Решени задачи C++

Предавање 1

1. Напиши С++ програм кој користи **inline** функција **circleArea** која бара влез од корисникот за радиус на круг, ја пресметува и ја печати површината на кругот.

double circleArea(double radius) {

return radius \* radius \* 3.14;

}

int main() {

double radius;

cin>>radius;

cout<<"Plostinata na krugot e "<<circleArea(radius)<<endl;

return 0;

}

1. Нaпиши С++ програм со две посебни функции кои едноставно ја триплираат вредноста на променливата **count** дефинирана во **main**. Овие функции се:
   1. Функцијата **tripleCallByValue** која предава копија од **count**, ја триплира нејзината вредност и ја враќа новата вредност
   2. Функцијата **tripleCallByReference** која ja предава **count** како референцен параметар, ја триплира самата променлива и ја враќа новата вредност на **count**

int tripleCallByValue(int value) {

return value \* 3;

}

void tripleCallByReference(int &value) {

value \*= 3;

}

int main() {

int value;

cin>>value;

tripleCallByValue(value);

cout<<value<<endl;

tripleCallByReference(value);

cout<<value<<endl;

return 0;

}

Предавање 2

1. Напиши програм кој кој користи функциски урнек **min** за да го определи помалиот од двата аргументи. Тестирај го прогамот со цели броеви, реални броеви и карактери.

template < class T >

T minimum( T value1, T value2, T value3 )

{

T min = value1;

if ( value2 < min )

min = value2;

if ( value3 < min )

min = value3;

return min;

}

int main() {

int int1, int2, int3;

cout << "Vnesi tri celi broja: ";

cin >> int1 >> int2 >> int3;

cout << minimum( int1, int2, int3 );

double double1, double2, double3;

cout << "\nVnesi 3 double vrednosti: ";

cin >> double1 >> double2 >> double3;

cout << minimum( double1, double2, double3 );

char char1, char2, char3;

cout << "\nVnesi tri karakteri: ";

cin >> char1 >> char2 >> char3;

cout << minimum( char1, char2, char3 )<< endl;

}

Предавање 3

1. Да се формира класа Квадар (скриени елементи: должина, ширина и висина; јавни функции: плоштина, волумен). Од неа да се имплементира поле од објекти. Да се најде оној кој има најмала плоштина, а не е коцка.

1. Креирај нов објект
2. Постави ги димензиите
3. Печати ги димензиите на сите објекти
4. Печати ги димензиите за поедин објект
5. Печати ги оние кои го задоволуваат условот

class Kvadar {

private:

double dolzina;

double sirina;

double visina;

public:

Kvadar(float dolzina, float sirina, float visina) : dolzina(dolzina), sirina(sirina), visina(visina) {}

float Plostina() {

return 2 \* (dolzina \* sirina + dolzina \* visina + sirina \* visina);

}

float Volumen() {

return visina \* sirina \* dolzina;

}

void Pecati() {

cout<<"Dolzina "<<dolzina<<", Sirina "<<sirina<<", Visina "<<visina<<endl;

}

};

1. Да се направи програма која ќе ја користи класата Штедач и ќе овозможи:
2. Внесување на нов штедач во системот
3. Приказ на податоците
4. Уплата на средства
5. Исплата на средства (со контрола дали е можна таква исплата)
6. Приказ на салдото за даден штедач

class Stedac

{

private:

int broj;

char imeprezime[30], adresa[50];

long saldo;

public:

void vnesi\_podatoci();

void prikazi\_podatoci();

void uplata(unsigned long i);

void isplata(unsigned long i);

long sostojba();

};

void Stedac::vnesi\_podatoci() {

cin >> broj;

cin.ignore();

cin.getline(imeprezime, sizeof(imeprezime));

cin.getline(adresa, sizeof(adresa));

cin >> saldo;

