

Name: Om Vivek Gharge

PRN: 2020BTEIT00041

Q1.

CODE:

```
1  /*
2  Name: Om Vivek Gharge
3  PRN: 2020BTEIT00041
4  Q1.
5  */
6
7  #include <stdio.h>
8  #include <conio.h>
9  #include <stdlib.h>
10 struct tree
11 {
12     int info;
13     struct tree *left;
14     struct tree *right;
15 };
16 struct tree *insert(struct tree *root, int x)
17 {
18     if (!root)
19     {
20         root = (struct tree *)malloc(sizeof(struct tree));
21         root->info = x;
22         root->left = NULL;
23         root->right = NULL;
24         return (root);
25     }
26     if (root->info > x)
27         root->left = insert(root->left, x);
28     else
29     {
30         if (root->info < x)
31             root->right = insert(root->right, x);
32     }
33     return (root);
34 }
35 void delete_tree(struct tree *ptr)
36 {
37     if (ptr)
38     {
39         delete_tree(ptr->left);
```

```

40         delete_tree(ptr->right);
41         free(ptr);
42     }
43 }
44 void inorder(struct tree *root)
45 {
46     if (root != NULL)
47     {
48         inorder(root->left);
49         printf(" %d", root->info);
50         inorder(root->right);
51         return;
52     }
53     return;
54 }
55 void postorder(struct tree *root)
56 {
57     if (root != NULL)
58     {
59         postorder(root->left);
60         postorder(root->right);
61         printf(" %d", root->info);
62     }
63     return;
64 }
65 void preorder(struct tree *root)
66 {
67     if (root != NULL)
68     {
69         printf(" %d", root->info);
70         preorder(root->left);
71         preorder(root->right);
72     }
73     return;
74 }
75 int main()
76 {
77     struct tree *root;
78     int choice, data, item_no;

```

```

79     root = NULL;
80     while (1)
81     {
82         printf("\n Menu\n");
83         printf("\n \t 1. Insert");
84         printf("\n\t 2. Delete");
85         printf("\n\t 3. Inorder traversal");
86         printf("\n\t 4. Postorder traversal");
87         printf("\n\t 5. Preorder traversal");
88         printf("\n\t 6. Exit ");
89         printf("\n\t Enter choice ");
90         scanf(" %d", &choice);
91         if (choice < 1 || choice > 5)
92             break;
93         switch (choice)
94         {
95             case 1:
96                 printf("\n Enter data ");
97                 scanf("%d", &data);
98                 root = insert(root, data);
99                 printf("\n root is %d", root->info);
100                break;
101             case 2:
102                 delete_tree(root);
103                 break;
104             case 3:
105                 printf("\n Inorder traversal of binary tree: ");
106                 inorder(root);
107                 break;
108             case 4:
109                 printf("\n Postorder traversal of binary tree: ");
110                 postorder(root);
111                 break;
112             case 5:
113                 printf("\n Preorder traversal of binary tree: ");
114                 preorder(root);
115             }
116         }
117         return 0;
118     }

```

OUTPUT:

```
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1

Enter data 1
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1

Enter data 2
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 1

Enter data 4
root is 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
```

```
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 3

Inorder traversal of binary tree: 1 2 4
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 4

Postorder traversal of binary tree: 4 2 1
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 5

Preorder traversal of binary tree: 1 2 4
Menu
1. Insert
2. Delete
3. Inorder traversal
4. Postorder traversal
5. Preorder traversal
6. Exit
Enter choice 6
```

Q2.

CODE:

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  struct BST
4  {
5      int data;
6      struct BST *left;
7      struct BST *right;
8  };
9  typedef struct BST NODE;
10 NODE *node;
11 NODE *createtree(NODE *node, int data)
12 {
13     if (node == NULL)
14     {
15         NODE *temp;
16         temp = (NODE *)malloc(sizeof(NODE));
17         temp->data = data;
18         temp->left = temp->right = NULL;
19         return temp;
20     }
21     if (data < (node->data))
22     {
23         node->left = createtree(node->left, data);
24     }
25     else if (data > node->data)
26     {
27         node->right = createtree(node->right, data);
28     }
29     return node;
30 }
31 NODE *search(NODE *node, int data)
32 {
33     if (node == NULL)
34         printf("\nElement not found");
35     else if (data < node->data)
36     {
37         node->left = search(node->left, data);
38     }
39     else if (data > node->data)
40     {
```

