HashTable -chaninng

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

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

struct nodLS
{
	student inf;
	nodLS* next;
};

struct hashT
{
	nodLS** vect;
	int size;
};

int functieHash(int cheie, hashT tabela)
{
	return cheie % tabela.size;
}

int inserare(hashT tabela, student s)
{
	int pozitie;
	if (tabela.vect != NULL)
	{
		pozitie = functieHash(s.cod, tabela);
		nodLS* nou = (nodLS*)malloc(sizeof(nodLS));
		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->next = NULL;
		if (tabela.vect[pozitie] == NULL)
```

```
                tabela.vect[pozitie] = nou;
            else
            {
                nodLS* temp = tabela.vect[pozitie];
                while (temp->next)
                    temp = temp->next;
                temp->next = nou;
            }
        }
        return pozitie;
}

void traversare(hashT tabela)
{
        if (tabela.vect != NULL)
        {
            for (int i = 0; i < tabela.size; i++)
                if (tabela.vect[i] != NULL)
                {
                    printf("\nPozitie: %d", i);
                    nodLS* temp = tabela.vect[i];
                    while (temp)
                    {
                        printf("\nCod=%d, Nume=%s, Medie=%5.2f",
temp->inf.cod, temp->inf.nume, temp->inf.medie);
                        temp = temp->next;
                    }
                }
        }
}

void dezalocare(hashT tabela)
{
        if (tabela.vect != NULL)
        {
            for (int i = 0; i < tabela.size; i++)
                if (tabela.vect[i] != NULL)
                {
                    nodLS* temp = tabela.vect[i];
                    while (temp)
                    {
                        nodLS* temp2 = temp->next;
                        free(temp->inf.nume);
                        free(temp);
                        temp = temp2;
                    }
```

```
                }
            free(tabela.vect);
        }
}

int stergere(hashT tabela, int cod)
{
      int pozitie;
      if (tabela.vect != NULL)
      {
            pozitie = functieHash(cod, tabela);
            if (tabela.vect[pozitie] == NULL)
                  return -1;
            else
            {
                  if (tabela.vect[pozitie]->inf.cod == cod)
                  {
                        if (tabela.vect[pozitie]->next == NULL)
                        {
                              nodLS* temp = tabela.vect[pozitie];
                              free(temp->inf.nume);
                              free(temp);
                              tabela.vect[pozitie] = NULL;
                        }
                        else
                        {
                              nodLS* temp = tabela.vect[pozitie];
                              tabela.vect[pozitie] = temp->next;
                              free(temp->inf.nume);
                              free(temp);
                        }
                  }
                  else
                  {
                        nodLS* temp = tabela.vect[pozitie];
                        while (temp->next != NULL && temp->next->inf.cod
!= cod)
                              temp = temp->next;
                        nodLS* p = temp->next;
                        if (p->next != NULL)
                        {
                              temp->next = p->next;
                              free(p->inf.nume);
                              free(p);
                        }
                        else
```

```
                        {
                              temp->next = NULL;
                              free(p->inf.nume);
                              free(p);
                        }
                  }
            }
      }
      return pozitie;
}

void main()
{
      int n;
      printf("Nr. studenti=");
      scanf("%d", &n);

      student s;
      char buffer[20];

      hashT tabela;
      tabela.size = 101;
      tabela.vect = (nodLS**)malloc(tabela.size * sizeof(nodLS*));
      for (int i = 0; i < tabela.size; i++)
            tabela.vect[i] = NULL;


      for (int i = 0; i < n; i++)
      {
            printf("\nCod=");
            scanf("%d", &s.cod);

            printf("\nNume=");
            scanf("%s", buffer);

            s.nume = (char*)malloc((strlen(buffer) + 1) * sizeof(char));
            strcpy(s.nume, buffer);

            printf("\nMedie=");
            scanf("%f", &s.medie);

            inserare(tabela, s);
      }
      traversare(tabela);

      stergere(tabela, 505);
```

```
	printf("\n----dupa stergere");
	traversare(tabela);
	dezalocare(tabela);
}
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <stdio.h>

using namespace std;

struct gradinita {
	int cod;
	char* nume;
	char* strada;
	int capacitate;
	float plata;
};

