Об’єктно-орієнтоване програмування

на основі мови C++

2й семестр

Лекція 1

Викладач:

Розова Людмила Вікторівна

доцент, кандидат технічних наук

каф.ДММ, к.13

email: lyudmyla.rozova@khpi.edu.ua

курс «Об’єктно-орієнтоване програмування»:

об’єм курсу: 150 годин (5 кредитів)

1 кредит (освітня одиниця) = 30 аудиторних годин (45 хв)

З них: **Лекції** 16 годин - 0,5 пари на тиждень

Лабораторні заняття 64 години – 2 пари на тиждень

Самостійна робота 70 годин ≈ 4 години на тиждень!

Оцінювання:

Лабораторні роботи (10 л/р) - 60 балів

Бали за рейтингом:

Робота на лекціях – 6 балів

Для тих виконав усі лабораторні роботи:

Проєкт - 14 балів

Теоретичний тест - 20 балів

Або екзамен, допуск до екзамену всі лабораторні роботи

Література

1. **Страуструп Б.** Язык программирования C++. Специальное издание [Текст] / Бьерн Страуструп; пер. с англ. – М. : ООО Бином-Пресс, 2007. –1104 с.
2. **Шилдт Г.** C++: базовый курс [Текст] / Герберт Шилдт. – 3-е изд. –М. : Вильямс, 2009. – 624 с.
3. **Алейников, Д. В., Холодова, Е. П., Силкович, Ю.** Н. Язык программирования C++ : введение в объектно-ориентированное программирование : учебно-методическое пособие – Минск : НациональнаябиблиотекаБеларуси, 2016. – 85 с.
4. **C++. Основи програмування. Теорія та практика :** підручник **/** [О.Г. Трофименко, Ю.В. Прокоп, І.Г. Швайко, Л.М. Буката та ін.] ; за ред. О.Г. Трофименко. – Одеса: Фенікс, 2010. – 544 с.
5. **Павловская Т.А.** C/C++. Программирование на языке высокого уровня. — СПб.: Питер, 2001, 2003
6. **Павловская Т. А., Щупак Ю. А.** C++. Объектно-ориентированное программирование: Практикум. — СПб.: Питер, 2006. — 265 с:

Література

7. Р. Лафоре. Объектно-ориентированное программирование в С++. – Спб.-2006.– 928с.

8. Б.Мейер. Основы объектно-ориентированного программирования

9. Основи програмування на С++: Навчальний посібник для студентів спеціальностей 113 – Прикладна математика та 122 – Комп`ютерні науки: навч. посіб./ Водка О.О., Дашкевич А.О., Іванченко К.В., Розова Л.В., Сенько А.В. – Харків: НТУ «ХПІ», 2021. – 114 с.

Посилання на деякі літературні джерела:

https://iiiii-my.sharepoint.com/:f:/g/personal/lyudmyla_rozova_khpi_edu_ua/EmLWclNwMsdBj_k3F3LnHdMB5X65aWJB8ICYFSov0BWe9g?e=JfLpdB

Репозитарій для лабораторних робіт та лекцій:

https://github.com/LRozova/OOP_ukr_2022

План лекції

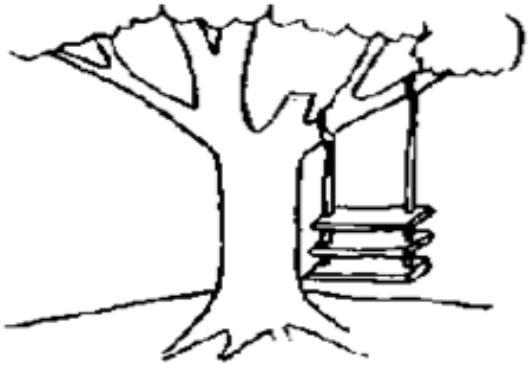
1. Введення в ООП
2. Поняття класу та об’єкту
3. Принципи ООП
4. Інкапсуляція, абстракція
5. Багатофайлові проекти

Парадигми програмування

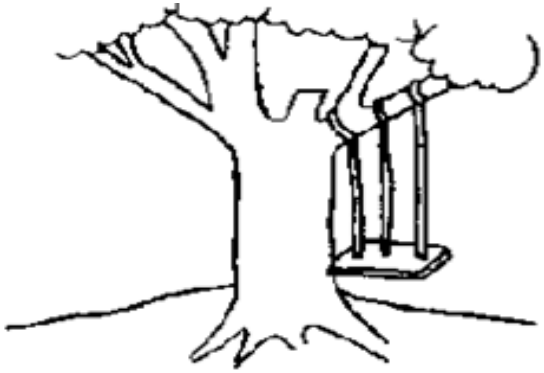
- **Структурне програмування** - проектування програми «зверху-вниз» з використанням певних алгоритмічних структур, покрокове їх виконання.
- В основі структурного підходу лежить розбиття програми на частини, підпрограми - процедурне програмування.

