Ministry of Education of the Republic of Moldova

Technical University of Moldova

Department of Applied Informatics

**REPORT**

Laboratory work nr.1

*at Object Oriented Programming*

Performed by:

st. gr. FAF-161 A.Chirosca

Verified by:

dr., conf.univ. M. Kulev

Chisinau 2017-2018

***Laboratory work nr. 1 (Variant no.5)***

**Topic:** Structure - a mechanism of abstractization

**Objectives:** - To obtain practical skills of programming using data abstractization

* Studying the rules of data structures definition and usage
* Creating variables of structure type, accessing structure’s components

**Condition of the problem:**

1. To create a structure named contributor, that has name, specialty, category and salary. Define functions for setting, modification and comparison of contributors. For string creation use new operator. Free the memory. In main to exemplify the sorting of contributors according to different criteria.
2. To create an abstract data type (structure)- vector, that has a pointer to long and the number of elements. To define the functions of initialization, elimination of the vector, setting/modifying of dimension, accessing vector’s elements, sum calculation of negative numbers. For example in main realize the adding of two vectors. (1)

**Short theory and memory used:**

An **abstract data type** (**ADT**) is basically a logical description or a specification of components of the data and the operations that are allowed, that is independent of the implementation.

A **data structure** is a group of data elements grouped together under one name. These data elements, known as *members*, can have different types and different lengths. Data structures can be declared in C++ using the following syntax:  
  
 struct type\_name {  
 member\_type1 member\_name1;  
 member\_type2 member\_name2;  
 member\_type3 member\_name3;  
 .  
 .  
 } object\_names;

Once the objects of a determined structure type are declared, its members can be accessed directly

If (pointer)

use (->) operator && (new && delete) keywords;

else

use (.) operator;

***new operator***

***new*** operator denotes a request for memory allocation on the Heap. If sufficient memory is available, new operator initializes the memory and returns the address of the newly allocated and initialized memory to the pointer variable.

***Syntax to use new operator****:* To allocate memory of any data type, the syntax is:

pointer-variable = **new** data-type;

**Initialize memory:** We can also initialize the memory using new operator:

pointer-variable = **new** data-type(value);

**Allocate block of memory:** new operator is also used to allocate a block(an array) of memory of type *data-type*.

pointer-variable = **new** data-type[size];

***delete operator***

Since it is programmer’s responsibility to deallocate dynamically allocated memory

**delete** pointer-variable;

// freed the allocated memory

***delete p***

// freed the block of allocated memory

***delete[] q;*** (2)(3)

1) ***Data analysis***

1. Function used to allocate memory:

void allocMemory(contributor \* person, char \* name, char \* specialty, char \* category, int salary)

It takes as parameters a pointer to our structure, pointers to char for name, specialty and category of our contributor. This function allocate memory for our pointers using new operator that return the address to a allocated memory and copy into this memory values of contributor name, specialty and category.

2.Function used to input the data:

contributor \* inputData(contributor \* person, int number\_of\_persons)

It takes as parameters a pointer to our structure and the number of persons we want to create. Here we allocate memory for our pointers from structure , then we set the necessary values , call the allocMemory function that will allocate strictly necessary amount of memory for our variables’ values and then will delete the unnecessary previously allocated memory.

Return data: a pointer to the structure

3.Modify data:

void modifyData(contributor \* person, int number\_of\_persons)

This functions simply takes as parameters the pointer to our structure and the number of objects, then it ask as if we want to change some information and what do we want to change. According to our answer it calls the necessary function and modify the data.

To modify the data of(char \*) type it deletes the already allocated memory and allocates the new memory of the necessary size.

4.Sorting functions:

void dataSorting(contributor \* person, int number\_of\_persons)

It asks the user about what type of sorting he want to perform and in accordance with his desire it calls the necessary function.

