Universitatea Tehnică a Moldovei

Facultatea *Calculatoare, Informatică şi Microelectronică*

Specialitatea *Tehnologii Informaționale*

Raport

la lucrarea de laborator nr. 3

Tema:*"Moştenirea şi compoziția"*

Disciplina: "Programarea orientata pe obiecte"

Varianta 3

A efectuat:	Student grupa TI-231 FR	Apareci Aurica
A verificat:	Asistent universitar	Mititelu Vitalie

Chișinău 2025

Cuprins

1. Cadru teoretic

Tema lucrării: Moştenirea şi compoziţia

Scopul lucrării:

- studierea moştenirii, avantajele şi dezavantajele
- studierea compoziției
- studierea regulilor de definire a moştenirii şi compoziției
- studierea formelor de moştenire
- studierea inițializatorilor
- principiul de substituție
- moştenire şi compoziție

Varianta 3:

a) Să se creeze clasa mobilă, care conține informație despre preț, stil şi domeniul de utilizare (oficiu, bucătărie şi altă mobilă). Pentru setarea câmpurilor textuale să se folosească memoria dinamică. Definiți clasele derivate masa şi scaun. Definiți constructorii, destructorul, operatorii de atribuire şi alte funcții necesare.

b) Să se creeze clasa garaj, care conține suprafața. Determinați constructorii şi metodele de acces. Creați clasa casa, care conține odăi, o bucătarie (suprafața ei) şi garaj. Definiți clasa derivată vilă (ca parametru adăugator – mărimea lotului de pamânt). Determinați constructorii, destructorul şi fluxul de ieşire.

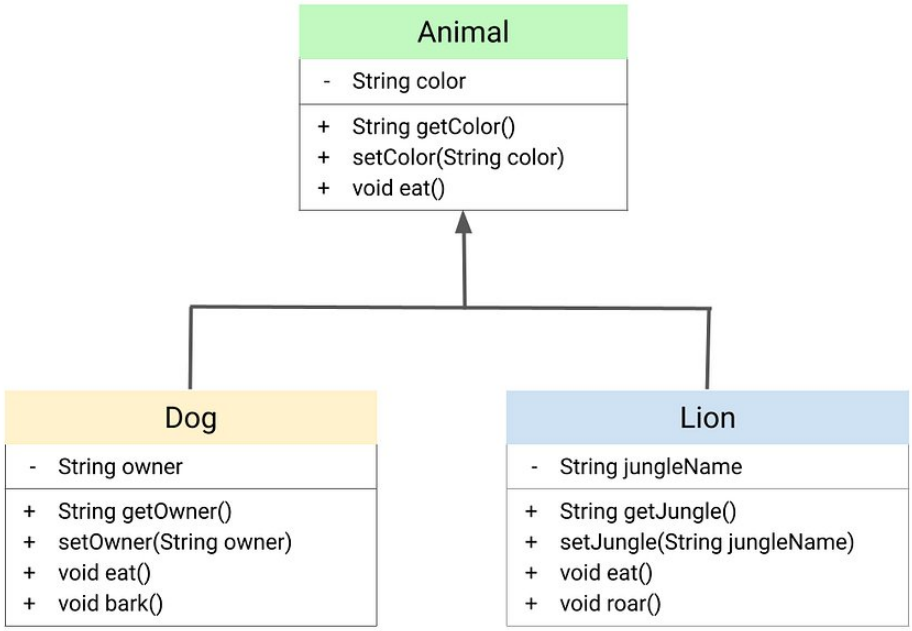
2. Repere teoretice

Tema acestei lucrări de laborator este axată pe două concepte fundamentale ale programării orientate pe obiect: moştenirea şi compoziția. Acestea reprezintă mecanisme prin care se pot crea relații între clase, facilitând reutilizarea codului, extinderea funcționalității şi modelarea logică a relațiilor din lumea reală în cadrul aplicațiilor software.

Moştenirea este mecanismul prin care o clasă derivată (sau subclasă) preia membrii (atributele şi metodele) unei clase de bază (superclasă), având posibilitatea de a-i utiliza, modifica sau extinde. Aceasta oferă o serie de avantaje semnificative, precum reutilizarea codului, creşterea clarității şi consistenței în proiectare, precum şi aplicarea principiului „o dată scris, oriunde reutilizat”. Cu toate acestea, moştenirea poate introduce şi anumite dezavantaje, cum ar fi creşterea interdependenței dintre clase şi dificultăți în mentenanța unui sistem extins ierarhic.

