Міністерство освіти і науки України
Національний університет «Львівська політехніка»
кафедра інформаційних систем та мереж

Григорович Віктор

Об’єктно-орієнтоване програмування

Класи та об’єкти

Навчальний посібник

2021

Григорович Віктор Геннадійович

Об’єктно-орієнтоване програмування. Класи та об’єкти. Навчальний посібник.

Дисципліна «Об’єктно-орієнтоване програмування» вивчається після курсу «Алгоритмізація та програмування», цією дисципліною продовжується цикл предметів, що стосуються програмування та розробки програмного забезпечення.

В посібнику містяться теоретичні відомості, приклади, методичні вказівки з їх розв’язування, варіанти лабораторних завдань та питання і завдання з контролю знань з теми «Класи та об’єкти».

Розглядаються наступні роботи лабораторного практикуму:

Лабораторна робота № 1.1. Поля та методи – дії над одним (поточним) об’єктом

Лабораторна робота № 1.2. Оголошення та будова класу

Лабораторна робота № 1.3. Об’єкти – параметри методів (дії над кількома об’єктами)

Лабораторна робота № 1.4. Використання класів

Лабораторна робота № 1.5. Композиція класів та об’єктів

Лабораторна робота № 1.6. Вкладені класи

Лабораторна робота № 1.7. Композиція класів та об’єктів – складніші завдання

Лабораторна робота № 1.8. Вкладені класи – складніші завдання

Відповідальний за випуск – Григорович В.Г.

Стислий зміст

Зміст

Вступ

Дисципліна «Об’єктно-орієнтоване програмування» вивчається після курсу «Алгоритмізація та програмування», цією дисципліною продовжується цикл предметів, що стосуються програмування та розробки програмного забезпечення.

В посібнику містяться теоретичні відомості, приклади, методичні вказівки з їх розв’язування, варіанти лабораторних завдань та питання і завдання з контролю знань з теми «Класи та об’єкти».

Тема 1. Класи та об’єкти

Стисло та головне про класи та об’єкти

Класи та об’єкти. Елементи класу

Класи та об’єкти

Поняття «клас» та «об’єкт»

Клас – це якісно новий тип даних, що поєднує в єдине ціле опис даних та опис дій над даними. Синонімом для терміну «клас» є «об’єктовий тип даних». Клас – це сукупність (множина) однотипних об’єктів. Саме класами мова С++ відрізняється від мови С (в мові С ніяких класів немає).

Об’єкт – це змінна об’єктового типу, іншими словами – екземпляр класу. Клас – множина, об’єкт – елемент множини.

Об’єкти як програмні ресурси в мові С++ мають безпосередні аналоги з об’єктами реального світу. Кожний об’єкт реального світу характеризується певними властивостями (наприклад, вагою, розмірами, кольором тощо) та поведінкою (діями, які може виконувати цей об’єкт – наприклад, пес кусає та гавкає, кіт спить або ловить мишей; чи діями, які ми можемо виконувати за допомогою цього об’єкту – наприклад, за столом можна їсти, читати, писати, на стільці можна сидіти і т.п.).

Аналогічно, до складу об’єктів в мові С++ входять змінні, що зберігають дані про значення характеристик (такі змінні в складі об’єкта називаються *поля*). До складу класів входять функції, що описують дії, які можна виконувати над об’єктами (ці функції називаються *методи*). Інших дій, окрім явно вказаних за допомогою методів, над об’єктами виконувати не можна.

Визначення класу

Синтаксис визначення класу розглянемо на прикладі класу Account, який описує банківський рахунок і містить поля прізвище власника (name), залишок коштів на рахунку (summa) та метод змінити кошти на рахунку (Change):

```
class Account
{
    string name;                          // поле = змінна в складі класу
    double summa;                         // поле = змінна в складі класу
```

```
    void Change(double value)                  // метод = функція в складі класу
    {
        if (value > 0)                         // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa)   // знімаємо кошти
            summa += value;
    }
};  // обов'язково в кінці має бути крапка з комою
```

В цьому прикладі показана *інкапсуляція* (в першому тлумаченні) – поєднання в єдине ціле *полів* та *методів* класу.

Елементи класу

Елементи класу поділяються на *поля* та *методи*.

Поля – це дані, які інкапсулює клас (змінні, які належать класу).

Методи – це дії, які можна виконувати над класом та його полями (функції, які належать класу).

Управління доступом до елементів класу

Слід зауважити, що у такому вигляді, як наведено у попередньому прикладі, клас Account використовувати не можна, бо не можна організувати доступу до полів та методів цього класу.

За умовчанням всі елементи (поля та методи) класу недоступні. Це означає, що до них можна звертатися лише із методів цього класу, і не можна – із інших функцій.

Для управління доступом до елементів класу використовуються директиви `private`, `protected`, `public`. Директива `protected` має сенс лише для ієрархії класів, утворених зв'язками успадковування, – її будемо розглядати при вивченні теми «Успадковування». Директива `private` встановлює максимальні обмеження доступу: елементи класу з рівнем доступу `private` повністю недоступні поза класом, до них можна звертатися лише із методів цього класу. Такі елементи – недоступні та приховані. Директива `public` скасовує всі обмеження – відповідні елементи класу доступні скрізь. Такі елементи – доступні та видимі.

Для того, щоб визначити певний рівень доступу до елементів класу, слід перед цими елементами вказати відповідну директиву доступу, яка складається із ключового слова (`private`, `protected`, `public`), за яким записується двокрапка:

```
class ім'я
{
private:
    // опис прихованих (недоступних) елементів
    ...
public:
    // опис видимих (доступних) елементів
    ...
};  // обов'язково має бути крапка з комою
```

Це – друге тлумачення *інкапсуляції*: управління доступом до елементів класу.

Прийнято максимально обмежувати доступ до полів: поля оголошують з директивою `private`. Це дозволяє запобігти неправильному використанню полів. Для того, щоб можна було записати та прочитати значення полів, оголошують *методи доступу* (методи *зчитування* та *запису*). Для нашого прикладу це буде виглядати так:

```
class Account
{
private:                                    // приховані елементи класу (поля)
    string name;                            // поле
    double summa;                           // поле

public:                                     // доступні елементи класу (методи)
    // методи доступу
    string getName()                        // метод зчитування поля name
    {
        return name;
    }

    void setName(string value)              // метод запису поля name
    {
        name = value;
    }

    double getSumma()                       // метод зчитування поля summa
    {
        return summa;
    }

    void setSumma(double value)             // метод запису поля summa
    {
        summa = value;
    }

    void Change(double value)               // метод зміни залишку коштів на рахунку
    {
        if (value > 0)                      // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa) // знімаємо кошти
            summa += value;
    }
};      // обов'язково в кінці має бути крапка з комою
```

Об’єкти

Створення / опис / оголошення об’єктів

Після того, як був визначений клас, ми можемо створювати об’єкти цього класу. Об’єкти створюються аналогічно до змінних інших типів:

```
    Account a;                  // об'єкт - звичайна змінна
    Account* b = new Account(); // об'єкт - динамічна змінна
    Account c[3];               // масив із трьох об'єктів
```

Доступ до елементів класу для об’єктів та вказівників на об’єкти

Звертання до елементів класу – подібне до звертання до елементів структури: використовуються операція «крапка» для доступу через звичайну змінну та операція «стрілка» для доступу через вказівник на динамічну змінну.

Оскільки всі поля класу Account – приховані та недоступні, то звертатися до них можна лише із методів цього ж класу:

```
a.name = "Іваненко";
a.summa = 100.0;

b->name = "Петренко";
b->summa = 150.0;

c[0].name = "Сидоренко";
c[0].summa = 175.0;
```

Всі методи – доступні, тому до них можна вертатися із будь-якої функції (наприклад, main()):

```
a.setName("Іваненко");
a.setSumma(100.0);

b->setName("Петренко");
b->setSumma(150.0);

c[0].setName("Сидоренко");
c[0].setSumma(175.0);
```

Методи

Визначення методів в класі та поза класом

У розглянутому прикладі всі методи визначалися в класі:

```
class Account
{
private:                                    // приховані елементи класу (поля)
    string name;                            // поле
    double summa;                           // поле

public:                                     // доступні елементи класу (методи)
    // методи доступу
    string getName()                        // метод зчитування поля name
    {
        return name;
    }
    void setName(string value)              // метод запису поля name
    {
        name = value;
    }

    double getSumma()                       // метод зчитування поля summa
    {
        return summa;
    }
```

```
    void setSumma(double value)              // метод запису поля summa
    {
        summa = value;
    }

    void Change(double value)                // метод зміни залишку коштів на рахунку
    {
        if (value > 0)                       // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa) // знімаємо кошти
            summa += value;
    }
};      // обов'язково в кінці має бути крапка з комою
```

Зазвичай в класі визначаються лише найпростіші методи – тіло яких містить одну-дві команди. У всіх інших випадках, щоб не робити визначення класу занадто громіздким, методи визначаються поза класом. Для цього в класі вказуємо лише заголовок (прототип) методу, а повне його визначення записуємо окремо. При цьому перед іменем методу вказуємо префікс: ім’я класу та дві двокрапки:

```
class Account
{
private:                        // приховані поля
    string name;
    double summa;

public:                         // доступні методи
    string getName()            // метод зчитування поля name
    {
        return name;
    }

    void setName(string value)  // метод запису поля name
    {
        name = value;
    }

    double getSumma()           // метод зчитування поля summa
    {
        return summa;
    }

    void setSumma(double value) // метод запису поля summa
    {
        summa = value;
    }

    void Change(double value); // прототип - оголошення методу

};  // обов'язково в кінці має бути крапка з комою

// визначення методу (повний опис методу) поза класом
void Account::Change(double value)   // перед іменем – префікс: ім'я класу
{
    if (value > 0)
        summa += value;
    if (value < 0 && abs(value) < summa)
        summa += value;
}
```

Префікс перед іменем методу вказує, що це – не звичайна функція, а метод, який належить до простору імен вказаного класу.

Вказівник this

В тілі кожного метода (за винятком *статичних* методів – див. далі) системою неявно визначено вказівник this, він налаштований на той об'єкт, який викликав цей метод. Цим методи відрізняються від звичайних функцій: ясна річ, що звичайні функції ніяких неявно визначених змінних не мають.

Таким чином, в тілі метода setName() з попереднього прикладу:

- якщо цей метод викликається об'єктом a, то вказівник this містить адресу об'єкта a;
- якщо цей метод викликається об'єктом *b, то вказівник this містить адресу об'єкта *b;
- якщо цей метод викликається об'єктом c[0], то вказівник this містить адресу об'єкта c[0].

Інтерфейс та друзі класу

Інтерфейс класу

Поняття «інтерфейс» та «реалізація» класу

Інтерфейс класу – це та його частина, яка доступна ззовні всім користувачам. До інтерфейсу належать доступні (`public`) елементи класу. Інтерфейс містить інформацію про те, **що** робить клас (його призначення).

Реалізація класу – це та частина, яка зазвичай ззовні недоступна (`private`, `protected`) для користувачів. Вона описує, **як** саме клас реалізує своє призначення.

Дружні функції та дружні класи

Інколи буває потрібно забезпечити безпосередній доступ ззовні до прихованих елементів класу, тобто розширити інтерфейс класу, відкрити клас для деяких програмних ресурсів. Для цього використовують дружні функції та дружні класи: при визначенні класу можна оголосити деякі зовнішні функції чи класи як дружні для цього класу, такі друзі будуть мати повний доступ до всіх елементів класу. При цьому клас сам визначає, які функції чи інші класи будуть для нього дружніми, а які – ні.

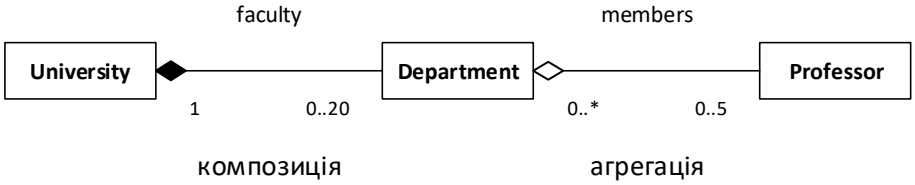
Композиція та агрегування. Делегування

Композиція та агрегування класів

В об’єктно-орієнтованому програмуванні під агрегуванням (в широкому значенні, його ще називають композицією або включенням) мають на увазі методику створення нового класу із вже створених класів – подібно до того, як певний виріб збирається із готових деталей. Агрегування часто називають «відношенням приналежності» за принципом «автомобіль має корпус, колеса та двигун».

При агрегуванні один клас містить поле-об’єкт, тип якого – іншого класу. Це поле може бути вказівником на об’єкт іншого класу (агрегування у вузькому значенні) або самим об’єктом іншого класу (композиція).

Позначення композиції та агрегації на UML-діаграмах класів:



Університет пов’язаний з кафедрами чи факультетами відношенням композиції, а кафедра з професорами – відношенням агрегації. Композиція означає, що при знищенні цілого всі частини теж знищуються, а агрегація – що при знищенні цілого частини продовжують існувати, тобто, – професори залишаються жити після ліквідації університету, тоді як факультети чи кафедри знищуються разом з університетом.

При позначенні композиції та агрегації рисують: ромбик – зі сторони цілого, лінію – зі сторони частини. Замальований ромбик означає більш сильний зв’язок (композиція), порожній ромбик – більш слабий зв’язок (агрегація).

Агрегація

Агрегація (агрегування за посиланням) – відношення «ціле-частина» між двома об’єктами, коли один об’єкт (контейнер) містить посилання (вказівник) на інший об’єкт. Обидва об’єкти можуть існувати незалежно один від одного: якщо контейнер буде знищено, то його вміст – залишиться.

```
class Professor;

class Department
{
private:
    Professor* members[5];
};
```

Композиція

Композиція (агрегування за значенням) – більш строгий варіант агрегування, коли внутрішній об’єкт може існувати лише як частина контейнера (контейнер містить не посилання чи вказівник, а сам внутрішній об’єкт). Якщо контейнер буде знищено, то і внутрішній об’єкт буде також знищений.

```
class Department;

class University
{
private:
    Department faculty[20];
};
```

Делегування дій іншим методам

На основі агрегування та композиції реалізується методика делегування, коли поставлена перед зовнішнім об’єктом задача передоручається іншому – внутрішньому – об’єкту, який спеціалізується на розв’язанні задач такого типу.