5.Free memory function:

void freeMem(contributor \* person, int number\_of\_persons)

Check if the memory is allocated and deletes it using delete operator;

6. Output function

void freeMem(contributor \* person, int number\_of\_persons)

It iterates through all objects and output the data stored in each member of the structure.

***Contributor.h***

#ifndef \_COLABORATOR\_H

#define \_COLABORATOR\_H

using namespace std;

typedef struct contributor

{

char \* name;

char \* specialty;

char \* category;

int salary;

}contributor;

void allocMemory(contributor \* person, char \* name, char \* specialty, char \* category, int salary);

contributor \* inputData(contributor \* person, int number\_of\_persons);

void modifyData(contributor \* person, int number\_of\_persons);

void dataSorting(contributor \* person, int number\_of\_persons);

void outputData(contributor \* person, int number\_of\_persons);

void freeMem(contributor \* person, int number\_of\_persons);

#endif

***Contributor.cpp***

#include <iostream>

#include <string>

#include <iomanip>

#include <cstring>

#include "Colaborator.h"

using namespace std;

void allocMemory(contributor \* person, char \* name, char \* specialty, char \* category, int salary)

{

person->name = new char[strlen(name) + 1];

strcpy(person->name, name);

person->specialty = new char[strlen(specialty) + 1];

strcpy(person->specialty, specialty);

person->category = new char[strlen(category) + 1];

strcpy(person->category, category);

person->salary = salary;

}

contributor \* inputData(contributor \* person, int number\_of\_persons)

{

char \* newName = new char[100];

char \* newSpecialty = new char[100];

char \* newCategory = new char[100];

int newSalary = 0;

for (int i = 0; i < number\_of\_persons; i++)

{

cout << "Enter the name of the person: " << endl;

cin >> newName;

cout << "Enter the specialty: " << endl;

cin >> newSpecialty;

cout << "Enter the category: " << endl;

cin >> newCategory;

cout << "Enter the salary" << endl;

cin >> newSalary;

allocMemory(&person[i], newName, newSpecialty, newCategory, newSalary);

}

delete[] newName;

delete[] newSpecialty;

delete[] newCategory;

return person;

}

//--------------------------------------------Sorting function-------------------------------------------------//

void changeName(contributor \* person)

{

char \* newName = new char[100];

cout << "Enter the new name: " << endl;

cin >> newName;

delete[] person->name;

person->name = new char[strlen(newName) + 1];

strcpy(person->name, newName);

}

void changeSpecialty(contributor \* person)

{

char \* newSpecialty = new char[100];

cout << "Enter the new specialty: " << endl;

cin >> newSpecialty;

delete[] person->specialty;

person->specialty = new char[strlen(newSpecialty) + 1];

strcpy(person->specialty, newSpecialty);

}

void changeCategory(contributor \* person)

{

char \* newCategory = new char[100];

cout << "Enter the new category: " << endl;

cin >> newCategory;

delete[] person->category;

person->category = new char[strlen(newCategory) + 1];

strcpy(person->category, newCategory);

}

void changeSalary(contributor \* person)

{

int newSalary = 0;

cout << "Enter the new salary: " << endl;

cin >> newSalary;

person->salary = newSalary;

}

void modifyData(contributor \* person, int number\_of\_persons)

{

string name;

string option;

cout << "Enter the name of the person you want to modify: " << endl;

cin >> name;

for (int i = 0; i < number\_of\_persons; i++)

{

if (name == person[i].name)

{

cout << "What do you want to modify?(name/specialty/category/salary) " << endl;

cin >> option;

if (option == "name" || option == "Name")

changeName(&person[i]);

if (option == "specialty" || option == "Specialty")

changeSpecialty(&person[i]);

if (option == "category" || option == "Category")

changeCategory(&person[i]);

if (option == "salary" || option == "Salary")

changeSalary(&person[i]);

}

}

}

//-----------------------------------------------------Sorting functions-----------------------------------------//

void nameSorting(contributor \* person, int number\_of\_persons)