Процедурне програмування

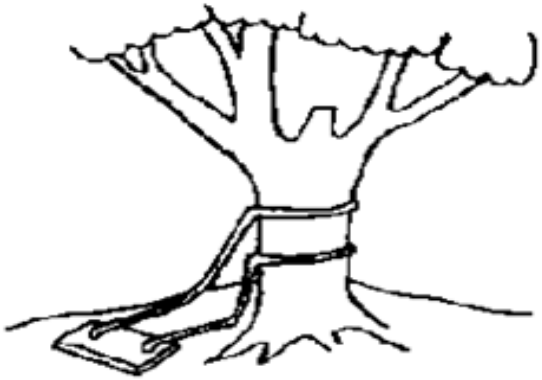
- C, Pascal, FORTRAN і інші подібні з ними мови програмування відносяться до категорії процедурних мов. Кожен оператор такої мови вказує комп'ютеру зробити деяку дію, *наприклад прийняти дані від користувача, зробити з ними певні дії і вивести результат цих дій на екран.* Програми, написані на процедурних мовах, являють собою послідовності інструкцій
- Дані + Алгоритм = Програма
- Програма, побудована на основі процедурного методу, розділена на функції, кожна з яких в ідеальному випадку виконує деяку закінчену послідовність дій і має явно виражені зв'язки з іншими функціями програми



1. Як було запропоновано організатором проекту



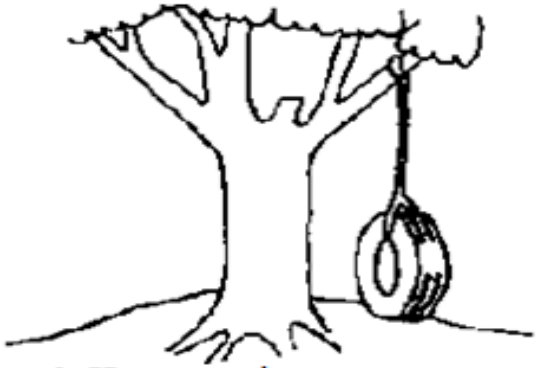
2. Як було описано в технічному завданні



3. Як було спроектовано провідним системним фахівцем

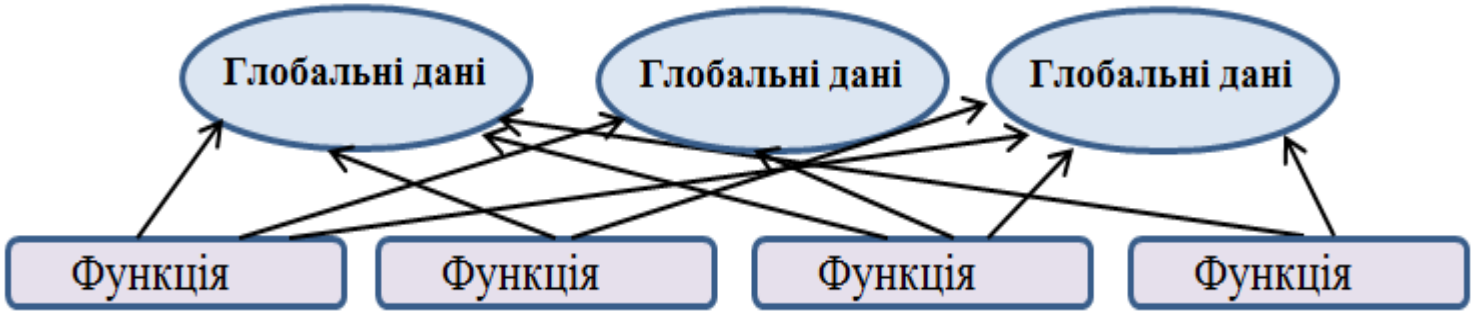
4. Як було реалізовано програмістами

5. Як було впроваджено



6. Чого хотів користувач

- У процедурній програмі, написаної, наприклад, на мові С, існує два типи даних: ***локальні і глобальні***
- **Локальні** дані знаходяться всередині функції і призначені для використання виключно цією функцією і не можуть бути змінені іншими функціями.
- Якщо існує необхідність спільного використання одних і тих ж даних декількома функціями, то дані повинні бути оголошені як **глобальні**.