```

41:         node->right = search(node->right, data);
42:     }
43:     else
44:         printf("\nElement found is: %d", node->data);
45:     return node;
46: }
47: void inorder(NODE *node)
48: {
49:     if (node != NULL)
50:     {
51:         inorder(node->left);
52:         printf("%d", node->data);
53:         inorder(node->right);
54:     }
55: }
56: void preorder(NODE *node)
57: {
58:     if (node != NULL)
59:     {
60:         printf("%d", node->data);
61:         preorder(node->left);
62:         preorder(node->right);
63:     }
64: }
65: void postorder(NODE *node)
66: {
67:     if (node != NULL)
68:     {
69:         postorder(node->left);
70:         postorder(node->right);
71:         printf("%d", node->data);
72:     }
73: }
74: NODE *findMin(NODE *node)
75: {
76:     if (node == NULL)
77:     {
78:         return NULL;
79:     }

```

```

80     if (node->left)
81         return findMin(node->left);
82     else
83         return node;
84 }
85 NODE *del(NODE *node, int data)
86 {
87     NODE *temp;
88     if (node == NULL)
89     {
90         printf("\nElement not found");
91     }
92     else if (data < node->data)
93     {
94         node->left = del(node->left, data);
95     }
96     else if (data > node->data)
97     {
98         node->right = del(node->right, data);
99     }
100     else
101     {
102         if (node->right && node->left)
103         {
104             temp = findMin(node->right);
105             node->data = temp->data;
106             node->right = del(node->right, temp->data);
107         }
108         else
109         {
110             temp = node;
111             if (node->left == NULL)
112                 node = node->right;
113             else if (node->right == NULL)
114                 node = node->left;
115             free(temp);
116         }
117     }
118     return node;
119 }

```

```

120 void main()
121 {
122     int data, ch, i;
123     NODE *root = NULL;
124     while (1)
125     {
126         printf("\n1.Insertion in Binary Search Tree");
127         printf("\n2.Search Element in Binary Search Tree");
128         printf("\n3.Delete Element in Binary Search Tree");
129         printf("\n4.Inorder\n5.Preorder\n6.Postorder\n7.Exit");
130         printf("\nEnter your choice: ");
131         scanf("%d", &ch);
132         switch (ch)
133         {
134             case 1:
135                 printf("\nEnter value ");
136                 scanf("%d", &data);
137                 root = createtree(root, data);
138                 break;
139             case 2:
140                 printf("\nEnter the element to search: ");
141                 scanf("%d", &data);
142                 root = search(root, data);
143                 break;
144             case 3:
145                 printf("\nEnter the element to delete: ");
146                 scanf("%d", &data);
147                 root = del(root, data);
148                 break;
149             case 4:
150                 printf("\nInorder Traversal: \n");
151                 inorder(root);
152                 break;
153             case 5:
154                 printf("\nPreorder Traversal: \n");
155                 preorder(root);
156                 break;
157             case 6:
158                 printf("\nPostorder Traversal: \n");
159                 postorder(root);
160                 break;
161             case 7:
162                 exit(0);
163             default:
164                 printf("\nwrong option");
165                 break;
166         }
167     }
168 }
169

```


OUTPUT:

```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1
```

Enter value 1

```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1
```

Enter value 2

```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 1
```

Enter value 3

```
1.Insertion in Binary Search Tree
2.Search Element in Binary Search Tree
3.Delete Element in Binary Search Tree
4.Inorder
5.Preorder
6.Postorder
7.Exit
Enter your choice: 2
```

Enter the element to search: 2

Element found is: 2

- 1.Insertion in Binary Search Tree
- 2.Search Element in Binary Search Tree
- 3.Delete Element in Binary Search Tree
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your choice: 4

Inorder Traversal:

1 2 3

- 1.Insertion in Binary Search Tree
- 2.Search Element in Binary Search Tree
- 3.Delete Element in Binary Search Tree
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your choice: 5

Preorder Traversal:

1 2 3

- 1.Insertion in Binary Search Tree
- 2.Search Element in Binary Search Tree
- 3.Delete Element in Binary Search Tree
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your choice: 6

Postorder Traversal:

3 2 1

- 1.Insertion in Binary Search Tree
- 2.Search Element in Binary Search Tree
- 3.Delete Element in Binary Search Tree
- 4.Inorder
- 5.Preorder
- 6.Postorder
- 7.Exit

Enter your choice: 7

Q3.

CODE:

