* + Definišite apstraktnu klasu **FlyingObject** koja predstavlja osnovu za leteće objekte. Klasa treba da sadrži:
    - **Atribut:** name (tipa **std::string**), koji predstavlja ime letećeg objekta.
    - **Apstraktnu metodu:** **fly()** koja simulira let objekta i vraća **void**.
    - **Konstruktor** koji postavlja vrednost atributa name.

**Zadatak:**

* Definišite dve klase Airplane i Helicopter, koje nasleđuju apstraktnu klasu FlyingObject.
  + **Airplane:**

Implementirajte metodu fly() tako da ispisuje "Avion [name] leti na visini od 10000m."

* + **Helicopter:**

Implementirajte metodu fly() tako da ispisuje "Helikopter [name] leti na visini od 2000m."

* + - **Definišite main funkciju u kojoj:**

**Kreirate kolekciju objekata klase FlyingObject**

**U kolekciju dodajte nekoliko aviona i helikoptera.**

**Implementirajte petlju koja prolazi kroz kolekciju i poziva metodu fly() za svaki objekat.**

**Sortiranje kolekcije po imenu**

* Implementirajte funkciju za sortiranje kolekcije letećih objekata po imenu.
  + Koristite **std::sort** i lambda funkciju za poređenje imena objekata.
  + Nakon sortiranja, ponovo ispišite imena letećih objekata.

**Pretraga objekata po imenu**

* Implementirajte funkciju koja omogućava pretragu letećih objekata po imenu koristeći **std::find\_if** i lambda funkciju.
  + Ako objekat postoji, ispišite poruku o objektu i pozovite njegovu metodu **fly()**.
  + Ako objekat ne postoji, ispišite poruku da nije pronađen.

**Definišite novu klasu Drone, koja nasleđuje FlyingObject.**

Implementirajte metodu **fly()** za dronove da ispisuje "Dron [name] leti autonomno na visini od 500m."

Dodajte nekoliko dronova u kolekciju i ponovo prikažite polimorfno ponašanje i sortiranje.

**Zadatak:** Podeliti kolekciju letećih objekata tako da svi Drone objekti budu na početku kolekcije, a ostali objekti iza njih.

**Zadatak:** Prebrojati koliko ima Airplane objekata u kolekciji. Koristiti std::count\_if i lambda izraz koji koristi dynamic\_cast

**Zadatak:** Naći prvi Airplane u kolekciji koji u imenu sadrži ime "Airline-32". Koristiti std::find\_if i lambda izraz koji kombinuje dynamic\_cast

**Zadatak:** koristeći std::stable\_sort sortiraj po imenu objekte.

Zadatak: sortiranje prva 3 objekta

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// 1. Apstraktna klasa

class FlyingObject {

protected:

string name;

public:

FlyingObject(string n) : name{ n } {}

virtual void fly() const = 0;

virtual ~FlyingObject() {}

string getName() const { return name; }

};

// 2. Izvedene klase

class Airplane : public FlyingObject {

public:

Airplane(string n) : FlyingObject(n) {}

void fly() const override {

cout << "Avion [" << name << "] leti na visini od 10000m" << endl;

}

};

class Helicopter : public FlyingObject {

public:

Helicopter(string n) : FlyingObject(n) {}

void fly() const override {

cout << "Helicoper [" << name << "] leti na visini od 30000m" << endl;

}

};

// 4. funkcija za sortiranje po imenu

void sortByName(vector<FlyingObject\*>& collection) {

sort(collection.begin(), collection.end(),

[](FlyingObject\* a, FlyingObject\* b) { return a->getName() < b->getName(); });

}

// 5. Funkcija za pretragu po imenu

void searchByName(const vector<FlyingObject\*>& collection, const string& targetName) {

auto it = find\_if(collection.begin(), collection.end(),

[&targetName](FlyingObject\* obj) { return obj->getName() == targetName; });

if (it != collection.end()) {

(\*it)->fly();

}

else {

cout << targetName << " objekat nije pronadjen" << endl;

}

}

// 6. Dodavanje klase Drone

class Drone : public FlyingObject

{

public:

Drone(string n) : FlyingObject(n) {}

virtual void fly() const override

{

cout << "Dron [" << name << "] leti autonomno na visini od 500m" << endl;

}

};

