#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define size 100000

struct data

{

char player\_name[30];

long int player\_id;

int kills;

int deaths;

float KD;

int assist;

int reports;

};

typedef struct node

{

struct node\* left;

struct data d;

struct node\* right;

}\*NODE;

NODE newNODE(struct data d);

void display\_preorder(NODE root);

void display\_inorder(NODE root);

void display\_postorder(struct node \*root);

void linkid(struct node\*\* root, struct data f);

void linktree(struct node\*\* root, struct data f);

NODE findMinNode(NODE node);

NODE deleteNode(struct node\* root,int report);

NODE rightRotate(NODE root);

NODE leftRotate(NODE root);

NODE leftrightRotate(NODE root);

NODE rightleftRotate(NODE root);

int getHeight(NODE root);

NODE balanceTree(NODE root);

void inorder\_LOP(NODE root,FILE \*fp,int a);

void main()

{

int n;

struct data f;

int reports;

long int id;

FILE \*fp;

fp=fopen("input.txt","r");

printf("enter the number of players:");

scanf("%d",&n);

NODE root\_reports=NULL;

NODE root\_id=NULL;

for(int i=0;i<n;i++)

{

fscanf(fp,"%s",f.player\_name);

fscanf(fp,"%ld",&f.player\_id);

fscanf(fp,"%d",&f.kills);

fscanf(fp,"%d",&f.deaths);

fscanf(fp,"%d",&f.assist);

fscanf(fp,"%d",&f.reports);

linktree(&root\_reports,f);

}

fclose(fp);

display\_inorder(root\_reports);

display\_inorder(root\_id);

root\_reports=balanceTree(root\_reports);

//lsit players who must be banned

FILE \*fp1;

int d;

fp1=fopen("bannedplayers.txt","w");

inorder\_LOP(root\_reports,fp1,100);

fclose(fp1);

printf("enter the number of players to be banned:");

scanf("%d",&d);

for(int i=0;i<d;i++)

{

printf("enter the reports:");

scanf("%d",&reports);

root\_reports=deleteNode(root\_reports,reports);

}

root\_reports=balanceTree(root\_reports);

display\_inorder(root\_reports);

}

NODE newNODE(struct data d)

{

NODE node=(struct node\*)malloc(sizeof(struct node));

node->d=d;

node->d.KD=(float)d.kills/d.deaths;

node->left=NULL;

node->right=NULL;

return node;

}

void display\_preorder(NODE root)

{

if(root)

{

printf("\nNAME:%s\nPLAYER ID:%ld\nDEATHS:%d\nKILLS:%d\nK/D:%.2f\nASSISTS:%d\nREPORTS:%d\n",root->d.player\_name,root->d.player\_id,root->d.deaths,root->d.kills,root->d.KD,root->d.assist,root->d.reports);

display\_preorder(root->left);

display\_preorder(root->right);

}

}

void display\_inorder(NODE root)

{

if(root)

{

display\_inorder(root->left);

printf("\nNAME:%s\nPLAYER ID:%ld\nDEATHS:%d\nKILLS:%d\nK/D:%.2f\nASSISTS:%d\nREPORTS:%d\n",root->d.player\_name,root->d.player\_id,root->d.deaths,root->d.kills,root->d.KD,root->d.assist,root->d.reports);

display\_inorder(root->right);

}

}

void display\_postorder(struct node \*root)

{

if(root)

{

display\_postorder(root->left);

display\_postorder(root->right);

printf("\nNAME:%s\nPLAYER ID:%ld\nDEATHS:%d\nKILLS:%d\nK/D:%.2f\nASSISTS:%d\nREPORTS:%d\n",root->d.player\_name,root->d.player\_id,root->d.deaths,root->d.kills,root->d.KD,root->d.assist,root->d.reports); }

}

void linktree(struct node\*\* root, struct data f)

{

if (\*root == NULL)

{

\*root = newNODE(f);

