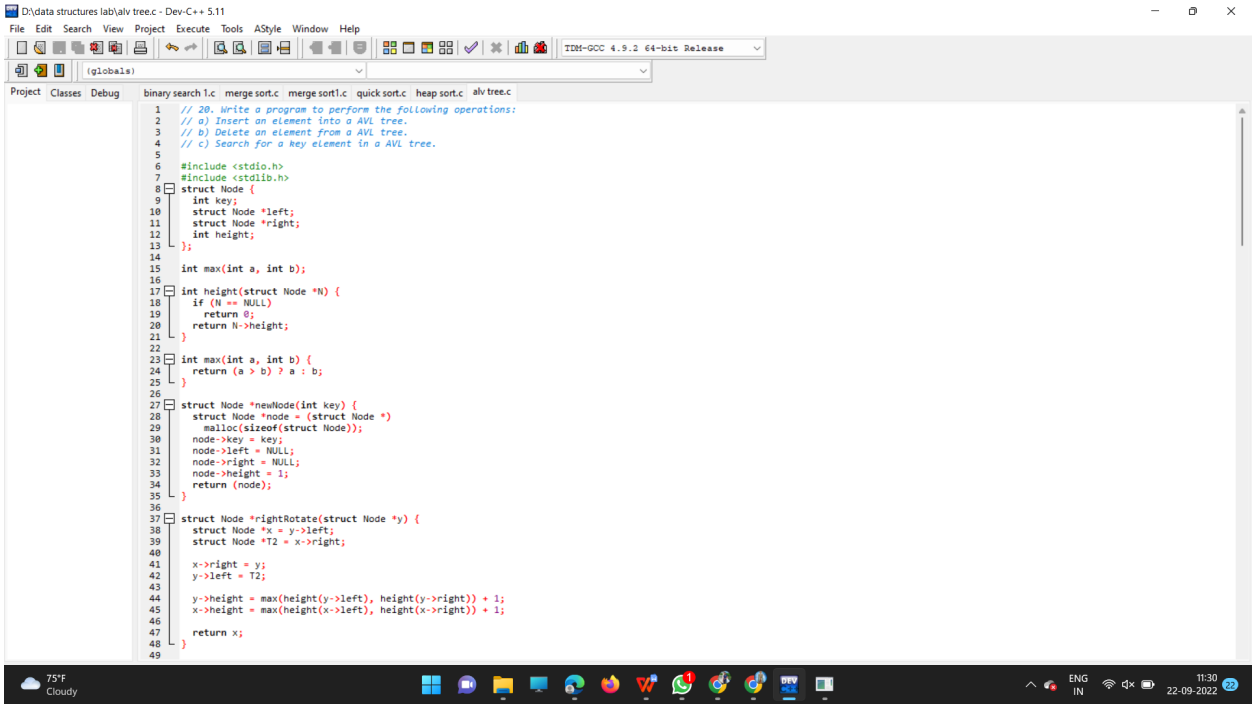
```
// 20. Write a program to perform the following operations:
// a) Insert an element into a AVL tree.
// b) Delete an element from a AVL tree.
// c) Search for a key element in a AVL tree.

#include <stdio.h>
#include <stdlib.h>
struct Node {
  int key;
  struct Node *left;
  struct Node *right;
  int height;
};

int max(int a, int b);

int height(struct Node *N) {
  if (N == NULL)
    return 0;
  return N->height;
}

int max(int a, int b) {
  return (a > b) ? a : b;
}

struct Node *newNode(int key) {
  struct Node *node = (struct Node *)
    malloc(sizeof(struct Node));
  node->key = key;
  node->left = NULL;
  node->right = NULL;
  node->height = 1;
  return (node);
}

struct Node *rightRotate(struct Node *y) {
  struct Node *x = y->left;
  struct Node *T2 = x->right;

  x->right = y;
  y->left = T2;

  y->height = max(height(y->left), height(y->right)) + 1;
```