{

contributor temp;

for (int i = 1; i < number\_of\_persons ; i++)

{

if(strcmp(person[i - 1].name, person[i].name) > 0)

{

temp = person[i - 1];

person[i - 1] = person[i];

person[i] = temp;

}

}

}

void specialtySorting(contributor \* person, int number\_of\_persons)

{

contributor temp;

for (int i = 1; i < number\_of\_persons; i++)

{

if (strcmp(person[i - 1].specialty, person[i].specialty) > 0)

{

temp = person[i - 1];

person[i - 1] = person[i];

person[i] = temp;

}

}

}

void categorySorting(contributor \* person, int number\_of\_persons)

{

contributor temp;

for (int i = 1; i < number\_of\_persons; i++)

{

if (strcmp(person[i - 1].category, person[i].category) > 0)

{

temp = person[i - 1];

person[i - 1] = person[i];

person[i] = temp;

}

}

}

void heapify(contributor \* person, int number\_of\_persons, int index)

{

int largest = index;

contributor temp;

int left = 2 \* index + 1;

int right = 2 \* index + 2;

if (left < number\_of\_persons && person[left].salary > person[largest].salary)

largest = left;

if (right < number\_of\_persons && person[right].salary > person[largest].salary)

largest = right;

if (largest != index)

{

temp = person[index];

person[index] = person[largest];

person[largest] = temp;

heapify(person, number\_of\_persons, largest);

}

}

void salarySorting(contributor \* person, int number\_of\_persons)

{

contributor temp;

for (int i = number\_of\_persons / 2 - 1; i >= 0; i--)

heapify(person, number\_of\_persons, i);

for (int i = number\_of\_persons - 1; i >= 0; i--)

{

temp = person[0];

person[0] = person[i];

person[i] = temp;

heapify(person, i, 0);

}

}

void dataSorting(contributor \* person, int number\_of\_persons)

{

string criteria;

cout << "Enter the criteria of sorting:(Name/Specialty/Category/Salary) " << endl;

cin >> criteria;

if (criteria == "name" || criteria == "Name")

nameSorting(person, number\_of\_persons);

if (criteria == "specialty" || criteria == "Specialty")

specialtySorting(person, number\_of\_persons);

if (criteria == "category" || criteria == "Category")

categorySorting(person, number\_of\_persons);

if (criteria == "salary" || criteria == "Salary")

salarySorting(person, number\_of\_persons);

}

void outputData(contributor \* person, int number\_of\_persons)

{

cout << "\n\n" << endl;

for (int i = 0; i < number\_of\_persons; i++)

{

cout << setw(10) << person[i].name << setw(15) << person[i].specialty << setw(5) << person[i].category << setw(10) << person[i].salary << endl;

}

cout << "\n\n" << endl;

}

void freeMem(contributor \* person, int number\_of\_persons)

{

for (int i = 0; i < number\_of\_persons; i++)

{

if(person[i].name)

delete[] person[i].name;

if(person[i].specialty)

delete[] person[i].specialty;

if(person[i].category)

delete[] person[i].category;

person[i].salary = 0;

}

}

***main.cpp***

#include <iostream>

#include <string>

#include <iomanip>

#include "Colaborator.h"

using namespace std;

int main()

{

int number\_of\_persons;

bool change = true;

cout << "Enter the number of people:";

cin >> number\_of\_persons;

contributor \* person = new contributor[number\_of\_persons];

inputData(person, number\_of\_persons);

outputData(person, number\_of\_persons);

while (1)

{

cout << "Do you want to modify some data? (1-Yes/0-No) " << endl;

cin >> change;

if (change)

modifyData(person, number\_of\_persons);

else

break;

}

while (1)

{

cout << "Do you want to sort data?(1/0)" << endl;

cin >> (change);

if (change)

dataSorting(person, number\_of\_persons);

else

break;