}

else

{

if (f.reports > (\*root)->d.reports)

{

linktree(&((\*root)->right),f);

}

else

{

linktree(&((\*root)->left),f);

}

}

}

NODE findMinNode(NODE node)

{

struct node\* current = node;

while (current && current->left != NULL) {

current = current->left;

}

return current;

}

NODE deleteNode(struct node\* root,int report)

{

if (root == NULL)

{

printf("Key element is not found\n");

return root;

}

if (report< root->d.reports)

{

root->left = deleteNode(root->left,report);

} else if (report > root->d.reports) {

root->right = deleteNode(root->right,report);

} else {

// Node to be deleted found

// Node with only one child or no child

if (root->left == NULL) {

struct node\* temp = root->right;

free(root);

return temp;

} else if (root->right == NULL) {

struct node\* temp = root->left;

free(root);

return temp;

}

// Node with two children: Replace with in-order successor (smallest value in the right subtree)

struct node\* temp = findMinNode(root->right);

root->d= temp->d;

root->right = deleteNode(root->right, temp->d.KD);

}

return root;

}

NODE rightRotate(NODE root)

{

NODE newRoot = root->left;

root->left = newRoot->right;

newRoot->right = root;

return newRoot;

}

// Function to perform a left rotation on a binary tree

NODE leftRotate(NODE root)

{

NODE newRoot = root->right;

root->right = newRoot->left;

newRoot->left = root;

return newRoot;

}

NODE leftrightRotate(NODE root)

{

root=leftRotate(root);

root=rightRotate(root);

return root;

}

NODE rightleftRotate(NODE root)

{

root=rightRotate(root);

root=leftRotate(root);

return root;

}

int getHeight(NODE root)

{

if (root == NULL)

return 0;

int leftHeight = getHeight(root->left);

int rightHeight = getHeight(root->right);

return (leftHeight > rightHeight ? leftHeight : rightHeight) + 1;

}

NODE balanceTree(NODE root)

{

int heightDiff = getHeight(root->left) - getHeight(root->right);

if (heightDiff > 1) {

if (getHeight(root->left->left) >= getHeight(root->left->right))

root = rightRotate(root);

else {

root->left = leftRotate(root->left);

root = rightRotate(root);

}

}

else if (heightDiff < -1) {

if (getHeight(root->right->right) >= getHeight(root->right->left))

root = leftRotate(root);

else {

root->right = rightRotate(root->right);

root = leftRotate(root);

}

}

return root;

}

void inorder\_LOP(NODE root,FILE \*fp,int a)

{

if(root)

{

inorder\_LOP(root->left,fp,a);

if(root->d.reports>a)

{

fprintf(fp,"%s\n",root->d.player\_name);

fprintf(fp,"%ld\n",root->d.player\_id);

fprintf(fp,"%d\n",root->d.reports);