Приклад:

```
class Engine                                // клас «двигун»
{
public:
    void Start() {/* ... */ }               // метод
};

class Automobile                            // клас «автомобіль»
{
private:
    Engine* engine;                         // агрегація

public:
    Automobile() { engine = new Engine(); } // конструктор
    void Start() { engine->Start(); }       // делегування
};
```

В наведеному прикладі метод `Start()` класу `Automobile` делегує свою роботу методу `Start()` класу `Engine`. Це реалізується як виклик метода внутрішнього класу із метода зовнішнього класу.

Статичні елементи класу

Статичні поля та методи

За допомогою модифікатора `static` можна описати статичні поля та методи класу. Їх можна розглядати як глобальні змінні чи функції, доступні лише в межах області класу.

Статичні поля

Статичні поля використовуються для зберігання даних, спільних для всіх об’єктів класу, наприклад, кількості об’єктів чи посилання на ресурс, що використовується усіма об’єктами. Статичні поля існують для всіх об’єктів класу в єдиному екземплярі, тобто не дублюються.

Статичні методи

Статичні методи призначені для доступу до статичних полів класу. Вони можуть звертатися лише до статичних полів та викликати лише інші статичні методи, тому що їм не передається вказівник `this`. Виклик статичних методів здійснюється так само, як і звертання до статичних полів – або через ім’я класу, або, якщо хоча би один об’єкт класу вже створений, – через ім’я об’єкту:

```
class A
{
    static int count;  // поле count – приховане; це – оголошення

public:
    static void inc_count() { count++; }
    ...
};

int A::count;              // визначення в глобальній області
                           // за умовчанням ініціалізується нулем

void f()
{
    A a;
    // a.count++;             - неможна, бо поле count - приховане
    //                        зміна поля за допомогою статичного методу:
    A::inc_count();        // або a.inc_count();
}
```

Статичні методи не можуть бути константними (`const`) та віртуальними (`virtual`).

Теоретичні відомості

Загальні поняття ООП

Процедурно-орієнтоване та об'єктно-орієнтоване програмування

Характерні особливості процедурно-орієнтованого програмування. Обмеження та недоліки процедурно-орієнтованого програмування

- *Відірваність опису даних від опису дій над даними*

  Розглянемо можливе представлення даних про Банківський рахунок в традиційному (до-об'єктному, процедурному) програмуванні:

```
string name;
double summa;
...
void Change(string name, double summa, double amount)
{ ... }
```

  У такому вигляді в одному місці програми описані дані (прізвище власника `name` та залишок коштів на рахунку `summa`), а у іншому – описані дії над цими даними (змінити, тобто зняти чи покласти кошти на рахунок – функція `Change`, параметри якої – прізвище власника `name`; залишок коштів на рахунку `summa`; величина, на яку ми змінюємо ці кошти `amount`).

  Слід зазначити, що при такому підході опис даних та опис дій над цими даними ніяк не пов'язані між собою: дані та функції їх опрацювання відірвані та не залежні одні від одного. Це приводить до потенційних проблем при їх використанні:

- *Неможливість захистити дані від неправильних дій*

  Оскільки опис даних не залежить від опису дій над даними, немає можливості захистити дані від неправильного використання.

  Для попереднього прикладу, – немає можливості заборонити наступну операцію:

```
summa = log(summa);
```

Поняття об'єктно-орієнтованого програмування

- Світ складається з реальних об'єктів
- Кожний такий об'єкт описується:
  - Характеристиками (вага, розмір, колір, положення тощо)

    - Поведінкою (дії, які може виконувати об’єкт, або які можна виконувати над об’єктом – за допомогою об’єкту)
- Характеристики та поведінка – невід’ємні якості об’єкта
- Створюємо нову програмну сутність, яка поєднує в єдине ціле опис даних та опис дій над даними = об’єкт в програмі
- Програмні об’єкти максимально точно відповідають об’єктам реального світу

Кожний об’єкт реального світу характеризується певними властивостями (наприклад, вагою, розмірами, кольором тощо) та поведінкою (діями, які може виконувати цей об’єкт – наприклад, пес кусає та гавкає, кіт спить або ловить мишей; чи діями, які ми можемо виконувати за допомогою цього об’єкту – наприклад, за столом можна їсти, читати, писати, на стільці можна сидіти і т.п.).

Що таке класи і для чого вони потрібні

Клас як принципово новий тип даних.
Класи та об’єкти

Клас – це якісно новий тип даних, що поєднує в єдине ціле опис даних та опис дій над даними. Синонімом для терміну «клас» є «об’єктовий тип даних». Клас – це сукупність (множина) однотипних об’єктів. Саме класами мова C++ відрізняється від мови C (в мові C ніяких класів немає).

Об’єкт – це змінна об’єктового типу, іншими словами – екземпляр класу. Клас – множина, об’єкт – елемент множини.

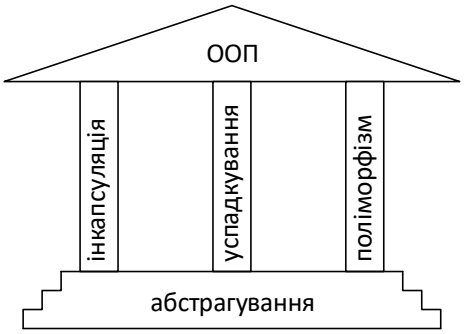
Об’єкти як програмні ресурси в мові C++ мають безпосередні аналоги з об’єктами реального світу.

До складу об’єктів в мові C++ входять змінні, що зберігають дані про значення характеристик (такі змінні в складі об’єкта називаються *поля*). До складу класів входять функції, що описують дії, які можна виконувати над об’єктами (ці функції називаються *методи*). Інших дій, окрім явно вказаних за допомогою методів, над об’єктами виконувати не можна.

Принципи об’єктно-орієнтованого програмування

Подібно до моделі «плоскої Землі», згідно якої Земля тримається на трьох слонах, які стоять на черепасі, можна уявити, що об’єктно орієнтоване програмування тримається на трьох «слонах»-принципах (інкапсуляції, успадкуванні, поліморфізмі), і це все базується на «черепасі»-фундаменті (абстрагуванні).

Зазначену модель плоскої землі рисувати досить складно, простіше це зобразити у наступному вигляді: ООП стоїть на трьох стовпах (інкапсуляція, успадкування, поліморфізм), а це все – на фундаменті (абстрагування).



Отже,

ООП базується на 3^{-x} принципах:

- Інкапсуляція
- Успадкування
- Поліморфізм

В основі всього лежить

- Абстрагування

Розглянемо ці принципи більш детально.

Поняття та терміни ООП

Абстрагування даних

= нехтування несуттєвими деталями.

- Ми нехтуємо деталями машинної реалізації: змінні, функції, байти, файли і т.п.
- Ми оперуємо термінами предметної області, які описують характеристики та поведінку об’єктів.
- Тепер програмування максимально наближене до предметної області.

Інкапсуляція

Інкапсуляція – один з трьох «китів» (разом з успадкуванням та поліморфізмом), на якому стоїть об’єктно-орієнтоване програмування.

Є два аспекти і, відповідно, два тлумачення поняття інкапсуляції.

Інкапсуляція – поєднання в єдине ціле опису даних та опису дій над даними

1. Поєднання в єдине ціле опису даних та опису дій над даними.

   Утворення нової програмної сутності – об’єкт, в якому нероздільно пов’язані дані та функції.

   Множина однотипних об’єктів – це клас.
   Дані, які описані у класі – це поля (так як і у структурі).
   Функції, які описані у класі – це методи.

   Елементи класу – це поля та методи.

Інкапсуляція – обмеження доступу до елементів класу

2. Управління доступом до елементів класу (до елементів всіх об’єктів цього класу) .

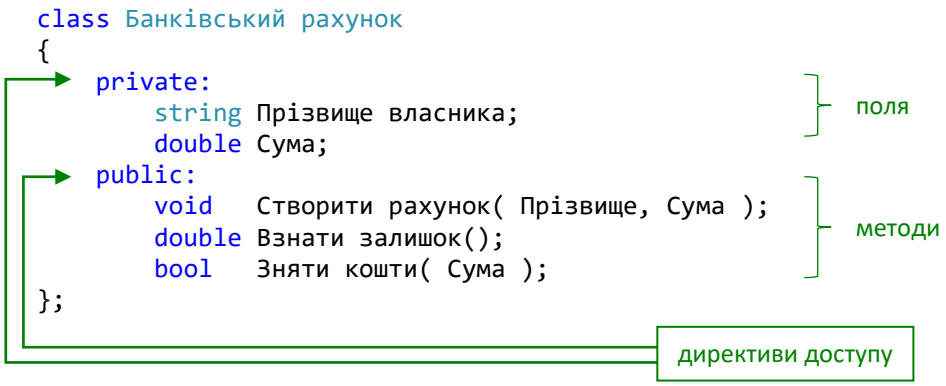
   Деякі елементи класу можна зробити загальнодоступними, можна обмежити доступ до інших елементів.

   Якщо користувач не має доступу до деякого елемента класу, він не зможе його пошкодити.

   Якщо користувач в принципі не може щось зробити, він ніколи не зробить цього неправильно.

Визначення класу

Розглянемо визначення класу `Банківський рахунок`. Для простоти будемо використовувати псевдокод, наближений до синтаксису мови C++.



Клас `Банківський рахунок` містить поля `Прізвище власника` та `Сума`.

В класі також описано методи `Створити рахунок( Прізвище, Сума )`, `Взнати залишок()`, та `Зняти кошти( Сума )`.

За допомогою директив управління доступом обмежено доступ до полів: вони оголошені як приховані (`private`). Методи описані як доступні (`public`).

Над об'єктами можна виконувати лише ті дії, які явно вказані при визначенні класу.

Поля – приховані, тому до них немає доступу ззовні, їх не можна довільно змінювати.

Всі дії над об'єктом виконуються за допомогою доступних методів.

Для класу `Банківський рахунок` доступними є дії:

- Створити рахунок;
- Взнати залишок;
- Зняти кошти.

Ніяких інших дій виконувати над об'єктами цього класу не можна! – ми захистили дані класу від неправильного використання!

Поля та методи – елементи класу

Елементи класу поділяються на *поля* та *методи*.

Поля – це дані, які інкапсулює клас (змінні, які належать класу).

Методи – це дії, які можна виконувати над класом та його полями (функції, які належать класу).

Інтерфейс та реалізація класу

Інтерфейс класу – це та його частина, яка доступна ззовні всім користувачам. До інтерфейсу належать доступні (`public`) елементи класу. Інтерфейс містить інформацію про те, **що** робить клас (його призначення).

Реалізація класу – це та частина, яка зазвичай ззовні недоступна (`private`, `protected`) для користувачів. Вона описує, **як** саме клас реалізує своє призначення.

Агрегування та композиція – використання одних класів іншими

В об'єктно-орієнтованому програмуванні під агрегуванням (в широкому значенні, його ще називають композицією або включенням) мають на увазі методику створення нового класу із вже створених класів – подібно до того, як певний виріб збирається із готових деталей. Агрегування часто називають «відношенням приналежності» за принципом «автомобіль має корпус, колеса та двигун».

При агрегуванні один клас містить поле-об'єкт, тип якого – іншого класу. Це поле може бути вказівником на об'єкт іншого класу (агрегування у вузькому значенні) або самим об'єктом іншого класу (композиція).

Перевантаження операцій

В C++ можна перевантажувати вбудовані операції (такі, як додавання, віднімання, присвоєння тощо) – це один із проявів *поліморфізму*. Перевантаження операцій не є обов'язковим в мовах, які реалізують об'єктно-орієнтоване програмування – наприклад, в мові java воно відсутнє. Проте наявність перевантаження операцій в C++ забезпечує додатковий рівень зручності при використанні нових типів даних.

Успадковування. Інтерпретація успадковування в термінах множин

Успадкування – один з трьох «китів» (разом з інкапсуляцією та поліморфізмом), на якому стоїть об'єктно-орієнтоване програмування. При успадкуванні обов'язково є *клас-предок* (*батьківський клас*; *той, що породжує*) та *клас-нащадок* (*дочірній клас*; *той, який породжений*). В C++ клас-предок називається *базовим*, а клас-нащадок – *похідним*. Всі терміни *батьківський клас*, *предок*, *базовий клас* – еквівалентні. Також еквівалентними є терміни *дочірній клас*, *нащадок*, *похідний клас*.

Наприклад, розглянемо класи Man (Людина) та Programmer (Програміст). Вони описують сукупності – множину Люди та множину Програмісти. Прийнято вважати, що кожний програміст є людиною (кодерів з Марсу та хакерів з планети Нібіру зараз не враховуємо), проте не кожна людина є програмістом: Програмісти – підмножина Людей.

Поліморфізм

Поліморфізм – це той третій «кит» (перші два – це інкапсуляція та успадковування), на якому тримається об’єктно-орієнтоване програмування.

Поліморфізм означає «багато форм», – характеризує «гнучкість» програмного коду.

Поліморфізм – це можливість об’єктів одного типу мати різну реалізацію; і найцікавіше – можливість об’єкта базового класу використовувати методи похідного класу, який ще не існує на момент створення базового.

Віртуальні функції (ті, що реалізують поліморфізм) – одна з найбільш цікавих та потужних концепцій C++, яка (якщо не розуміти механізму її реалізації) зазвичай виглядає містикою.

Поліморфізм можна поділити на 2 види:

- *статичний поліморфізм*, який представлений перевантаженням функцій (на одному рівні можна описати багато однойменних функції, – головне, щоб вони відрізнялися набором параметрів) та шаблонами функцій – ці теми розглядали раніше;
- *динамічний поліморфізм*, який власне реалізований за допомогою віртуальних функцій і який буде детально розглядатися у відповідній темі.

Характерні особливості розробки об'єктно-орієнтованих програм

Цілісність об'єктів

При об'єктно-орієнтованому проектуванні насамперед розглядаємо цілісні об'єкти – тобто, не розриваємо зв'язок між описом даних та описом дій над даними.

Проектування класів

При проектуванні класів ми розглядаємо не статичну картину предметної області (фотознімок), як це робиться при структурному програмуванні, а динамічну (потік відео). Це означає, що ми маємо враховувати не лише дані, які описують предметну область, а і дії над цими даними. Тобто, слід не лише спроектувати структуру даних, а (що не менш важливо) – структуру функцій, які опрацьовують ці дані.