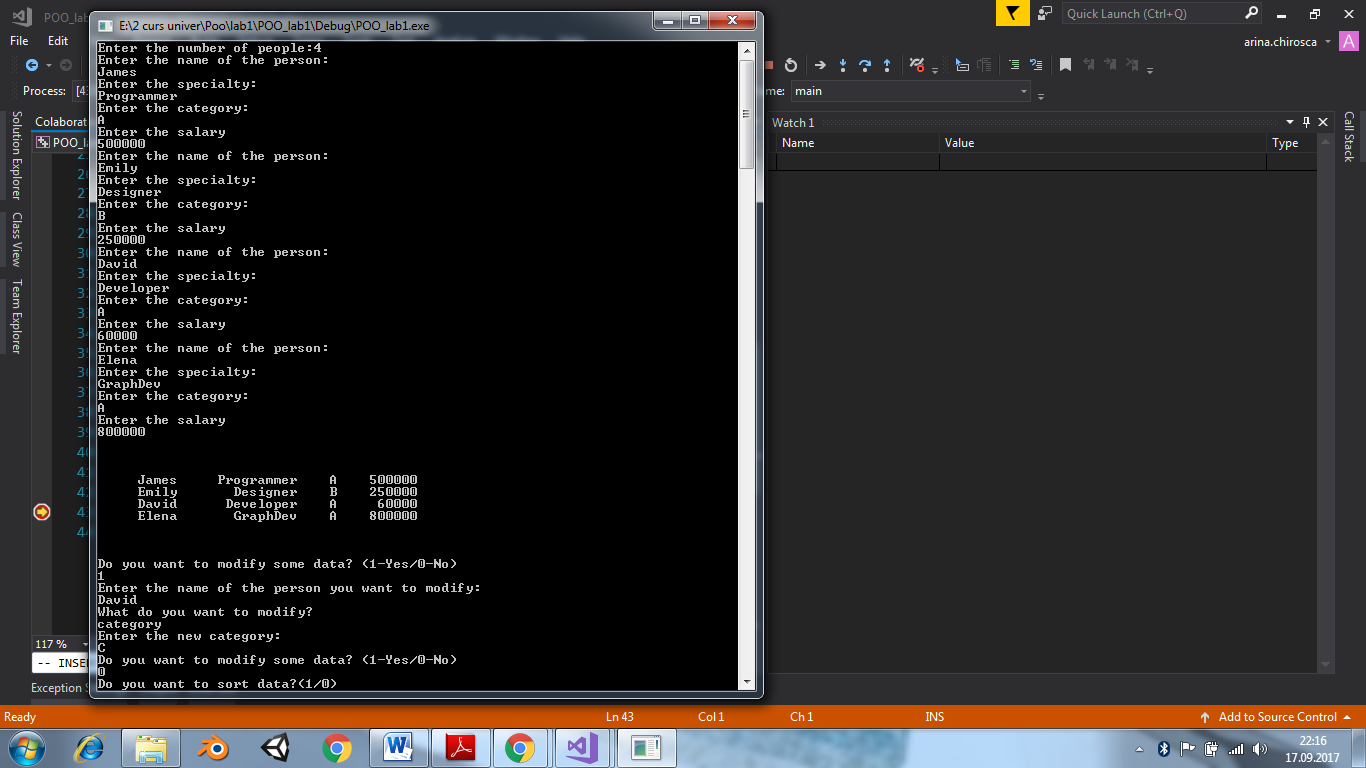
}

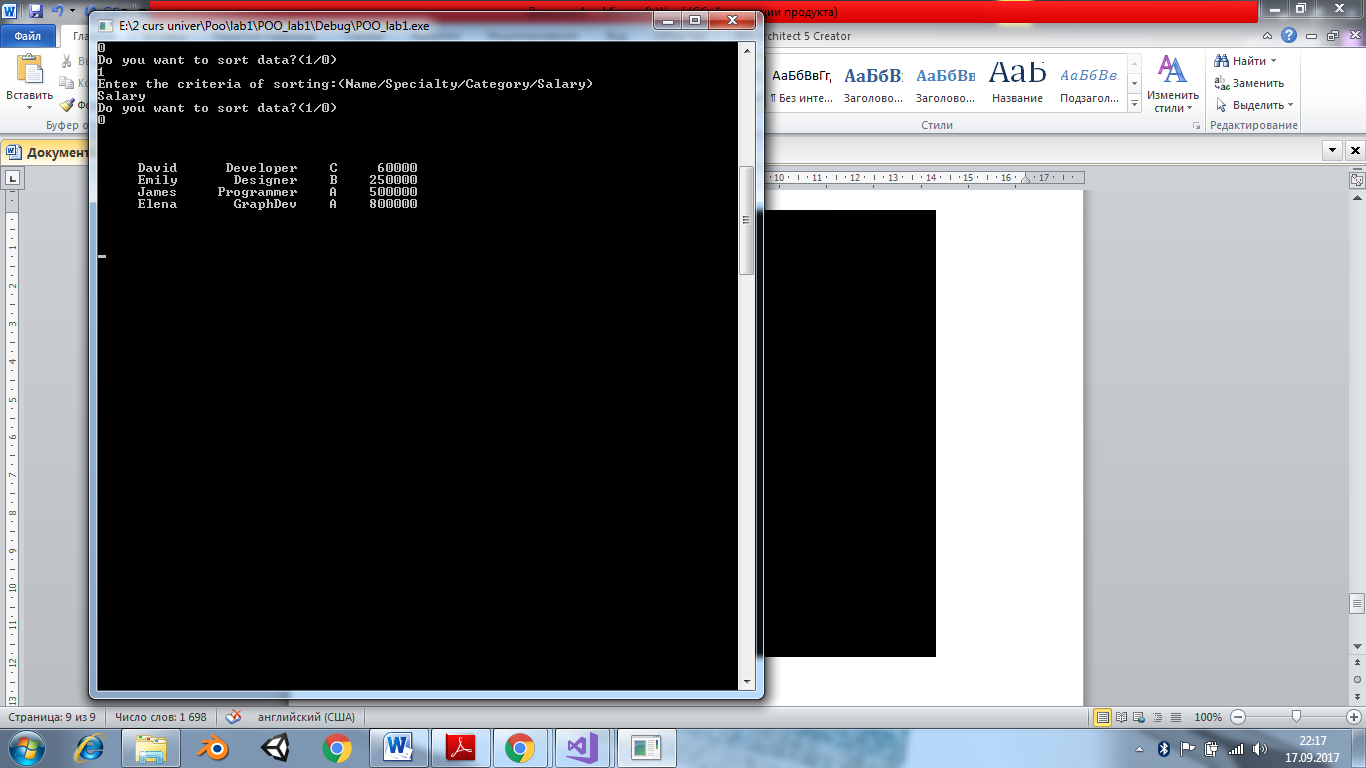
outputData(person, number\_of\_persons);

freeMem(person, number\_of\_persons);

return 0;

}





2) ***Data analysis***

1.vector initialization

void initVec(vector \* self, int number\_of\_elem)

Parameters : pointer to the structure, the number of elements in a vector, which maximum size can be 4.

Allocate the memory for our vector according to the number of elements.

