```
#include <stdio.h>
#include <iostream>
using namespace std;

struct student
{
	int cod;
	char* nume;
	float medie;
};

struct nodarb
{
	int BF;
	student inf;
	nodarb* left, * right;
};

nodarb* creare(student s, nodarb* st, nodarb* dr)
{
	nodarb* nou = (nodarb*)malloc(sizeof(nodarb));
	nou->inf.cod = s.cod;
	nou->inf.nume = (char*)malloc((strlen(s.nume) + 1) *
sizeof(char));
	strcpy(nou->inf.nume, s.nume);
	nou->inf.medie = s.medie;
	nou->left = st;
	nou->right = dr;
	return nou;
}

nodarb* inserare(student s, nodarb* rad)
{
	nodarb* aux = rad;
	if (rad == NULL)
	{
		aux = creare(s, NULL, NULL);
		return aux;
	}
	else
		while (true)
		{
			if (s.cod < aux->inf.cod)
				if (aux->left != NULL)
```

```
                        aux = aux->left;
                    else
                    {
                        aux->left = creare(s, NULL, NULL);
                        return rad;
                    }
                else
                    if (s.cod > aux->inf.cod)
                        if (aux->right != NULL)
                            aux = aux->right;
                        else
                        {
                            aux->right = creare(s, NULL, NULL);
                            return rad;
                        }
                    else
                        return rad;
            }
}

void preordine(nodarb* rad)
{
    if (rad != NULL)
    {
        printf("\nCod=%d, Nume=%s, Medie=%5.2f, BF=%d", rad-
>inf.cod, rad->inf.nume, rad->inf.medie, rad->BF);
        preordine(rad->left);
        preordine(rad->right);
    }
}

void inordine(nodarb* rad)
{
    if (rad != NULL)
    {
        inordine(rad->left);
        printf("\nCod=%d, Nume=%s, Medie=%5.2f, BF=%d", rad-
>inf.cod, rad->inf.nume, rad->inf.medie, rad->BF);
        inordine(rad->right);
    }
}

void postordine(nodarb* rad)
{
    if (rad != NULL)
    {
```

```
            postordine(rad->left);
            postordine(rad->right);
            printf("\nCod=%d, Nume=%s, Medie=%5.2f, BF=%d", rad-
>inf.cod, rad->inf.nume, rad->inf.medie, rad->BF);
      }
}

void dezalocare(nodarb* rad)
{
      if (rad != NULL)
      {
            nodarb* st = rad->left;
            nodarb* dr = rad->right;
            free(rad->inf.nume);
            free(rad);
            dezalocare(st);
            dezalocare(dr);
      }
}

nodarb* cautare(nodarb* rad, int cheie)
{
      if (rad != NULL)
      {
            if (cheie == rad->inf.cod)
                  return rad;
            else
                  if (cheie < rad->inf.cod)
                        return cautare(rad->left, cheie);
                  else
                        return cautare(rad->right, cheie);
      }
      else
            return NULL;
}

int maxim(int a, int b)
{
      int max = a;
      if (max < b)
            max = b;
      return max;
}

int nrNiveluri(nodarb* rad)
{
```

```
	if (rad != NULL)
		return 1 + maxim(nrNiveluri(rad->left), nrNiveluri(rad-
>right));
	else
		return 0;
}

void conversieArboreVector(nodarb* rad, student* vect, int* nr)
{
	if (rad != NULL)
	{
		/*vect[*nr] = rad->inf;
		(*nr)++;
		conversieArboreVector(rad->left, vect, nr);
		conversieArboreVector(rad->right, vect, nr);*/

		vect[*nr].cod = rad->inf.cod;
		vect[*nr].nume = (char*)malloc((strlen(rad->inf.nume) + 1) *
sizeof(char));
		strcpy(vect[*nr].nume, rad->inf.nume);
		vect[*nr].medie = rad->inf.medie;
		(*nr)++;

		nodarb* st = rad->left;
		nodarb* dr = rad->right;
		free(rad->inf.nume);
		free(rad);
		conversieArboreVector(st, vect, nr);
		conversieArboreVector(dr, vect, nr);
	}
}

/*5 0
3 0		7 1
1 0	4 0		10 0
	3 2
	1 0		4 2
				7 1
					10 0
		7
   3		10
1		4*/


nodarb* stergeRad(nodarb* rad)
{
```

```
	nodarb* aux = rad;
	if (aux->left != NULL)
	{
		rad = aux->left;
		if (aux->right != NULL)
		{
			nodarb* temp = aux->left;
			while (temp->right)
				temp = temp->right;
			temp->right = aux->right;
		}
	}
	else
		if (aux->right != NULL)
			rad = aux->right;
		else
			rad = NULL;
	free(aux->inf.nume);
	free(aux);
	return rad;
}

nodarb* stergeNod(nodarb* rad, int cheie)
{
	if (rad == NULL)
		return NULL;
	else
		if (rad->inf.cod == cheie)
		{
			rad = stergeRad(rad);
			return rad;
		}
		else
		{
			nodarb* aux = rad;
			while (true)
			{
				if (cheie < aux->inf.cod)
					if (aux->left == NULL)
						break;
					else
						if (aux->left->inf.cod == cheie)
							aux->left = stergeRad(aux-
>left);
						else
							aux = aux->left;
```

```
                        else
                              if (cheie > aux->inf.cod)