При структурному проектуванні ми розподіляли дані між сутностями як атрибути відповідних сутностей.

При об'єктно-орієнтованому проектуванні ми маємо розподілити дані та функції як поля та методи відповідних класів.

Правило проектування класів

Принцип **KISS**: **K**eep **i**t **s**imple, **s**tupid! (не потребує перекладу)

- Кожний клас має бути якомога простіший і має представляти лише одну сутність (тобто, лише один набір однотипних екземплярів предметної області);
- Краще багато простих класів, ніж один чи кілька складних. Слід уникати класу, який поєднує у собі все (всі сутності предметної області) – тобто, уникати «класу-Бога»;
- Класи мають мати значущі імена (ім'я класу – це ім'я відповідної сутності) та очевидну будову і функціонал.

Правило проектування методів

Принцип **KISS**: **K**eep **i**t **s**imple, **s**tupid! (не потребує перекладу)

- Кожний метод має бути якомога простішим і має виконувати лише одну роль;
- Краще багато простих методів, ніж однин чи кілька складних. Слід уникати методу, який виконує всі дії– тобто, уникати «методу-Бога»;
- Методи мають мати значущі імена (ім'я методу – це ім'я відповідної ролі) та очевидний функціонал.

Проектування ієрархії класів

Послідовність дій

- Виконується «знизу догори»
- Починаємо від конкретних класів (вони будуть на нижніх рівнях ієрархії)
- Всі спільні характеристики та спільні дії об’єднуємо та виносимо у більш загальні базові класи, які утворюють вищі рівні ієрархії
- Продовжуємо, поки не дійдемо до найбільш загальних класів

Межі застосовності об’єктно-орієнтованого програмування

Хоча ООП – дуже потужна технологія, не потрібно для примітивних задач робити громіздку ієрархію класів.

Прості задачі потребують простих рішень.

Класи та об’єкти C++

Визначення класу

Визначення класу

Загальний синтаксис визначення класу:

```
class ім’я
{
private:
    // опис прихованих елементів
    ...
public:
    // опис доступних елементів
    ...
};  // обов'язково має бути крапка з комою
```

Всі елементи класу за умовчанням – приховані та не доступні, тому директива `private:` – не обов’язкова. Структура в C++ – теж є своєрідним класом:

```
struct ім’я
{
private:
    // опис прихованих елементів
    ...
public:
    // опис доступних елементів
    ...
};  // обов'язково має бути крапка з комою
```

Єдина відмінність структур від класів: елементи структури за умовчанням – доступні.

При визначенні класу можна вказувати кілька секцій `private` та `public`, порядок їх розташування значення не має.

Синтаксис визначення класу розглянемо на прикладі класу Account, який описує банківський рахунок і містить поля прізвище власника (name), залишок коштів на рахунку (summa) та метод змінити кошти на рахунку (Change):

```
class Account
{
    string name;               // поле = змінна в складі класу
    double summa;              // поле = змінна в складі класу

    void Change(double value)  // метод = функція в складі класу
    {
        if (value > 0)                       // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa) // знімаємо кошти
            summa += value;
    }
};  // обов'язково в кінці має бути крапка з комою
```

В цьому прикладі показана *інкапсуляція* (в першому тлумаченні) – поєднання в єдине ціле *полів* та *методів* класу.

Елементи класу

Елементи класу поділяються на *поля* та *методи*.

Поля – це дані, які інкапсулює клас (змінні, які належать класу).

Методи – це дії, які можна виконувати над класом та його полями (функції, які належать класу).

Директиви доступу до елементів класу

Елементи, описані після директиви `private:`, доступні лише всередині класу (в т.ч. і в усіх методах класу). Цей вид доступу діє в класі за умовчанням.

Інтерфейс класу описують після директиви `public:` – елементи, описані після цієї директиви, доступні скрізь: і всередині, і поза класом.

Слід зауважити, що у такому вигляді клас Account з попереднього прикладу використовувати не можна, бо не можна організувати доступу до полів та методів цього класу – адже всі вони приховані (`private`).

За умовчанням всі елементи (поля та методи) класу недоступні. Це означає, що до них можна звертатися лише із методів цього класу, і не можна – із інших функцій.

Для управління доступом до елементів класу використовуються директиви `private`, `protected`, `public`.

Директива `protected` має сенс лише для ієрархії класів, утворених зв'язками успадковування, – її будемо розглядати при вивченні теми «Успадковування».

Директива `private` встановлює максимальні обмеження доступу: елементи класу з рівнем доступу `private` повністю недоступні поза класом, до них можна звертатися лише із методів цього класу. Такі елементи – недоступні та приховані.

Директива `public` скасовує всі обмеження – відповідні елементи класу доступні скрізь. Такі елементи – доступні та видимі.

Для того, щоб визначити певний рівень доступу до елементів класу, слід перед цими елементами вказати відповідну директиву доступу, яка складається із ключового слова (`private`, `protected`, `public`), за яким записується двокрапка:

```
class ім’я
{
private:
    // опис прихованих (недоступних) елементів
    ...
public:
    // опис видимих (доступних) елементів
    ...
};  // обов'язково має бути крапка з комою
```

Це – друге тлумачення *інкапсуляції*: управління доступом до елементів класу.

Прийнято максимально обмежувати доступ до полів: поля оголошують з директивою `private`. Це дозволяє запобігти неправильному використанню полів. Для того, щоб можна було записати та прочитати значення полів, оголошують *методи доступу* (методи *зчитування* та *запису*). Для нашого прикладу це буде виглядати так:

```
class Account
{
private:                        // приховані елементи класу (поля)
    string name;                // поле
    double summa;               // поле

public:                         // доступні елементи класу (методи)
    // методи доступу
    string getName()            // метод зчитування поля name
    {
        return name;
    }

    void setName(string value) // метод запису поля name
    {
        name = value;
    }

    double getSumma()           // метод зчитування поля summa
    {
        return summa;
    }

    void setSumma(double value)// метод запису поля summa
    {
        summa = value;
    }

    void Change(double value)  // метод зміни залишку коштів на рахунку
    {
        if (value > 0)                      // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa) // знімаємо кошти
            summa += value;
    }
};  // обов'язково в кінці має бути крапка з комою
```

Поля класу

Поля – це дані, які інкапсулює клас (змінні, які належать класу).

Поля класу:

– можуть мати будь-який тип, крім типу цього класу (проте можуть бути вказівниками чи посиланнями на цей клас), – як і поля структур;
– можуть бути описані з модифікатором `const`, при цьому вони ініціалізуються лише один раз (за допомогою конструктора) і не можуть змінюватися (про конструктори буде в розділі «Конструктори»);
– можуть бути описані з модифікатором `static` (про це буде в розділі «Статичні елементи класу»), але не як `auto`, `extern` чи `register`.

При визначенні полів їх ініціалізація не допускається.

Приклад:

```
class Student
{
private:
    string name;
    int kurs;

public:
    string getName() const { return name; }
    int    getKurs() const { return kurs; }
    void setName(string value);
    void setKurs(int value);               // оголошення методу
};

void Student::setName(string value)
{
    name = value;
}

void Student::setKurs(int value)           // визначення методу поза класом
{
    if (value > 0 && value < 7)
        kurs = value;
    else
    {
        kurs = 0;
        cerr << "Wrong kurs value!" << endl;
    }
}
```

Оголошення класу

Приклад оголошення класу:

```
class Student;
```

Це – не визначення, а лише оголошення класу, не можна створювати об’єктів класу, який не визначений, а лише оголошений, проте можна створювати вказівники та посилання на об’єкти оголошеного класу.

Зазвичай оголошення класу використовують, коли один клас залежить від іншого, а визначення цього іншого класу – недоступне, наприклад:

```
class A
{
    B* b;
};

class B
{
    A* a;
};
```

В наведеному прикладі в класі A оголошено поле b – вказівник на об’єкт класу B. В свою чергу, в класі B оголошено поле a – вказівник на об’єкт класу A.

На момент компіляції класу A система нічого не знає про клас B, тому цей код скомпільований не буде.

Якщо поміняти місцями визначення класу A та класу B, – це не усуне проблему, бо тепер при компіляції класу B система нічого не знає про клас A:

```
class B
{
    A* a;
};

class A
{
    B* b;
};
```

Щоб виправити цю помилку і вірно скомпілювати два взаємозалежні класи, слід використовувати оголошення класу: спочатку оголошуємо клас B, потім – визначаємо клас A (який буде вірно скомпільований, бо система вже знає, що ім’я B означає клас).

Після оголошення класу далі за текстом програми має бути визначення відповідного класу:

```
class B;

class A
{
    B* b;
};

class B
{
    A* a;
};
```

Оголошення класу подібні до прототипів функцій.

Створення / опис / оголошення об’єктів

Після того, як був визначений клас, ми можемо створювати об’єкти цього класу. Об’єкти створюються аналогічно до змінних інших типів:

```
Account a;                  // об'єкт - звичайна змінна
Account* b = new Account(); // об'єкт - динамічна змінна
Account c[3];               // масив із трьох об'єктів
```

Ще приклади:

```
Student s;
Student* p = new Student;
Student* q = new Student();
Student g[25];
Student& c = s;
```

Для об’єкта, створеного командою

```
Student *q = new Student();
```

виділена пам’ять ініціалізується нулями, а для об’єкта, створеного командою

```
Student *p = new Student;
```

– ні.

Доступ до елементів класу для об’єктів та вказівників на об’єкти

Звертання до елементів класу – подібне до звертання до елементів структури: використовуються операція «крапка» для доступу через звичайну змінну та операція «стрілка» для доступу через вказівник на динамічну змінну.

Для об’єктів:

```
об’єкт.поле
об’єкт.метод(аргументи)
```

Для вказівників на об’єкти::

```
вказівник_на_об’єкт -> поле
вказівник_на_об’єкт -> метод(аргументи)
```

Приклади:

Для об’єктів класу Student:

```
s.setKurs(2);
s.setName("Вася");

q->setKurs(1);
q->setName("Петя");
```

Оскільки всі поля класу Account – приховані та недоступні, то звертатися до них можна лише із методів цього ж класу:

```
a.name = "Іваненко";
a.summa = 100.0;
```

```
b->name = "Петренко";
b->summa = 150.0;

c[0].name = "Сидоренко";
c[0].summa = 175.0;
```

Всі методи – доступні, тому до них можна вертатися із будь-якої функції (наприклад, main()):

```
a.setName("Іваненко");
a.setSumma(100.0);

b->setName("Петренко");
b->setSumma(150.0);

c[0].setName("Сидоренко");
c[0].setSumma(175.0);
```

Розподіл пам’яті для об’єктів.

Зберігання звичайних полів та методів

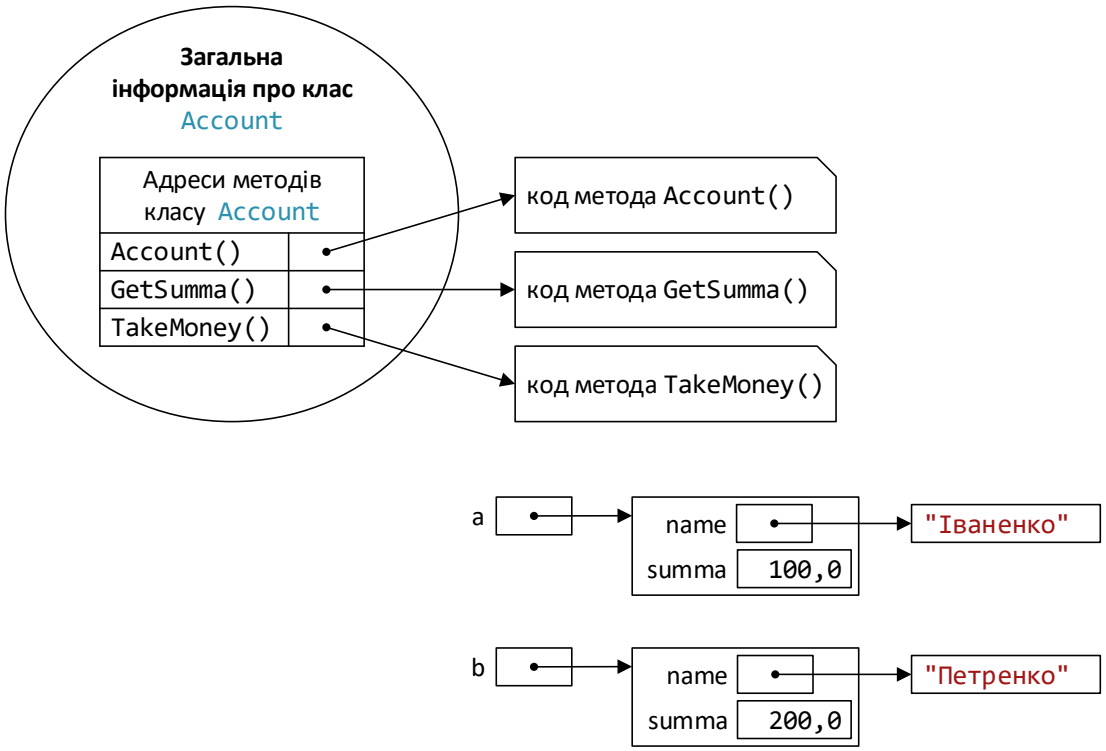
Розглянемо розподіл пам’яті для об’єктів на прикладі класу Account:

```
class Account
{
private:
    string name;
    double summa;

public:
    Account(string name, double summa)
    {
        /* ... */
    }

    double GetSumma()
    {
        return summa;
    }

    bool TakeMoney(double summa)
    {
        /* ... */
    }
};

int main()
{
    Account* a = new Account("Іваненко", 100);
    Account* b = new Account("Петренко", 200);

    cout << a->GetSumma() << endl;
}
```

В наведеній програмі створено два динамічних об’єкти класу Account.

Кожний об’єкт містить свої екземпляри звичайних полів. Це означає, що поля name та summa об’єкта *a – займають інші області пам’яті, ніж поля name та summa об’єкта *b.

Крім того, в певній області пам’яті, відведеній для загальної інформації про клас Account, міститься таблиця адрес методів цього класу.



Опис методів

Методи – це дії, які можна виконувати над класом та його полями (функції, які належать класу).