```
1  /*
2   * Name: On Vivek Gharge
3   * PRN: 2020011700041
4   * Q3.
5   */
6
7  #include <stdio.h>
8  #include <stdlib.h>
9  #define bool int
10 struct tNode
11 {
12     int data;
13     struct tNode *left;
14     struct tNode *right;
15 };
16 struct sNode
17 {
18     struct tNode *t;
19     struct sNode *next;
20 };
21 void push(struct sNode **top_ref, struct tNode *t);
22 struct tNode *pop(struct sNode **top_ref);
23 bool isEmpty(struct sNode *top);
24 void inorder(struct tNode *root)
25 {
26     struct tNode *current = root;
27     struct sNode *s = NULL;
28     bool done = 0;
29     while (!done)
30     {
31         if (current != NULL)
32         {
33             push(&s, current);
34             current = current->left;
35         }
36         else
37         {
38             if (!isEmpty(s))
39             {
```

```

39         {
40             current = pop(&s);
41             printf("%d ", current->data);
42             current = current->right;
43         }
44         else
45             done = 1;
46     }
47 }
48 }
49 void push(struct sNode **top_ref, struct tNode *t)
50 {
51     struct sNode *new_tNode =
52         (struct sNode *)malloc(sizeof(struct sNode));
53     if (new_tNode == NULL)
54     {
55         printf("Stack Overflow \n");
56         getchar();
57         exit(0);
58     }
59     new_tNode->t = t;
60     new_tNode->next = (*top_ref);
61     (*top_ref) = new_tNode;
62 }
63 bool isEmpty(struct sNode *top)
64 {
65     return (top == NULL) ? 1 : 0;
66 }
67 struct tNode *pop(struct sNode **top_ref)
68 {
69     struct tNode *res;
70     struct sNode *top;
71     if (isEmpty(*top_ref))
72     {
73         printf("Stack Underflow \n");
74     }
75     else
76     {

```

```

77         top = *top_ref;
78         res = top->t;
79         *top_ref = top->next;
80         free(top);
81         return res;
82     }
83 }
84 struct tNode *newtNode(int data)
85 {
86     struct tNode *tNode = (struct tNode *)
87         malloc(sizeof(struct tNode));
88     tNode->data = data;
89     tNode->left = NULL;
90     tNode->right = NULL;
91     return (tNode);
92 }
93 int main()
94 {
95     struct tNode *root = newtNode(1);
96     root->left = newtNode(2);
97     root->right = newtNode(3);
98     root->left->left = newtNode(4);
99     root->left->right = newtNode(5);
100     printf("\n Inorder traversal is:");
101     inOrder(root);
102     return 0;
103 }

```

OUTPUT:

```
Inorder traversal is:4 2 5 1 3
```

Q4.

CODE:

```
1  /*
2     Name: Om Vivek Gharge
3     PRN: 2020011108041
4     Q4.
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #define MAX_SIZE 100
9  struct Node
10 {
11     int data;
12     struct Node *left, *right;
13 };
14 struct Stack
15 {
16     int size;
17     int top;
18     struct Node **array;
19 };
20 struct Node *newNode(int data)
21 {
22     struct Node *node = (struct Node *)malloc(sizeof(struct Node));
23     node->data = data;
24     node->left = node->right = NULL;
25     return node;
26 }
27 struct Stack *createStack(int size)
28 {
29     struct Stack *stack = (struct Stack *)malloc(sizeof(struct Stack));
30     stack->size = size;
31     stack->top = -1;
32     stack->array = (struct Node **)malloc(stack->size * sizeof(struct Node *));
33     return stack;
34 }
35 int isFull(struct Stack *stack)
36 {
37     return stack->top + 1 == stack->size;
38 }
39 int isEmpty(struct Stack *stack)
40 {
```