2.Set Data

vector \* setData(vector \* self, int num\_of\_elem)

Parameters : pointer to the structure, the number of elements in a vector

Call the function initVec and set the data of the vector

Return type: pointer to structure

1. Free the memory

void freeMemory(vector \* self)

It sets all values to 0 then deletes the structure;

1. Addition

vector \* vectorAddition(vector \* vec1, vector \* vec2, vector \* result)

parameters: pointer to vectors to be added, and to the result vector

return type: pointer to the result vector

1. Vector resize

void vectorResize(vector \* self, int num\_of\_elem)

In order to change the size it free the memory and then again set the data;

1. Change one of the vector’s value

Parameters: pointer to the vector , index of the value you want to change, the new value;

It iterates through all values and at the necessary index it stops and changes the value

***Vector.h***

#ifndef \_VECTOR\_H

#define \_VECTOR\_H

#define VECTOR\_INIT\_CAPACITY 4

typedef struct vector

{

long \* vectorValues;

int number\_of\_elements;

}vector;

vector \* setData(vector \* self, int num\_of\_elem);

vector \* vectorAddition(vector \* vec1, vector \* vec2, vector \* result);

void printVector(vector \* self);

int getVectorSize(vector \* self);

void vectorResize(vector \* self, int num\_of\_elem);

void changeVectorValue(vector \* self, int index, int value);

#endif

***Vector.cpp***

#include <iostream>

#include <iomanip>

#include <stdio.h>

#include "Vector.h"

using namespace std;

void initVec(vector \* self, int number\_of\_elem)

{

self->number\_of\_elements = number\_of\_elem;

self->vectorValues = new long[self->number\_of\_elements];

}

int setNumberOfElements(vector \* self)

{

int number\_of\_elements;

cout << "Enter the number of vector's elements between (maximum 4): " << endl;

cin >> number\_of\_elements;

return number\_of\_elements;

}

vector \* setData(vector \* self, int num\_of\_elem)

{

if (num\_of\_elem > VECTOR\_INIT\_CAPACITY)

num\_of\_elem = 4;

initVec(self, num\_of\_elem);

cout << "Enter the vector's values:\n" << endl;

for (int i = 0; i < num\_of\_elem; i++)

{

cout << "Enter the value: " << i + 1 << endl;

cin >> self->vectorValues[i];

}

return self;

}

void freeMemory(vector \* self)

{

for (int i = 0; i < self->number\_of\_elements; i++)

{

self->vectorValues[i] = 0;

}

self->number\_of\_elements = 0;

delete[] self;

}

void vectorResize(vector \* self, int num\_of\_elem)

{

cout << "Resize the vector from: " << self->number\_of\_elements << " to " << num\_of\_elem << endl;

freeMemory(self);

setData(self, num\_of\_elem);

}

vector \* vectorAddition(vector \* vec1, vector \* vec2, vector \* result)

{

int num\_of\_elem = 0;

if (vec1->number\_of\_elements > vec2->number\_of\_elements)

num\_of\_elem = vec1->number\_of\_elements;

else

num\_of\_elem = vec2->number\_of\_elements;

initVec(result, num\_of\_elem);

for (int i = 0; i < num\_of\_elem; i++)

{

result->vectorValues[i] = vec1->vectorValues[i] + vec2->vectorValues[i];

}

return result;

}

int getVectorSize(vector \* self)

{

return self->number\_of\_elements;

}

void changeVectorValue(vector \* self, int index, int value)

{

int temp;

for (int i = 0; i < index; i++)

{

if ((i + 1) == index)

{

temp = self->vectorValues[i];

self->vectorValues[i] = value;

}

}

}

void printVector(vector \* self)

{

for (int i = 0; i < self->number\_of\_elements; i++)

cout << setw(5) << self->vectorValues[i];

cout << "\n";

}

***main.cpp***

#include <iostream>

#include "Vector.h"

using namespace std;

int main()

{

vector \* vec1 = new vector[VECTOR\_INIT\_CAPACITY];

vector \* vec2 = new vector[VECTOR\_INIT\_CAPACITY];

vector \* result = new vector[VECTOR\_INIT\_CAPACITY];

int length;

int newSize;

int number\_of\_elem1, number\_of\_elem2 = 0;

number\_of\_elem1 = setNumberOfElements(vec1);

setData(vec1, number\_of\_elem1);

number\_of\_elem2 = setNumberOfElements(vec2);

setData(vec2, number\_of\_elem2);

system("CLS");

result = vectorAddition(vec1, vec2, result);

cout << "vector1: ";

printVector(vec1);

cout << "vector2: ";