Методи можна описувати в класі (це рекомендується лише для дуже компактних методів, які не містять циклів та рекурсивних викликів), такі методи вважаються inline-функціями (не генеруються команди виклику і повернення, а код функції підставляється в місце виклику).

Методи також можна описувати і поза класом, тоді для імені методу необхідно вказувати доступ до простору імен класу за допомогою префіксу: імені класу та операції доступу.

Вказівник `this` та відмінність методів від звичайних функцій

Кожний метод, крім явно вказаних параметрів, приймає ще один параметр, визначений неявно: вказівник на об’єкт, який викликає цей метод. Цей вказівник має ім’я `this`, його не потрібно описувати!

Приклад:

Опис метода в програмі:

```
void Student::setName(string value)
{
    name = value;
}
```

Фактично генерується наступний код методу:

```
void Student::setName(Student *this, string value)
{
    this->name = value;
}
```

Для об’єкта, створеного командою

```
Student s;
```

коли в програмі викликаємо цей метод командою:

```
s.setName("Вася");
```

– то фактично генерується наступний код виклику

```
s.setName(&s, "Вася");
```

– адреса того об’єкта, який викликав цей метод, передається у метод і стає значенням вказівника `this`.

Реалізація інкапсуляції

Для динамічного об’єкта, створеного командою

```
Account* a = new Account("Іваненко", 100);
```

– що буде виконувати наступна команда?

```
cout << a->GetSumma() << endl;
```

– напевно, виведе інформацію про суму (залишок коштів) на рахунку об’єкта `a`.

Звідки метод `GetSumma()` буде знати, що слід повернути значення поля `summa` саме об’єкта `a`? – адже в тілі цього метода ніякої інформації про це немає:

```
class Account
{
...
public:
    double GetSumma()
    {
        return summa;
    }
};
```

Кожний звичайний (не статичний) метод, крім явно вказаних параметрів, приймає ще один параметр, визначений неявно: вказівник на той об’єкт, який викликав цей метод. Цей вказівник має ім’я `this`, його не потрібно описувати!

Команда

```
cout << a->GetSumma() << endl;
```

буде виконуватися наче

```
cout << a->GetSumma( a ) << endl;
```

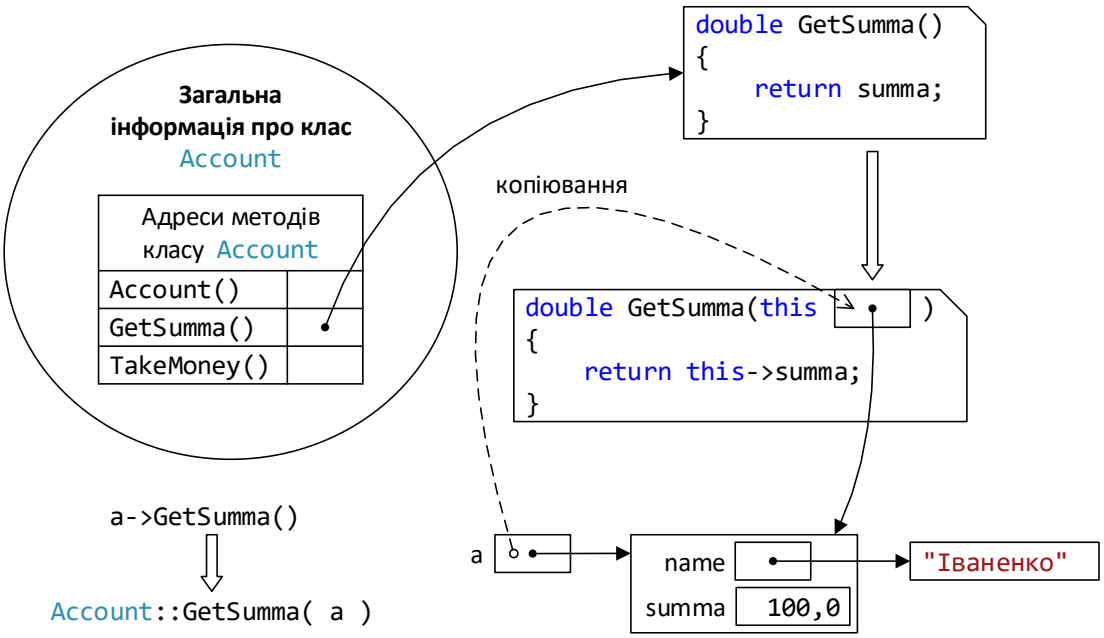
– компілятор генерує код, у якому адреса динамічного об’єкта (тобто, значення вказівника a) передається у метод і стає значенням вказівника this.

Фактично, на основі визначення класу

```
class Account
{
...
public:
    double GetSumma()
    {
        return summa;
    }
};
```

генерується такий код

```
class Account
{
...
public:
    double GetSumma( Account *this )
    {
        return this->summa;
    }
};
```



Виклик a->GetSumma() компілятором опрацьовується як Account::GetSumma(a).

Адреса об’єкта (тобто, значення вказівника a) передається у метод і копіюється як значення вказівника this. Тепер this вказує на той ж самий динамічний об’єкт, що і a.

В тілі метода команда return summa; виконується як return this->summa;. Таким чином, метод GetSumma() повертає значення поля summa того об’єкта, на який налаштований вказівник this – тобто, нашого динамічного об’єкта: a->summa.

Визначення методів в класі та поза класом

У розглянутому прикладі всі методи визначалися в класі:

```
class Account
{
private:                        // приховані елементи класу (поля)
    string name;                // поле
    double summa;               // поле

public:                         // доступні елементи класу (методи)
    // методи доступу
    string getName()            // метод зчитування поля name
    {
        return name;
    }

    void setName(string value) // метод запису поля name
    {
        name = value;
    }

    double getSumma()           // метод зчитування поля summa
    {
        return summa;
    }

    void setSumma(double value)// метод запису поля summa
    {
        summa = value;
    }

    void Change(double value)  // метод зміни залишку коштів на рахунку
    {
        if (value > 0)                          // надходять кошти
            summa += value;
        if (value < 0 && abs(value) < summa)  // знімаємо кошти
            summa += value;
    }
};  // обов'язково в кінці має бути крапка з комою
```

Зазвичай в класі визначаються лише найпростіші методи – тіло яких містить одну-дві команди. У всіх інших випадках, щоб не робити визначення класу занадто громіздким, методи визначаються поза класом. Для цього в класі вказуємо лише заголовок (прототип) методу, а повне його визначення записуємо окремо. При цьому перед іменем методу вказуємо префікс: ім’я класу та дві двокрапки:

```
class Account
{
private:                         // приховані поля
    string name;
    double summa;

public:                          // доступні методи
    string getName()             // метод зчитування поля name
    {
        return name;
    }
```

```
    void setName(string value)  // метод запису поля name
    {
        name = value;
    }

    double getSumma()           // метод зчитування поля summa
    {
        return summa;
    }

    void setSumma(double value) // метод запису поля summa
    {
        summa = value;
    }

    void Change(double value);  // прототип - оголошення методу

};  // обов'язково в кінці має бути крапка з комою

// визначення методу (повний опис методу) поза класом
void Account::Change(double value)  // перед іменем – префікс ім'я класу
{
    if (value > 0)
        summa += value;
    if (value < 0 &&
        abs(value) < summa)
        summa += value;
}
```

Префікс перед іменем методу вказує, що це – не звичайна функція, а метод, який належить до простору імен вказаного класу.

Визначення методів в класі

Методи можна описувати в класі (це рекомендується лише для дуже компактних методів, які не містять циклів та рекурсивних викликів), такі методи вважаються inline-функціями (не генеруються команди виклику і повернення, а код функції підставляється в місце виклику):

```
class Student
{
private:
    string name;
    int kurs;

public:
    string getName() const { return name; }
    int    getKurs() const { return kurs; }
    void setName(string value);
    void setKurs(int value);
};
```

– методи `getName()` та `getKurs()` визначені в класі

Визначення методів поза класом

Методи також можна описувати і поза класом, тоді для імені методу необхідно вказувати доступ до простору імен класу за допомогою префіксу: імені класу та операції доступу.

При визначенні класу вказуються прототипи таких методів:

```
class Student
{
private:
    string name;
    int kurs;

public:
    string getName() const { return name; }
    int    getKurs() const { return kurs; }
    void setName(string value);
    void setKurs(int value);
};

void Student::setName(string value)
{
    name = value;
}

void Student::setKurs(int value)
{
    if (value > 0 && value < 7)
        kurs = value;
    else
    {
        kurs = 0;
        cerr << "Wrong kurs value!" << endl;
    }
}
```

– методи `setName()` та `setKurs()` визначені поза класом.

Перевантаження методів

Як і звичайні функції, методи можуть бути перевантажені. Перевантажений метод мусить мати інший набір параметрів (тобто, параметри мають бути іншого типу і / або має бути різна кількість параметрів).

Приклад:

```
class Addition
{
public:
    int    Add(int a,    int b)    { return a + b; }
    double Add(double a, int b)    { return a + b; }
    double Add(int a,    double b) { return a + b; }
    double Add(double a, double b) { return a + b; }
};
```

Константні методи

Можна створити *константний об'єкт*:

```
const Student t;
```

Поля константного об'єкту змінювати не можна, для нього можна викликати лише константні методи:

```
cout << t.GetName() << endl;
```

Константний метод:

– оголошується з ключовим словом `const` після списку параметрів;

– не може змінювати полів класу;

– може викликати лише константні методи;

– може викликатися будь-якими (не лише константними) методами;

– може викликатися для будь-яких (не лише константних) об'єктів.

Рекомендується всі методи, які призначені для отримання значень полів, та які не змінюють поля об'єктів, описувати явно як константні методи.

Приклад:

```
class Student
{
private:
    string name;
    int kurs;

public:
    string GetName() const { return name; }
    int    GetKurs() const { return kurs; }
    void SetName(string value) { name = value; }
    void SetKurs(int value)    { kurs = value; }
};
```

Методи доступу

Зазвичай поля класу – скриті (приватні), для доступу до полів створюють відповідні методи: сетери – для встановлення значення (англ. *set* – встановити) – починаються з префіксу `set` та гетери – для отримання значення (англ. *get* – отримати) – починаються з префіксу `get`.

Методи доступу, які повертають значення булевих полів, зазвичай починаються з префіксу `is` (англ. *is* – є).

Рекомендується надавати полям та методам значущі імена.

Наприклад, методи з наступними іменами означають: `isEmpty()` – перевірка, чи якесь поле – порожнє, `isValid()` – перевірка, чи значення поля – допустиме.

Створювати методи доступу – хороший стиль, навіть якщо для поточної задачі вони не є необхідними.

Методи доступу дозволяють сховати реалізацію класу від користувача. Це, в свою чергу, дозволяє змінювати реалізацію класу, не змінюючи його інтерфейсу, – тобто, не змінюючи всіх клієнтських програм, які використовують цей клас.

Розглянемо приклад: клас – планувальник подій, одне із полів якого – дата події.

Створюємо в класі приховане поле `date` та доступні методи `GetDate()` та `SetDate()`.

Припустимо, що для зберігання дати використовується масив із трьох цілих чисел розміром 1 байт – елементи якого містять номери дня, місяця, останніх двох цифр року:

```
unsigned char date[3];
```

– елементи `date[0]`, `date[1]`, `date[2]` – це відповідно, день, місяць, рік.

Метод `GetDate()` перетворює дату із вказаного формату у літерний рядок виду `"DD/MM/YY"` – тобто, у вигляд, зручний для людини.

Метод `SetDate()` – виконує зворотне перетворення: літерний рядок виду `"DD/MM/YY"` парсить, щоб отримати значення елементів масиву `unsigned char date[3]`.

Припустимо тепер, що замовник нашої програми вирішив розширити її функціонал: тепер нам слід передбачити, що події можуть відбуватися не лише у цьому столітті, а і у інших століттях (наприклад, замовник придбав еліксир вічного життя, чи винайшов машину часу, – зараз це не важливо).

В цьому випадку для представлення року вже не буде вистачати 1 байту – слід поміняти представлення даних. Тепер будемо представляти дату як наступну структуру:

```
struct Date
{
    unsigned char day, month;
    int year;                   // від'ємні роки – це дати до нашої ери
};
...
Date date;
```

Щоб наша програма коректно опрацьовувала нове представлення дати, достатньо поміняти лише код методів `GetDate()` та `SetDate()`, – звісно, за умови, що ми скрізь використовуємо лише методи доступу, а не пряме звертання до полів!

Тепер старий формат літерного рядка `"DD/MM/YY"` буде означати дату `DD/MM/20YY`, а новий формат `"DD/MM/YYYY"` – дату для вказаного року.

Таким чином, використання методів доступу дозволяє безболісно змінювати внутрішнє представлення даних класу без необхідності переписувати увесь код клієнтських програм, які використовують цей клас.

Навіть якщо для поточної задачі методи доступу не є необхідними, вони можуть стати потрібними в майбутньому – і тоді вони вже будуть у класі.

Міркування економії пам’яті не є поважною причиною не створювати методів доступу: Якщо компілятор залишає код невикористаних методів у виконуваному файлі (бо деякі компілятори можуть видаляти код таких методів) – для сучасних технологій розробки програм обсяг пам’яті не є критичним показником. Набагато важливішими показниками є зручність і зрозумілість коду та його швидкодія.

Дружні функції та дружні класи

Поняття інтерфейсу класу

Інтерфейс класу – це доступні елементи класу. Рекомендується доступними робити лише методи, потрібні користувачеві; всі поля мають бути прихованими (не доступними); допоміжні методи, які не потрібні користувачеві, а допомагають реалізувати відкриті методи – також мають бути прихованими.

Поняття та призначення дружніх функцій та дружніх класів

Інколи буває потрібно забезпечити безпосередній доступ ззовні до прихованих елементів класу, тобто розширити інтерфейс класу, відкрити клас для деяких програмних ресурсів. Для цього використовують дружні функції та дружні класи: при визначенні класу можна оголосити деякі зовнішні функції чи класи як дружні для цього класу, такі друзі будуть мати повний доступ до всіх елементів класу. При цьому клас сам визначає, які функції чи інші класи будуть для нього дружніми, а які – ні.

