**Instructions: Please read carefully**

* Please rename this file as only your ID number **(e.g. 18-\*\*\*\*\*-1.doc or 18-\*\*\*\*\*-1.pdf).**
* Submit the file by **Next Class** in the Portal Lab Performance section labeled **Lab task 10. If you cannot complete the full task, do not worry. Just upload what you have completed.**

|  |  |  |
| --- | --- | --- |
| 1. **Write a C++ code to implement Binary Search Tree operations (insertion, traversal and searching)**   **Do the following to write program for a BST:**   1. To construct a binary search tree of integers (**insert** one by one). 2. To **traverse** the tree using all the methods i.e., in order, preorder and post order. 3. To **search** an element on the BST. 4. There are three cases when you delete a node.  * Case 1: Node with zero child (Leaf node) * Case 2: Node with one child * Case 3: Node with both children   **Implement the logic of 3 cases one by one.**  **Hint**: Your program should ask the user to input the choice what operation the user wants to perform.   1. Insert 2. Travers 3. Search 4. Delete  |  |  | | --- | --- | | Class Node{  Data  lptr , rptr  Node(){  Everything null  }  Node(int a){  Data =a  Everything else null  }  } | Class BST{  root  BST(){  Root = NULL  }  insert(x){  if(Root == null){  Node \* nptr = new Node();  Nptr->Data = x;  Root = nptr;  ……}// for inserting root node  else {………..}// for inserting rest of the nodes  }  Preorder(Node \* tptr){  If(tptr !=Null){  Print(tptr->Data)  Preorder(tptr->lptr)  Preorder(tptr->rptr)    }  }  } |   main(){  BST b  b.insert(10)  b.insert(20)  b.Preorder(b.root)  } |
| **Your code here:**  **# include <iostream>**  **# include <cstdlib>**  **using namespace std;**  **struct node**  **{**  **int info;**  **struct node \*left;**  **struct node \*right;**  **}\*root;**  **class BST**  **{**  **public:**  **void find(int, node \*\*, node \*\*);**  **void insert(node \*, node \*);**  **void del(int);**  **void case\_a(node \*,node \*);**  **void case\_b(node \*,node \*);**  **void case\_c(node \*,node \*);**  **void preorder(node \*);**  **void inorder(node \*);**  **void postorder(node \*);**  **void display(node \*, int);**  **BST()**  **{**  **root = NULL;**  **}**  **};**  **void BST::find(int item, node \*\*par, node \*\*loc)**  **{**  **node \*ptr, \*ptrsave;**  **if (root == NULL)**  **{**  **\*loc = NULL;**  **\*par = NULL;**  **return;**  **}**  **if (item == root->info)**  **{**  **\*loc = root;**  **\*par = NULL;**  **return;**  **}**  **if (item < root->info)**  **ptr = root->left;**  **else**  **ptr = root->right;**  **ptrsave = root;**  **while (ptr != NULL)**  **{**  **if (item == ptr->info)**  **{**  **\*loc = ptr;**  **\*par = ptrsave;**  **return;**  **}**  **ptrsave = ptr;**  **if (item < ptr->info)**  **ptr = ptr->left;**  **else**  **ptr = ptr->right;**  **}**  **\*loc = NULL;**  **\*par = ptrsave;**  **}**  **void BST::insert(node \*tree, node \*newnode)**  **{**  **if (root == NULL)**  **{**  **root = new node;**  **root->info = newnode->info;**  **root->left = NULL;**  **root->right = NULL;**  **cout<<"Root Node is Added"<<endl;**  **return;**  **}**  **if (tree->info == newnode->info)**  **{**  **cout<<"Element already in the tree"<<endl;**  **return;**  **}**  **if (tree->info > newnode->info)**  **{**  **if (tree->left != NULL)**  **{**  **insert(tree->left, newnode);**  **}**  **else**  **{**  **tree->left = newnode;**  **(tree->left)->left = NULL;**  **(tree->left)->right = NULL;**  **cout<<"Node