L07. TABELE DE DISPERSIE (HASH TABLES)

Header.h

```
#pragma once
#include <iostream>
using namespace std;

#define M 10000

struct HT{
	char* cheie;
	HT* urm;
};

HT* Find(HT* tab[M], char* key);
void Insert(HT* tab[M], char* key);
int fct(char* key);
void ListAll(HT* tab[M]);
void Remove(HT* tab[M], char* key);
double Test(HT* tab[M], double& proc);
```

Functii.cpp

```
#include "Header.h"

int fct(char* key)
{
	int rez = 0;
	for (unsigned int i = 0;i < strlen(key);i++)
		rez += key[i];
	return rez%M;
}

HT* Find(HT* tab[M], char* key)
{
	HT* p = tab[fct(key)];
	while (p)
	{
		if (strcmp(key, p->cheie) == 0)
			return p;
		p = p->urm;
	}
	return NULL;
}
```

```
void Insert(HT* tab[M], char* key)
{
	HT* p = new HT;
	p->cheie = new char[strlen(key) + 1];
	strcpy_s(p->cheie, strlen(key) + 1, key);
	HT* q = Find(tab, key);
	if (!q)
	{
		p->urm = tab[fct(key)];
		tab[fct(key)] = p;
	}
	else
		cout << "\n cheie dubla";
}

void ListAll(HT* tab[M])
{
	HT* p = NULL;
	for (int i = 0; i < M; i++)
	{
		if (tab[i] != 0)
		{
			cout << "\nInregistrari avand codul de dispersie "
<< i << ": ";
			p = tab[i];
			while (p)
			{
				if (p->cheie)
					cout << p->cheie << "\t";
				p = p->urm;
			}
			cout << endl;
		}
	}
}

void Remove(HT* tab[M], char* key)
{
	HT* p = tab[fct(key)];
	HT* q = p;
	if (p)
	{
		//cout << "\n stergere " << key << endl;
		if (strcmp(p->cheie, key) == 0) //delete front
		{
			//cout << "\n delete front\n";
			q = p->urm;
			if (p->cheie)
				delete[] p->cheie;
			p->cheie = nullptr;
```

```
                  if (p)
                        delete p;
                  if(!q)
                        tab[fct(key)] = NULL;
            }
            else
            {
                  while (p)
                  {
                        p = p->urm;
                        if (strcmp(p->cheie, key) == 0)
                        {
                              //cout << "\n delete interm\n";
                              q->urm = p->urm;
                              if (p->cheie)
                                    delete[] p->cheie;
                              p->cheie = nullptr;
                              if (p)
                                    delete p;
                              p = NULL;
                        }
                        q = p;
                  }
            }
      }
      else
            cout << endl << key << " " << fct(key) << " nu exista in
tabela hash\n";
}

double Test(HT* tab[M], double& rez)
{
      double total = 0;
      double notFound = 0;
      FILE* f;
      HT*p = NULL;
      int j;
      char buff[100];
      char temp[100];
      if (fopen_s(&f, "text.in", "r"))
      {
            fprintf(stderr, "\nEroare la deschiderea fisierului\n");
            return false;
      }

      while (fgets(buff, 99, f) != NULL)
      {
            buff[strlen(buff) - 1] = '\0';
            j = 0;
            for (unsigned int i = 0;i <= strlen(buff);i++)
            {
```

```
                if (buff[i] == ' ' || buff[i]=='\0')
                {
                        temp[j] = '\0';
                        total+=1;
                        p = Find(tab, temp);
                        if (!p)
                                notFound++;
                        j = 0;
                }
                else
                {
                        temp[j] = buff[i];
                        j++;
                }
        }
    }
    cout << endl;


    if (fclose(f))
    {
        fprintf(stderr, "\nEroare la inchiderea fisierului\n");
        return false;
    }

    rez = (double)(total - notFound) / total * 100;
    return true;
}
```

Main.cpp

```
#include "Header.h"

int main()
{
    HT* tab[M];
    int x[M];
    int N = 0;
    char buff[100];
    char text[10000];
    FILE* f;
    double C = 0;
    double val = 0;
    int ok;

    cout << "\n Initializare tabela...\n";
    for (int i = 0;i < M;i++)
    {
        tab[i] = NULL;
        x[i] = 0;
```

```
	}

	cout << "\n Inserare date in tabela...\n";
	if (fopen_s(&f, "date.in", "r"))
	{
		fprintf(stderr, "\nEroare la deschiderea fisierului\n");
		exit(EXIT_FAILURE);
	}

	while (fgets(buff, 99, f) != NULL)
	{
		buff[strlen(buff) - 1] = '\0';
		N++;
		x[fct(buff)]++;

		Insert(tab, buff);
	}
	cout << endl;


	if (fclose(f))
	{
		fprintf(stderr, "\nEroare la inchiderea fisierului\n");
		exit(EXIT_FAILURE);
	}

	cout << "\n Tabela completa:\n";
	ListAll(tab);
	cout << endl << endl;

	for (int i = 0;i < M;i++)
		C += (double)x[i] / N;
	C -= (double)N / M;
	cout << endl << "Factorul de incarcare: " << C << endl << endl;

	if (Find(tab, "volum"))
		cout << "\n Cautare volum: gasit" << endl;
	else
		cout << "\n Cautare volum: nu exista\n";

	if (Find(tab, "USB"))
		cout << "\n Cautare USB: gasit" << endl;
	else
		cout << "\n Cautare USB: nu exista\n";

	cout << "\nEliminare hub...";
	Remove(tab, "hub");

	cout << "\nEliminare script...";
	Remove(tab, "script");
	cout << "\n Tabela dupa eliminare:\n";
```

```
      ListAll(tab);


      ok = Test(tab, val);
      if (ok)
            cout << "\n Corectitudinea textului din fisierul text.in:
" << val << "%.\n";
      return 0;
}
```