****

**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informatică şi Microelectronică**

**Departamentul Informatică şi Ingineria Sistemelor**

**Samarchin Maxim**

**Gr. MI-222**

**Raport**

**pentru lucrarea de laborator Nr.1**

***la cursul “Programarea orientată pe obiecte”***

***Tema:*** *: Clase și obiecte. Constructori și destructor pentru clasa*

Verificat:

**Kulev Mihail ,** *dr., conf. univ.*

Facultatea FCIM, UTM

**Chișinău – 2023**

**Scopul lucrării:**

Studierea programării prin abstractizarea datelor;

Studierea regulilor de definire şi utilizare a structurilor de date;

Crearea variabilelor de tip structură, accesarea componentelor unei structuri.

Noţiuni de bază

Structura – este o mulţime de date grupate, conform unei ierarhii, de obicei de tipuri diferite. Structura poate fi comparată cu tabloul, însă tabloul conţine date de un acelaşi tip, iar elementele tabloului se accesează cu ajutorul indicilor, pe cînd accesarea componentelor structurii are o formă specifică. Structura reprezintă un tip abstract de date, un tip definit de utilizator prin utilizarea tipurilor deja existente.

Sarcina Nr.1:

а) Să se creeze tipul abstract de date (structura) – cîine, care se compune din rasă, nume şi vîrstă. Să se definească funcţiile de setare şi modificare a datelor şi de eliberare a memoriei alocate. Pentru iniţializarea componentelor textuale să se utilizeze operatorul de alocare dinamică a memoriei new. Să se definească funcţia pentru sortarea tabloului de tip cîine conform vîrstei şi numelui. Adică cîinii de aceeaşi vîrstă se sortează în ordine alfabetică.

#include <iostream>

#include <string>

#include <algorithm>

using namespace std;

struct Caine {

string rasa;

string nume;

int varsta;

};

// Functie pentru setarea datelor unui caine

void setare\_caine(Caine &caine, const string &rasa, const string &nume, int varsta) {

caine.rasa = rasa;

caine.nume = nume;

caine.varsta = varsta;

}

// Functie pentru eliberarea memoriei alocate

void eliberare\_memorie(Caine \*&caini, int numar\_caini) {

for (int i = 0; i < numar\_caini; ++i) {

delete &caini[i];

}

delete[] caini;

}

// Functie de comparare pentru sortarea cainilor dupa varsta si nume

bool comparare\_caini(const Caine &caine1, const Caine &caine2) {

if (caine1.varsta == caine2.varsta) {

return caine1.nume < caine2.nume; // Sortare alfabetică după nume

}

return caine1.varsta < caine2.varsta; // Sortare după varsta

}

int main() {

// Exemplu de utilizare a structurii Caine și a funcțiilor definite

const int numar\_caini = 3;

Caine \*caini = new Caine[numar\_caini];

// Setarea datelor pentru fiecare caine

setare\_caine(caini[0], "Labrador", "Rex", 3);

setare\_caine(caini[1], "Bulldog", "Max", 5);

setare\_caine(caini[2], "Pudel", "Bella", 3);

// Sortarea tabloului de caini conform varstei si numelui

sort(caini, caini + numar\_caini, comparare\_caini);

// Afisare rezultatul sortarii

for (int i = 0; i < numar\_caini; ++i) {

cout << "Rasa: " << caini[i].rasa << ", Nume: " << caini[i].nume << ", Varsta: " << caini[i].varsta << endl;

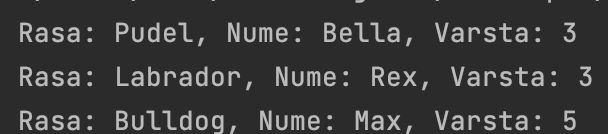
}

// Eliberarea memoriei alocate

eliberare\_memorie(caini, numar\_caini);

return 0;

}



Sarcina Nr.2

b) Să se creeze tipul abstract de date (structura) – vector, care se compune dintr-un pointer spre float şi din numărul de elemente. Să se definească funcţiile: de iniţializare, de eliminare a unui vector; de setare/ modificare a dimensiunii vectorului; de acces la elementele vectorului; de calcul a sumei elementelor unui vector. În funcţia main să se exemplifice înmulţirea a doi vectori.

