**MINISTERUL EDUCAȚIEI ȘI CERCETĂRII AL REPUBLICII MOLDOVA**

**Universitatea Tehnică a Moldovei**

**RAPORT**

Lucrare de laborator nr. 1

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

***Tema: Clase și obiecte. Constructorii și destructorul clasei***

**Varianta 21**

**A efectuat :**   **St. gr. CR-221FR Serba Cristina**

**A verificat: asist. univ. Mantaluța Marius**

**Chișinău 2024**

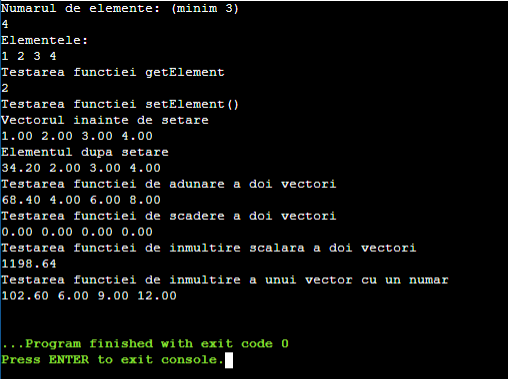
# Sarcina lucrării:

Să se creeze clasa Vector – vector. Clasa conţine pointer spre float, numărul de elemente şi o variabilă – codul erorii. Să se definească constructorul fără parametri (constructorul implicit), constructorul cu un parametru şi constructorul cu doi parametri. Să se definească funcţiile membru de acces: returnarea şi setarea valorii elementului (i,j). Să se definească funcţiile de adunare şi scădere a doi vectori; înmulţirea scalară unui vector cu un alt vector; înmulţirea unui vector cu un număr. Să se testeze funcţionarea clasei. În caz de insuficienţă de memorie, necorespondenţă a dimensiunilor vectorilor, depăşire a limitei memoriei utilizate să se stabilească codul erorii.

# Mersul programului:

|  |
| --- |
| #include <iostream>  using namespace std;  typedef enum{  E\_OK = 0,  E\_NULL\_PTR,  E\_INSUFFICIENT\_MEMORY,  E\_VECTOR\_UNMATCHED,  E\_MEMORY\_OVERFLOW  } Error\_Type;  class Vector{  private:  float \*vectorElements;  int size;  Error\_Type errorCode;    public:  Vector() {  vectorElements = NULL;  size = 0;  errorCode = E\_NULL\_PTR;  }    Vector(int newSize)  {  vectorElements = new float[newSize];  if(vectorElements != NULL)  {  size = newSize;  for(int i = 0; i < newSize; i++)  {  vectorElements[i] = 0;  }  errorCode = E\_OK;  }  else  {  size = 0;  errorCode = E\_INSUFFICIENT\_MEMORY;  }  }  Vector(int newSize, float element) {  vectorElements = new float[newSize];  if(vectorElements != NULL)  {  size = newSize;  for(int i = 0; i < newSize; i++)  {  vectorElements[i] = element;  }  errorCode = E\_OK;  }  else  {  size = 0;  errorCode = E\_INSUFFICIENT\_MEMORY;  }  }    ~Vector()  {  delete[] vectorElements;  }  void setElement(int position, float value)  {  if(position < size)  {  vectorElements[position] = value;  errorCode = E\_OK;  }  else  {  errorCode = E\_MEMORY\_OVERFLOW;  }  }  float getElement(int position)  {  float returnElement = 0;  if(position < size)  {  returnElement = vectorElements[position];  errorCode = E\_OK;  }  else  {  errorCode = E\_VECTOR\_UNMATCHED;  }  return returnElement;  }  Vector& operator=(const Vector& second)  {  if (this != &second)  {  delete[] vectorElements;    size = second.size;  vectorElements = new float[size];    for (int i = 0; i < size; ++i)  {  vectorElements[i] = second.vectorElements[i];  }  }  return \*this;  }    friend float operator \*(const Vector &A, const Vector &B);  friend Vector operator \*(const Vector &A, float factor);    friend Vector operator +(const Vector &A, const Vector &B);    friend Vector operator -(const Vector &A, const Vector &B);    friend ostream& operator <<(ostream &out, const Vector &vector);  };  float operator \*(const Vector &A, const Vector &B)  {  float result = 0;  if (A.size == B.size)  {  for(int i = 0; i < A.size; i++)  {  result += A.vectorElements[i] \* B.vectorElements[i];  }  }  return result;  }  Vector operator \*(const Vector &A, float factor)  {  Vector temp(A.size);  for(int i = 0; i < A.size; i++)  {  float value = A.vectorElements[i] \* factor;  temp.setElement(i, value);  }  return temp;  }  Vector operator +(const Vector &A, const Vector &B)  {  Vector temp(A.size);  if(A.size == B.size)  {  for(int i = 0; i < A.size; i++)  {  float value = A.vectorElements[i] + B.vectorElements[i];  temp.setElement(i, value);  }  }  return temp;  }  Vector operator -(const Vector &A, const Vector &B)  {  Vector temp(A.size);  if(A.size == B.size)  {  for(int i = 0; i < A.size; i++)  {  float value = A.vectorElements[i] - B.vectorElements[i];  temp.setElement(i, value);  }  }  return temp;  }  ostream& operator <<(ostream &out, const Vector &example)  {  out << fixed;  out.precision(2);  for(int i = 0; i < example.size; i++)  {  out << example.vectorElements[i] << " ";  }  out << endl;  return out;  }  int main(void)  {  int numberOfElements;  float temp;  cout << "Numarul de elemente: (minim 3)" << endl;  cin >> numberOfElements;  Vector vectorList(numberOfElements);  Vector result(numberOfElements);  cout << "Elementele: \n";  for(int i = 0; i < numberOfElements; i++)  {  cin >> temp;  vectorList.setElement(i, temp);  }  temp = vectorList.getElement(1);  cout << "Testarea functiei getElement\n" << temp << endl;    cout << "Testarea functiei setElement()\n";  cout << "Vectorul inainte de setare\n" << vectorList;  vectorList.setElement(0, 34.2);  cout << "Elementul dupa setare\n" << vectorList;  result = vectorList + vectorList;  cout << "Testarea functiei de adunare a doi vectori\n" << result;    result = vectorList - vectorList;  cout << "Testarea functiei de scadere a doi vectori\n" << result;    temp = vectorList \* vectorList;  cout << "Testarea functiei de inmultire scalara a doi vectori\n" << temp << endl;    result = vectorList \* 3;  cout << "Testarea functiei de inmultire a unui vector cu un numar\n" << result;    return 0;  } |

# Rezultatele obținute:



# Concluzii:

În concluzie, cunoștințele și practica în domeniul constructorilor și destructorilor sunt esențiale pentru a scrie cod C++ eficient și sigur, asigurând utilizarea corectă a resurselor și funcționarea corespunzătoare a programului. Posibilitatea de a defini mai mulți constructori într-o clasă, fie cu parametri diferiți, fie cu valori implicite, oferă flexibilitate și adaptabilitate în crearea și inițializarea obiectelor. Principiile de definire și utilizare a constructorilor și destructorilor în C++ sunt cruciale pentru dezvoltarea de aplicații robuste și eficiente.