// 7. funkcija stavljanja prvo dronova na pocetku vektora

void sortParticion(vector<FlyingObject\*>& collection) {

auto part= partition(collection.begin(), collection.end(),

[](FlyingObject\* obj) {

return dynamic\_cast<Drone\*>(obj) != nullptr;

});

}

// funkcija broji avione

int countAirplane(const vector<FlyingObject\*>& numb) {

int count = count\_if(numb.begin(), numb.end(), [](FlyingObject\* obj) {

return dynamic\_cast<Airplane\*>(obj) != nullptr;

});

return count;

}

// stabilnim sortiranjem , oni koji su prvi dosli oni ce i biti prvi

void sortByName\_StableSort(vector<FlyingObject\*>& objects) {

stable\_sort(objects.begin(), objects.end(),

[](FlyingObject\* a, FlyingObject\* b) {

return a->getName() < b->getName();

});

}

// sortiranje prva 3 objekta

void partial\_Sort(vector<FlyingObject\*>& o) {

partial\_sort(o.begin(), o.begin() + 3, o.end(), [](FlyingObject\* a, FlyingObject\* b) {return a->getName() < b->getName();});

}

// Da li se u kolekciji nalazi avion po umenu "Airline-32"

void find\_Airplane\_name(vector<FlyingObject\*>& o, const string& targetName) {

auto iterator= find\_if(o.begin(), o.end(),

[&](FlyingObject\* obj)

{ Airplane\* a = dynamic\_cast<Airplane\*>(obj);

return a != nullptr && a->getName() == targetName;});

if (iterator != o.end())

(\*iterator)->fly();

else

cout << "Nije pronadjen avion sa imenom "<<targetName<<endl;

}

int main() {

// kolekcija objekata

vector<FlyingObject\*> flyingObjects;

// dodavanje objekata

flyingObjects.push\_back(new Airplane("Boeing"));

flyingObjects.push\_back(new Airplane("Airbus"));

flyingObjects.push\_back(new Helicopter("Apache"));

flyingObjects.push\_back(new Helicopter("BlackHawk"));

flyingObjects.push\_back(new Airplane("A"));

flyingObjects.push\_back(new Helicopter("K"));

cout << "--- prikaz letenja objekata ---" << endl;

for (auto obj : flyingObjects) {

obj->fly();

}

sortByName(flyingObjects);

cout << "\n\n--- prikaz letenja objekata posle sortiranja ---" << endl;

for (auto obj : flyingObjects) {

obj->fly();

}

cout << "\n--------Pretraga objekta po imenu--------" << endl;

searchByName(flyingObjects, "Boeing");

searchByName(flyingObjects, "DJI Drone");

cout << "\n------Dodavanje dronova ---" << endl;

flyingObjects.push\_back(new Drone("DJI Phantom"));

flyingObjects.push\_back(new Drone("Parrot Anafi"));

for (auto obj : flyingObjects) {

obj->fly();

}

cout << "\n----------------particija prvo Dronovi----------" << endl;

sortParticion(flyingObjects);

for (auto p : flyingObjects)

p->fly();

cout << "\n" << endl;

int brojAviona = countAirplane(flyingObjects);

cout << "Ukupan broj aviona je: " << brojAviona << endl;

cout << "\n--------------- Sortiranja prva 3 elementa po imenu--------" << endl;

partial\_Sort(flyingObjects);

for (auto p : flyingObjects)

p->fly();

cout << "\n--------------- stable sort imena--------" << endl;

sortByName\_StableSort(flyingObjects);

for (auto p : flyingObjects)

p->fly();

cout << "\n--------------- Pronalazenja aviona Airline - 32 --------" << endl;

find\_Airplane\_name(flyingObjects, "Airline-32");

}

**const string&:**

* **string**: Ovaj tip predstavlja niz karaktera — u ovom slučaju string, tj. tekst, kao što je "Airline-32".
* **&**: Ovaj znak označava **referencu**, što znači da se ne pravi kopija objekta, već se koristi originalni objekat. Dakle, kad prosleđuješ objekat funkciji kao referencu, ne moraš da praviš novu kopiju te vrednosti, već samo "pokazuješ" na postojeći objekat.
* **const**: Ovaj ključna reč znači da se **vrednost objekta ne može menjati** unutar funkcije. To znači da ne možeš da menjaš originalni string unutar funkcije. Ako bi pokušao, dobio bi grešku.