Оголошення та опис дружньої функції

Дружні функції використовуються для доступу до прихованих полів класу та являють собою альтернативу методам. Методи зазвичай використовують для реалізації властивостей об’єктів. У вигляді дружніх функцій подаються дії, які хоча і не описують властивостей класу, проте концептуально належать до його інтерфейсу і потребують доступу до його прихованих полів, наприклад, операції введення-виведення об’єктів (ці операції будуть розглядатися в наступних розділах).

Правила опису та особливості використання дружніх функцій:

– Дружня функція оголошується *всередині класу*, до елементів якого їй потрібно відкрити доступ, з ключовим словом `friend`. В якості параметру їй необхідно передати об’єкт або посилання на об’єкт такого класу, оскільки дружня функція – не метод і вказівник `this` їй не передається.
– Дружня функція може бути звичайною функцією або методом іншого вже визначеного класу, на неї не поширюється дія специфікаторів доступу (для неї несуттєві директиви `private:` чи `public:`), місце розміщення в класі оголошення дружньої функції – не суттєве.
– Одна функція може бути дружньою одразу кільком класам.

Розглянемо приклад визначення двох дружніх функцій:

Функція f1() – метод класу A, дружній для класу B; функція f2() – звичайний метод класу B; функція f3() – не належить жодному класу, вона – теж дружня для класу B.

```
class B;                    // попереднє оголошення класу

class A
{
public:
    void f1(B&);            // прототип методу класу A - приймає посилання на B
};

class B
{
public:
    void f2();              // прототип звичайного методу класу B

    friend void f3(B&);     // дружня зовнішня функція
    friend void A::f1(B&);  // дружній метод з класу A
};                          // клас A має бути визначений раніше

void A::f1(B& x) { /* ... */ }
void B::f2()     { /* ... */ }
void f3(B& x)    { /* ... */ }
```

Слід, де можливо, уникати використання дружніх функцій, оскільки вони порушують принцип інкапсуляції і тим самим ускладнюють модифікацію та налагодження програми.

Оголошення та опис дружнього класу

В попередньому прикладі метод f1() класу A мусив мати доступ до прихованих елементів класу B, тому цей метод в класі B був оголошений як дружня функція.

Якщо всі методи деякого класу мають мати доступ до прихованих елементів іншого класу, то весь клас оголошується дружнім за допомогою ключового слова `friend`. В наступному прикладі клас B оголошено як дружній для класу A:

```
class A
{
    friend class B;
};

class B
{
    void f1();
    void f2();
};

void B::f1() { /* ... */ }
void B::f2() { /* ... */ }
```

Методи f1() та f2() класу B є дружніми функціями по відношенню до класу A (хоча вони і описані без ключового слова `friend`) і мають доступ до всіх елементів класу A.

Оголошення `friend` не є специфікатором доступу і не успадковується (успадковування пояснюється в наступних темах).

Вкладені класи. Композиція та агрегування класів

Вкладені класи

Оголошення вкладених класів

В блоці визначення класу (тобто, в області його дії) можна оголосити змінні – це будуть поля класу, можна оголосити функції – вони будуть методами класу, а можна також оголосити інший клас.

Клас, оголошений всередині іншого класу, називається *вкладеним*. Будемо називати той клас, який охоплює вкладений клас, *зовнішнім* класом. Вкладений клас є елементом того класу, що його охоплює, тобто – елементом зовнішнього класу. Як і будь-який інший елемент класу, він підпорядковується директивам доступу: визначення вкладеного класу може бути відкритим (`public`) або закритим (`private`). Рівень вкладеності не обмежується.

Ім’я вкладеного класу має бути унікальним в зовнішньому класі, проте може збігатися з іншими іменами поза класом.

Ні вкладений клас, ні зовнішній не можуть безпосередньо звертатися до методів один одного; як і у випадку звичайних не вкладених класів, необхідно оголосити об’єкт (або вказівник), для якого вже викликати потрібний метод. Об’єкт зовнішнього класу може передаватися методу вкладеного класу як аргумент, наприклад:

```
class External
{
public:
    void methodExternal() {}

    class Inner                 // вкладений клас
    {
    public:
        void methodInner(External t);
    };
};

void External::Inner::methodInner(External t)
{
    t.methodExternal();         // виклик метода зовнішнього класу
}
```

– метод вкладеного класу `methodInner()` отримує в якості параметру об’єкт `t` зовнішнього класу і звичним способом викликає метод зовнішнього класу `methodExternal()`. Цей метод визначено поза визначенням зовнішнього класу.

В області глобальної видимості поза визначенням зовнішнього класу можна визначити і сам вкладений клас, якщо в зовнішньому класі вказати оголошення вкладеного класу, наприклад:

```
class External
{
    // ...

    class Inner;                // оголошення вкладеного класу
    // ...
};

class External::Inner           // зовнішнє визначення вкладеного класу
{
    // ...
};
```

Вкладені класи знаходяться в області зовнішнього класу і доступні для використання всередині цієї області.

Для звертання до вкладеного класу із області, яка відрізняється від області, що безпосередньо містить цей вкладений клас, необхідно використовувати повне ім’я.

Наступний приклад демонструє оголошення вкладеного класу.

```
// nested_class_declarations.cpp

class BufferedIO
{
public:
    enum IOError { None, Access, General };

    // визначаємо вкладений клас BufferedInput:
    class BufferedInput
    {
    public:
        int read();

        int good()
        {
            return _inputerror == None;
        }

    private:
        IOError _inputerror;
    };

    // визначаємо вкладений клас BufferedOutput:
    class BufferedOutput
    {
        // список елементів
    };
};

int main()
{
}
```

Класи `BufferedIO::BufferedInput` та `BufferedIO::BufferedOutput` оголошені всередині класу `BufferedIO`. Проте об'єкти типу `BufferedIO` не містять об'єктів типу `BufferedIO::BufferedInput` чи `BufferedIO::BufferedOutput` (оголошення вкладеного класу – це лише вкладеність блоків опису даних, а не композиція чи агрегування).

Вкладені класи можуть безпосередньо використовувати імена елементів, імена типів, імена статичних елементів та переліки лише із свого зовнішнього класу. Для використання імен інших членів класу необхідно використовувати вказівники, посилання або імена об'єктів.

В прикладі класу `BufferedIO` до перелічуваного типу `IOError` можна отримати доступ безпосередньо за допомогою методів вкладених класів `BufferedIO::BufferedInput` чи `BufferedIO::BufferedOutput`, як показано для методу `good()`.

Вкладені класи оголошують лише типи в межах області зовнішнього класу. Вони не створюють об'єктів ні в зовнішньому, ні у вкладеному класі. В попередньому прикладі оголошено два вкладені класи, але не визначено об'єктів цих класів.

Винятком із видимості області оголошення вкладеного класу є оголошення імені типу разом з попереднім описом. В цьому випадку ім'я класу, оголошене за допомогою попереднього опису, видиме за межами зовнішнього класу, область видимості визначається як найменша область, яка містить клас. Наприклад:

```
// nested_class_declarations_2.cpp

class C
{
public:
    typedef class U u_t;    // клас U видимий поза областю класу C
    typedef class V {} v_t; // клас V не видимий поза областю класу C
};                          // u_t та v_t – синоніми класів U та V відповідно
                            // U – попередній опис, V – визначення
int main()
{
    // вірно, використовується попередній опис, область його видимості – «файл»
    U* pu;          // на основі попереднього опису можна визначити вказівник

    // помилка, ім'я синоніму існує лише в області класу C
    u_t* pu2;       // E0020

    // помилка, клас V визначений як вкладений по відношенню до класу C
    V* pv;          // E0020

    // вірно, використовується повне ім'я класу V
    C::V* pv2;
}
```

Права доступу та вкладені класи

Вкладення одного класу у інший клас не надає програмі ніяких особливих прав доступу до елементів вкладеного класу. Аналогічно, методи зовнішнього класу не мають ніяких особливих прав доступу до елементів вкладеного класу.

```
class External
{
    // ...
    class Inner  // вкладений клас
    {
        // ...
    };
};
```

Якщо вкладений клас оголошено доступним (в секції `public`), то його можна використовувати в якості типу по всій програмі, – тоді його ім'я слід писати з префіксом, вказуючи область дії зовнішнього класу та операцію доступу:

```
External::Inner *p;
```

Якщо ж вкладений клас оголошено в закритій частині зовнішнього класу (в секції `private`), то він доступний лише елементам зовнішнього класу та його друзям. В цьому випадку елементи вкладеного класу зазвичай оголошують відкритими – тоді немає потреби і оголошувати другом зовнішній клас, наприклад:

```
class External
{
    // ...
    friend class Inner;         // Inner доступні приватні елементи External

    struct Inner                // вкладений клас (структура – теж клас!)
    {
        // ...                     всі елементи якого – відкриті та доступні
    };
};
```

Цей приклад буде вірно скомпільований. Можна оголосити другом внутрішню структуру, наприклад:

```
class External
{
    // ...
    friend struct Inner;        // Inner доступні приватні елементи External

    struct Inner                // вкладений клас (структура – теж клас!)
    {
        // ...                     всі елементи якого – відкриті та доступні
    };
};
```

Методи вкладених класів

Всередині методу вкладеного класу ключове слово `this` є вказівником на об'єкт вкладеного класу, для якого викликається цей метод.

Методи вкладеного класу можна реалізувати безпосередньо всередині класу.

Якщо ж методи вкладеного класу визначаються поза класом, то визначення слід записувати поза найбільш зовнішнім із тих класів, які охоплюють такий клас – в області глобальної видимості. Ім'я методу в цьому випадку слід записувати з префіксами, які будуть уточнювати операцію доступу до області дії відповідного класу; кількість префіксів відповідає рівню вкладеності класів.

Якщо використати визначення методів, оголошених у вкладених класах, поза класом (в області файлу), то попередній приклад можна записати так:

```
// member_functions_in_nested_classes.cpp

class BufferedIO
{
public:
    enum IOError { None, Access, General };

    class BufferedInput
    {
    public:
        int read(); //  оголошення, але не визначення
        int good(); //  методів read() та good()

    private:
        IOError _inputerror;
    };

    class BufferedOutput
    {
        //  список елементів
    };
};

//  визначаємо методи read() та good()
//  поза класом – в області видимості файлу

int BufferedIO::BufferedInput::read()
{
    return (1);
}

int BufferedIO::BufferedInput::good()
{
    return _inputerror == None;
}

int main()
{
}
```

В цьому прикладі префікс *повне-ім'я-типу* використовується для оголошення імені функції.

Оголошення

```
BufferedIO::BufferedInput::read()
```

означає «функція `read()`, яка є елементом класу `BufferedInput`, який міститься в області класу `BufferedIO`». Оскільки при цьому використовується синтаксис *повне-ім'я-типу*, то можливі наступні конструкції:

```
typedef BufferedIO::BufferedInput BIO_INPUT;

int BIO_INPUT::read()
```

Це оголошення еквівалентне попередньому, але в ньому використовується команда `typedef`, яка створює синонім імені типу, замість складеного імені класу.

Дружні функції та вкладені класи

За умовчанням вкладений клас не має доступу до приватних елементів зовнішнього класу, так само які і зовнішній клас – до приватних елементів вкладеного класу. Обійти це обмеження можна за допомогою механізму дружніх відношень (з використанням ключового слова `friend`), наприклад:

```
class External
{
    // ...
    friend class Inner;         // Inner доступні приватні елементи External

    class Inner                 // вкладений клас
    {
        // ...
        friend class External;  // External доступні приватні елемент Inner
    };
};
```

Дружні функції, оголошені у вкладеному класі, містяться в області вкладеного, а не зовнішнього класу. Тому ці дружні функції не отримують ніяких особливих прав доступу до елементів зовнішнього класу.

Якщо необхідно використати ім’я, оголошене у вкладеному класі, в дружній функції, і ця дружня функція визначена в області файлу, слід використовувати повні імена типів:

```
// friend_functions_and_nested_classes.cpp

enum
{
    sizeOfMessage = 255  // визнавчили цілу константу
};

char* rgszMessage[sizeOfMessage];


class BufferedIO
{
public:
    class BufferedInput
    {
    public:
        friend int GetExtendedErrorStatus();
        static char* message;
        static int  messageSize;
        int iMsgNo;
    };
};

char* BufferedIO::BufferedInput::message;
int BufferedIO::BufferedInput::messageSize;

int GetExtendedErrorStatus()
{
    int iMsgNo = 1; // присвоїли довільне значення номеру повідомлення
```

```
    strcpy_s(BufferedIO::BufferedInput::message,
             BufferedIO::BufferedInput::messageSize,
             rgszMessage[iMsgNo]);

    return iMsgNo;
}

int main()
{
}
```

В цьому прикладі функція `GetExtendedErrorStatus()` оголошена як дружня. В тілі функції (визначеної в області файлу) повідомлення копіюється із статичного масиву в поле класу. Для оптимальної реалізації функції `GetExtendedErrorStatus()` її рекомендується оголосити так:

```
int GetExtendedErrorStatus( char *message )
```

Композиція та агрегування класів

В об’єктно-орієнтованому програмуванні під агрегуванням (в широкому значенні, його ще називають композицією або включенням) мають на увазі методику створення нового класу із вже створених класів – подібно до того, як певний виріб збирається із готових деталей. Агрегування часто називають «відношенням приналежності» за принципом «автомобіль має корпус, колеса та двигун».

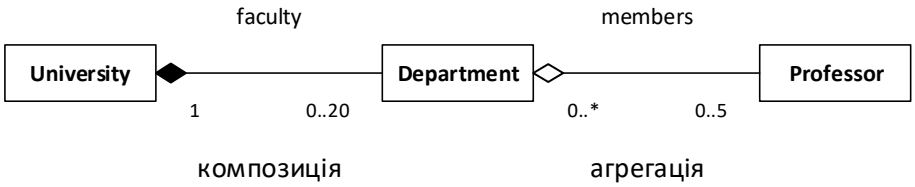
На відміну від вкладених класів, коли область одного класу містила визначення іншого класу, при агрегуванні один клас містить поле-об’єкт, тип якого – іншого класу. Це поле може бути вказівником на об’єкт іншого класу (агрегування у вузькому значенні) або самим об’єктом іншого класу (композиція).

Вкладені об’єкти нового класу зазвичай оголошуються закритими, що робить їх недоступними для програмістів, які використовують такий клас. Тоді розробник класу може змінювати ці об’єкти, не порушуючи при цьому роботи наявного клієнтського коду. Крім того, заміна вкладених об’єктів на етапі виконання програми дозволяє динамічно змінювати її поведінку. На цьому принципі працюють патерни програмування, механізм успадкування не має такої гнучкості.