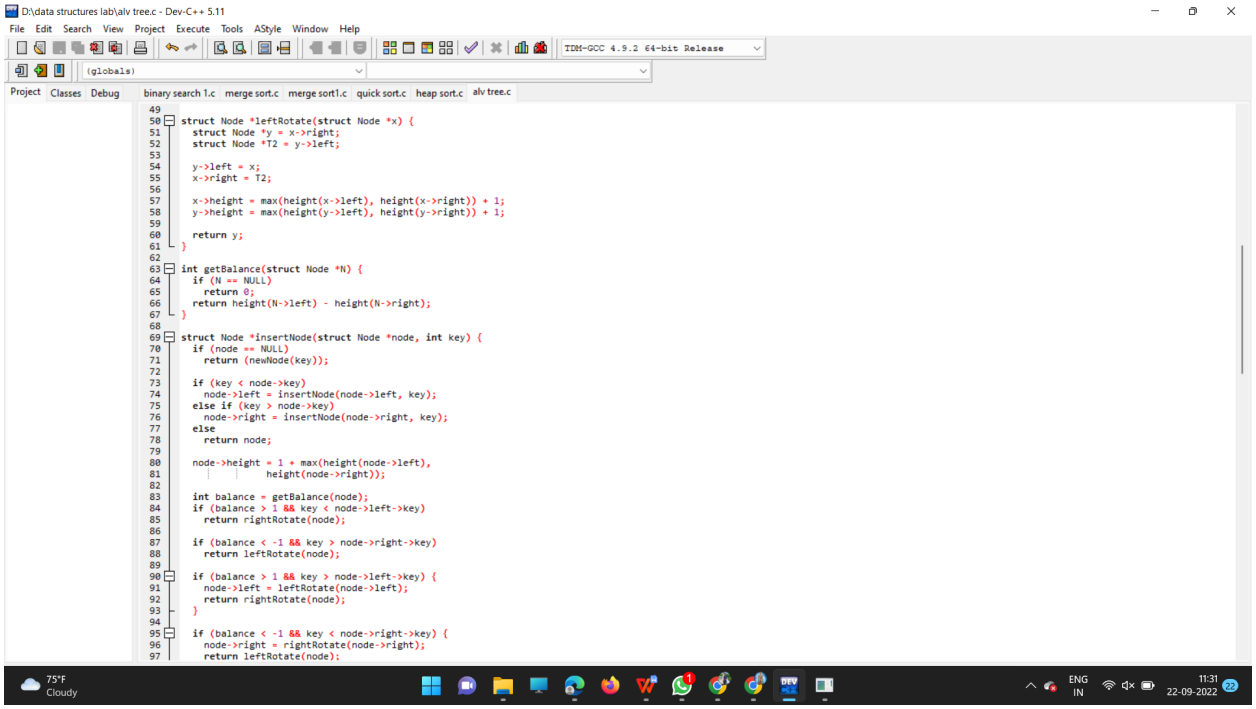
```
  x->height = max(height(x->left), height(x->right)) + 1;

  return x;
}

struct Node *leftRotate(struct Node *x) {
  struct Node *y = x->right;
  struct Node *T2 = y->left;

  y->left = x;
  x->right = T2;

  x->height = max(height(x->left), height(x->right)) + 1;
  y->height = max(height(y->left), height(y->right)) + 1;

  return y;
}

int getBalance(struct Node *N) {
  if (N == NULL)
    return 0;
  return height(N->left) - height(N->right);
}

struct Node *insertNode(struct Node *node, int key) {
  if (node == NULL)
    return (newNode(key));

  if (key < node->key)
    node->left = insertNode(node->left, key);
  else if (key > node->key)
    node->right = insertNode(node->right, key);
  else
    return node;

  node->height = 1 + max(height(node->left),
               height(node->right));

  int balance = getBalance(node);
  if (balance > 1 && key < node->left->key)
    return rightRotate(node);

  if (balance < -1 && key > node->right->key)
    return leftRotate(node);
```

```
  if (balance > 1 && key > node->left->key) {
    node->left = leftRotate(node->left);
    return rightRotate(node);
  }

  if (balance < -1 && key < node->right->key) {
    node->right = rightRotate(node->right);
    return leftRotate(node);
  }

  return node;
}

struct Node *minValueNode(struct Node *node) {
  struct Node *current = node;

  while (current->left != NULL)
    current = current->left;

  return current;
}


struct Node *deleteNode(struct Node *root, int key) {

  if (root == NULL)
    return root;

  if (key < root->key)
    root->left = deleteNode(root->left, key);