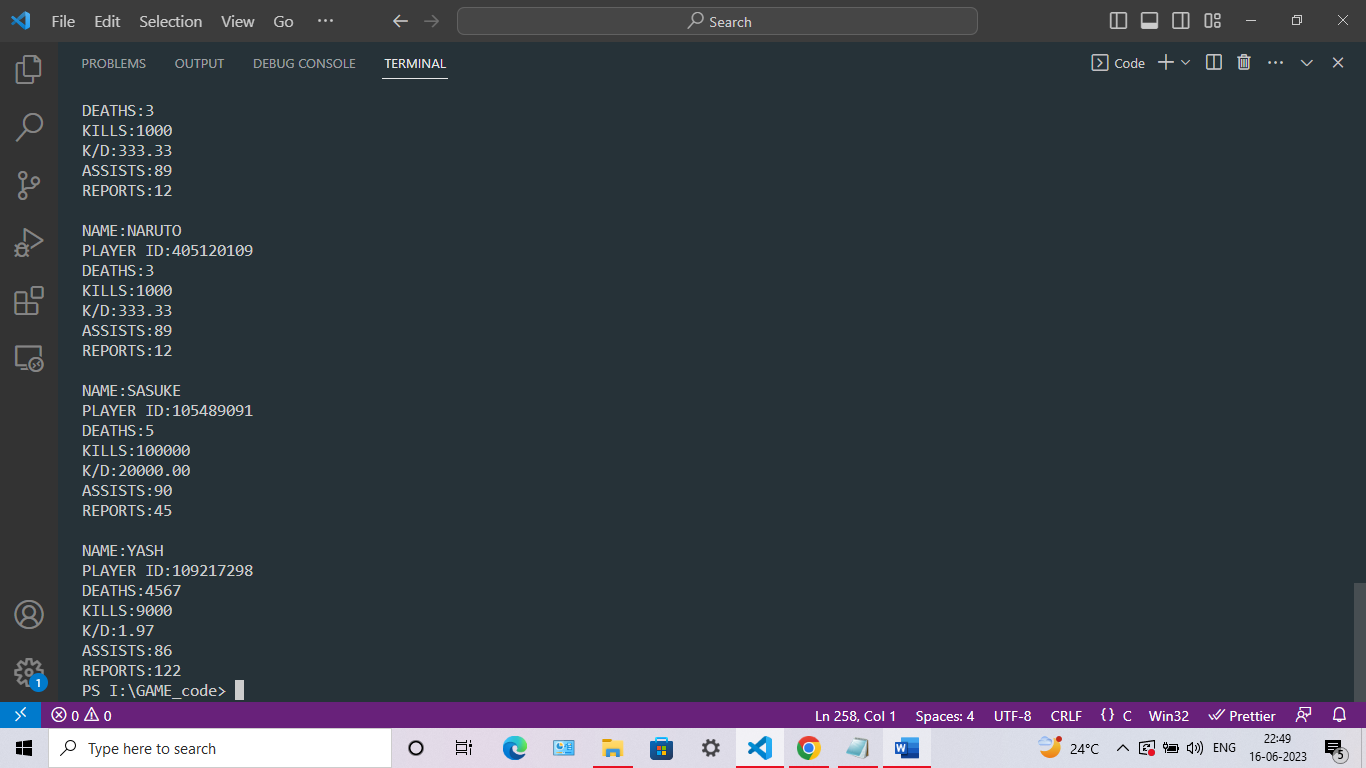
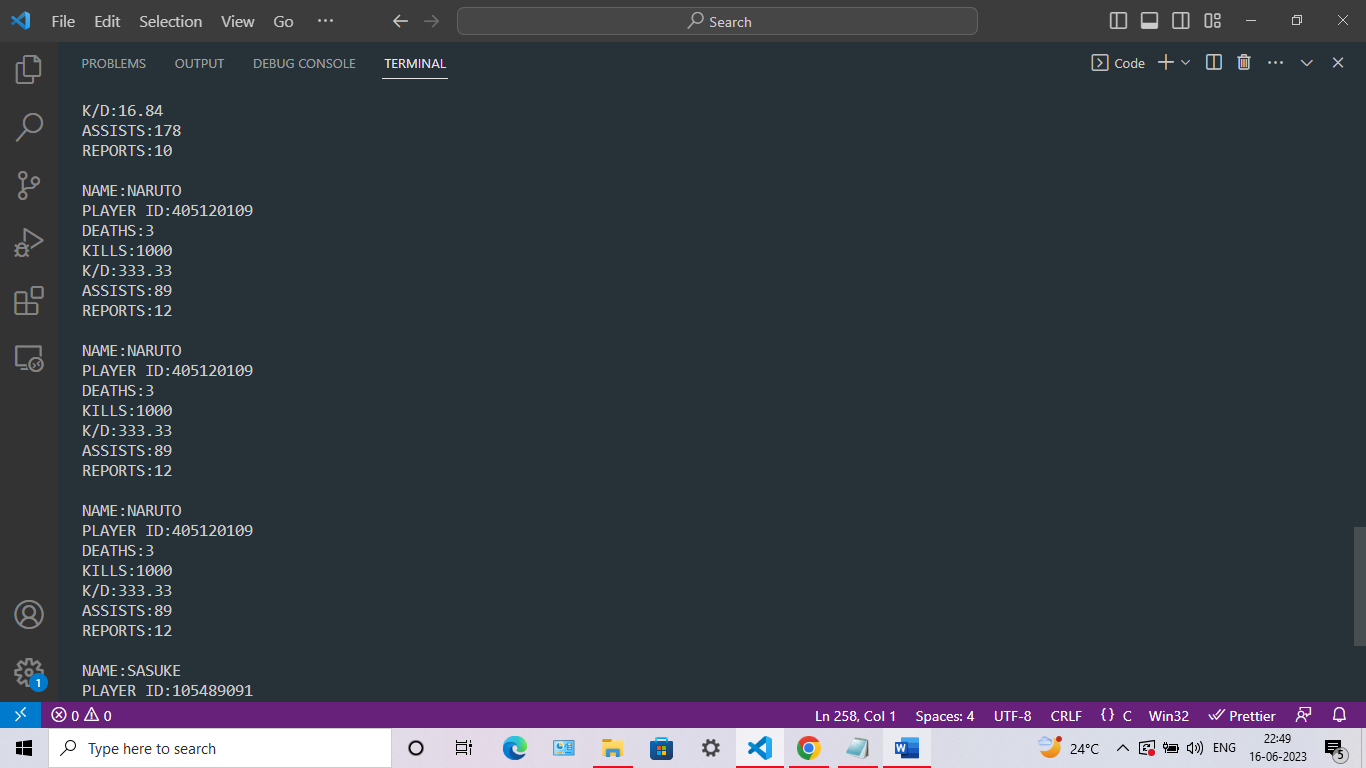