#include <iostream>

using namespace std;

struct Vector {

float\* data;

int numar\_elemente;

};

// Funcție pentru inițializarea vectorului cu un număr dat de elemente

void initializare\_vector(Vector &vec, int numar\_elemente) {

vec.data = new float[numar\_elemente];

vec.numar\_elemente = numar\_elemente;

}

// Funcție pentru eliminarea vectorului și eliberarea memoriei alocate

void eliminare\_vector(Vector &vec) {

delete[] vec.data;

vec.numar\_elemente = 0;

}

// Funcție pentru setarea/modificarea dimensiunii vectorului

void setare\_dimensiune(Vector &vec, int nou\_numar\_elemente) {

float\* nou\_data = new float[nou\_numar\_elemente];

// Copierea datelor existente în noul vector (atât cât este posibil)

for (int i = 0; i < min(vec.numar\_elemente, nou\_numar\_elemente); ++i) {

nou\_data[i] = vec.data[i];

}

delete[] vec.data; // Eliberarea vechii alocări de memorie

vec.data = nou\_data; // Actualizarea pointerului către noul vector

vec.numar\_elemente = nou\_numar\_elemente; // Actualizarea numărului de elemente

}

// Funcție pentru accesarea elementului de la un anumit index

float accesare\_element(Vector &vec, int index) {

if (index >= 0 && index < vec.numar\_elemente) {

return vec.data[index];

}

return -1; // Sau altă valoare semnificativă pentru eroare

}

// Funcție pentru calculul sumei elementelor unui vector

float suma\_elemente(Vector &vec) {

float suma = 0.0;

for (int i = 0; i < vec.numar\_elemente; ++i) {

suma += vec.data[i];

}

return suma;

}

int main() {

// Exemplu de utilizare a structurii Vector și a funcțiilor definite

Vector vector1, vector2;

initializare\_vector(vector1, 3); // Inițializare vector1 cu 3 elemente

initializare\_vector(vector2, 3); // Inițializare vector2 cu 3 elemente

// Setarea elementelor pentru vector1 și vector2 (doar pentru exemplificare)

vector1.data[0] = 1.0;

vector1.data[1] = 2.0;

vector1.data[2] = 3.0;

vector2.data[0] = 4.0;

vector2.data[1] = 5.0;

vector2.data[2] = 6.0;

// Exemplu de calcul al sumei elementelor pentru vector1 și vector2

cout << "Suma elementelor vectorului 1: " << suma\_elemente(vector1) << endl;

cout << "Suma elementelor vectorului 2: " << suma\_elemente(vector2) << endl;

// Exemplu de înmulțire a doi vectori

if (vector1.numar\_elemente == vector2.numar\_elemente) {

Vector rezultat\_inmultire;

initializare\_vector(rezultat\_inmultire, vector1.numar\_elemente);

for (int i = 0; i < vector1.numar\_elemente; ++i) {

rezultat\_inmultire.data[i] = vector1.data[i] \* vector2.data[i];

cout << "Element " << i << " din vectorul rezultat: " << rezultat\_inmultire.data[i] << endl;

}

eliminare\_vector(rezultat\_inmultire); // Eliberare memorie pentru rezultatul înmulțirii

} else {

cout << "Vectorii nu pot fi inmultiti, dimensiunile sunt diferite." << endl;

}

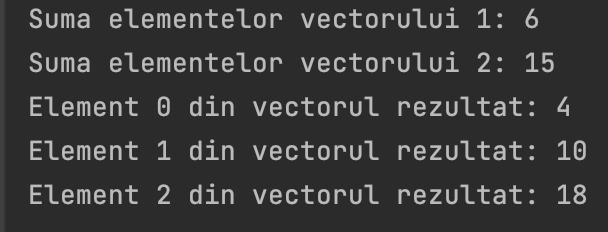
// Eliberarea memoriei alocate pentru vectori

eliminare\_vector(vector1);

eliminare\_vector(vector2);

return 0;

}



Concluzie

Aceste exemple demonstrează utilizarea conceptelor de structuri și funcții în C++ pentru a crea și gestiona structuri de date abstracte, oferind un început pentru lucrul cu astfel de entități în limbajul de programare C++. Este important să adăugați tratarea erorilor și alte funcționalități în implementarea finală, pentru a asigura o utilizare corectă și robustă a acestor structuri.