  else if (key > root->key)
    root->right = deleteNode(root->right, key);

  else {
    if ((root->left == NULL) || (root->right == NULL)) {
      struct Node *temp = root->left ? root->left : root->right;

      if (temp == NULL) {
        temp = root;
        root = NULL;
      } else
        *root = *temp;
```

```
      free(temp);
    } else {
      struct Node *temp = minValueNode(root->right);

      root->key = temp->key;

      root->right = deleteNode(root->right, temp->key);
    }
  }

  if (root == NULL)
    return root;

  root->height = 1 + max(height(root->left),
              height(root->right));

  int balance = getBalance(root);
  if (balance > 1 && getBalance(root->left) >= 0)
    return rightRotate(root);

  if (balance > 1 && getBalance(root->left) < 0) {
    root->left = leftRotate(root->left);
    return rightRotate(root);
  }

  if (balance < -1 && getBalance(root->right) <= 0)
    return leftRotate(root);

  if (balance < -1 && getBalance(root->right) > 0) {
    root->right = rightRotate(root->right);
    return leftRotate(root);
  }

  return root;
}


void printPreOrder(struct Node *root) {
  if (root != NULL) {
    printf("%d ", root->key);
    printPreOrder(root->left);
    printPreOrder(root->right);
  }
}
```

```
int main() {
  struct Node *root = NULL;

  root = insertNode(root, 2);
  root = insertNode(root, 1);
  root = insertNode(root, 7);
  root = insertNode(root, 4);
  root = insertNode(root, 5);
  root = insertNode(root, 3);
  root = insertNode(root, 8);

  printPreOrder(root);

  root = deleteNode(root, 3);

  printf("\nAfter deletion: ");
  printPreOrder(root);

  return 0;
}
```