}

void Stedac::prikazi\_podatoci() {

cout << "Broj: " << broj << ", Ime i prezime: " << imeprezime

<< ", Adresa: " << adresa << ", Saldo: " << saldo <<endl;

}

void Stedac::uplata(unsigned long i) {

saldo += i;

cout << "Uplata izvrsena. Novo saldo: " << saldo <<endl;

}

void Stedac::isplata(unsigned long i) {

if (saldo >= i) {

saldo -= i;

cout << "Isplata izvrsena. Novo saldo: " << saldo <<endl;

} else {

cout << "Nemate dovolno sredstva za isplata." <<endl;

}

}

3. Создади класа **Dropki** која извршува операции со дропки. Напиши „драјвер“ за твојот програм да манипулира со оваа класа.

Користи целоборјни променливи за броител и именител кои мора да се приватни. Обезбеди потребен број на контруктори кои нема да дозволат дефинирање на грешни вредности (со 0 не се дели!). По секоја од наредните операции, дропката треба да се зачува во редуцирана форма, т.е дропката 2/4 треба да се зачува како 1/2. Обезбеди ги следните јавни функции членки:

а) Собирање на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма.

б) Одземање на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

в) Множење на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

г) Делење на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

д) Печатење на објектот во форма **a/b**.

class Dropki {

private:

int broitel;

int imenitel;

void poednostavi() {

int GDC = gcd(broitel, imenitel);

broitel /= GDC;

imenitel /= GDC;

if (imenitel < 0) {

broitel = -broitel;

imenitel = -imenitel;

}

}

public:

Dropki() : broitel(0), imenitel(1) {}

Dropki(int broitel, int imenitel) {

if(imenitel == 0)

throw invalid\_argument("Именителот не може да биде 0");

this->broitel = broitel;

this->imenitel = imenitel;

poednostavi();

}

void pecati() const {

cout << broitel << "/" << imenitel << endl;

}

Dropki soberi(const Dropki& other) const {

int novBroitel = broitel \* other.imenitel + other.broitel \* imenitel;

int novImenitel = imenitel \* other.imenitel;

return Dropki(novBroitel, novImenitel);

}

Dropki odzemi(const Dropki& other) const {

int novBroitel = broitel \* other.imenitel - other.broitel \* imenitel;

int novImenitel = imenitel \* other.imenitel;

return Dropki(novBroitel, novImenitel);

}

Dropki mnozenje(const Dropki& other) const {

int novBroitel = broitel \* other.broitel;

int novImenitel = imenitel \* other.imenitel;

return Dropki(novBroitel, novImenitel);

}

Dropki mnozenje(const Dropki& other) const {

int novBroitel = broitel \* other.imenitel;

int novImenitel = imenitel \* other.broitel;

return Dropki(novBroitel, novImenitel);

}

};

4. Во една зоолошка градина се води евиденција за различни типови животни кои се чуваат во неа: птици, влекачи, цицачи.... Користејќи ги концептите на ООП да се моделира класа според спецификациите дадени на слика 5.1. Цицачите се вакцинираат секоја година, птиците на секои 6 месеци, а влекачите на секои 8 месеци. Датумите да се дефинираат во посебна класа **Datum** со приватни податочни членки **den**, **mesec** и **godina**. Да се имплементира кориснички посредник преку мени со следните опции:

1. Додавање на животно во зоолошка градина
2. Прикажување на бројна состојба
3. Прикажување на животни од дадена група (тип: **c**- цицачи, **v**-влекачи, **p**-птици)
4. Прикажување на целата листа
5. Прикажување на сите животни кои треба да се вакцинираат во рок од 1 недела
6. Излез од менито

enum SEX {

M,

Z

};

ostream& operator<<(ostream& os, SEX sex) {

switch (sex) {

case M: os << "M"; break;

case Z: os << "Z"; break;

default: os << "Unknown Sex"; break;

}

return os;