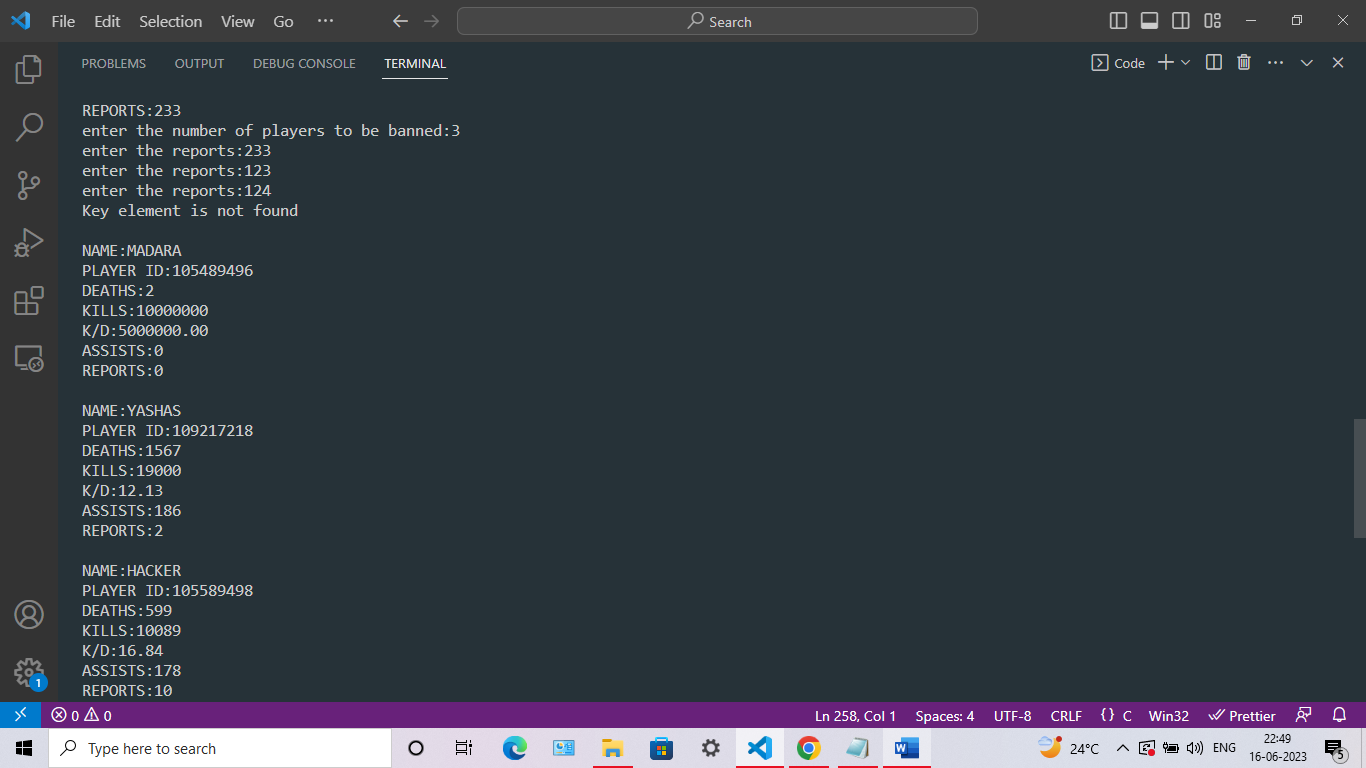