Pe de altă parte, compoziția este o relație prin care o clasă conține obiecte ale altor clase ca membri, oferind astfel o alternativă la moştenire. Compoziția permite asamblarea funcționalităților din mai multe clase într-una nouă, fără a crea dependențe rigide ierarhice. Acest principiu susține o

mai bună încapsulare și flexibilitate, întrucât modificările aduse unei clase componente nu afectează în mod direct alte clase care o utilizează.



Definirea corectă a moștenirii și a compoziției presupune respectarea unor reguli stricte de structurare. În cazul moștenirii, se utilizează specificatorii public, protected sau private, care influențează vizibilitatea membrilor moșteniți în cadrul clasei derivate. Totodată, limbajul C++ permite mai multe forme de moștenire: moștenirea simplă (o singură clasă de bază), moștenirea multiplă (derivarea din mai multe clase de bază), precum și moștenirea ierarhică, în care mai multe clase derivă dintr-o clasă comună.

Un aspect esențial în definirea relațiilor de moștenire îl constituie inițializatorii de clasă, care permit apelarea constructorilor claselor de bază în mod explicit din constructorul clasei derivate, asigurând astfel o inițializare corectă și completă a tuturor membrilor. De asemenea, în contextul moștenirii, un concept esențial este principiul de substituție Liskov, conform căruia un obiect al unei clase derivate trebuie să poată fi utilizat în locul unui obiect al clasei de bază fără a altera corectitudinea programului. Acest principiu susține ideea de polimorfism și este fundamental pentru scrierea unui cod robust și extensibil.

În concluzie, moștenirea și compoziția sunt două tehnici complementare de structurare a programelor orientate pe obiect, fiecare având roluri bine definite. Alegerea între cele două se face în funcție de natura relației logice dintre clase și de nevoia de flexibilitate, reutilizare sau izolare funcțională în cadrul arhitecturii software.

3. Listingul programului

main.cpp

```
#include "Scaun.h"
#include "Masa.h"

int main()
{
	Scaun * s1 = new Scaun();
	Masa * m1 = new Masa();
	cout << "Introduceti datele scaunului: " << endl;
	s1->Read();
	cout << "Introduceti datele mesei:   " << endl;
	m1->Read();
	system("cls");
	cout << "Datele scaunului:   " << endl;
	cout<<s1->ToString()<<endl;
	cout << "Datele mesei:   " << endl;
	cout << m1->ToString() << endl;
	return 0;
}
```

Masa.h

```
#pragma once
#include "Mobila.h"

class Masa : public Mobila
{
	private:
		double Inaltime;
		double Greutate;
		string Material;
	public:
		Masa() :Mobila(0, "Undefined", Other)
		{
			Inaltime = 0;
			Greutate = 0;
			Material = "";
		}
		Masa(string nume, double pret, double inaltime, double greutate,
string material, FurnitureType type) : Mobila(pret, nume, type)
		{
			Inaltime = inaltime;
			Greutate = greutate;
			Material = material;
		}

		double GetInaltime() { return Inaltime; }
		double GetGreutate() { return Greutate; }
		string GetMaterial() { return Material; }

		void SetInaltime(double inaltime) { Inaltime = inaltime; }
		void SetGreutate(double greutate) { Greutate = greutate; }
		void SetMaterial(string material) { Material = material; }
		string ToString()
		{
			string result = Mobila::ToString();
			result += "Inaltime: " + to_string(Inaltime) + "\n";
			result += "Greutate: " + to_string(Greutate) + "\n";
			result += "Material: " + Material + "\n";
			return result;
		}
```

```
	void Read() override
		{
			Mobila::Read();
			cout << "Inaltime: ";
			cin >> Inaltime;
			cout << "Greutate: ";
			cin >> Greutate;
			cout << "Material: ";
			cin >> Material;
		}
};
```

