

LAB REPORT

**CSE2011 – DATA STRUCTURES AND ALGORITHMS LAB**



**(B.Tech. CSE Specialisation in Bioinformatics)**

**WINTER SEMESTER 2020-2021**

|  |  |
| --- | --- |
| **Name:** | ALOK MATHUR |
| **Reg. No:** | 20BCB0086 |
| **Slot:** | L51+L52 |
| **Faculty Name:** | SRIVANI A Ma’am |

**VIT – A Place to Learn; A Chance to Grow**

**ASSIGNMENT 3**

1. Write a program to perform the following operations:

1. Create a binary search tree
2. Insert an element into a binary search tree
3. Deletion of an element(all the options)

d) Sort the elements of the BST.

e) Find the minimum and maximum element in the BST

f) Find the kth minimum element in the BST

**CODE**

#include <stdio.h>

#include <iostream>

#include <string.h>

using namespace std;

struct node

{

    int data;

    struct node \*left\_child;

    struct node \*right\_child;

};

struct node \*newNode(int *n*)

{

    struct node \*node = new struct node;

    node->data = *n*;

    node->left\_child = NULL;

    node->right\_child = NULL;

};

struct node \*find\_minimum(struct node \**root*);

void inorder\_traversal(struct node \**p*)

{

    if (*p* == NULL)

        return;

    inorder\_traversal(*p*->left\_child);

    cout << *p*->data << " ";

    inorder\_traversal(*p*->right\_child);

}

void preorder\_traversal(struct node \**p*)

{

    if (*p* == NULL)

        return;

    cout << *p*->data << " ";

    preorder\_traversal(*p*->left\_child);

    preorder\_traversal(*p*->right\_child);

}

void postorder\_traversal(struct node \**p*)

{

    if (*p* == NULL)

        return;

    postorder\_traversal(*p*->left\_child);

    postorder\_traversal(*p*->right\_child);

    cout << *p*->data << " ";

}

struct node \*insert(struct node \**root*, int *n*)

{

    if (*root* == NULL)

    {

        return newNode(*n*);*//Creating the main root node*

    }

    else if (*n* > *root*->data)

    {

*root*->right\_child = insert(*root*->right\_child, *n*);

    }

    else if (*n* < *root*->data)

    {

*root*->left\_child = insert(*root*->left\_child, *n*);

    }

    return *root*;

}

struct node \*del(struct node \**root*, int *key*)

{

    if (*root* == NULL)

        return *root*;

    if (*key* < *root*->data)

*root*->left\_child = del(*root*->left\_child, *key*);

    else if (*key* > *root*->data)

*root*->right\_child = del(*root*->right\_child, *key*);

    else

    {

*// node with only one child or no child*

        if (*root*->left\_child == NULL)

        {

            struct node \*temp = *root*->right\_child;

            free(*root*);

            return temp;

        }

        else if (*root*->right\_child == NULL)

        {

            struct node \*temp = *root*->left\_child;

            free(*root*);

            return temp;

        }

*// node with two children:*

        struct node \*temp = find\_minimum(*root*->right\_child);

*// Copy the inorder*

*root*->data = temp->data;

*// Delete the inorder successor*

*root*->right\_child = del(*root*->right\_child, temp->data);

    }

    return *root*;

}

struct node \*find\_minimum(struct node \**root*)

{

    if (*root* == NULL)

        return NULL;

    else if (*root*->left\_child != NULL)

        return find\_minimum(*root*->left\_child);

    return *root*;

}

struct node \*max\_element(struct node \**root*)

{

    struct node\*temp=*root*;

    while(temp->right\_child!=NULL)

        temp=temp->right\_child;

    return temp;

}

int main()

{

*/\* Let us create following BST*

*50*

*/     \*

*30      70*

*/  \    /  \*

*20   40  60   80 \*/*

    struct node \*root = NULL;

    root = insert(root, 50);

    insert(root, 30);

    insert(root, 70);

    insert(root, 20);

    insert(root, 40);

    insert(root, 60);

    insert(root, 80);

    cout << "\*\*BST\*\*" << endl;

    int choice, x,k,a;

    struct node \*t;

    struct node \*temp;

    while (choice != 10)

    {

        cout << "Options available are:-" << endl;

        cout << "1.Insert an element into BST\n2.Deletion of Node\n3.Inorder Traversal\n4.Preorder Traversal\n5.Postorder Traversal\n6.Minimum element\n7.Maximum element\n8.Exit" << endl;

        cin >> choice;

        switch (choice)