}

enum TYPE {

CICACI,

VLEKACI,

PTICI

};

ostream& operator<<(ostream& os, TYPE type) {

switch (type) {

case CICACI: os << "CICACI"; break;

case VLEKACI: os << "VLEKACI"; break;

case PTICI: os << "PTICI"; break;

default: os << "Unknown Type"; break;

}

return os;

}

class Date {

private:

int day;

int month;

int year;

public:

Date() {}

Date(int d, int m, int y) : day(d), month(m), year(y) {}

friend std::ostream& operator<<(std::ostream& os, const Date& date) {

os << date.day << "/" << date.month << "/" << date.year;

return os;

}

int operator-(const Date& today) const {

int days1 = this->year \* 365 + this->month \* 30 + this->day;

int days2 = today.year \* 365 + today.month \* 30 + today.day;

return days2 - days1;

}

Date& operator=(const Date& other) {

if(this != &other) {

day = other.day;

month = other.month;

year = other.year;

}

return \*this;

}

Date nextVaccine(int period) const {

int newDay = day;

int newMonth = month + period;

int newYear = year;

if(newMonth > 12) {

newYear += 1;

newMonth -= 12;

}

return Date(newDay, newMonth, newYear);

}

};

class Animal {

private:

const Date birthday;

const Date arrival;

SEX sex;

TYPE type;

char name[20];

Date last\_vaccine;

public:

Animal(Date b, Date a, SEX sex, TYPE type, char name[], Date last\_vaccine) : birthday(b), arrival(a) {

this->sex = sex;

this->type = type;

strcpy(this->name, name);

this->last\_vaccine = last\_vaccine;

}

void print() const {

cout<<"Name:"<<name<<" Type:"<<type<<" Sex:"<<sex<<" Birthday:"<<this->birthday<<" Arrival:"<<this->arrival<<" Last Vaccine:"<<this->last\_vaccine<<endl;

}

void calculateVaccineDate(Date today) const {

if(this->type == CICACI) {

if(this->last\_vaccine.nextVaccine(6) - today <= 7)

this->print();

} else if(this->type == PTICI) {

if(this->last\_vaccine.nextVaccine(8) - today <= 7)

this->print();

} else if(this->type == VLEKACI) {

if(this->last\_vaccine.nextVaccine(12) - today <= 7)

this->print();

}

}

void printByType(TYPE type) const {

if(this->type == type)

this->print();

}

};

int main() {

vector<Animal> Zoo;

Animal animal = Animal(Date(1,1,2012),Date(4,8,2016),M,CICACI,"Joe",Date(18,11,2024));

Animal animal2 = Animal(Date(2,2,2013),Date(4,8,2016),M,PTICI,"Boe",Date(21,11,2024));

//Add animals in the ZOO

Zoo.emplace\_back(animal);

Zoo.emplace\_back(animal2);

//Size

cout<<"Number of animals in the ZOO: "<<Zoo.size()<<endl;

//Print by type

TYPE type = CICACI;

for (const auto& animal : Zoo) {

animal.printByType(type);

}

//Print all the animals in the ZOO

for (const auto& animal : Zoo) {

animal.print();

}

//Print the animals that need to be vaccine in a week

Date today = Date(25,11,2024);

for (const auto& animal : Zoo) {

animal.calculateVaccineDate(today);

}

}

Предавање 4

1. Создади класа Stedac во која ќе користиш статичка податочна членка **kamata** која ќе ја чува вредноста на годишната камата за сите штедачи и статична податочна членка **brojStedaci** кој во секој момент кажува колку штедачи има во банката. Останатите податочни членки се **lice**, **bilans** и **kredit**. Името, презимето, адресата и телефонот на секој објeкт од класата **Stedac** се сместени во приватна податочна членка **lice** која е од класaта **Covek**.

class Covek {

public:

string ime;

string prezime;

string adresa;

string broj;

Covek() {}

Covek(string ime, string prezime, string adresa, string broj) {

this->ime = ime;

this->prezime = prezime;

this->adresa = adresa;

this->broj = broj;

