversal of tree 3 :");
84.     inorder(root);
85.     largest(root);
86.     return 0;
87.     }
```

Output:

```
Inorder traversal of tree 1 : 13  17  19  25  27  35  55
Largest value is 55

Inorder traversal of tree 2 : 1  2  3  4  5
Largest value is 5

Inorder traversal of tree 3 : 15
Largest value is 15
```

Question

9. C Program to Implement Depth First Search Traversal using Post Order

Code:

```
1. #include <stdio.h> 2.
#include <stdlib.h>
3.
4.      struct btnode {
5.      int value;
6.      struct btnode *l;
7.      struct btnode *r;
8.      };
9.
10. typedef struct btnode bt;
11. bt *root;
12. bt *new, *list;
13. bt *create_node();
14. void display(bt *);
15. void construct_tree();
16. void dfs(bt *);
17.
18.      void main()
19.      {
20.      construct_tree();
21.      display(root);
22.      printf("\n");
23.      printf("Depth first traversal\n ");
24.      dfs(root);
25.      }
26.
27.      /* Creates an empty node */
28.      bt * create_node()
29.      {
30.      new=(bt *)malloc(sizeof(bt));
```

```
31.     new->l = NULL;
32.     new->r = NULL;
33.     }
34.
35.     /* Constructs a tree */
36.     void construct_tree()
37.     {
38.     root = create_node();
39.     root->value = 50;
```

```
40.    root->l = create_node(); 41.
root->l->value = 20;
42.       root->r = create_node();
43.       root->r->value = 30;
44.       root->l->l = create_node(); 45.     root->l->l->value = 70;
46.       root->l->r = create_node();
47.       root->l->r->value = 80;
48.       root->l->r->r = create_node(); 49.     root->l->r->r->value = 60;
50.    root->l->l->l = create_node(); 51.
root->l->l->l->value = 10;
52.       root->l->l->r = create_node();
53.       root->l->l->r->value = 40;
54.       }
55.
56.            /* Display the elements in a tree using inorder */
57.            void display(bt * list)
58.            {
59.            if (list == NULL)
60.            {
61.            return;
62.            }
63.            display(list->l);
64.            printf("->%d", list->value);
65.            display(list->r);
66.            }
67.
68.            /* Dfs traversal using post order */
69.            void dfs(bt * list)
70.            {
```

```
71.         if (list == NULL)
72.         {
73.         return;
74.         }
75.         dfs(list->l);
76.         dfs(list->r);
77.         printf("->%d ", list->value);
78.         }
```

Output:

```
        50
       /  \
     20    30
    / \
   70   80
  /  \    \
 10  40    60
$ cc tree29.c
$ a.out
->10->70->40->20->80->60->50->30
Depth first traversal
->10->40->70->60->80->20->30->50
```

Question:

10.C Program to Find the Largest value in a Tree using Inorder Traversal

Code:

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  struct node
4.  {
5.  int info;
6.  struct node *left, *right;
```

```
};
struct node *createnode(int key)
{
struct node *newnode = (struct node*)malloc(sizeof(struct node));
newnode->info = key;
newnode->left = NULL;
newnode->right = NULL;
return(newnode);
}
void inorder(struct node *root)
{
if(root != NULL)
{
```

```
20.     inorder(root->left);
21.     printf(" %d ",root->info);
22.     inorder(root->right);
23.     }
24.     }
25.     void largest(struct node *root)
26.     {
27.     while (root != NULL && root->right != NULL)
28.     {
29.     root = root->right;
30.     }
31.     printf("\nLargest value is %d\n", root->info);
32.     }
33.     /*
34.      * Main Function
35.      */
36.     int main()
37.     {
38.     /* Creating first Tree. */
39.     struct node *newnode = createnode(25);
40.     newnode->left = createnode(17);
41.     newnode->right = createnode(35);
42.     newnode->left->left = createnode(13);
43.     newnode->left->right = createnode(19);
44.     newnode->right->left = createnode(27); 45.    newnode->right->right =
createnode(55);
46.       /* Sample Tree 1:
47.        *          25
48.        *        /   \
49.        *       17    35
```

```
50.     *      /\   /\
51.     *     13 19  27 55
52.     */
53.     printf("Inorder traversal of tree 1 :");
54.     inorder(newnode);
55.     largest(newnode);
56.
57.     /* Creating second Tree. */
58.     struct node *node = createnode(1);
59.     node->right = createnode(2);
60.     node->right->right = createnode(3);
61.     node->right->right->right = createnode(4);
62.     node->right->right->right->right = createnode(5);
63.     /* Sample Tree 2:   Right Skewed Tree (Unbalanced).
64.     *         1
65.     *          \
```

```
66.     *         2
67.     *          \
68.     *           3
69.     *            \
70.     *             4
71.     *              \
72.     *               5
73.     */
74.     printf("\nInorder traversal of tree 2 :");
75.     inorder(node);
76.     largest(node);
77.
78.     /* Creating third Tree. */
79.     struct node *root = createnode(15);
80.     /* Sample Tree 3- Tree having just one root node.
81.     *         15
82.     */
83.     printf("\nInorder traversal of tree 3 :");
84.     inorder(root);
85.     largest(root);
86.     return 0;
87.     }
```

Output:

```
Inorder traversal of tree 1 : 13  17  19  25  27  35  55
Largest value is 55

Inorder traversal of tree 2 : 1  2  3  4  5
Largest value is 5

Inorder traversal of tree 3 : 15
Largest value is 15
```

****

Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):

Sr. No.	Parameters	Marks Obtained	Maximum Marks
1.			
2.			
3.			