        {

        case 1:

            cout << "Enter the number to be inserted" << endl;

            cin >> x;

            insert(root, x);

            break;

        case 2:

            cout << "Enter the element to be deleted " << endl;

            cin >> x;

            del(root, x);

            inorder\_traversal(root);

            cout << "\n";

            break;

        case 3:

            inorder\_traversal(root);

            cout << "\n";

            break;

        case 4:

            preorder\_traversal(root);

            cout << "\n";

            break;

        case 5:

            postorder\_traversal(root);

            cout << "\n";

            break;

        case 6:

            t = find\_minimum(root);

            cout << t->data<<endl;

            break;

        case 7:

            temp = max\_element(root);

            cout << temp->data<<endl;

            break;

        case 8:

            cout << "Exiting..." << endl;

            break;

        default:

            cout << "Invalid Input" << endl;

            break;

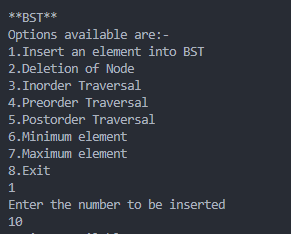
        }

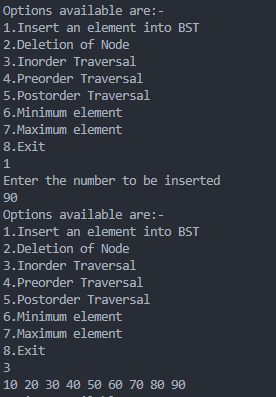
    }

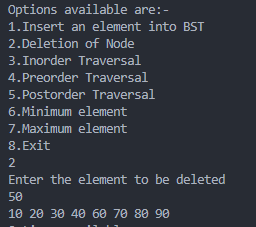
    return 1;

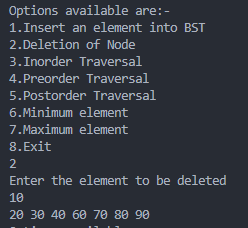
}

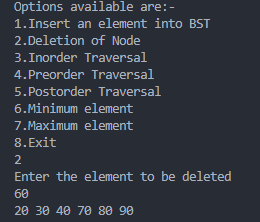
**OUTPUT**

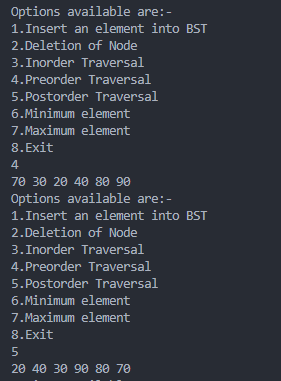


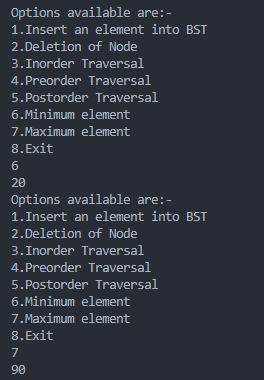


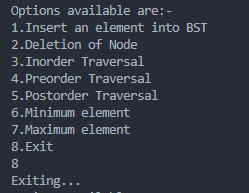












2. Consider a postfix expression, construct the expression tree, and traverse the tree using . Other possible traversals and display the corresponding expressions.

**CODE**

#include <iostream>

using namespace std;

struct n

{

    char d;

    n \*l;

    n \*r;

};

char pf[50];

int top = -1;

n \*a[50];

int r(char *inputch*)

{

    if (*inputch* == '+' || *inputch* == '-' || *inputch* == '\*' || *inputch* == '/')

        return (-1);

    else if (*inputch* >= 'A' || *inputch* <= 'Z')

        return (1);

    else if (*inputch* >= 'a' || *inputch* <= 'z')

        return (1);

    else

        return (-100);

}

void push(n \**tree*)

{

    top++;

    a[top] = *tree*;

}

n \*pop()

{

    top--;

    return (a[top + 1]);

}

void construct\_expression\_tree(char \**suffix*)

{

    char s;

    n \*newl, \*p1, \*p2;

    int flag;

    s = *suffix*[0];

    for (int i = 1; s != 0; i++)

    {

        flag = r(s);

        if (flag == 1)

        {

            newl = new n;

            newl->d = s;

            newl->l = NULL;

            newl->r = NULL;

            push(newl);

        }

        else

        {

            p1 = pop();

            p2 = pop();

            newl = new n;

            newl->d = s;

            newl->l = p2;

            newl->r = p1;

            push(newl);

        }

        s = *suffix*[i];

    }

}

void preOrder(n \**tree*)