```

41     return stack->top == -1;
42 }
43 void push(struct Stack *stack, struct Node *node)
44 {
45     if (isFull(stack))
46         return;
47     stack->array[++stack->top] = node;
48 }
49 struct Node *pop(struct Stack *stack)
50 {
51     if (isEmpty(stack))
52         return NULL;
53     return stack->array[stack->top--];
54 }
55 struct Node *peek(struct Stack *stack)
56 {
57     if (isEmpty(stack))
58         return NULL;
59     return stack->array[stack->top];
60 }
61 void postOrderIterative(struct Node *root)
62 {
63     if (root == NULL)
64         return;
65     struct Stack *stack = createStack(MAX_SIZE);
66     do
67     {
68         while (root)
69         {
70             if (root->right)
71                 push(stack, root->right);
72             push(stack, root);
73             root = root->left;
74         }
75         root = pop(stack);
76         if (root->right && peek(stack) == root->right)
77         {
78             pop(stack);
79             push(stack, root);
80             root = root->right;
81         }
82         else
83         {
84             printf("%d ", root->data);
85             root = NULL;
86         }
87     } while (!isEmpty(stack));
88 }
89 int main()
90 {
91     struct Node *root = NULL;
92     root = newNode(1);
93     root->left = newNode(2);
94     root->right = newNode(3);
95     root->left->left = newNode(4);
96     root->left->right = newNode(5);
97     root->right->left = newNode(6);
98     root->right->right = newNode(7);
99     printf("Post order traversal of binary tree is :");
100     postOrderIterative(root);
101     return 0;
102 }

```

OUTPUT:

```

Post order traversal of binary tree is :4 5 2 6 7 3 1

```

Q5.

CODE:

```
1  /*
2   * Name: Om Vivesh Chavge
3   * PGN: 2020HTEI100041
4   * Q5.
5   */
6  #include <stdio.h>
7  #include <stdlib.h>
8  struct node
9  {
10     struct node *left;
11     struct node *right;
12     int data;
13 };
14 struct stack
15 {
16     struct node *data;
17     struct stack *next;
18 };
19 void push(struct stack **top, struct node *n);
20 struct node *pop(struct stack **top);
21 int isEmpty(struct stack *top);
22 int tree_traversal(struct node *root)
23 {
24     struct node *temp = root;
25     struct stack *s_temp = NULL;
26     int flag = 1;
27     while (flag)
28     {
29         if (temp)
30         {
31             printf("%d", temp->data);
32             push(&s_temp, temp);
33             temp = temp->left;
34         }
35         else
36         {
37             if (isEmpty(s_temp))
38             {
39                 temp = pop(&s_temp);
40                 temp = temp->right;
41             }
42         }
43     }
44 }
```



```

41     }
42     else
43         flag = 0;
44     }
45 }
46 }
47 void push(struct stack **top, struct node *n)
48 {
49     struct stack *new_n = (struct stack *)malloc(sizeof(struct stack));
50     new_n->data = n;
51     new_n->next = (*top);
52     (*top) = new_n;
53 }
54 int isEmpty(struct stack *top)
55 {
56     if (top == NULL)
57         return 1;
58     else
59         return 0;
60 }
61 struct node *pop(struct stack **top_n)
62 {
63     struct node *item;
64     struct stack *top;
65     top = *top_n;
66     item = top->data;
67     *top_n = top->next;
68     free(top);
69     return item;
70 }
71 struct node *create_node(int data)
72 {
73     struct node *new_n = (struct node *)malloc(sizeof(struct node));
74     new_n->data = data;
75     new_n->left = NULL;
76     new_n->right = NULL;
77     return (new_n);
78 }
79 int main()
80 {
81     struct node *root;
82     root = create_node(8);
83     root->left = create_node(5);
84     root->right = create_node(4);
85     root->left->left = create_node(7);
86     root->left->right = create_node(6);
87     printf("\n Preorder traversal is:");
88     tree_traversal(root);
89     return 0;
90 }

```

OUTPUT:

```
Preorder traversal is:  8  5  7  6  4
```