}

friend ostream& operator<<(ostream& os, const Covek& covek) {

os<<"Ime: "<<covek.ime<<", Prezime: "<<covek.prezime<<", Broj: "<<covek.broj<<endl;

return os;

}

Covek& operator=(const Covek& other) {

if(this != &other) {

ime = other.ime;

prezime = other.prezime;

adresa = other.adresa;

broj = other.broj;

}

return \*this;

}

};

class Stedac {

public:

static float kamata;

static int brojStedaci;

Covek lice;

float bilans;

float kredit;

Stedac(Covek lice, float bilans, float kredit) {

this->lice = lice;

this->bilans = bilans;

this->kredit = kredit;

}

friend ostream& operator<<(ostream& os, const Stedac& stedac) {

os << "Kamata: " << Stedac::kamata << ", Broj na stedaci: " << Stedac::brojStedaci

<< ", Lice: " << stedac.lice << ", Bilans: " << stedac.bilans

<< ", Kredit: " << stedac.kredit << endl;

return os;

}

};

float Stedac::kamata = 3.4;

int Stedac::brojStedaci = 4;

int main() {

Covek covek = Covek("Filip","Petkovski","Adresa","078907908");

Stedac stedac1 = Stedac(covek, 5000.20, 30000.20);

Stedac stedac2 = Stedac(covek, 5000.20, 30000.20);

cout<<stedac1;

cout<<stedac2;

stedac1.kamata = 10;

cout<<stedac1;

cout<<stedac2;

return 0;

}

Предавање 5

1. Создади класа **Dropki** која извршува операции со дропки. Напиши „драјвер“ за твојот програм да манипулира со оваа класа.

Користи целоборјни променливи за броител и именител кои мора да се приватни. Обезбеди потребен број на контруктори кои нема да дозволат дефинирање на грешни вредности (со 0 не се дели). По секоја од наредните операции, дропката треба да се зачува во редуцирана форма, т.е дропката 2/4 треба да се зачува како 1/2. Обезбеди ги следните операторски функции членки:

а) Собирање на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма.

б) Одземање на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

в) Множење на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

г) Делење на два објекти од класа **Dropki**. Резултатот да се даде во редуцирана форма

д) Собирање на објект од класа **Dropki** со цел број. Резултатот да се даде во редуцирана форма.

ѓ) Одземање на објект од класа **Dropki** со цел број. Резултатот да се даде во редуцирана форма.

е) Множење на објект од класа **Dropki** со цел број. Резултатот да се даде во редуцирана форма.

ж) Делење на објект од класа **Dropki** со цел број. Резултатот да се даде во редуцирана форма.

Да се обезбедат и следните функции членки на класата:

з) Печатење на објектот во форма **a/b**.

class Dropki {

private:

int broitel;

int imenitel;

int gcd(int a, int b) {

while (b != 0) {

int temp = b;

b = a % b;

a = temp;

}

return a;

}

void poednostavi() {

int GDC = gcd(broitel, imenitel);

broitel /= GDC;

imenitel /= GDC;

if (imenitel < 0) {

broitel = -broitel;

imenitel = -imenitel;

}

}

public:

Dropki() {}

Dropki(int broitel, int imenitel) {

if(imenitel == 0)

throw new invalid\_argument("Invalid imenitel");

this->broitel = broitel;

this->imenitel = imenitel;

this->poednostavi();