{

    if (*tree* != NULL)

    {

        cout << *tree*->d;

        preOrder(*tree*->l);

        preOrder(*tree*->r);

    }

}

void inOrder(n \**tree*)

{

    if (*tree* != NULL)

    {

        inOrder(*tree*->l);

        cout << *tree*->d;

        inOrder(*tree*->r);

    }

}

void postOrder(n \**tree*)

{

    if (*tree* != NULL)

    {

        postOrder(*tree*->l);

        postOrder(*tree*->r);

        cout << *tree*->d;

    }

}

int main(int *argc*, char \*\**argv*)

{

    cout << "Enter Postfix Expression : ";

    cin >> pf;

    construct\_expression\_tree(pf);

    cout << "\nIn-Order Traversal : ";

    inOrder(a[0]);

    cout << "\nPre-Order Traversal : ";

    preOrder(a[0]);

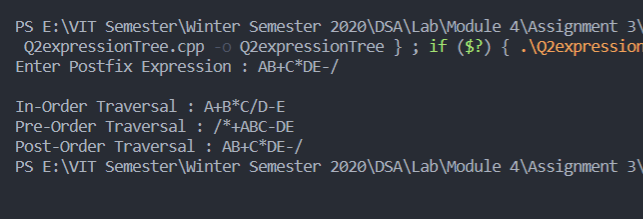
    cout << "\nPost-Order Traversal : ";

    postOrder(a[0]);

    return 0;

}

**OUTPUT**



3. Write a program that takes the details of mobile phone (model name, year, camera resolution, RAM , memory card size and Operating system) and sort the mobile phones in ascending order based on their RAM size using insertion sort.

**CODE**

#include <stdio.h>

#include <iostream>

#include <string.h>

using namespace std;

struct phone{

    char model[100];

    int year;

    float cam;

    int ram;

    int mem;

    char os[100];

}s[10];

void insertion\_sort(struct phone *p*[],int *n*)

{

    int i,key,j;

    for(i=1;i<*n*;i++)

    {

        key=*p*[i].ram;

        j=i-1;

        while(j>=0 && *p*[j].ram>key)

        {

*p*[j+1].ram=*p*[j].ram;

            j--;

        }

*p*[j+1].ram=key;

    }

}

int main()

{

    int i,n;

    cout<<"Pls enter Number of mobile phones"<<endl;

    cin>>n;

    for(i=0;i<n;i++)

    {

        cout<<"Pls enter the model of phone"<<" number "<<i+1<<endl;

        cin>>s[i].model;

        cout<<"Pls enter the year of phone"<<" number "<<i+1<<endl;

        cin>>s[i].year;

        cout<<"Pls enter the camera resoluton of phone"<<" number "<<i+1<<endl;

        cin>>s[i].cam;

        cout<<"Pls enter the RAM of phone"<<" number "<<i+1<<endl;

        cin>>s[i].ram;

        cout<<"Pls enter the memory of phone"<<" number "<<i+1<<endl;

        cin>>s[i].mem;

        cout<<"Pls enter the OS of phone"<<" number "<<i+1<<endl;

        cin>>s[i].os;

    }

    insertion\_sort(s,n);

    cout<<"\n";

    cout<<"\n";

    for(i=0;i<n;i++)

    {

        cout<<s[i].year<<endl;

        cout<<s[i].model<<endl;

        cout<<s[i].cam<<endl;

        cout<<s[i].ram<<endl;

        cout<<s[i].mem<<endl;

        cout<<s[i].os<<endl;

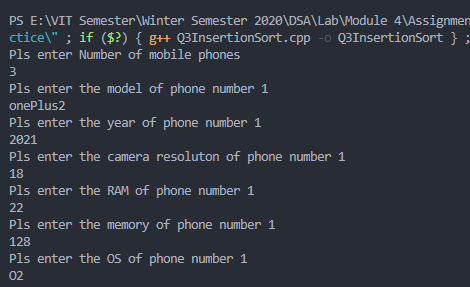
        cout<<"\n";

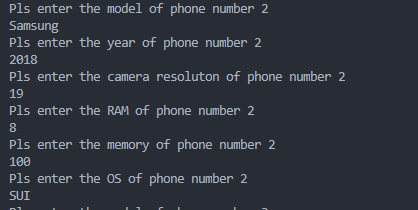
    }

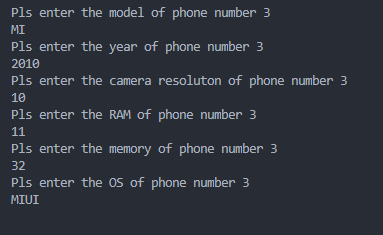
    return 1;