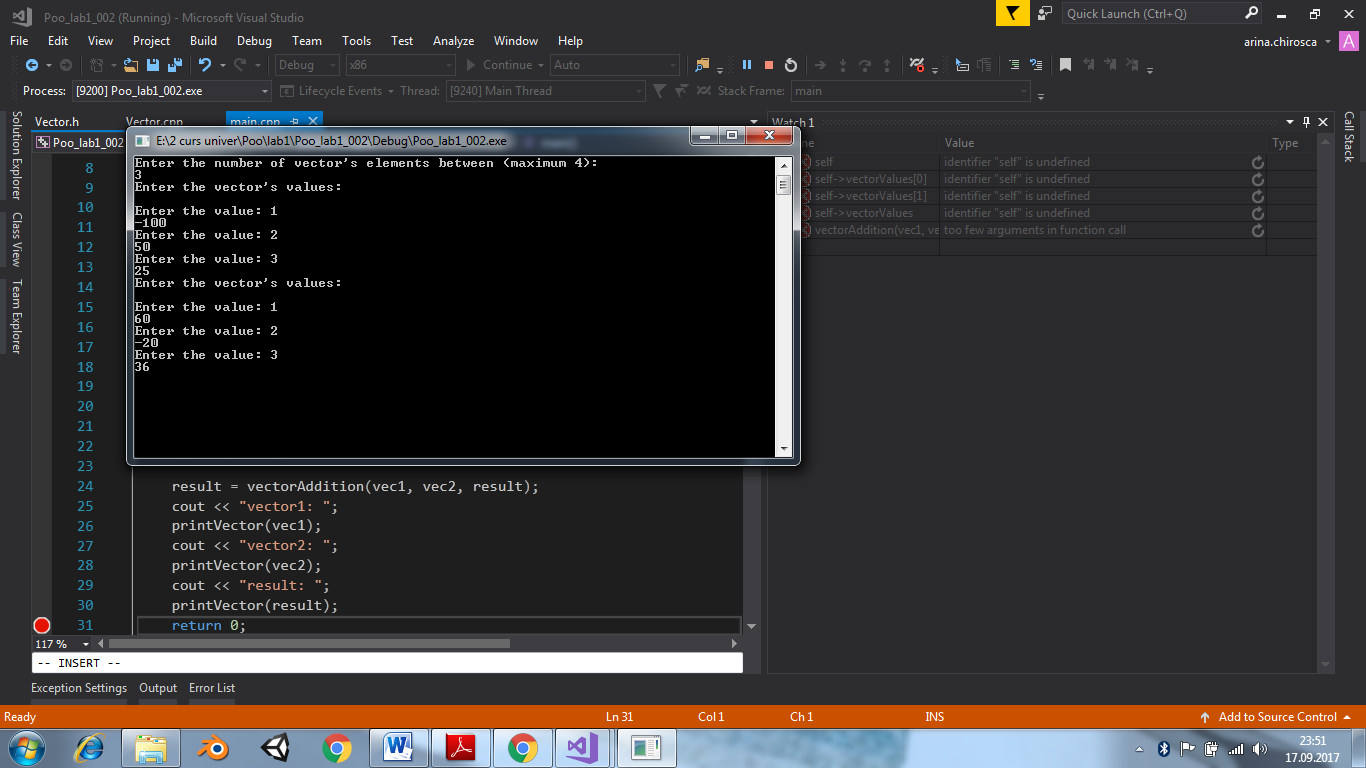
printVector(vec2);