- Великі програми зазвичай містять безліч функцій і глобальних змінних. Проблема процедурного підходу полягає в тому, що число можливих зв'язків між глобальними змінними і функціями може бути дуже велике.
- Велике число зв'язків між функціями і даними породжує кілька проблем:

➢ ускладнюється структура програми

➢ в програму стає важко вносити зміни.

Моделювання реального світу

Друга, більш важлива, проблема процедурного підходу полягає в тому, що відділення даних від функцій виявляється малопридатним для відображення картини **реального світу**.

У реальному світі нам доводиться мати справу з фізичними об'єктами, такими, наприклад, як люди або машини, інше.

Ці об'єкти не можна віднести ні до даних, ні до функцій, оскільки реальні речі являють собою сукупність **властивостей** і **поведінки**.

Прикладами *властивостей* (характеристиками) для людей можуть бути колір очей або довжина волосся; для машин - потужність двигуна і кількість дверей.
Таким чином, **властивості** об'єктів рівносильні **даним** в програмах. Вони мають певне значення

Поведінка - це деяка реакція об'єкта у відповідь на зовнішній вплив.

- Посмішка у людини на жарт
- Зупинка атомобіля при натисканні на гальмо
- Лай у собаки при сторонніх

Поведінка схожа з **функцією**: ви викликаєте функцію, щоб здійснити будь-яку дію, характерну лише цьому об’єкту.

Таким чином, ні окремо взяті дані, ні окремо взяті функції не здатні адекватно відобразити об'єкти реального світу.

Об'єктно-орієнтований підхід

- Основною ідеєю об'єктно-орієнтованого підходу є об'єднання даних і дій, що здійснюються над цими даними в єдине ціле, яке називається **об'єктом**.
- Функції об'єкта (**методи**) зазвичай призначені для доступу до даних об'єкта. Якщо необхідно взяти будь-які дані об'єкта, потрібно викликати відповідний метод. Прямий доступ до даних неможливий. Моделлю реального об’єкта є клас

В основі ООП лежить поняття класа і об’єкта.

- **Клас** є абстрактним типом даних, що визначається користувачем, і являє собою модель реального об'єкта у вигляді **даних** і **функцій** для роботи з ними.
- **Об'єкт** (або екземпляр класу) – це представник класу, побудований згідно створеному в класі опису. Кожному *класу* може належати одночасно декілька об'єктів, кожен з яких має унікальне ім'я. *Об'єкт* характеризується фізичним існуванням і є конкретним екземпляром *класу*.

Клас борщ

Дані класу

екземпляри класу - об'єкти

Методи класу:

- ✓ Де взяти інгредієнти;
- ✓ Помити, почистити, порізати;
- ✓ У якій послідовності застосувати, як;
- ✓ Скільки часу варити;
- ✓ …….

Клас (class)

Дані класу називаються **полями** (по аналогії з полями структури),

Функції класу - **методами.**

Поля і методи називаються **елементами класу**.

Опис класу в першому наближенні виглядає так:

```
class <Ім'я> {
  private:
  // Опис прихованих елементів
  // встановлюється за замовчуванням
  public:
  // Опис публічних елементів
  protected:
  // Опис захищених елементів
}; // Опис закінчується крапкою з комою
```

Основні принципи ООП

- **Інкапсуляція** – об’єднання даних і методів у єдине ціле (class);
- **Успадкування** – створення нових класів на базі існуючих
- **Поліморфізм** – «один інтерфейс, багато методів»

Механічна кавомолка

Електрична кавомолка
-мотор
-кнопки
-схеми
…

Класс «Електрична кавомолка»:

***Успадкування*:** створення класу «Електрична кавомолка» на базі класу «Механічна кавомолка»

***Поліморфізм*:** перевизначення основних функцій роботи з кавомолкою в класі «Електрична кавомолка»

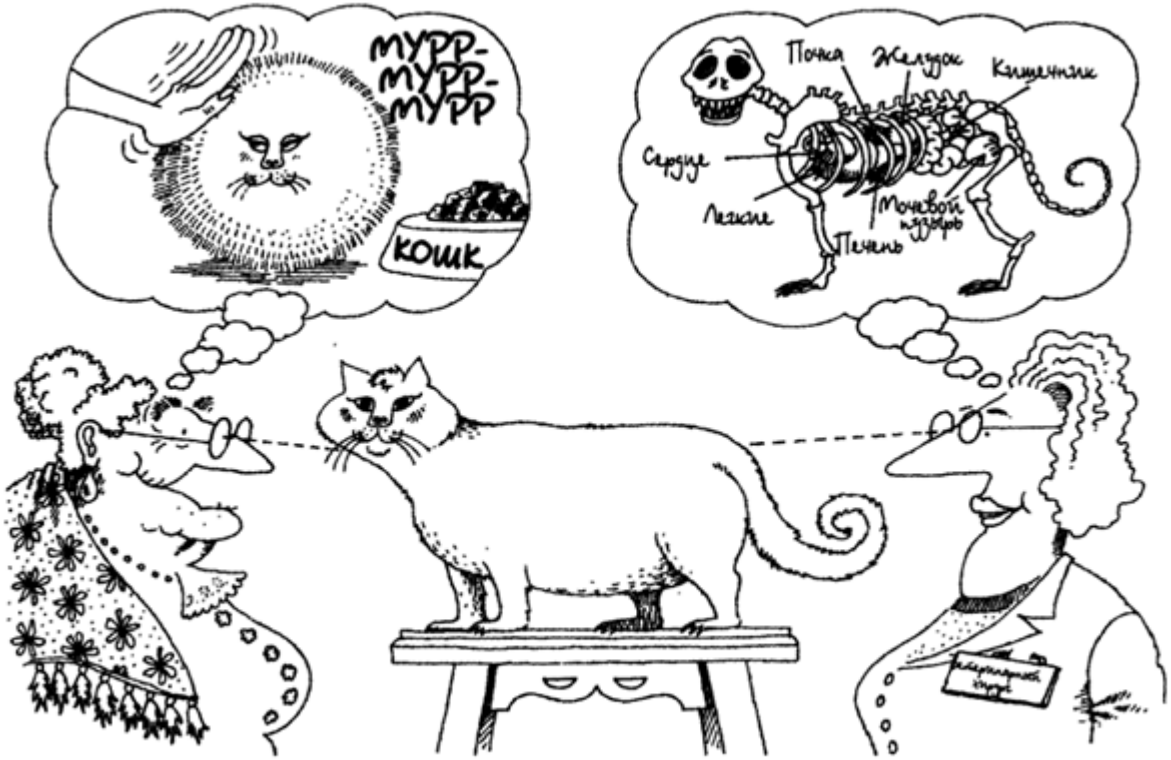
Інкапсуляція : приховування всієї начинки кавомолки від користувача. Лише взаємодія по інструкції

Інкапсуляція

- об'єднання даних і методів для роботи з ними в один об'єкт.
- Інкапсуляція також реалізує приховування даних від зовнішнього впливу, що захищає їх від випадкових змін.

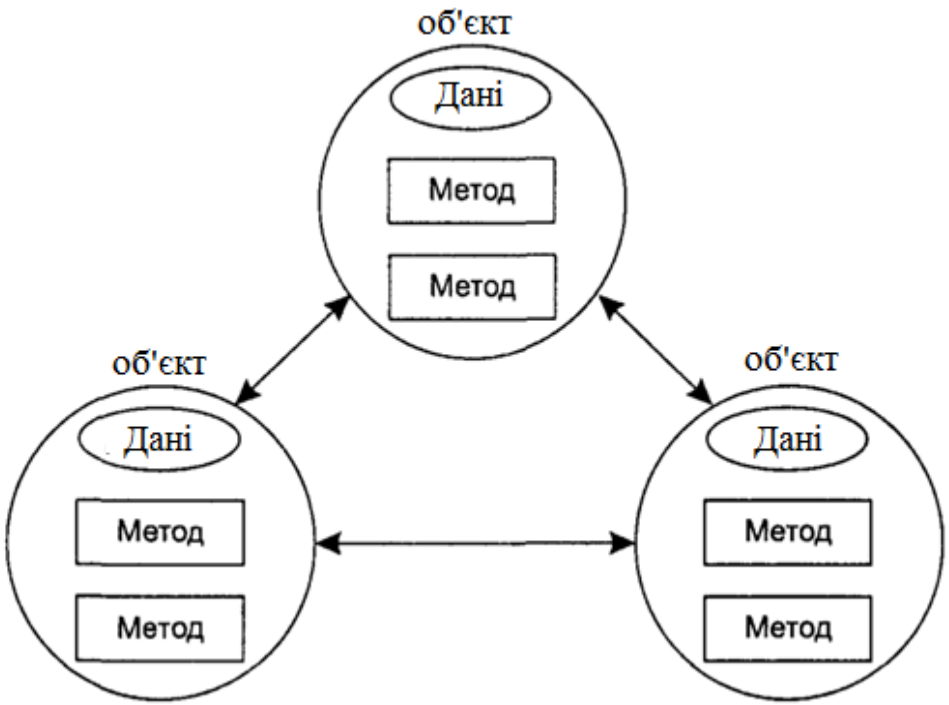
Для моделювання класу необхідно застосувати метод **абстракції -** виділення істотних характеристик деякого об'єкта, що відрізняють його від всіх інших видів об'єктів і таким чином, чітко описує його концептуальні межі з точки зору спостерігача.