Mobila.h

```
#pragma once
#include <string>
#include <iostream>

using namespace std;
enum FurnitureType
{
	Oficiu,
	Bucatarie,
	Other
};

class Mobila
{
	protected:
		double Pret;
		string Stil;
		FurnitureType Type;
	public:
		Mobila(double pret, string stil, FurnitureType type)
		{
			Pret = pret;
			Stil = stil;
			Type = type;
		}

		double GetPret() { return Pret; }
		string GetStil() { return Stil; }
		FurnitureType GetType() { return Type; }

		void SetPret(double pret) { Pret = pret; }
		void SetStil(string stil) { this->Stil = stil; }
		void SetType(FurnitureType type) { this->Type = type; }

		virtual string ToString()
		{
			string result = "Pret: " + to_string(Pret) + "\n";
			result += "Stil: " + Stil + "\n";
			result += "Tip: ";
			switch (Type)
			{
				case Oficiu:
					result += "Mobila de oficiu\n";
					break;
				case Bucatarie:
					result += "Mobila pentru bucatarie\n";
					break;
```

```
                        case Other:
                              result += "Alt tip de mobila\n";
                              break;
                        default:
                              break;
                  }
                  return result;
            }

            virtual void Read()
            {
                  cout << "Pret: ";
                  cin >> Pret;
                  cout << "Stilul piesei de mobila: ";
                  cin >> Stil;
                  cout << "Tip (1-Oficiu/2-Bucatarie/3-altul): ";
                  int tip;
                  cin >> tip;
                  switch (tip)
                  {
                        case 1:
                              Type = Oficiu;
                              break;
                        case 2:
                              Type = Bucatarie;
                              break;
                        default:
                        case 3:
                              Type = Other;
                              break;
                  }
            }
};
```

Scaun.h

```
#pragma once
#include "Mobila.h"
class Scaun : public Mobila
{
      private:
            double GreutateAdmisa;
            string MaterialCarcasa;
            string Culoare;
      public:
            Scaun() :Mobila(0,"Undefined", Other)
            {
                  GreutateAdmisa = 0;
                  MaterialCarcasa = "";
                  Culoare = "";
            }
            Scaun(string nume, double pret, double greutateAdmisa, string ma-
terialCarcasa, string culoare, FurnitureType type) : Mobila(pret, nume, type)
            {
                  GreutateAdmisa = greutateAdmisa;
                  MaterialCarcasa = materialCarcasa;
                  Culoare = culoare;
            }

            double GetGreutateAdmisa() { return GreutateAdmisa; }
            string GetMaterialCarcasa() { return MaterialCarcasa; }
            string GetCuloare() { return Culoare; }
```

```
            void SetGreutateAdmisa(double greutateAdmisa) { GreutateAdmisa =
greutateAdmisa; }
            void SetMaterialCarcasa(string materialCarcasa) { MaterialCarcasa
= materialCarcasa; }
            void SetCuloare(string culoare) { Culoare = culoare; }
            string ToString()
            {
                  string result = Mobila::ToString();
                  result += "Greutate admisa: " + to_string(GreutateAdmisa) +
"\n";
                  result += "Material carcasa: " + MaterialCarcasa + "\n";
                  result += "Culoare: " + Culoare + "\n";
                  return result;
            }
            void Read() override
            {
                  Mobila::Read();
                  cout << "Greutate maxima admisa: ";
                  cin >> GreutateAdmisa;
                  cout << "Material carcasa: ";
                  cin >> MaterialCarcasa;
                  cout << "Culoare: ";
                  cin >> Culoare;
            }
};
```

Rezultatele testării

```
Datele scaunului:
Pret: 5300.000000
Stil: modern
Tip: Mobila de oficiu
Greutate admisa: 250.000000
Material carcasa: piele
Culoare: maroon

Datele mesei:
Pret: 4700.000000
Stil: modern
Tip: Alt tip de mobila
Inaltime: 190.000000
Greutate: 45.000000
Material: wood
```