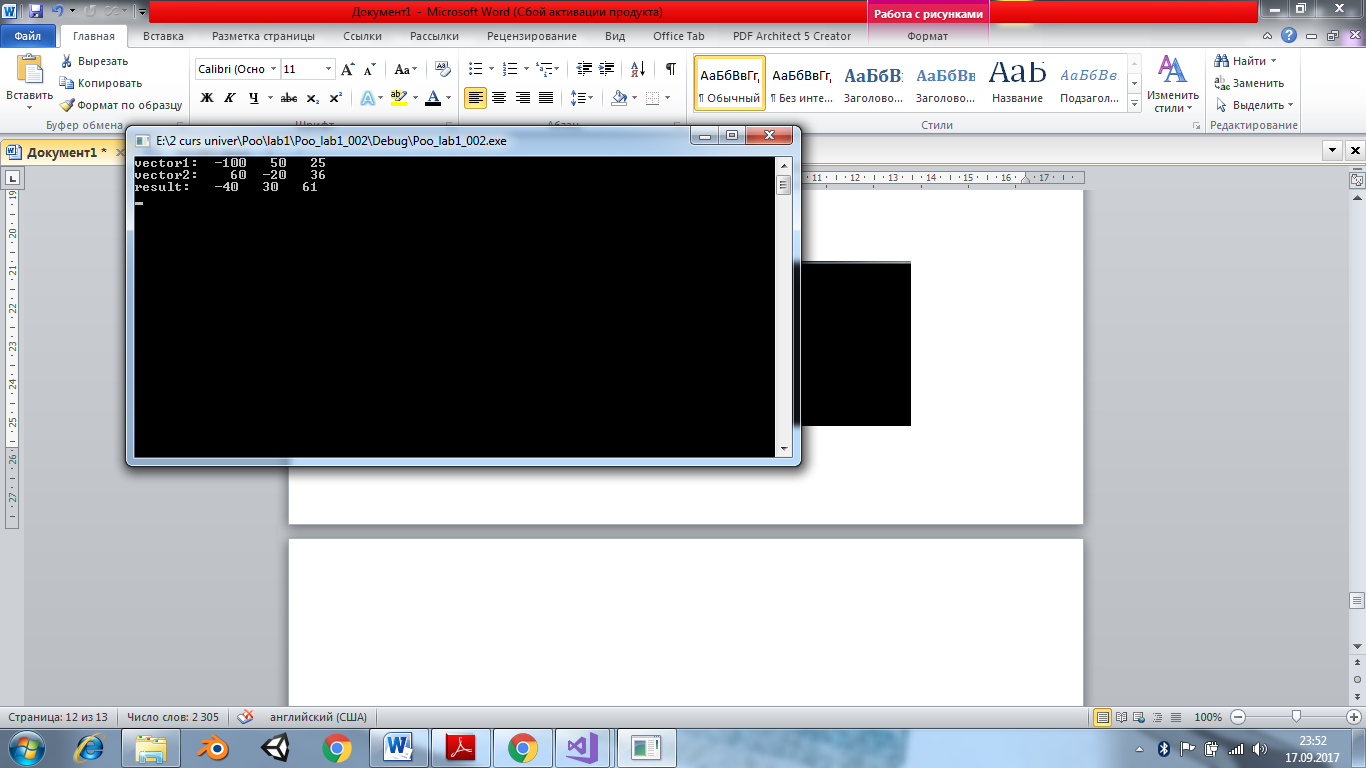
cout << "result: ";

printVector(result);

return 0;

}





***Conclusion***

*Advantages:*

* Structures allows easier processing of data.
* Structure can store heterogeneous values(Different Data types)
* Structure can even hold the value of an array.

***Bibliography***

1)Handnotes on OOP lesson of Lector dr., conf. univ. M.Kulev. for the FAF – 161 group Chișinău: UTM, 2017.

2) http://www.cplusplus.com/doc/tutorial/structures/

3) https://www.programiz.com/cpp-programming/structure