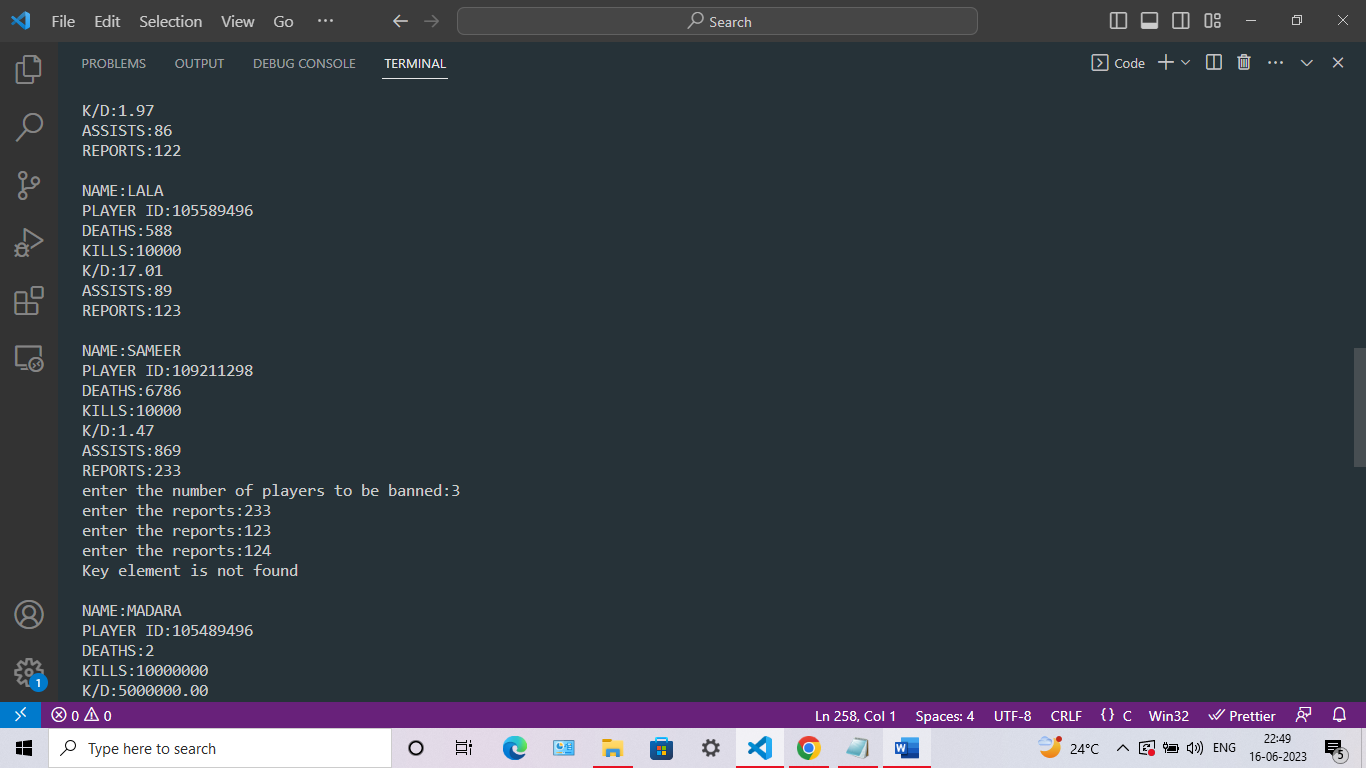
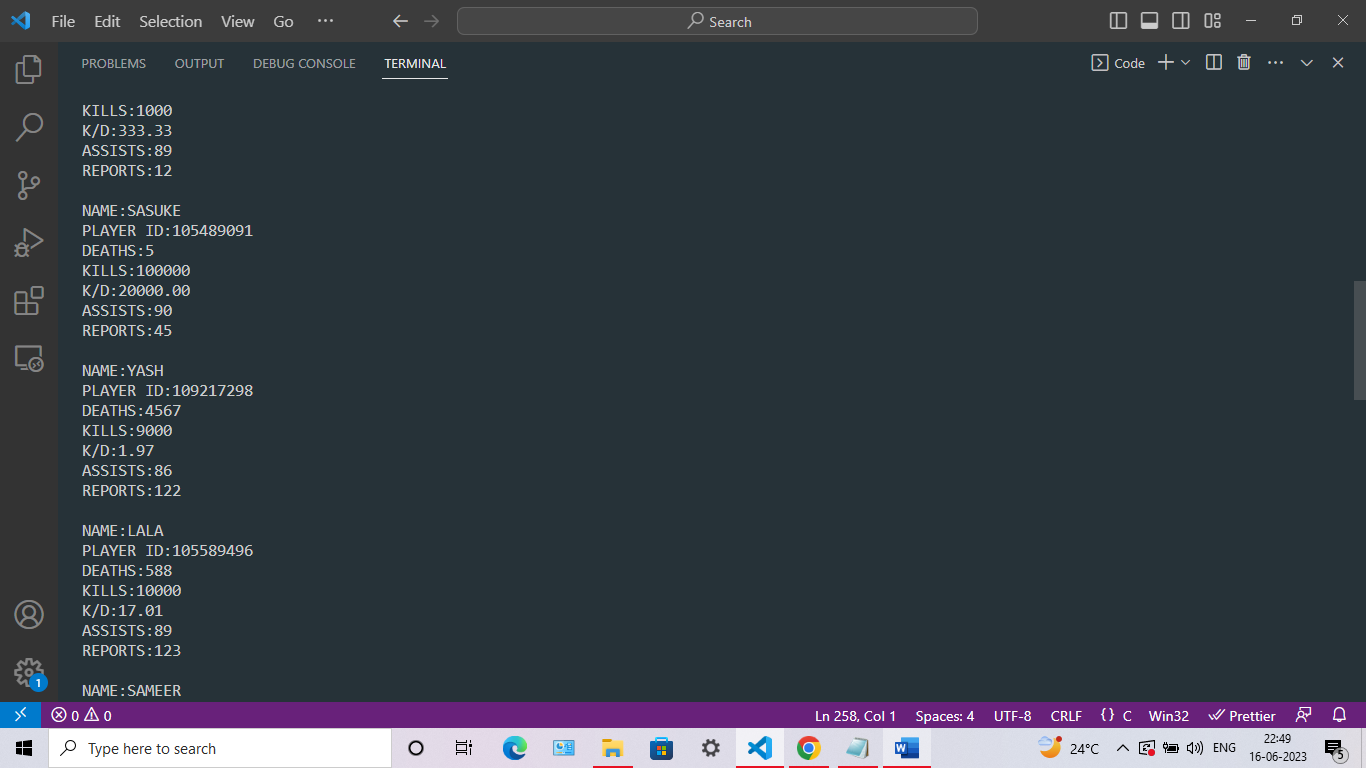