}

friend Dropki operator+(const Dropki& dropka1, const Dropki& dropka2) {

int newBroitel = dropka1.broitel \* dropka2.imenitel + dropka1.imenitel \* dropka2.broitel;

int newImenitel = dropka1.imenitel \* dropka2.imenitel;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator-(const Dropki& dropka1, const Dropki& dropka2) {

int newBroitel = dropka1.broitel \* dropka2.imenitel - dropka1.imenitel \* dropka2.broitel;

int newImenitel = dropka1.imenitel \* dropka2.imenitel;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator\*(const Dropki& dropka1, const Dropki& dropka2) {

int newBroitel = dropka1.broitel \* dropka2.broitel;

int newImenitel = dropka1.imenitel \* dropka2.imenitel;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator/(const Dropki& dropka1, const Dropki& dropka2) {

int newBroitel = dropka1.broitel \* dropka2.imenitel;

int newImenitel = dropka1.imenitel \* dropka2.broitel;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator+(const Dropki& dropka1, int i) {

int newBroitel = dropka1.broitel \* i + dropka1.imenitel \* i;

int newImenitel = dropka1.imenitel \* i;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator-(const Dropki& dropka1, int i) {

int newBroitel = dropka1.broitel \* i - dropka1.imenitel \* i;

int newImenitel = dropka1.imenitel \* i;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator\*(const Dropki& dropka1, int i) {

int newBroitel = dropka1.broitel \* i;

int newImenitel = dropka1.imenitel \* i;

return Dropki(newBroitel, newImenitel);

}

friend Dropki operator/(const Dropki& dropka1, int i) {

int newBroitel = dropka1.broitel \* i;

int newImenitel = dropka1.imenitel \* i;

return Dropki(newBroitel, newImenitel);

}

friend ostream& operator<<(ostream& os, const Dropki& dropka) {

os<<dropka.broitel<<"/"<<dropka.imenitel<<endl;

return os;

}

};

int main() {

Dropki dropka1 = Dropki(2,5);

Dropki dropka2 = Dropki(3,7);

Dropki sobiranje = dropka1 + dropka2;

sobiranje = dropka1 - dropka2;

sobiranje = dropka1 \* dropka2;

sobiranje = dropka1 / dropka2;

sobiranje = dropka1 + 3;

sobiranje = dropka1 - 3;

sobiranje = dropka1 \* 3;

sobiranje = dropka1 / 3;

cout<<sobiranje<<endl;

return 0;

}

2. Нека е дадена класата **Complex** со кодовите презентирани подолу (дефиниција на класа, дефиниција на функции цхленки и драјвер за класатa **Complex**). Оваа класа овозможува операции со комплексни броеви. Тоа се броеви од облик realenDel + imaginarenDel \* i, каде i има вредност √1

а) да се модифицира класата така да овозможи влез и излез на комплексни броеви преку преклопување на операторите **<<** и **>>.** Ова треба да ја замени функцијата **print()** од класата, која во вашето решение не треба да постои.

б) да се преклопи оператор за множење кој ќе овозможи множење на: два комплексни броја, комплексен со цел број и цел број со комплексен број.

в) да се преклопат операорите **==** и **!=** кои ќе овозможат споредување на два комплексни броја.

г) да се преклопат операторите **+=**, **-=** и **\*=** со помош на ваќе направените функции за преклопување на **+**, **-** и **\***.

friend ostream& operator<<(ostream& os, const Complex& complex) {

os<< '(' << complex.real << ", " << complex.imaginary << ')';

return os;

}

friend Complex operator\*(const Complex& complex1, const Complex& complex2) {

return Complex(complex1.real\*complex2.real, complex1.imaginary\*complex2.imaginary);

}

friend bool operator==(const Complex& complex1, const Complex& complex2) {

if(complex1.real == complex2.real && complex2.imaginary == complex1.imaginary)

return true;

else

false;

}

friend bool operator!=(const Complex& complex1, const Complex& complex2) {

if(complex1 == complex2)

return false;

else

return true;

}

Complex operator+=(const Complex& complex2) {

\*this = \*this+complex2;

}

Complex operator-=(const Complex& complex1, const Complex& complex2) {

\*this = \*this-complex2;

}

Complex operator\*=(const Complex& complex1, const Complex& complex2) {

\*this = \*this \* complex2;

