|  |
| --- |
|  |
| #include<stdio.h>  #include<stdlib.h>  struct Node  {  struct Node \*lchild;  int info;  struct Node \*rchild;  };  void insert();  void preorder(struct Node \*);  void inorder(struct Node \*);  void postorder(struct Node \*);  void search();  void smallest();  void largest();  void delete1();  void case\_A(struct Node \*, struct Node \*);  void case\_B(struct Node \*, struct Node \*);  void case\_C(struct Node \*, struct Node \*);  struct Node \*root = NULL;  void insert()  {  struct Node \*temp, \*ptr, \*par;  int item;  temp = (struct Node \*)malloc(sizeof(struct Node));  if (temp == NULL)  {  printf("\n Memory is not allocated\n");  }  else  {  printf("Enter value to insert: ");  scanf("%d", &item);  temp->info = item;  temp->lchild = NULL;  temp->rchild = NULL;  if (root == NULL)  {  root = temp;  }  else  {  ptr = root;  while (ptr != NULL)  {  par = ptr;  if (item < ptr->info)  {  ptr = ptr->lchild;  }  else if (item > ptr->info)  {  ptr = ptr->rchild;  }  else  {  printf("\nDuplicate element is not allowed\n");  break;  }  }  if (ptr == NULL)  {  if (item < par->info)  {  par->lchild = temp;  }  else  {  par->rchild = temp;  }  }  }  }  }  void delete1()  {  struct Node \*ptr, \*par;  int item;  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  ptr = root;  par = NULL;  printf("\nEnter value to delete: ");  scanf("%d", &item);  while (ptr != NULL)  {  if (ptr->info == item)  {  break;  }  par = ptr;  if (item < ptr->info)  {  ptr = ptr->lchild;  }  else  {  ptr = ptr->rchild;  }  }  if (ptr == NULL)  {  printf("\nNode not found\n");  }  else  {  printf("\nDeleted node = %d\n", ptr->info);  if ((ptr->lchild == NULL) && (ptr->rchild == NULL))  {  case\_A(par, ptr); // Node having zero child  }  else if ((ptr->lchild == NULL) || (ptr->rchild == NULL))  {  case\_B(par, ptr); // Node having one child  }  else  {  case\_C(par, ptr); // Node having both children  }  }  }  }  void case\_A(struct Node \*par, struct Node \*ptr)  {  if (par == NULL)  {  root = NULL;  }  else if (ptr == par->lchild)  {  par->lchild = NULL;  }  else  {  par->rchild = NULL;  }  free(ptr);  }  void case\_B(struct Node \*par, struct Node \*ptr)  {  struct Node \*child;  if (ptr->lchild != NULL)  {  child = ptr->lchild;  }  else  {  child = ptr->rchild;  }  if (par == NULL)  {  root = child;  }  else if (ptr == par->lchild)  {  par->lchild = child;  }  else  {  par->rchild = child;  }  free(ptr);  }  void case\_C(struct Node \*par, struct Node \*ptr)  {  struct Node \*succ, \*parsucc;  parsucc = ptr;  succ = ptr->rchild;  while (succ->lchild != NULL)  {  parsucc = succ;  succ = succ->lchild;  }  ptr->info = succ->info;  if (succ->lchild == NULL && succ->rchild == NULL)  {  case\_A(parsucc, succ);  }  else  {  case\_B(parsucc, succ);  }  }  void preorder(struct Node \*temp)  {  if (temp != NULL)  {  printf("%d ", temp->info);  preorder(temp->lchild);  preorder(temp->rchild);  }  }  void postorder(struct Node \*temp)  {  if (temp != NULL)  {  postorder(temp->lchild);  postorder(temp->rchild);  printf("%d ", temp->info);  }  }  void inorder(struct Node \*temp)  {  if (temp != NULL)  {  inorder(temp->lchild);  printf("%d ", temp->info);  inorder(temp->rchild);  }  }  void smallest()  {  struct Node \*temp;  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  temp = root;  while (temp->lchild != NULL)  {  temp = temp->lchild;  }  printf("\nSmallest element of the tree is %d\n", temp->info);  }  }  void largest()  {  struct Node \*temp;  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  temp = root;  while (temp->rchild != NULL)  {  temp = temp->rchild;  }  printf("\nLargest element of the tree is %d\n", temp->info);  }  }  void search()  {  int item;  struct Node \*temp;  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  printf("\nEnter the value to search: ");  scanf("%d", &item);  temp = root;  while (temp != NULL)  {  if (temp->info == item)  {  printf("\nSearched item is present\n");  return;  }  else if (item < temp->info)  {  temp = temp->lchild;  }  else  {  temp = temp->rchild;  }  }  printf("\nSearched item is not present\n");  }  }  int main()  {  int choice;  while (1)  {  printf("\n1. Insert\n");  printf("2. Preorder\n");  printf("3. Inorder\n");  printf("4. Postorder\n");  printf("5. Smallest Node\n");  printf("6. Largest Node\n");  printf("7. Delete\n");  printf("8. Search an element\n");  printf("9. Exit\n");  printf("\nEnter your choice: ");  scanf("%d", &choice);  switch (choice)  {  case 1:  insert();  break;  case 2:  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  printf("\nPreorder traversal: ");  preorder(root);  }  break;  case 3:  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  printf("\nInorder traversal: ");  inorder(root);  }  break;  case 4:  if (root == NULL)  {  printf("\nTree is empty\n");  }  else  {  printf("\nPostorder traversal: ");  postorder(root);  }  break;  case 5:  smallest();  break;  case 6:  largest();  break;  case 7:  delete1();  break;  case 8:  search();  break;  case 9:  exit(0);  default:  printf("\nWrong Choice\n");  }  }  return 0;  } |  |