Added To Left"<<endl;**  **return;**  **}**  **}**  **else**  **{**  **if (tree->right != NULL)**  **{**  **insert(tree->right, newnode);**  **}**  **else**  **{**  **tree->right = newnode;**  **(tree->right)->left = NULL;**  **(tree->right)->right = NULL;**  **cout<<"Node Added To Right"<<endl;**  **return;**  **}**  **}**  **}**  **void BST::del(int item)**  **{**  **node \*parent, \*location;**  **if (root == NULL)**  **{**  **cout<<"Tree empty"<<endl;**  **return;**  **}**  **find(item, &parent, &location);**  **if (location == NULL)**  **{**  **cout<<"Item not present in tree"<<endl;**  **return;**  **}**  **if (location->left == NULL && location->right == NULL)**  **case\_a(parent, location);**  **if (location->left != NULL && location->right == NULL)**  **case\_b(parent, location);**  **if (location->left == NULL && location->right != NULL)**  **case\_b(parent, location);**  **if (location->left != NULL && location->right != NULL)**  **case\_c(parent, location);**  **free(location);**  **}**  **void BST::case\_a(node \*par, node \*loc )**  **{**  **if (par == NULL)**  **{**  **root = NULL;**  **}**  **else**  **{**  **if (loc == par->left)**  **par->left = NULL;**  **else**  **par->right = NULL;**  **}**  **}**  **void BST::case\_b(node \*par, node \*loc)**  **{**  **node \*child;**  **if (loc->left != NULL)**  **child = loc->left;**  **else**  **child = loc->right;**  **if (par == NULL)**  **{**  **root = child;**  **}**  **else**  **{**  **if (loc == par->left)**  **par->left = child;**  **else**  **par->right = child;**  **}**  **}**  **void BST::case\_c(node \*par, node \*loc)**  **{**  **node \*ptr, \*ptrsave, \*suc, \*parsuc;**  **ptrsave = loc;**  **ptr = loc->right;**  **while (ptr->left != NULL)**  **{**  **ptrsave = ptr;**  **ptr = ptr->left;**  **}**  **suc = ptr;**  **parsuc = ptrsave;**  **if (suc->left == NULL && suc->right == NULL)**  **case\_a(parsuc, suc);**  **else**  **case\_b(parsuc, suc);**  **if (par == NULL)**  **{**  **root = suc;**  **}**  **else**  **{**  **if (loc == par->left)**  **par->left = suc;**  **else**  **par->right = suc;**  **}**  **suc->left = loc->left;**  **suc->right = loc->right;**  **}**  **void BST::preorder(node \*ptr)**  **{**  **if (root == NULL)**  **{**  **cout<<"Tree is empty"<<endl;**  **return;**  **}**  **if (ptr != NULL)**  **{**  **cout<<ptr->info<<" ";**  **preorder(ptr->left);**  **preorder(ptr->right);**  **}**  **}**  **void BST::inorder(node \*ptr)**  **{**  **if (root == NULL)**  **{**  **cout<<"Tree is empty"<<endl;**  **return;**  **}**  **if (ptr != NULL)**  **{**  **inorder(ptr->left);**  **cout<<ptr->info<<" ";**  **inorder(ptr->right);**  **}**  **}**  **void BST::postorder(node \*ptr)**  **{**  **if (root == NULL)**  **{**  **cout<<"Tree is empty"<<endl;**  **return;**  **}**  **if (ptr != NULL)**  **{**  **postorder(ptr->left);**  **postorder(ptr->right);**  **cout<<ptr->info<<" ";**  **}**  **}**  **void BST::display(node \*ptr, int level)**  **{**  **int i;**  **if (ptr != NULL)**  **{**  **display(ptr->right, level+1);**  **cout<<endl;**  **if (ptr == root)**  **cout<<"Root->: ";**  **else**  **{**  **for (i = 0;i < level;i++)**  **cout<<" ";**  **}**  **cout<<ptr->info;**  **display(ptr->left, level+1);**  **}**  **}**  **int main()**  **{**  **int c, num;**  **BST bst;**  **node \*temp;**  **while (1)**  **{**  **cout<<"1.Insert Element "<<endl;**  **cout<<"2.Delete Element "<<endl;**  **cout<<"3.Inorder Traversal"<<endl;**  **cout<<"4.Preorder Traversal"<<endl;**  **cout<<"5.Postorder Traversal"<<endl;**  **cout<<"6.Quit"<<endl;**  **cout<<"Enter choice : ";**  **cin>>c;**  **switch(c)**  **{**  **case 1:**  **temp = new node;**  **cout<<"Enter the number to be inserted : ";**  **cin>>temp->info;**  **bst.insert(root, temp);**  **break;**  **case 2:**  **if (root == NULL)**  **{**  **cout<<"Tree is empty, nothing to delete"<<endl;**  **continue;**  **}**  **cout<<"Enter the number to be deleted : ";**  **cin>>num;**  **bst.del(num);**  **break;**  **case 3:**  **cout<<"Inorder Traversal of BST:"<<endl;**  **bst.inorder(root);**  **cout<<endl;**  **break;**  **case 4:**  **cout<<"Preorder Traversal of BST:"<<endl;**  **bst.preorder(root);**  **cout<<endl;**  **break;**  **case 5:**  **cout<<"Postorder Traversal of BST:"<<endl;**  **bst.postorder(root);**  **cout<<endl;**  **break;**  **case 6:**  **exit(1);**  **default:**  **cout<<"Wrong choice"<<endl;**  **}**  **}**  **}** |
| **Your whole Screenshot here: (Console Output):** |

|  |
| --- |
| 1. Implement a program using BST to store information of students with the following details such as id, name and CGPA. You can use unique id of a student to search and insert different data.   Note: Each node will contain three information: id, name and CGPA but id will be used for searching data in the BST |
| **Your code here:**  #include<iostream>  # include <cstdlib>  struct{  char studID[6];  char fname[30];  char lname[30];  char mname[3];  float qpa;  }  struct node \*nd;  struct node  {  STUD stu;  nd left;  nd rigth;  }  void inputStud(nd \*);  bool treeEmpty(nd \*);  void deleteNode(nd \*);  void editStud(nd \*);  void viewAll(nd);  nd searchNode(nd \*,nd \*,char[]);  int main(void)  {  int ans,c;  bool e;  nd root = NULL;  do  {  system("cls");  cout<<"1] Add New Student Record\n";  cout<<"2] Edit Student Record\n";  cout<<"3] View Specific Record\n";  cout<<"4] Display All\n";  cout<<"5] Delete Student Record\n";  cout<<"6] Exit\n");  cout<<"Pick your choice: ";  cin>>ans;  switch(ans)  {  case 1: temp = new node;  cin>>temp->info;  break;  case 2: editStud(&root);  break;  case 3: viewAll(root);  getch();  break;  case 4: viewAll(root);  getch();  break;  case 5: deleteNode(&root);  break;  case 6: cout<<"\n\nclosing the program in 5 seconds....";  }  }while (ans != 6);  return 0;  }  void inputStud(nd \*root)  {  nd t, t1, temp;  char sID[6];  char lnam[20];  char fnam[30];  char mnam[3];  float qp;  STUD st;  cout<<"Enter Student ID: /n";  gets(sID);  cout<<"Enter Student's Last Name: /n");  gets(lnam);  cout<<"Enter Student's First Name: /n");  gets(fnam);  cout<<"Enter Student's Middle Initial(s): /n");  gets(mnam);  cout<<"Enter Student's QPA: /n");  cin>>qp;  strcpy(st.studID, sID);  strcpy(st.lname, lnam);  strcpy(st.fname, fnam);  strcpy(st.mname, mnam);  st.qpa=qp;  temp = malloc(sizeof(NODE));  temp -> stu = st;  temp -> left = NULL;  temp -> rigth = NULL;  if (\*root == NULL)  {  \*root = temp;  }  else  {  t = \*root;  while (t != NULL)  {  t1 = t;  if (strcmp(temp -> stu.studID,t -> stu.studID)==0)  t = t -> left;  else  t = t -> rigth;  }  if (strcmp(temp -> stu.studID,t1 -> stu.studID) < 0)  t1 -> left = temp;  else  t1 -> rigth = temp;  }  t1 = NULL;  t = NULL;  temp = NULL;  return;  }  void editStud(nd \*root)  {  char sID[6];  char lnam[20];  char fnam[30];  char mnam[3];  float qp;  STUD st;  cout<<"Please enter the ID of the Student you would like to edit: ";  gets(sID); |
| **Your whole Screenshot here: (Console Output):** |