D:\data structures lab\alv tree.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
binary search 1.c merge sort.c merge sort1.c quick sort.c heap sort.c alv tree.c
// 20. Write a program to perform the following operations:
// a) Insert an element into a AVL tree.
// b) Delete an element from a AVL tree.
// c) Search for a key element in a AVL tree.
#include <stdio.h>
#include <stdlib.h>
struct Node {
int key;
struct Node *left;
struct Node *right;
int height;
};
int max(int a, int b);
int height(struct Node *N) {
if (N == NULL)
return 0;
return N->height;
}
int max(int a, int b) {
return (a > b) ? a : b;
}
struct Node *newNode(int key) {
struct Node *node = (struct Node *)
malloc(sizeof(struct Node));
node->key = key;
node->left = NULL;
node->right = NULL;
node->height = 1;
return (node);
}
struct Node *rightRotate(struct Node *y) {
struct Node *x = y->left;
struct Node *T2 = x->right;
x->right = y;
y->left = T2;
y->height = max(height(y->left), height(y->right)) + 1;
x->height = max(height(x->left), height(x->right)) + 1;
return x;
}
75°F
Cloudy
ENG
IN
11:30
22-09-2022


D:\data structures lab\alv tree.c - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
(globals)
Project Classes Debug
binary search 1.c merge sort.c merge sort1.c quick sort.c heap sort.c alv tree.c
struct Node *leftRotate(struct Node *x) {
struct Node *y = x->right;
struct Node *T2 = y->left;
y->left = x;
x->right = T2;
x->height = max(height(x->left), height(x->right)) + 1;
y->height = max(height(y->left), height(y->right)) + 1;
return y;
}
int getBalance(struct Node *N) {
if (N == NULL)
return 0;
return height(N->left) - height(N->right);
}
struct Node *insertNode(struct Node *node, int key) {
if (node == NULL)
return (newNode(key));
if (key < node->key)
node->left = insertNode(node->left, key);
else if (key > node->key)
node->right = insertNode(node->right, key);
else
return node;
node->height = 1 + max(height(node->left),
height(node->right));
int balance = getBalance(node);
if (balance > 1 && key < node->left->key)
return rightRotate(node);
if (balance < -1 && key > node->right->key)
return leftRotate(node);
if (balance > 1 && key > node->left->key) {
node->left = leftRotate(node->left);
return rightRotate(node);
}
if (balance < -1 && key < node->right->key) {
node->right = rightRotate(node->right);
return leftRotate(node);
75°F
Cloudy
ENG
IN
11:31
22-09-2022

D:\data structures lab\alv tree.c - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug

binary search 1.c | merge sort.c | merge sort1.c | quick sort.c | heap sort.c | alv tree.c

```
97      return leftRotate(node);
98    }
99
100   return node;
101 }
102
103 struct Node *minValueNode(struct Node *node) {
104   struct Node *current = node;
105
106   while (current->left != NULL)
107     current = current->left;
108
109   return current;
110 }
111
112
113 struct Node *deleteNode(struct Node *root, int key) {
114
115   if (root == NULL)
116     return root;
117
118   if (key < root->key)
119     root->left = deleteNode(root->left, key);
120
121   else if (key > root->key)
122     root->right = deleteNode(root->right, key);
123
124   else {
125     if ((root->left == NULL) || (root->right == NULL)) {
126       struct Node *temp = root->left ? root->left : root->right;
127
128       if (temp == NULL) {
129         temp = root;
130         root = NULL;
131       } else
132         *root = *temp;
133       free(temp);
134     } else {
135       struct Node *temp = minValueNode(root->right);
136
137       root->key = temp->key;
138
139       root->right = deleteNode(root->right, temp->key);
140     }
141   }
142
143   if (root == NULL)
144     return root;
145
```

75°F Cloudy

ENG IN 11:31 22-09-2022

D:\data structures lab\alv tree.c - [Executing] - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 64-bit Release

(globals)

Project Classes Debug

binary search 1.c | merge sort.c | merge sort1.c | quick sort.c | heap sort.c | alv tree.c

```
149   int balance = getBalance(root);
150   if (balance > 1 && getBalance(root->left) >= 0)
151     return rightRotate(root);
152
153   if (balance > 1 && getBalance(root->left) < 0) {
154     root->left = leftRotate(root->left);
155     return rightRotate(root);
156   }
157
158   if (balance < -1 && getBalance(root->right) <= 0)
159     return leftRotate(root);
160
161   if (balance < -1 && getBalance(root->right) > 0) {
162     root->right = rightRotate(root->right);
163     return leftRotate(root);
164   }
165
166   return root;
167 }
168
169
170 void printPreOrder(struct Node *root) {
171   if (root != NULL) {
172     printf("%d ", root->key);
173     printPreOrder(root->left);
174     printPreOrder(root->right);
175   }
176 }
177
178 int main() {
179   struct Node *root = NULL;
180
181   root = insertNode(root, 2);
182   root = insertNode(root, 1);
183   root = insertNode(root, 7);
184   root = insertNode(root, 4);
185   root = insertNode(root, 5);
186   root = insertNode(root, 3);
187   root = insertNode(root, 8);
188
189   printPreOrder(root);
190
191   root = deleteNode(root, 3);
192
193   printf("\nAfter deletion: ");
194   printPreOrder(root);
195
196   return 0;
197 }
```

D:\data structures lab\alv tree.exe

```
4 2 1 3 7 5 8
After deletion: 4 2 1 7 5 8
--------------------------------
Process exited after 3.175 seconds with return value 0
Press any key to continue . . .
```

75°F Cloudy

ENG IN 11:32 22-09-2022