Абстракція - один з основних методів, що дозволяють впоратися зі складністю вихідного завдання, бо основна задача проектувальника створити ілюзію простоти



Абстракція концентрує увагу на суттєвих властивостях об'єкта з точки зору спостерігача

Об'єктно-орієнтоване програмування ООП (object-oriented programming - OOP) - це метод програмування, заснований на представленні програми у вигляді сукупності взаємодіючих об'єктів (через їх інтерфейси), кожен з яких є екземпляром певного класу, а класи є членами певної ієрархії при успадкування.



Моделі реальних об'єктів (абстракції) +
+ поведінка об'єктів (взаємодії) = програма

Що може бути об’єктом реального світу (класом)?

Фізичні об'єкти:

- Автомобілі під час моделювання вуличного рухи.
- схемні елементи при моделюванні ланцюга електричного струму.
- країни при створенні економічної моделі.
- літаки при моделюванні диспетчерської системи.

Елементи інтерфейсу:

- вікна.
- меню.
- графічні об`єкти (лінії, прямокутники, кола).
- миша, клавіатура, дискові пристрої, принтери.

Структури даних

- масиви.
- стеки.
- пов'язані списки.
- бінарні дерева.

Що може бути об’єктом реального світу (класом)?

Групи людей

- Співробітники.
- Студенти.
- Покупці.
- Продавці.

Сховища даних

- Описи інвентарю.
- Списки співробітників.
- Словники.
- Географічні координати міст світу.

Призначені для користувача типи даних

- Час.
- Величини кутів.
- Комплексні числа.
- Точки на площині.

Учасники комп'ютерних ігр

- Автомобілі в гонках.
- Позиції в настільних іграх (шашки, шахи).
- Друзі та вороги в пригодницьких іграх.

Та інше

Режими доступу **private** і **public**

- специфікатори доступу **private** і **public** керують видимістю елементів класу.

Елементи, описані після службового слова ***private***, видимі тільки всередині класу. Цей вид доступу прийнятий в класі за замовчуванням. Елементи, описані після слова *public*, доступні зовні.

- Інтерфейс класу описується після специфікатора public.

Дія будь-якого специфікатора поширюється до наступного специфікатора або до кінця класу. Можна, задавати кілька секцій private і public, Порядок їх слідування значення не має.

```
class Employee
{private:
    char name[25];//ім'я
    int age; //вік
    char position[25]; //посада
    int stage; //стаж
public:
    void setName(char *n);
    void setAge(int s) { age = s;}
    int getAge() { return age;}
    int getStage(int currentYear, int
currentMonth, int currentDay);
};
void Employee::setName(char* n)
{
    strcpy(name, n);
}
```

Опис об’єктів (екземплярів класу)

```
Employee John;
Employee managers[20];
Employee *chef;
```

Доступ до елементів об'єкта

```
John.setName(name);
managers[2].setAge(30);
chef->getStage(2020, 2, 18);
```

Геттери і сеттери - методи класу, які організовують доступ до приватних елементів класу, застосовуючи певну логіку доступу, враховуючі небажані ситуації