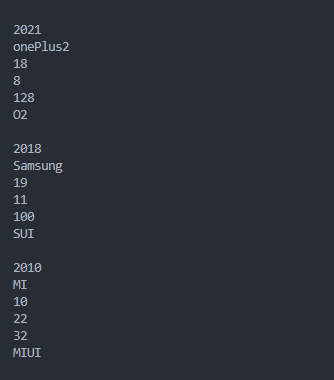
}

**OUTPUT**









4. Write a program that takes the details of a patient (hospital number, patient name, age, token number, height, , weight, reason(disease ) and sort the patients in ascending order based on their token number using quick sort.

**CODE**

#include <stdio.h>

#include <iostream>

#include <string.h>

using namespace std;

struct patient

{

   int hn;

   char name[200];

   int age;

   int tn;

   float height;

   float weight;

   char dis[200];

};

void quicksort(struct patient *s*[], int *first*, int *last*)

{

   int i, j, pivot;

   patient temp;

   if (*first* < *last*)

   {

      pivot = *first*;

      i = *first*;

      j = *last*;

      while (i < j)

      {

         while (*s*[i].tn <= *s*[pivot].tn && i < *last*)

            i++;

         while (*s*[j].tn > *s*[pivot].tn)

            j--;

         if (i < j)

         {

            temp = *s*[i];

*s*[i] = *s*[j];

*s*[j] = temp;

         }

      }

      temp = *s*[pivot];

*s*[pivot] = *s*[j];

*s*[j] = temp;

      quicksort(*s*, *first*, j - 1);

      quicksort(*s*, j + 1, *last*);

   }

}

int main()

{

   patient s[10];

   int choice, n,i;

   cout << "\*\*\*Quick Sort Application\*\*\*" << endl;

   cout << "Number of patients\n";

   cin >> n;

   for(i=0;i<n;i++)

   {

      cin>>s[i].hn>>s[i].name>>s[i].age>>s[i].tn>>s[i].height>>s[i].weight>>s[i].dis;

   }

   quicksort(s,0,n-1);

   cout<<"\n";

   cout<<"\n";

   for(i=0;i<n;i++)

   {

      cout<<s[i].hn<<endl;

      cout<<s[i].name<<endl;

      cout<<s[i].age<<endl;

      cout<<s[i].tn<<endl;

      cout<<s[i].height<<endl;

      cout<<s[i].weight<<endl;

      cout<<s[i].dis<<endl;

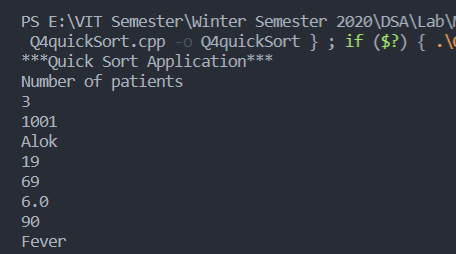
      cout<<"\n";

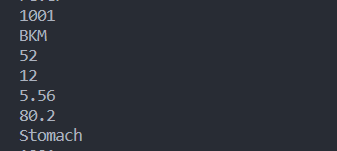
   }

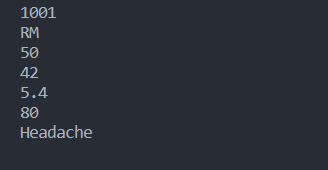
   return 1;

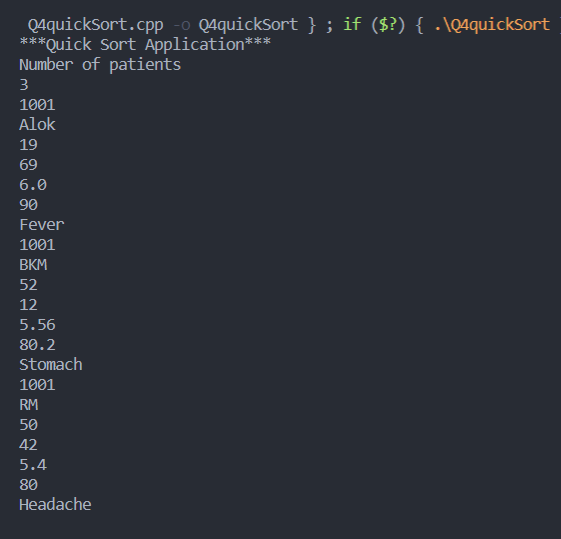
}

**OUTPUT**









***Sorted According to Token Number***