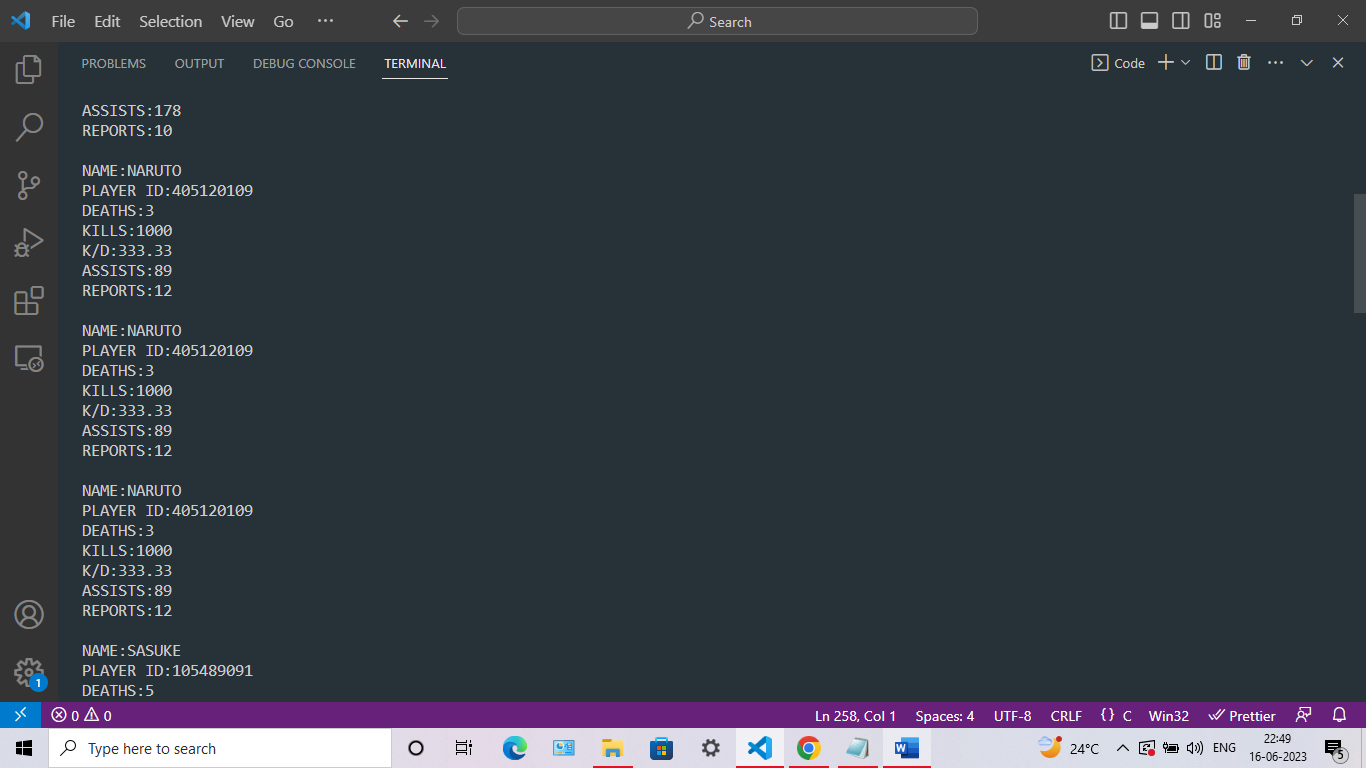