На основі агрегування та композиції реалізується методика делегування, коли поставлена перед зовнішнім об’єктом задача передоручається іншому – внутрішньому – об’єкту, який спеціалізується на розв’язанні задач такого типу.

Приклад

Позначення композиції та агрегації на UML-діаграмах класів:



Університет пов’язаний з кафедрами чи факультетами відношенням композиції, а кафедра з професорами – відношенням агрегації. Композиція означає, що при знищенні цілого всі частини теж знищуються, а агрегація – що при знищенні цілого частини продовжують існувати, тобто, – професори залишаються жити після ліквідації університету, тоді як факультети чи кафедри знищуються разом з університетом.

При позначенні композиції та агрегації рисують: ромбик – зі сторони цілого, лінію – зі сторони частини. Замальований ромбик означає більш сильний зв’язок (композиція), порожній ромбик – більш слабий зв’язок (агрегація).

Агрегація

Агрегація (агрегування за посиланням) – відношення «ціле-частина» між двома об’єктами, коли один об’єкт (контейнер) містить посилання (вказівник) на інший об’єкт. Обидва об’єкти можуть існувати незалежно один від одного: якщо контейнер буде знищено, то його вміст – залишиться.

```
class Professor;

class Department
{
private:
    Professor* members[5];
};
```

Композиція

Композиція (агрегування за значенням) – більш строгий варіант агрегування, коли внутрішній об’єкт може існувати лише як частина контейнера (контейнер містить не посилання чи вказівник, а сам внутрішній об’єкт). Якщо контейнер буде знищено, то і внутрішній об’єкт буде також знищений.

```
class Department;

class University
{
private:
    Department faculty[20];
};
```

Інший приклад:

```
// Composition vs Aggregation
class Carburetor;

class Automobile
{
private:
    Carburetor* itsCarb;  // агрегація

public:
    Automobile() { itsCarb = new Carburetor(); }  // конструктор
    virtual ~Automobile() { delete itsCarb; }     // деструктор
};
```

Делегування дій іншим методам

Делегування – це коли поставлена перед зовнішнім об’єктом задача передоручається іншому – внутрішньому – об’єкту, який спеціалізується на розв’язанні задач такого типу.

Приклад:

```
class Engine
{
public:
    void Start() {/* ... */}                // метод
};
```

```
class Automobile
{
private:
    Engine* engine;                          // агрегація

public:
    Automobile() { engine = new Engine(); }  // конструктор
    void Start() { engine->Start(); }        // делегування
};
```

В наведеному прикладі метод `Start()` класу `Automobile` делегує свою роботу методу `Start()` класу `Engine`. Це реалізується як виклик метода внутрішнього класу із метода зовнішнього класу.

Ще приклад:

```
class Man
{
public:
    void Display()
    {}
};

class StudentC
{
    Man man;             // композиція

public:
    void Display()
    {
        man.Display();   // делегування
    }
};
```

Семантика зв’язку композиції Студент – Людина: студент містить людину, – тобто, вважаємо, що десь глибоко всередині кожного студента схована людина (схована – бо всі поля традиційно робимо прихованими).

У цьому прикладі метод `Display()` класу `StudentC` делегує свою роботу методу `Display()` класу `Man`. Це реалізується як виклик метода внутрішнього класу із метода зовнішнього класу.

Статичні елементи класу. Розмір класу

За допомогою модифікатора `static` можна описати статичні поля та методи класу. Їх можна розглядати як глобальні змінні чи функції, доступні лише в межах області класу.

Статичні поля

Статичні поля використовуються для зберігання даних, спільних для всіх об'єктів класу, наприклад, кількості об'єктів чи посилання на ресурс, що використовується усіма об'єктами. Статичні поля існують для всіх об'єктів класу в єдиному екземплярі, тобто не дублюються.

Особливості статичних полів:

- Пам'ять для статичного поля виділяється один раз при його ініціалізації незалежно від кількості створених об'єктів (і навіть за їх відсутності) та ініціалізується за допомогою операції доступу до області дії, а не операції вибору елемента класу (визначення статичного поля має бути записане глобально, поза функціями):

```
class A
{
public:
    static int count;     // оголошення в класі
};
...

int A::count;             // визначення в глобальній області
                          // за умовчанням ініціалізується нулем

// int A::count = 10;     // приклад ініціалізації довільним значенням
```

- Статичні поля доступні як через ім'я класу, так і через ім'я об'єкту:

```
A *a, b;
...
cout << A::count << a->count << b.count; // буде виведено одне і те ж
```

- На статичні поля поширюється дія специфікаторів доступу, тому статичні поля, описані як `private`, не можна змінити за допомогою операції доступу до області дії, як описано вище. Це можна зробити лише за допомогою статичних методів.
- Пам'ять, яка виділяється для статичних полів, не враховується при визначенні розміру об'єкта за допомогою операції `sizeof`. Це пояснюється тим, що пам'ять для статичних полів виділяється не в області, виділеній для кожного окремого об'єкту, а в області класу, єдиній для всіх об'єктів.

Статичні методи

Статичні методи призначені для доступу до статичних полів класу. Вони можуть звертатися лише до статичних полів та викликати лише інші статичні методи, тому що їм не передається вказівник `this`. Виклик статичних методів здійснюється так само, як і звертання до статичних полів – або через ім'я класу, або, якщо хоча би один об'єкт класу вже створений, – через ім'я об'єкту:

```
class A
{
    static int count;     // поле count – приховане; це – оголошення

public:
    static void inc_count() { count++; }
    ...
};
                          // визначення в глобальній області
int A::count;             // за умовчанням ініціалізується нулем

void f()
{
    A a;
    // a.count++;            - неможна, бо поле count - приховане
    //                       зміна поля за допомогою статичного методу:
    A::inc_count();       // або a.inc_count();
}
```

Статичні методи не можуть бути константними (`const`) та віртуальними (`virtual`).

Розміри об'єктів класу – звичайні та статичні поля

Особливості розподілу пам'яті:

– для класу і структури пам'ять виділяється абсолютно однаково;

– навіть порожній клас займає в пам'яті певну кількість байт (при тестуванні у Visual C++.NET 2003 – один байт);

– якщо в класі визначено лише одне поле, то пам'яті виділяється рівно стільки, скільки займає значення відповідного типу в пам'яті;

– якщо в класі є кілька полів, то виділений об'єм пам'яті кратний найбільшому розміру поля;

– методи в класі місця не займають.

При виділенні пам'яті для об'єктів статичні поля не враховуються – при створенні об'єкта пам'ять виділяється лише для звичайних полів за вищевказаними правилами. Статичні поля займають окрему область, кожне статичне поле – єдине для всього класу, пам'ять для статичного поля виділяється лише один раз для всіх об'єктів, незалежно від їх кількості (навіть якщо об'єктів взагалі не буде, пам'ять для статичного поля все рівно буде виділена).

Розподіл пам’яті для об’єктів.

Зберігання статичних полів та методів

Розглянемо розподіл пам’яті для об’єктів на прикладі класу Account:

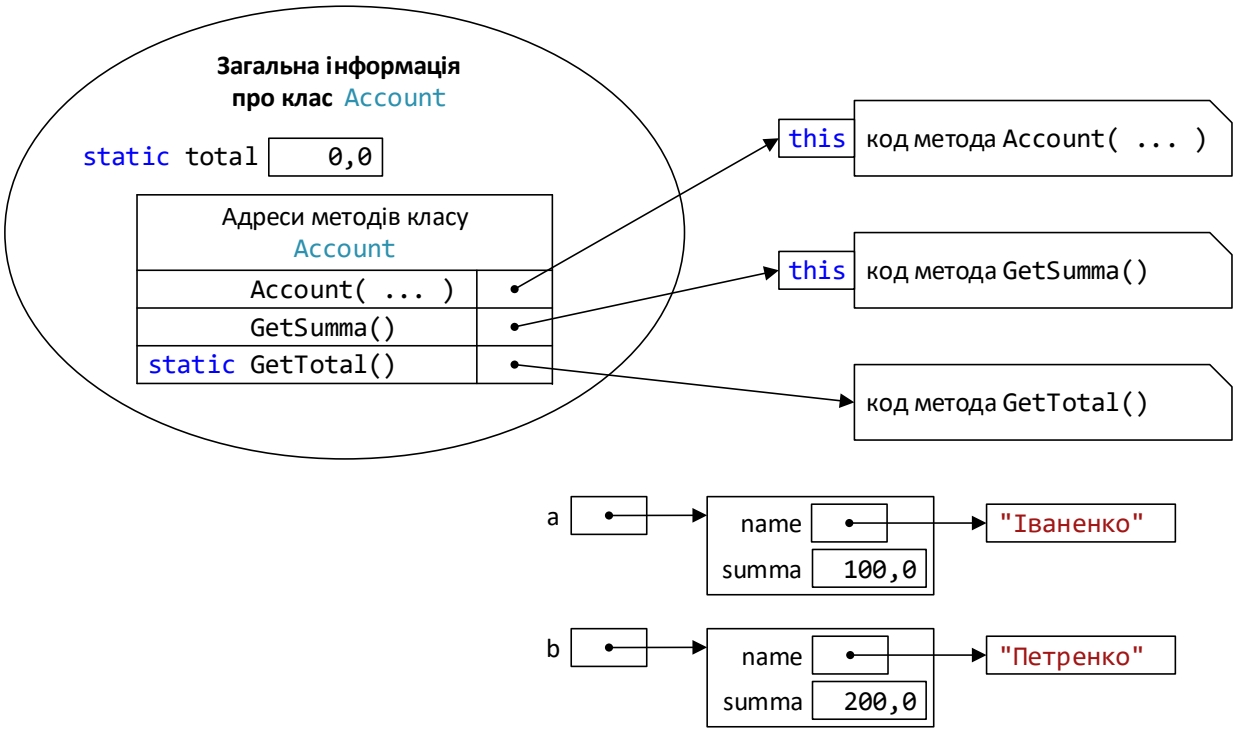
```
#include <string>
#include <iostream>

using namespace std;

class Account
{
private:
    string name;
    double summa;
    static double total;

public:
    Account(string name, double summa)
    {
        this->name = name;
        this->summa = summa;
        total += summa;
    }

    double GetSumma()
    {
        return summa;
    }

    static double GetTotal()
    {
        return total;
    }
};

// в глобальній області слід
// ініціалізувати статичне поле:
double Account::total = 0;

int main()
{
    cout << Account::GetTotal() << endl;

    Account* a = new Account("Іваненко", 100);
    cout << Account::GetTotal() << endl;

    Account* b = new Account("Петренко", 200);
    cout << Account::GetTotal() << endl;

    cout << a->GetSumma() << endl;
}
```

В наведеній програмі створено два динамічних об’єкти класу Account.

Кожний об’єкт містить свої екземпляри звичайних полів. Це означає, що поля name та summa об’єкта *a – займають інші області пам’яті, ніж поля name та summa об’єкта *b.

Крім того, в певній області пам'яті, відведеній для загальної інформації про клас Account, міститься таблиця адрес методів цього класу.

Також у цій загальній для класу області пам'яті міститься комірка для зберігання значення статичного поля total. На відміну від звичайних полів, екземпляри яких – це окремі (різні) комірки пам'яті для окремих (різних) об'єктів, статичне поле – це одна і та ж сама комірка для усіх об'єктів відповідного класу:



Директива #pragma pack – встановлення режиму вирівнювання даних

Поля, вказані при визначенні класу першими, розміщуються в комірках з молодшими адресами.

В пам'яті для об'єкту виділяється неперервна область, розмір якої дорівнює сумі розмірів всіх полів – але є один нюанс, який називається вирівнюванням. Розмір області пам'яті, виділеної для об'єкту, може бути більшим за суму розмірів полів цього об'єкту.

Розглянемо, що таке вирівнювання і як воно працює:

Для об'єкту

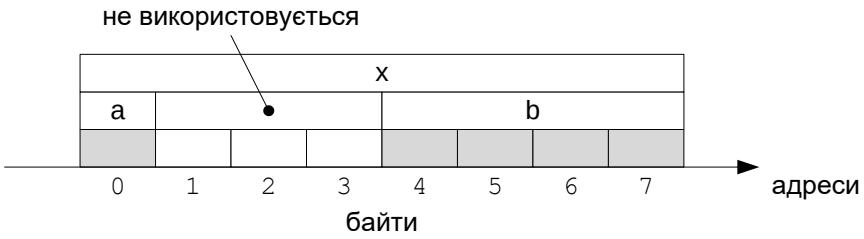
```
class A
{
    char a;
    int b;
} x;
```

його розмір `sizeof(A)` (чи `sizeof(x)`) не завжди буде дорівнювати 5 – все залежить від налаштування компілятора та директив в тексті програми.

Правило: поля в пам'яті вирівнюються по границі, кратній своєму розміру.

Тобто, 1-байтові поля не вирівнюються, 2-байтові – вирівнюються на позиції парних адрес, 4-байтові – на позиції адрес, кратних чотирьом і т.д.

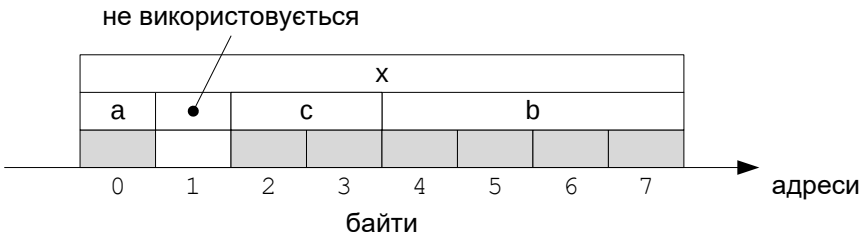
В більшості випадків за умовчанням вирівнювання розміру об'єктів в пам'яті становить 4 байти. Таким чином, `sizeof(A) == 8`:



Розглянемо тепер розподіл пам'яті для наступного об'єкту:

```
class A
{
    char a;
    short c;
    int b;
} x;
```

– він виглядає так:



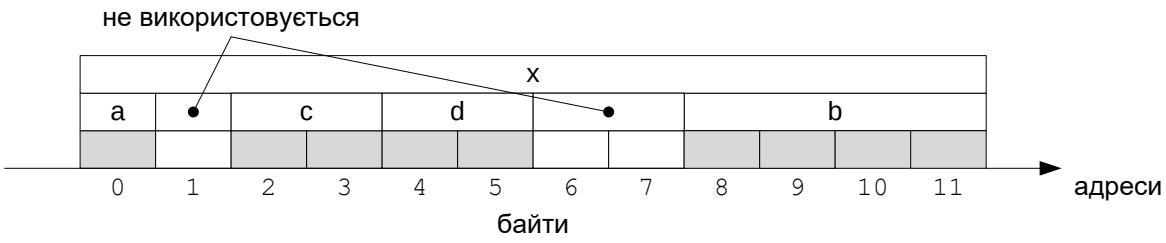
– нове поле не привело до збільшення розміру об'єкту!