}

Предавање 6

1. Користејќи ги концептите на ООП да се моделираат соодветни класи според спецификациите дадени на слика 1, а потоа во функцијата main() да се констуира низа со објекти од долунаведените класи (професор, асистент, демонстратор) и имплементира кориснички посредник преку мени со следните опции:

1.    Додавање на нов вработен во листата

2.    Прикажување на листата

3.    Крај

class Vraboten {

private:

string name;

int arrivalYear;

public:

Vraboten() {}

Vraboten(string name, int arrivalYear) {

this->name = name;

this->arrivalYear = arrivalYear;

}

string getName() {

return name;

}

int getYear() {

return arrivalYear;

}

void postavi() {

}

virtual void print() {

cout<<"Name: "<<name<<", Arrival year:"<<arrivalYear;

}

};

class Professor : public Vraboten {

private:

string zvanje;

string oblast;

int brPredmeti;

public:

Professor() {}

Professor(string name, int year, string zvanje, string oblast, int brPredmeti)

:Vraboten(name,year) {

this->zvanje = zvanje;

this->oblast = oblast;

this->brPredmeti = brPredmeti;

}

void print() override {

Vraboten::print();

cout << " Zvanje: " << zvanje << ", Oblast: " << oblast << ", Number of Subjects: " << brPredmeti<<endl;

}

Professor operator=(Professor& prof) {

return Professor(prof.getName(), prof.getYear(), prof.zvanje, prof.oblast,prof.brPredmeti);

}

};

class Asssistant : public Vraboten {

private:

string zvanje;

Professor mentor;

int brPredmeti;

public:

Asssistant(string name, int year, string zvanje, Professor mentor, int brPredmeti)

:Vraboten(name,year) {

this->zvanje = zvanje;

this->mentor = mentor;

this->brPredmeti = brPredmeti;

}

void print() override {

Vraboten::print();

cout << " Zvanje: " << zvanje << " Mentor: "<<mentor.getName()<<", Number of Subjects: " << brPredmeti << endl;

}

};

class Demonstrator : public Vraboten {

private:

int od;

int doo;

public:

Demonstrator(string name, int year, int od, int doo)

:Vraboten(name,year) {

this->od = od;

this->doo = doo;

}

void print() override {

Vraboten::print();

cout<<"Od: "<<od<< " Do: "<<doo<<endl;

}

};

int main() {

list<Vraboten\*> lista;

Professor prof = Professor("Pavle",2023,"Programer","IT",4);

Asssistant assis = Asssistant("Petar",2024,"Programer",prof,2);

// prof.print();

// assis.print();

lista.push\_back(&prof);

lista.push\_back(&assis);

for(auto \*vraboten : lista) {

vraboten->print();

}

return 0;

}

1. Во една зоолошка градина се води евиденција за животните кои се чуваат во неа: птици, влекачи и цицачи. Користејќи ги концептите на ООП да се моделираат соодветни класи според спецификациите дадени на слика 1. Цицачите се вакцинираат секоја година, птиците на секои 6 месеци, а за влекачите овој податок варира. Во main() функцијата да се конструира низа од долунаведените класи (влекачи, птици и цицачи) и да се имплементира кориснички посредник преку мени со следните опции:

enum SEX {

M,

Z

};

ostream& operator<<(ostream& os, SEX sex) {

switch (sex) {

case M: os << "M"; break;

case Z: os << "Z"; break;

default: os << "Unknown Sex"; break;

}

return os;

}

class Date {

private:

int day;

int month;

int year;

public:

Date() {}

Date(int d, int m, int y) : day(d), month(m), year(y) {}

friend std::ostream& operator<<(std::ostream& os, const Date& date) {

os << date.day << "/" << date.month << "/" << date.year;

return os;

}

int operator-(const Date& today) const {

int days1 = this->year \* 365 + this->month \* 30 + this->day;

int days2 = today.year \* 365 + today.month \* 30 + today.day;

return days1 - days2;