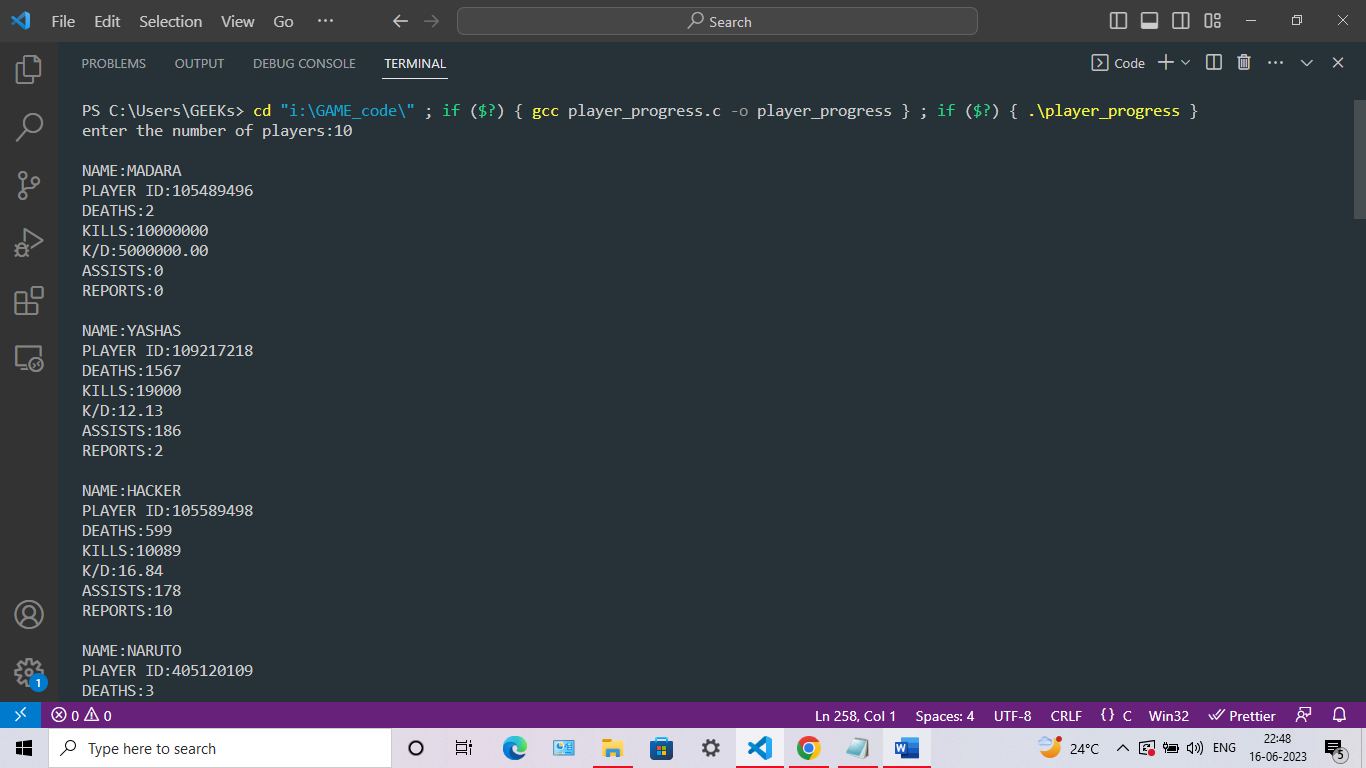
fprintf(fp,"\n");

}

inorder\_LOP(root->right,fp,a);

}

}



OUTPUT OF PLAYERS TO BE BANNED