struct nodLDS {
	gradinita info;
	nodLDS* next, * prev;
};

struct nodls {
	gradinita info;
	nodls* next;
```

```
};

struct hashT {
	nodls** vect;
	int size;
};

int functieHash(int cheie, hashT tabela)
{
	return cheie % tabela.size;
}

int inserare(hashT tabela, gradinita g) {
	int pozitie;
	if (tabela.vect != NULL) {
		pozitie = functieHash(g.cod, tabela);
		nodls *nou = (nodls*)malloc(sizeof(nodls));
		nou->info.cod = g.cod;
		nou->info.nume = (char*)malloc((strlen(g.nume) + 1) * sizeof(char));
		strcpy(nou->info.nume, g.nume);
		nou->info.strada = (char*)malloc((strlen(g.strada) + 1) * sizeof(char));
		strcpy(nou->info.strada, g.strada);
		nou->info.capacitate = g.capacitate;
		nou->info.plata = g.plata;

		nou->next = NULL;
		if (tabela.vect[pozitie] == NULL)
```

```
                tabela.vect[pozitie] = nou;
        else {
                nodls* temp = tabela.vect[pozitie];
                while (temp->next)
                        temp = temp->next;
                temp->next = nou;
        }
    }
    return pozitie;
}

void traversare(hashT tabela) {
    if (tabela.vect != NULL)
    {
        for (int i = 0; i < tabela.size; i++)
            if (tabela.vect[i] != NULL)
            {
                printf("\nPozitie=%d", i);
                nodls* temp = tabela.vect[i];
                while (temp)
                {
                    printf("\nCod=%d, Nume=%s, Strada=%s, Capacitate=%d,
Plata=%5.2f",
                        temp->info.cod, temp->info.nume, temp-
>info.strada, temp->info.capacitate, temp->info.plata);
                    temp = temp->next;
                }
            }
```

```
	}
}

int stergere(hashT tabela, int cod) {
	int pozitie;
	if (tabela.vect != NULL) {
		pozitie = functieHash(cod, tabela);
		if (tabela.vect[pozitie] == NULL) {
			return -1;
		}
		else {
			if (tabela.vect[pozitie]->info.cod == cod) {
				if (tabela.vect[pozitie]->next == NULL) {
					nodls* temp = tabela.vect[pozitie];
					free(temp->info.nume);
					free(temp->info.strada);
					free(temp);
					tabela.vect[pozitie] = NULL;
				}
				else {
					nodls* temp = tabela.vect[pozitie];
					tabela.vect[pozitie] = temp->next;
					free(temp->info.nume);
					free(temp->info.strada);
					free(temp);
				}
			}
```

```
                else {
                        nodls* temp = tabela.vect[pozitie];
                        while (temp->next != NULL && temp->next->info.cod != cod)
                                temp = temp->next;
                        nodls* p = temp->next;
                        if (p->next != NULL) {
                                temp->next = p->next;
                                free(p->info.nume);
                                free(p->info.strada);
                                free(p);
                        }
                        else {
                                temp->next = NULL;
                                free(p->info.nume);
                                free(p->info.strada);
                                free(p);
                        }
                }
        }
        return pozitie;
}

void modificareCheie(hashT tabela, int cod, int codNou) {
        gradinita g;

        if (tabela.vect != NULL)
```

```
	{
		if (tabela.vect[cod] != NULL)
		{
			nodls* temp = tabela.vect[cod];
			while (temp->next)
				temp = temp->next;
			g.cod = codNou;
			g.nume = (char*)malloc((strlen(temp->info.nume) + 1) * sizeof(char));
			strcpy(g.nume, temp->info.nume);
			g.strada = (char*)malloc((strlen(temp->info.strada) + 1) * sizeof(char));
			strcpy(g.strada, temp->info.strada);
			g.capacitate = temp->info.capacitate;
			g.plata = temp->info.plata;
			stergere(tabela, cod);
			inserare(tabela, g);
		}
	}
}

void dezalocare(hashT tabela)
{
	if (tabela.vect != NULL)
	{
		for (int i = 0; i < tabela.size; i++)
			if (tabela.vect[i] != NULL)
			{
				nodls* temp = tabela.vect[i];
```