main.cpp

```
#include <iostream>
#include "Building.h"

int main()
{
	Building building = Building(BuildingType::CivilBuilding, "Bloc loca-
tiv", "Chisinau, Decebal 19");
	for (size_t i = 0; i < 5; i++)
	{//generam 10 apartamente cu pretul 1199
		Apartment ap(1199);
		IRoom dormitor1 = IRoom("dormitor1",2,1,4,25,Bedroom);
		ap.AddRoom(dormitor1);
		IRoom dormitor2 = IRoom("dormitor2", 2, 1, 4, 30, Bedroom);
		ap.AddRoom(dormitor2);
		IRoom living = IRoom("living", 2, 1, 4, 25, LivingRoom);
		ap.AddRoom(living);
		IRoom bucatarie = IRoom("bucatarie", 2, 1, 4, 25, Kitchen);
		ap.AddRoom(bucatarie);
		IRoom baie = IRoom("baie", 2, 1, 4, 25, Bathroom);
		ap.AddRoom(baie);
		building.AddApartment(ap);
	}
	Store store = Store("Magazin");
	store.SetPrice(790);
	IRoom magazin = IRoom("magazin", 2, 1, 4, 25, Other);
	store.AddRoom(magazin);
	IRoom depozit = IRoom("depozit", 2, 1, 4, 50, Other);
	store.AddRoom(depozit);
	IRoom birou = IRoom("birou", 2, 1, 4, 25, Other);
	store.AddRoom(birou);
	IRoom wc = IRoom("wc", 2, 1, 4, 25, Bathroom);
	store.AddRoom(wc);

	building.AddStore(store);

	IRoom DepozitBloc = IRoom("Depozit bloc", 2, 1, 4, 100, Other);
	building.AddRoom(DepozitBloc);
	IRoom BirouBloc = IRoom("Birou bloc", 2, 1, 4, 25, Other);
	building.AddRoom(BirouBloc);
	IRoom CameraTehnica = IRoom("Camera tehnica", 2, 1, 4, 25, Other);
	building.AddRoom(CameraTehnica);

	cout << "=================\n";
	cout << building.ToString();
	cout << "-----------------\n";
	//print the price of every apartment with the nr of rooms
	for (size_t i = 0; i < building.GetNrOfApartments(); i++)
	{
		Apartment ap = building.GetApartment(i);
		cout << "Apartment " << i + 1 <<endl;
		cout << ap.ToString();
	}

	cout << "-----------------\n";
	cout << "\tStores:\n";
	cout << "-----------------\n";
	for (size_t i = 0; i < building.GetNrOfStores(); i++)
	{
		Store store = building.GetStore(i);
		cout << "\tStore " << i + 1 << endl;
		cout << store.ToString();
	}
```

```
	cout << "-----------------\n";
	cout << "Technical Rooms:\n";
	cout << "-----------------\n";
	for (size_t i = 0; i < building.GetNrOfRooms(); i++)
	{
		IRoom room = building.GetRoom(i);
		cout << "\tRoom " << i + 1 << endl;
		cout << room.ToString();
	}
}
```

Store.h

```
#pragma once

class Store
{
	private:
		string StoreName;
		double Price;
		vector<IRoom> Rooms;
	public:
		Store(string name)
		{
			StoreName = name;
			Rooms = vector<IRoom>();
		}

		void SetPrice(double price)
		{
			Price = price;
		}

		double GetPrice()
		{
			return Price;
		}

		void AddRoom(IRoom room)
		{
			Rooms.push_back(room);
		}

		void RemoveRoom(int index)
		{
			Rooms.erase(Rooms.begin() + index);
		}

		IRoom GetRoom(int index)
		{
			return Rooms[index];
		}

		string ToString()
		{
			string result = "\tName: " + StoreName + "\n";
			result += "\tPrice: " + to_string(Price) + "\n";
			result += "\tNr of rooms: " + to_string(Rooms.size()) +
"\n";
			return result;
		}
};
```

IRoom.h

```
#pragma once
#include <string>
using namespace std;
enum RoomType
{ Kitchen, LivingRoom, Bedroom, Bathroom, Garage, Garden, Other };

class IRoom{
	private:
		string RoomName;
		int NrOfDors;
		int NrOfWindows;
		int NrOfWalls;
		double Area;
		RoomType Type;
	public: IRoom(string roomName, int nrOfDors, int nrOfWindows, int
nrOfWalls, double area ,RoomType type)
		{
			RoomName = roomName;
			NrOfDors = nrOfDors;
			NrOfWindows = nrOfWindows;
			NrOfWalls = nrOfWalls;
			Area = area;
			Type = type;
		}
		string GetRoomName() { return RoomName; }
		int GetNrOfDors() { return NrOfDors; }
		int GetNrOfWindows() { return NrOfWindows; }
		int GetNrOfWalls() { return NrOfWalls; }
		RoomType GetType() { return Type; }
		double GetArea() { return Area; }
		void SetRoomName(string roomName) { RoomName = roomName; }
		void SetNrOfDors(int nrOfDors) { NrOfDors = nrOfDors; }
		void SetNrOfWindows(int nrOfWindows){ NrOfWindows = nrOfWindows; }
		void SetNrOfWalls(int nrOfWalls) { NrOfWalls = nrOfWalls; }
		void SetType(RoomType type) { Type = type; }
		void SetArea(double area) { Area = area; }
		string ToString()
		{
			string result = "Room name: " + RoomName + "\n";
			result += "Area: " + to_string(Area) + "\n";
			result += "Room type: ";
			switch (Type)
			{
			case Kitchen:
				result += "Kitchen";
				break;
			case LivingRoom:
				result += "Living room";
				break;
			case Bedroom:
				result += "Bedroom";
				break;
			case Bathroom:
				result += "Bathroom";
				break;
			case Garage:
				result += "Garage";
				break;
			case Other:
				result += "Other";
				break;
			default:
				break;
			}
			result += "\n";
			return result;
		}
```