}

Date& operator=(const Date& other) {

if(this != &other) {

day = other.day;

month = other.month;

year = other.year;

}

return \*this;

}

Date nextVaccine(int period) const {

int newDay = day;

int newMonth = month + period;

int newYear = year;

if(newMonth > 12) {

newYear += 1;

newMonth -= 12;

}

return Date(newDay, newMonth, newYear);

}

};

class Animal {

private:

const Date birthday;

const Date arrival;

SEX sex;

string name;

Date last\_vaccine;

public:

Animal(Date b, Date a, SEX sex, string name, Date last\_vaccine) : birthday(b), arrival(a) {

this->sex = sex;

this->name = name;

this->last\_vaccine = last\_vaccine;

}

Date getBirthday() const {

return birthday;

}

Date getArrival() const {

return arrival;

}

SEX getSex() const {

return sex;

}

string getName() const {

return name;

}

Date getLastVaccine() const {

return last\_vaccine;

}

virtual void print() const {

cout<<"Name:"<<name<<" Sex:"<<sex<<" Birthday:"<<this->birthday<<" Arrival:"<<this->arrival<<" Last Vaccine:"<<this->last\_vaccine<<endl;

}

virtual Date calculateVaccineDate() const = 0;

virtual string getByType() const = 0;

};

class Reptile : public Animal {

private:

Date nextVaccine;

public:

Reptile(Date b, Date a, SEX sex, string name, Date last\_vaccine) :Animal(b, a, sex, name, last\_vaccine) {

this->nextVaccine = calculateVaccineDate();

}

Date calculateVaccineDate() const {

return getLastVaccine().nextVaccine(12);

}

void print() const {

Animal::print();

cout<<this->nextVaccine<<endl;

}

string getByType() const {

return "Reptile";

}

};

class Bird : public Animal {

private:

char letac;

public:

Bird(Date b, Date a, SEX sex, string name, Date last\_vaccine, char l) :Animal(b, a, sex, name, last\_vaccine) {

this->letac = l;

}

Date calculateVaccineDate() const {

return this->getLastVaccine().nextVaccine(8);

}

void print() const {

Animal::print();

cout<<this->letac<<endl;

}

string getByType() const {

return "Bird";

}

};

class Mammals : public Animal {

private:

char prezivar;

public:

Mammals(Date b, Date a, SEX sex, string name, Date last\_vaccine, char p) :Animal(b, a, sex, name, last\_vaccine), prezivar(p) {}

Date calculateVaccineDate() const {

return this->getLastVaccine().nextVaccine(6);

}

void print() const {

Animal::print();

cout<<this->prezivar<<endl;

}

string getByType() const {

return "Mammal";

}

};

int main() {

vector<Animal\*> Zoo;

Reptile vlekac = Reptile(Date(1,1,2012),Date(4,8,2016),M,"Joe",Date(18,11,2024));

Bird ptica = Bird(Date(2,2,2013),Date(4,8,2016),M,"Boe",Date(18,11,2024),'d');

Mammals cicac = Mammals(Date(2,2,2013),Date(4,8,2016),M,"Goe",Date(18,8,2024),'n');

//Add animals in the ZOO

Zoo.emplace\_back(&vlekac);

Zoo.emplace\_back(&ptica);

Zoo.emplace\_back(&cicac);

// Size

cout<<"Number of animals in the ZOO: "<<Zoo.size()<<endl;

// //Print by type

for (const auto& animal : Zoo) {

animal.printByType(type);

}

// Print all the animals in the ZOO

string type = "Bird";

for (const auto\* animal : Zoo) {

if(animal->getByType() == type)

animal->print();

}

//Print the animals that need to be vaccine in a week

Date today = Date(14,2,2025);

for (const auto\*animal : Zoo) {

if(animal->calculateVaccineDate()-today <= 7)

animal->print();

}

return 0;

}