                                    if (aux->right == NULL)
                                          break;
                                    else
                                          if (aux->right->inf.cod ==
cheie)
                                                aux->right = stergeRad(aux-
>right);
                                          else
                                                aux = aux->right;

                        }
                        return rad;
                  }
}

void calculBF(nodarb* rad)
{
      if (rad != NULL)
      {
            rad->BF = nrNiveluri(rad->right) - nrNiveluri(rad->left);
            calculBF(rad->left);
            calculBF(rad->right);
      }
}

nodarb* rotatie_dreapta(nodarb* rad)
{
      printf("\nRotatie dreapta\n");
      nodarb* nod1 = rad->left;
      rad->left = nod1->right;
      nod1->right = rad;
      rad = nod1;
      return rad;
}

nodarb* rotatie_stanga(nodarb* rad)
{
      printf("\nRotatie stanga\n");
      nodarb* nod1 = rad->right;
      rad->right = nod1->left;
      nod1->left = rad;
      rad = nod1;
      return rad;
}
```

```
nodarb* rotatie_dreapta_stanga(nodarb* rad)
{
	printf("\nRotatie dreapta-stanga\n");
	nodarb* nod1 = rad->right;
	nodarb* nod2 = nod1->left;
	nod1->left = nod2->right;
	nod2->right = nod1;
	rad->right = nod2->left;
	nod2->left = rad;
	rad = nod2;
	return rad;
}

nodarb* rotatie_stanga_dreapta(nodarb* rad)
{
	printf("\nRotatie stanga-dreapta\n");
	nodarb* nod1 = rad->left;
	nodarb* nod2 = nod1->right;
	nod1->right = nod2->left;
	nod2->left = nod1;
	rad->left = nod2->right;
	nod2->right = rad;
	rad = nod2;
	return rad;
}

nodarb* reechilibrare(nodarb* rad)
{
	calculBF(rad);
	if (rad->BF <= -2 && rad->left->BF <= -1)
	{
		rad = rotatie_dreapta(rad);
		calculBF(rad);
	}
	else
		if (rad->BF >= 2 && rad->right->BF >= 1)
		{
			rad = rotatie_stanga(rad);
			calculBF(rad);
		}
		else
			if (rad->BF >= 2 && rad->right->BF <= -1)
			{
				rad = rotatie_dreapta_stanga(rad);
				calculBF(rad);
```

```
                }
                else
                        if (rad->BF <= -2 && rad->left->BF >= 1)
                        {
                                rad = rotatie_stanga_dreapta(rad);
                                calculBF(rad);
                        }
        return rad;
}

void main()
{
        int n;

        FILE* f = fopen("fisier.txt", "r");

        //printf("Nr. studenti: ");
        fscanf(f, "%d", &n);

        nodarb* rad = NULL;
        student s;
        char buffer[20];

        for (int i = 0; i < n; i++)
        {
                //printf("\nCod: ");
                fscanf(f, "%d", &s.cod);
                //printf("\nNume: ");
                fscanf(f, "%s", buffer);
                s.nume = (char*)malloc((strlen(buffer) + 1) * sizeof(char));
                strcpy(s.nume, buffer);
                //printf("\nMedie: ");
                fscanf(f, "%f", &s.medie);

                rad = inserare(s, rad);

                rad = reechilibrare(rad);
        }
        fclose(f);

        //calculBF(rad);

        /*preordine(rad);
        printf("\n----------------\n");
        inordine(rad);
        printf("\n----------------\n");
```

```
	postordine(rad);*/

	/*nodarb *nodCautat = cautare(rad, 6);
	if(nodCautat!=NULL)
		printf("\nStudentul cautat se numeste %s", nodCautat-
>inf.nume);
	else
		printf("\nStudentul nu exista!");*/



		//printf("\n------Vector----------\n");

		/*student *vect = (student*)malloc(n*sizeof(student));
		int nr = 0;
		conversieArboreVector(rad, vect, &nr);
		for(int i=0;i<nr;i++)
			printf("\nCod=%d, Nume=%s, Medie=%5.2f", vect[i].cod,
vect[i].nume, vect[i].medie);
		for(int i=0;i<nr;i++)
			free(vect[i].nume);
		free(vect);*/

		//rad = stergeRad(rad);
		//rad = stergeNod(rad, 5);
		//calculBF(rad);

	inordine(rad);
	printf("\n----------------\n");
	inordine(rad->left);
	printf("\n----------------\n");
	inordine(rad->right);

	printf("\nInaltime arbore: %d", nrNiveluri(rad));
	printf("\nInaltime subarbore stang: %d", nrNiveluri(rad->left));
	printf("\nInaltime subarbore drept: %d", nrNiveluri(rad->right));

	rad = stergeNod(rad, 9);

	rad = reechilibrare(rad);

	inordine(rad);
	printf("\n----------------\n");
	inordine(rad->left);
	printf("\n----------------\n");
	inordine(rad->right);
```

```
printf("\nInaltime arbore: %d", nrNiveluri(rad));
printf("\nInaltime subarbore stang: %d", nrNiveluri(rad->left));
printf("\nInaltime subarbore drept: %d", nrNiveluri(rad->right));
```