**Zadatak 1: Apstraktna klasa Shape**

Definišite apstraktnu klasu Shape, koja predstavlja osnovu za različite oblike. Klasa treba da sadrži:

* Atribut: color (tipa std::string), koji predstavlja boju oblika.
* Apstraktnu metodu: area(), koja vraća površinu oblika (tipa double).
* Apstraktnu metodu: perimeter(), koja vraća obim oblika (tipa double).
* Konstruktor koji postavlja vrednost atributa color.

Zadatak:

* Definišite dve klase Circle i Rectangle, koje nasleđuju apstraktnu klasu Shape.
  + Circle: Implementirajte metodu area() tako da računa površinu kruga (pi \* radius^2).
  + Circle: Implementirajte metodu perimeter() tako da računa obim kruga (2 \* pi \* radius).
  + Rectangle: Implementirajte metodu area() tako da računa površinu pravougaonika (width \* height).
  + Rectangle: Implementirajte metodu perimeter() tako da računa obim pravougaonika (2 \* (width + height)).
* U main funkciji kreirajte kolekciju objekata klase Shape (krugova i pravougaonika).
* Implementirajte petlju koja prolazi kroz kolekciju i ispisuje površinu i obim svakog oblika.

**Zadatak 2: Sortiranje oblika prema površini**

* Koristite std::sort i lambda funkciju da sortirate kolekciju oblika prema njihovoj površini.
* Nakon sortiranja, ispišite boje i površine oblika.

**Zadatak 3: Pretraga oblika po boji**

* Implementirajte funkciju koja omogućava pretragu oblika po boji koristeći std::find\_if i lambda funkciju.
* Ako oblik sa tom bojom postoji, ispišite informaciju o njemu (boja, površina, obim).
* Ako oblik ne postoji, ispišite poruku da nije pronađen.

**Zadatak 4: Polimorfizam sa više oblika**

* Dodajte nove klase Triangle i Square koje nasleđuju klasu Shape.
  + Triangle: Implementirajte metodu area() za trougao (0.5 \* base \* height).
  + Square: Implementirajte metodu area() za kvadrat (side \* side).
* Kreirajte kolekciju svih oblika i pozovite polimorfnu metodu area() za svaki oblik.

**Zadatak 5: Prebrojavanje objekata u kolekciji**

* Koristite std::count\_if i lambda izraz koji koristi dynamic\_cast da prebrojite koliko ima objekata klase Rectangle u kolekciji.

**Zadatak 6: Pretraga objekta po površini**

* Nađite prvi oblik u kolekciji koji ima površinu veću od 100. Koristite std::find\_if i lambda izraz koji kombinuje dynamic\_cast.

**Zadatak 1: Rad sa klasama Employee i Manager**

Definišite apstraktnu klasu Employee koja sadrži:

* Atribut: name (tipa std::string).
* Apstraktnu metodu: salary() koja vraća godišnju platu zaposlenog (tipa double).
* Konstruktor koji postavlja vrednost atributa name.

Zadatak:

* Definišite klasu Manager koja nasleđuje Employee i implementira metodu salary(), vraćajući godišnju platu menadžera (osnovna plata + bonus).
* Definišite klasu Developer koja nasleđuje Employee i implementira metodu salary(), vraćajući godišnju platu programera (osnovna plata + bonus za projekte).
* U main funkciji kreirajte kolekciju objekata klase Employee (menadžeri i programeri).
* Implementirajte petlju koja prolazi kroz kolekciju i ispisuje ime i platu svakog zaposlenog.

**Zadatak 2: Sortiranje zaposlenih prema plati**

* Koristite std::sort i lambda funkciju za sortiranje zaposlenih prema njihovoj plati.
* Nakon sortiranja, ispišite imena i plate zaposlenih.

**Zadatak 3: Pretraga zaposlenog po imenu**

* Implementirajte funkciju koja omogućava pretragu zaposlenih po imenu koristeći std::find\_if i lambda funkciju.
* Ako zaposleni postoji, ispišite informacije o njemu (ime, pozicija, plata).
* Ako zaposleni ne postoji, ispišite poruku da nije pronađen.

**Zadatak 4: Prebrojavanje tipova zaposlenih**

* Koristite std::count\_if i lambda izraz sa dynamic\_cast da prebrojite koliko ima zaposlenih tipa Manager u kolekciji.