```
                while (temp)
                {
                        nodls* temp2 = temp->next;
                        free(temp->info.nume);
                        free(temp->info.strada);
                        free(temp);
                        temp = temp2;
                }
        }
    free(tabela.vect);
    }
}

nodLDS* inserareLDS(nodLDS* cap, nodLDS** coada, gradinita g)
{
    nodLDS* nou = (nodLDS*)malloc(sizeof(nodLDS));
    nou->info.cod = g.cod;
    nou->info.nume = (char*)malloc((strlen(g.nume) + 1) * sizeof(char));
    strcpy(nou->info.nume, g.nume);
    nou->info.strada = (char*)malloc((strlen(g.strada) + 1) * sizeof(char));
    strcpy(nou->info.strada, g.strada);
    nou->info.capacitate = g.capacitate;
    nou->info.plata = g.plata;
    nou->next = NULL;
    nou->prev = NULL;
    if (cap == NULL)
    {
```

```
		cap = nou;
		*coada = nou;
	}
	else
	{
		nodLDS* temp = cap;
		while (temp->next != NULL)
			temp = temp->next;
		temp->next = nou;
		nou->prev = temp;
		*coada = nou;
	}
	return cap;
}

void traversareLDS(nodLDS* cap)
{
	nodLDS* temp = cap;
	while (temp != NULL)
	{
		printf("\nCod=%d, Nume=%s, Strada=%s, Capacitate=%d, Plata=%5.2f",
			temp->info.cod, temp->info.nume, temp->info.strada, temp->info.capacitate, temp->info.plata);
		temp = temp->next;
	}
}
```

```
void splitListaDubla(nodLDS* lista) {
	nodLDS* p = lista;
	nodLDS* capListaPar = NULL, *capListaImpar = NULL;
	nodLDS* coadaPar = NULL, *coadaImpar = NULL;
	while (p) {
		if (p->info.capacitate % 2 == 0)
			capListaPar = inserareLDS(capListaPar, &coadaPar, p->info);
		else
			capListaImpar = inserareLDS(capListaImpar, &coadaImpar, p->info);
		p = p->next;
	}

	printf("\nGRADINITE CU CAPACITATE IMPARA\n");
	traversareLDS(capListaImpar);
	printf("\nGRADINITE CU CAPACITATE PARA\n");
	traversareLDS(capListaPar);
}

void main() {
	hashT tabela;
	tabela.size = 101;
	tabela.vect = (nodls**)malloc(tabela.size * sizeof(nodls*));
	for (int i = 0; i < tabela.size; i++) {
		tabela.vect[i] = NULL;
	}

	nodLDS* capLSpar = NULL, * coadaLSpar = NULL;
```

```
nodLDS* capLD = NULL, * coadaLD = NULL;

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

int n;
fscanf(f, "%d", &n);
gradinita g;
char buffer1[30];
char buffer2[40];
for (int i = 0; i < n; i++)
{
	fscanf(f, "%d", &g.cod);
	fscanf(f, " %[^\n]s", buffer1);
	g.nume = (char*)malloc((strlen(buffer1) + 1) * sizeof(char));
	strcpy(g.nume, buffer1);
	fscanf(f, " %[^\n]s", buffer2);
	g.strada = (char*)malloc((strlen(buffer2) + 1) * sizeof(char));
	strcpy(g.strada, buffer2);
	fscanf(f, "%d", &g.capacitate);
	fscanf(f, "%f", &g.plata);
	inserare(tabela, g);

}

if (tabela.vect != NULL)
{
	for (int i = 0; i < tabela.size; i++)
```

```
		{
			if (tabela.vect[i] != NULL)
			{
				nodls* temp = tabela.vect[i];
				while (temp)
				{
					capLD = inserareLDS(capLD, &coadaLD, temp->info);
					if ((temp->info.capacitate) % 2 == 0)
					{
						capLSpar = inserareLDS(capLSpar, &coadaLSpar,
temp->info);
					}
					temp = temp->next;
				}
			}
		}

		printf("\n_____\n");
		traversareLDS(capLSpar);

		printf("\nSPLIT LISTA DUBLA\n");
		splitListaDubla(capLD);

		traversare(tabela);
		printf("\n_____");
		modificareCheie(tabela, 2, 14);
```

```
        traversare(tabela);

}
```