Building.h

```
#pragma once
#include "IRoom.h"
#include "Apartment.h"
#include "Store.h"

using namespace std;

enum BuildingType {
	CivilBuilding,
	IndustrialBuilding,
	AgroBuilding
};

class Building
{
	private:
		BuildingType Type;
		string Name;
		string Address;
		vector<Apartment> Apartments;
		vector<IRoom> Rooms;
		vector<Store> Stores;
	public:
		Building(BuildingType type, string name, string address)
		{
			Type = type;
			Name = name;
			Address = address;
			Rooms = vector<IRoom>();
			Stores = vector<Store>();
			Apartments = vector<Apartment>();
		}
		void AddRoom(IRoom room)
		{
			Rooms.push_back(room);
		}
		void RemoveRoom(int index)
		{
			Rooms.erase(Rooms.begin() + index);
		}
		IRoom GetRoom(int index)
		{
			return Rooms[index];
		}
		void AddStore(Store store)
		{
			Stores.push_back(store);
		}
		void RemoveStore(int index)
		{
			Stores.erase(Stores.begin() + index);
		}
		Store GetStore(int index)
		{
			return Stores[index];
		}
		void AddApartment(Apartment apartment)
		{
			Apartments.push_back(apartment);
		}
		void RemoveApartment(int index)
		{
			Apartments.erase(Apartments.begin() + index);
		}
		Apartment GetApartment(int index)
		{
			return Apartments[index];
```

```
            int GetNrOfApartments()
            {
                  return Apartments.size();
            }
            int GetNrOfStores()
            {
                  return Stores.size();
            }
            int GetNrOfRooms()
            {
                  return Rooms.size();
            }
            string ToString()
            {
                  string res = "";
                  res += "Building type: ";
                  switch (Type)
                  {
                  case CivilBuilding:
                        res += "Civil";
                        break;
                  case IndustrialBuilding:
                        res += "Industrial";
                        break;
                  case AgroBuilding:
                        res += "Agro";
                        break;
                  default:
                        break;
                  }
                  res += "\nBuilding name: " + Name + "\n";
                  res += "Building address: " + Address + "\n";
                  res += "Number of stores: " + to_string(Stores.size()) +
"\n";
                  res += "Number of apartments: " + to_string(GetNrOfApart-
ments()) + "\n";
                  return res;
            }
};
```

Apartament.h

```
#pragma once
#include "IRoom.h"
#include <vector>

class Apartment
{
      private:
            vector<IRoom> Rooms;
            double PricePerSqMeter;
      public:
            Apartment(double price)
            {
                  Rooms = vector<IRoom>();
                  PricePerSqMeter = price;
            }
            void AddRoom(IRoom room)
            {
                  Rooms.push_back(room);
            }
            void RemoveRoom(int index)
            {
                  Rooms.erase(Rooms.begin() + index);
            }
```

```
            IRoom GetRoom(int index)
            {
                  return Rooms[index];
            }
            double GetTotalArea()
            {
                  double totalArea = 0;
                  for (int i = 0; i < Rooms.size(); i++)
                  {
                        totalArea += Rooms[i].GetArea();
                  }
                  return totalArea;
            }
            double GetPrice()
            {
                  return GetTotalArea() * PricePerSqMeter;
            }
            int GetNrOfRooms()
            {
                  return Rooms.size();
            }
            string ToString()
            {
                  string result = "\tNr of rooms: " + to_string(GetNrOf-
Rooms()) + "\n";
                  result += "\tTotal area: " + to_string(GetTotalArea()) +
"\n";
                  result += "\tPrice per sq meter: " + to_string(PricePerSqMe-
ter) + "\n";
                  result += "\tTotal price: " + to_string(GetPrice()) + "\n";
                  return result;
            }

};
```