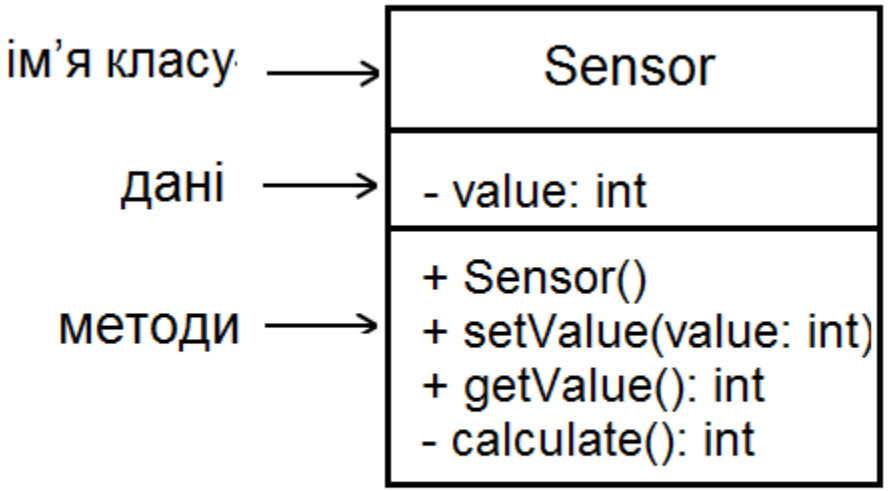
UML діаграми класів

Одним з способів опису та представлення об'єктно-орієнтованих програм та класів зокрема, є використання ***UML-нотації***.

UML (*unified modeling language*) — узагальнена мова опису предметних областей, зокрема, програмних систем, об'єктно-орієнтованого аналізу і проєктування. Використовується для візуалізації, специфікації, конструювання і документування програмних систем. В контексті ООП в UML містяться такі базові компоненти, як сутності, зв'язки між ними, та набори сутностей

Структурна сутність клас уявляє собою опис набору об'єктів з однаковими властивостями (даними або атрибутами), операціями (методами), зв'язками та поведінкою

UML діаграма класу



Відділення інтерфейсу від реалізації

Один з найбільш фундаментальних принципів розробки хорошого програмного забезпечення полягає в відділенні інтерфейсу від реалізації:

при побудові програми на C++ для зручності подальшої підтримки та сприйняття об'єктно-орієнтованого коду програми, слід розділяти об'явлення класу та його реалізацію, Для цього об'явлення класу слід надавати в заголовкових файлах, а реалізацію — в файлах вихідного коду

Заголовки - імя_класса.h

Файл вихідного коду (або файл реалізації класу) - імя_класса.cpp

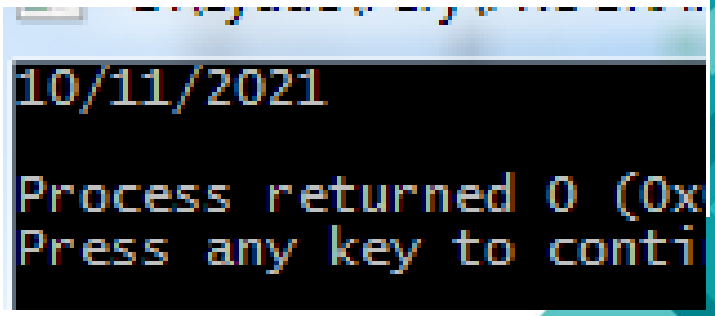
Заголовки включаються (через #include) в кожен файл, в якому використовується клас, а файли з вихідними кодами компілюються і компонуються з файлом, що містить головну програму (main).

```
//Файл Date.h – заголовковий файл (Header file)
#ifndef DATE_H//умовна компіляція захищає від
#define DATE_H//багаторазового включення файлу

class Date
{
private:
    int m_day;
    int m_month;
    int m_year;
public:
    int getDay(); // геттер для day
    void setDay(int day); // сеттер для day
    int getMonth(); // геттер для month
    void setMonth(int month); // сеттер для month
    int getYear(); // геттер для year
    void setYear(int year); // сеттер для year
};
#endif // DATE_H
//-----------------Конец файла-----
```

```
//Файл Date.cpp – файл реалізації
#include <iostream>
#include "Date.h"
using namespace std;
 int Date::getDay()
  { return m_day; }
 void Date::setDay(int day)
  { m_day = day; }
 int Date::getMonth()
  { return m_month; }
 void Date::setMonth(int month)
  { m_month = month; }
 int Date::getYear()
  { return m_year; }
 void Date::setYear(int year)
  { m_year = year; }
//-------Кінець файлу-------------
```

```
// Файл main.cpp
#include <iostream>
#include "Date.h"
using namespace std;
int main()
{   Date D1;
    D1.setDay(10);
    D1.setMonth(11);
    D1.setYear(2021);
    cout<<D1.getDay()<<"/"<<D1.getMonth()
    <<"/"<<D1.getYear()<<endl;
    return 0;
}
```



Дякую за увагу!