Rezultatele testării

```
================
Building type: Civil
Building name: Bloc locativ
Building address: Chisinau, Decebal 19
Number of stores: 1
Number of apartments: 5
----------------
Apartment 1
        Nr of rooms: 5
        Total area: 130.000000
        Price per sq meter: 1199.000000
        Total price: 155870.000000
Apartment 2
        Nr of rooms: 5
        Total area: 130.000000
        Price per sq meter: 1199.000000
        Total price: 155870.000000
Apartment 3
        Nr of rooms: 5
        Total area: 130.000000
        Price per sq meter: 1199.000000
        Total price: 155870.000000
Apartment 4
        Nr of rooms: 5
        Total area: 130.000000
        Price per sq meter: 1199.000000
        Total price: 155870.000000
Apartment 5
        Nr of rooms: 5
        Total area: 130.000000
        Price per sq meter: 1199.000000
        Total price: 155870.000000
```

```
-----------------
        Stores:
-----------------
        Store 1
        Name: Magazin
        Price: 790.000000
        Nr of rooms: 4
-----------------
Technical Rooms:
-----------------
        Room 1
Room name: Depozit bloc
Area: 100.000000
Room type: Other
        Room 2
Room name: Birou bloc
Area: 25.000000
Room type: Other
        Room 3
Room name: Camera tehnica
Area: 25.000000
Room type: Other
```

4. Concluzii

Lucrarea de laborator a avut ca obiectiv aprofundarea conceptelor fundamentale ale programării orientate pe obiect (OOP), cu un accent deosebit pe utilizarea moștenirii, compoziției și gestionarea memoriei dinamice în C++.

În cadrul primei sarcini, a fost elaborată clasa de bază Mobila, care conține informații despre preț, stil și domeniul de utilizare, cu alocarea dinamică a memoriei pentru câmpurile textuale. Această clasă a fost extinsă prin intermediul moștenirii de către clasele derivate Masa și Scaun, ilustrând clar avantajele reutilizării codului și ale specializării comportamentale. Prin implementarea constructorilor, a destructorului și a operatorului de atribuire, s-a asigurat o gestionare corectă a memoriei, evitând scurgerile de resurse și asigurând o funcționalitate robustă.

A doua sarcină a evidențiat utilizarea compoziției și a moștenirii multiple într-un context mai complex. Clasa Casa a fost compusă din elemente precum Garaj și bucătărie, ilustrând relația de tip „are un”, în timp ce clasa Vila a fost derivată din Casa, extinzând structura prin adăugarea unui nou atribut – mărimea lotului de pământ. Astfel, s-a consolidat înțelegerea diferențelor între compoziție și moștenire, dar și a modului în care acestea pot fi combinate într-un sistem coerent.

Realizarea acestei lucrări a demonstrat în mod practic beneficiile OOP, precum modularitatea, extensibilitatea, claritatea semantică a relațiilor între clase și ușurința în întreținerea codului. Moștenirea a permis extinderea elegantă a funcționalității fără duplicare inutilă de cod, în timp ce compoziția a oferit un model flexibil de construire a obiectelor complexe din componente mai simple. Prin integrarea acestor principii, au fost create structuri de date coerente, expresive și scalabile, relevante pentru scenarii din viața reală. Din perspectivă personală, consider că această experiență a contribuit semnificativ la consolidarea înțelegerii mele asupra principiilor OOP, oferindu-mi în același timp încredere în aplicarea acestora în proiecte practice, concrete. Prin abordarea progresivă a cerințelor, am reușit să îmbin teoria cu practica într-un mod eficient și formativ.

5. Webobrafie

- https://else.fcim.utm.md/course/view.php?id=663
- https://www.w3schools.com/cpp/cpp_oop.asp
- https://www.programiz.com/cpp-programming/oop,
- https://code.visualstudio.com/docs/cpp/config-linux
- https://chatgpt.com/