Те поле, яке можна розмістити до границі вирівнювання по адресі, кратній 4 байти – не приводить до збільшення розміру структури в пам'яті.

Додамо ще одне поле:

```
class A
{
    char a;
    short c;
    short d;
    int b;
} x;
```

Тепер поля розміщуються в пам'яті наступним чином:



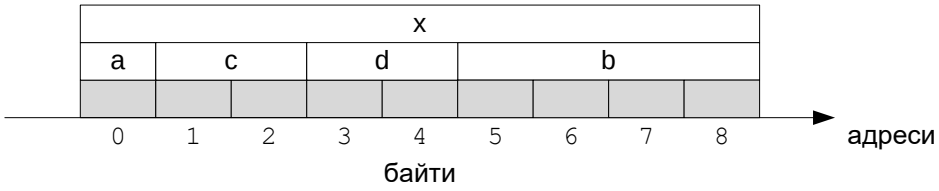
Є можливість керувати вирівнюванням безпосередньо в програмі:

```
#pragma pack(push, 1)

    class A
    {
        char a;
        short c;
        short d;
        int b;
    } x;

#pragma pack(pop)
```

– встановили вирівнювання в 1 байт, визначили об’єкт та вернули попередні налаштування.

Повертати попередні налаштування – категорично рекомендується!

Тепер розподіл пам’яті виглядає так:



Об'єкти – параметри методів.
Реалізація методів, які описують дії над кількома об'єктами

Розглянемо наступний приклад:

Умова завдання

Створити клас Number для роботи з дійсними числами. Число має бути представлене полем x дійсного типу. Тобто, клас Number – це оболонка для стандартного типу double, кожний об'єкт класу Number містить поле (змінну) типу double.

Реалізувати арифметичну операцію додавання двох об'єктів класу Number.

Реалізувати метод перетворення до літерного рядка toString( ).

Аналіз та пояснення різних способів реалізації операції додавання

В першому наближенні визначення класу Number матиме вигляд:

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }
};
```

Кожний об'єкт класу Number буде мати таку будову (припустимо, що створено два об'єкти a та b):

```
Number a, b;
```

– кожний об'єкт містить свою комірку пам'яті, виділену для поля x.

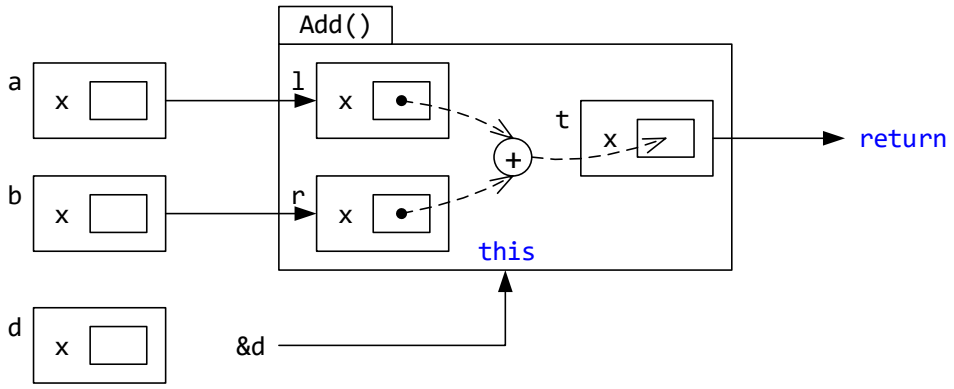
Реалізація звичайним методом з двома параметрами

Реалізуємо операцію додавання двох об'єктів класу Number.

Мабуть, найперше, що хочеться зробити – написати звичайний метод, який буде приймати два параметри класу Number та повертати результат типу Number (добавлені рядки виділено сірим фоном):

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    Number Add(Number l, Number r);
};

Number Number::Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

Проте слід пам’ятати, що кожний звичайний метод, крім явно вказаних параметрів, приймає ще один параметр – вказівник `this`, який отримує значення адреси того об’єкта, що викликає цей метод. Таким чином, метод Add() буде отримувати три параметри:



Використовувати таку операцію можна, наприклад, так;

```
int main()
{
    Number a, b, c, d;
    a.setX(1);
    b.setX(2);

    c = d.Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до звичайного методу використовуємо префікс, який містить ім’я об’єкта.

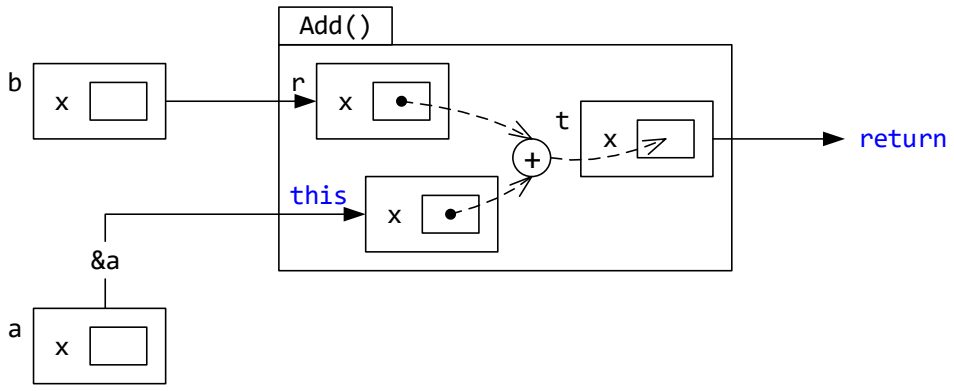
Як бачимо, метод Add() насправді приймає три параметри: два – явно описані в заголовку методу, та параметр `this` – неявний, він містить адресу об'єкта, що викликає цей метод (адресу d в нашому прикладі). Також видно, що параметр `this` ніде в методі не використовується, тому один з цих трьох параметрів – зайвий. Тому цей спосіб, хоча і вірний з точки зору синтаксису, не правильний з точки зору ефективного використання параметрів.

Реалізація звичайним методом з одним параметром

Для того, щоб позбутися зайвих параметрів, перепишемо приклад, використовуючи метод Add(), який приймає лише один параметр (змінені рядки виділено сірим фоном):

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    Number Add(Number r);
};

Number Number::Add(Number r)
{
    Number t;
    t.x = this->x + r.x;
    return t;
}
```

У цій версії діаграма виконання методу Add() буде мати наступний вигляд:



Використовувати цей метод можна так:

```
int main()
{
    Number a, b, c;
```

```
    a.setX(1);
    b.setX(2);

    c = a.Add(b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до звичайного методу використовуємо префікс, який містить ім'я об'єкта.

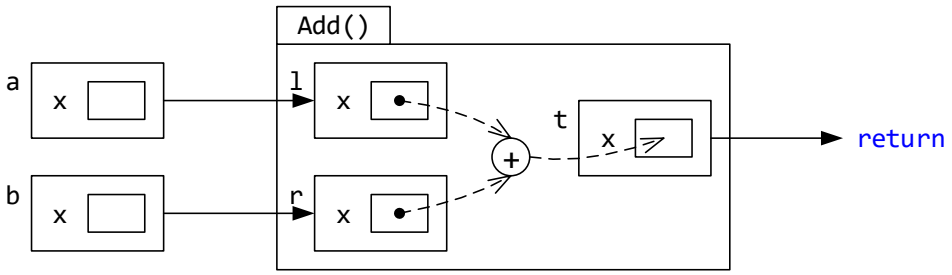
Як бачимо, ми позбулися зайвих параметрів. Проте є ще одна проблема: в такій реалізації операції додавання її лівий та правий операнди – нерівнозначні. Перший (лівий) операнд – це поточний об'єкт, що викликає метод Add(). Другий (правий) операнд – це параметр метода Add(). Завжди поточний об'єкт, який викликає метод, буде більш важливий за параметр цього методу! Але в операції додавання обидва операнди – рівносильні, однакові за важливістю. Тому цей спосіб, хоча і вірний з точки зору синтаксису та правильний з точки зору відсутності параметрів, які не використовуються, – не правильний концептуально: якщо в предметній області (в математиці) для операції додавання лівий та правий операнди – рівносильні, то і в програмній реалізації слід зберегти таку рівносильність.

Реалізація дружньою функцією з двома параметрами

Перепишемо наш приклад, але тепер реалізуємо операцію додавання за допомогою дружньої функції Add(), яка приймає два параметри (змінені рядки виділено сірим фоном):

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    friend Number Add(Number l, Number r);
};

Number Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

В цій версії діаграма виконання дружньої функції Add() буде мати наступний вигляд:



Приклад використання цієї дружньої функції:

```
int main()
{
    Number a, b, c;
    a.setX(1);
    b.setX(2);

    c = Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

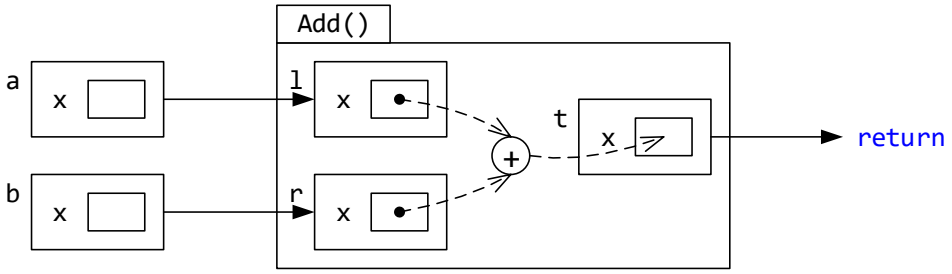
Цей спосіб, вірний як з точки зору синтаксису так і з точки зору відсутності параметрів, що не використовуються. Також цей спосіб правильний концептуально: операція додавання – симетрична, бо її лівий та правий операнди – рівносильні.

Реалізація статичним методом з двома параметрами

Ще один можливий спосіб реалізації операції додавання – використовувати статичний метод (він не приймає параметра `this`). Змінені рядки виділено сірим фоном:

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    static Number Add(Number l, Number r);
};

Number Number::Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

Діаграма виконання статичного методу Add() нічим не буде відрізнятися від діаграми використання дружньої функції:



Приклад використання такого статичного методу:

```
int main()
{
    Number a, b, c;
    a.setX(1);
    b.setX(2);

    c = Number::Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до статичного методу використовуємо префікс, який містить ім’я класу.

Реалізація методу перетворення до літерного рядка toString( )

Повернемося до початкового визначення класу Number, яке мало вигляд:

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }
};
```

– це дозволить не захаращувати код реалізацією операції додавання та сконцентруватися на поясненні методу перетворення до літерного рядка toString().

Добавимо метод toString() (нові рядки виділено сірим фоном):

```
#include <string>        // підключаємо бібліотеку, яка описує літерні рядки
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки
```

```
using namespace std;

class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    string toString() const;
};

string Number::toString() const
{
    stringstream sout;            // створили об'єкт класу "літерний потік"

    sout << "x = " << x << endl;  // направили в літерний потік виведення
                                  // даних про об'єкт

    return sout.str();            // метод str() перетворює літерний потік
                                  // до літерного рядка
}
```

Пояснення методу перетворення до літерного рядка toString( )

Наведений спосіб реалізації методу toString() використовує літерні потоки.

Потік введення / виведення – це послідовність символів, які пересилаються із консолі в пам'ять (введення) чи з пам'яті на консоль (виведення). Потоки введення / виведення ми використовували, коли реалізовували ефективне потокове введення / виведення даних.

Літерний потік – це послідовність символів, які пересилаються із однієї області пам'яті в іншу.

Для використання літерних потоків необхідно підключити бібліотеку <sstream> (подібно до того, як для використання потоків введення / виведення необхідно підключити бібліотеку <iostream>).

Крім того, бажано добавити директиву використання стандартного простору імен, яка дозволить не вказувати префікс std:: перед іменем кожного ресурсу з цього простору, який ми використовуємо в програмі:

```
#include <string>        // підключаємо бібліотеку, яка описує літерні рядки
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки

using namespace std;
```

Метод перетворення до літерного рядка не міняє стану об’єкта (не змінює його полів). Тому стилістично правильно визначити його як константний метод:

```
class Number
{
    ...

    string toString() const;
};
```

Розглянемо реалізацію цього методу:

```
string Number::toString() const
{
    stringstream sout;              // створили об'єкт класу "літерний потік"

    sout << "x = " << x << endl;    // направили в літерний потік виведення
                                    // даних про об'єкт

    return sout.str();              // метод str() перетвоює літерний потік
                                    // до літерного рядка
}
```

Спочатку оголошуємо змінну sout – об’єкт класу stringstream. Клас stringstream описує тип «літерні потоки», а об’єкт sout (String OUTput) названий за аналогією з іменем cout – об’єкт, який відповідає за консольне виведення (Console OUTput).

Потім ми направляємо до літерного потоку ту інформацію, яку бажаємо перетворити до літерного представлення. Запис у літерний потік відбувається аналогічно до запису у потік виведення: всі дані автоматично перетворюються до літерного представлення (числа подаються не у машинних кодах, а як групи символів, що містять зображення відповідних чисел).

Нарешті, при поверненні результату, для об’єкта sout викликаємо метод str() – цей метод на основі літерного потоку формує літерний рядок, який буде містити символи, що зображують всі ті дані, що були направлені у літерний потік.

UML-діаграми

UML діаграми ми вже розглядали в курсі «Алгоритмізація та програмування» - тоді для графічного зображення алгоритму використовували блок-схеми та UML-activity діаграми.

По мірі вивчення ООП будемо вивчати відповідні UML-діаграми та позначення на них.

Класифікація UML-діаграм

Діаграми, що описують **концептуальну модель**:

1) UML-use case diagram – UML-діаграма прецедентів (варіантів використання)

Діаграми, що описують **модель структури**:

2) UML-class diagram – UML-діаграма класів
3) UML-composite strucrure diagram – UML-діаграма композитної структури
4) UML-package diagram – UML-діаграма пакетів
5) UML-object diagram – UML-діаграма об’єктів

Діаграми, що описують **модель поведінки**:

6) UML-sequence diagram – UML-діаграма послідовностей
7) UML-activity diagram – UML-діаграма діяльності
8) UML-communication diagram – UML-діаграма комунікації
9) UML-interaction overview diagram – UML-діаграма огляду взаємодії
10) UML-timing diagram – UML-часова діаграма
11) UML-state machine diagram – UML-діаграма станів (UML-діаграма скінченого автомата)

Діаграми, що описують **модель реалізації**:

12) UML-component diagram – UML-діаграма компонентів
13) UML-deployment diagram – UML-діаграма розгортання

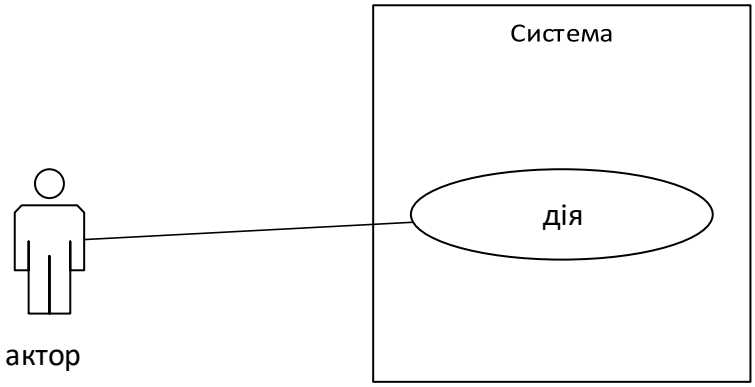
UML-use case diagram

UML-діаграма прецедентів (варіантів використання)

UML-діаграма прецедентів (варіантів використання) показує хто і як саме буде використовувати створену інформаційну систему (програму).

Хто використовує – це «актори».

Як саме використовує – це прецедент (дія).

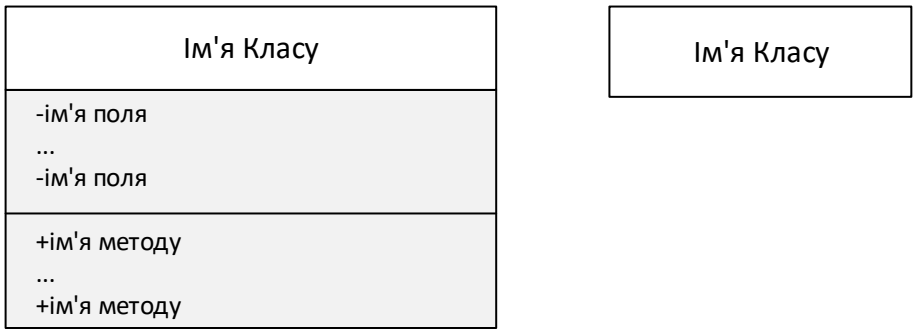


UML-class diagram

UML-діаграма класів

UML-діаграма класів показує будову класів та їх взаємозв’язки.

Позначення класів на UML-діаграмах:



Є дві форми позначень класів на UML-діаграмах – повна та скорочена.

В скороченій формі клас позначається прямокутником, в якому записується ім’я класу.

В повній формі клас позначається прямокутником, розділеним горизонтальними лініями на три частини (секції). У верхній частині записується ім’я класу.

В середній частині записуються поля, а в нижній – методи.

Якщо полів чи методів у класі – немає, то відповідна секція залишається порожньою.

Формат позначень полів:

ім’я_поля

ім’я_поля : тип

Формат позначень методів:

ім’я_метода

ім’я_метода (список_параметрів) : тип_результату

де

список_параметрів

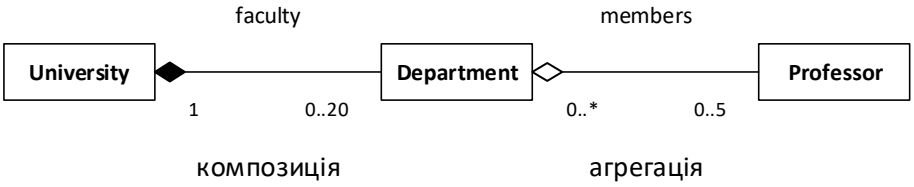
параметр : тип [, …]

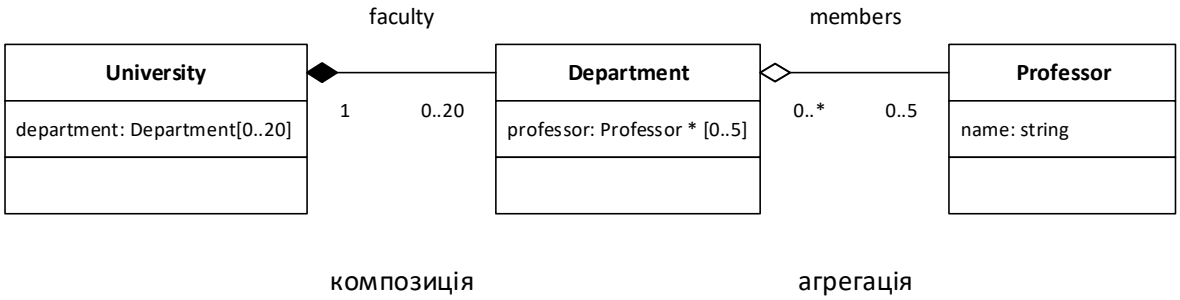
Перед іменем поля чи методу може бути символ, який позначає обмеження доступу:

+ – відповідний елемент класу доступний (традиційно для полів);

- – відповідний елемент класу – прихований (традиційно для методів).

Позначення композиції та агрегації на UML-діаграмах класів:





Університет пов’язаний з кафедрами чи факультетами відношенням композиції, а кафедра з професорами – відношенням агрегації. Композиція означає, що при знищенні цілого всі частини теж знищуються, а агрегація – що при знищенні цілого частини продовжують існувати, тобто, – професори залишаються жити після ліквідації університету, тоді як факультети чи кафедри знищуються разом з університетом.

При позначенні композиції та агрегації рисують: ромбик – зі сторони цілого, лінію – зі сторони частини. Замальований ромбик означає більш сильний зв’язок (композиція), порожній ромбик – більш слабий зв’язок (агрегація).

Структурна схема програми

Структурна схема відображає склад і взаємодію частин програмного забезпечення.

Структурна схема програми зображує взаємозв'язки програми та всіх її програмних одиниць: схему вкладеності та охоплення підпрограм, програми та модулів; а також схему звертання одних програмних одиниць до інших.

Структурну схему програми зазвичай будують методом покрокової деталізації.

Компонентами структурної схеми програмної системи або програмного комплексу можуть служити програми, підсистеми, підпрограми (процедури та функції), бази даних, бібліотеки ресурсів і т.п.

Елементи структурної схеми зображуються прямокутниками, а зв'язки між ними – суцільними лініями зі стрілками, що показують напрям дії «виклик», тобто, – передачі управління.

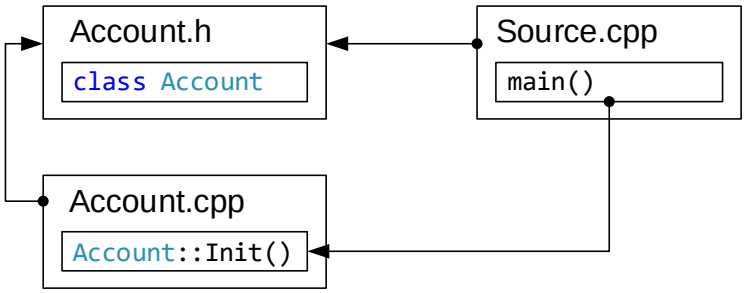
Приклад. Для наступного проекту

```
////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include <iostream>
#include "Account.h"

using namespace std;

int main()
{
    Account a1;
    a1.Init(1, "Вася", 100.0);

    return 0;
}

////////////////////////////////////////////////////////////////////////
// Account.h
//                заголовний файл – визначення класу

#pragma once
#include <string>

using namespace std;

class Account
{
private:
    int No;
    string Name;
    double Summa;

public:
    void Init(int No, string Name, double Summa);
};
```

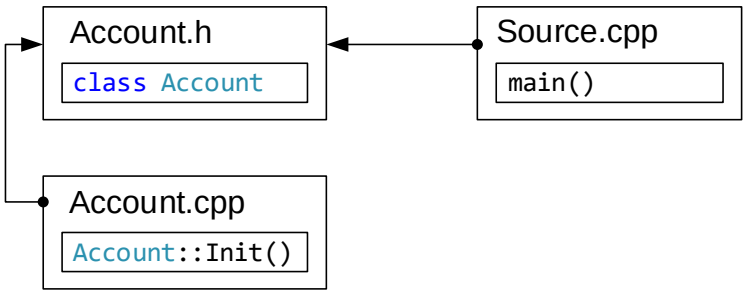
```
////////////////////////////////////////////////////////////////////////
// Account.cpp
//                файл реалізації – реалізація методів класу

#include "Account.h"

void Account::Init(int No, string Name, double Summa)
{
    this->No = No;
    this->Name = Name;
    this->Summa = Summa;
}

////////////////////////////////////////////////////////////////////////
```

– структурна схема буде такою:

Повний варіант – показано всі залежності – зв’язки між модулями та виклики функцій:



Скорочений варіант – зображуються лише зв’язки між модулями:



Лабораторний практикум

Оформлення звіту про виконання лабораторних робіт

Вимоги до оформлення звіту про виконання лабораторних робіт №№ 1.1–1.8

Звіт про виконання лабораторних робіт №№ 1.1–1.8 має містити наступні елементи:

1) заголовок;
2) мету роботи;
3) умову завдання;

   Умова завдання має бути вставлена у звіт як фрагмент зображення (скрін) сторінки посібника.
4) UML-діаграму класів;
5) структурну схему програми;

   Структурна схема програми зображує взаємозв'язки програми та всіх її програмних одиниць: схему вкладеності та охоплення підпрограм, програми та модулів; а також схему звертання одних програмних одиниць до інших.
6) текст програми;

   Текст програми має бути правильно відформатований: відступами і порожніми рядками слід відображати логічну структуру програми; програма має містити необхідні коментарі – про призначення підпрограм, змінних та параметрів – якщо їх імена не значущі, та про призначення окремих змістовних фрагментів програми. Текст програми слід подавати моноширинним шрифтом (`Courier New` *розміром 10 пт. або* `Consolas` *розміром 9,5 пт.*) *з одинарним міжрядковим інтервалом;*
7) посилання на git-репозиторій з проектом (див. інструкції з Лабораторної роботи № 2.2 з предмету «Алгоритмізація та програмування»);
8) хоча б для одної функції, яка повертає результат (як результат функції чи як параметр-посилання) – результати unit-тесту: текст програми unit-тесту та скрін результатів її виконання (див. інструкції з Лабораторної роботи № 5.6 з предмету «Алгоритмізація та програмування»);
9) висновки.

Зразок оформлення звіту про виконання лабораторних робіт №№ 1.1–1.8

ЗВІТ

про виконання лабораторної роботи № < *номер* >

« *назва теми лабораторної роботи* »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента(ки) групи КН-17

< *Прізвище Ім’я По_батькові* >

Мета роботи:

…

Умова завдання:

…

UML-діаграма класів:

…

Структурна схема програми:

…

Текст програми:

…

Посилання на git-репозиторій з проектом:

…

Результати unit-тесту:

…

Висновки:

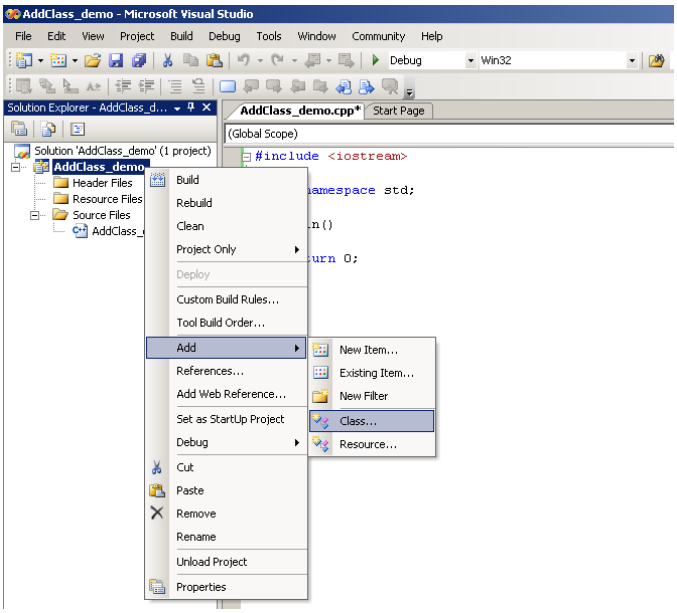
…

Створення та підключення C++ класу в середовищі Visual Studio

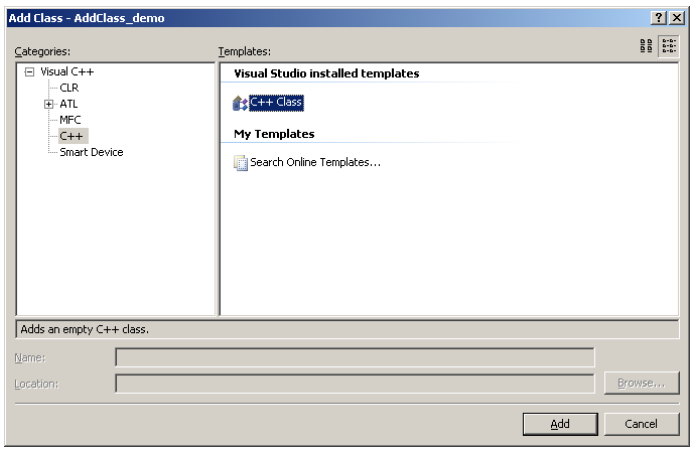
Створення класу

Після створення проекту та головного файлу (який містить функцію `main()`), добавимо до проекту клас.

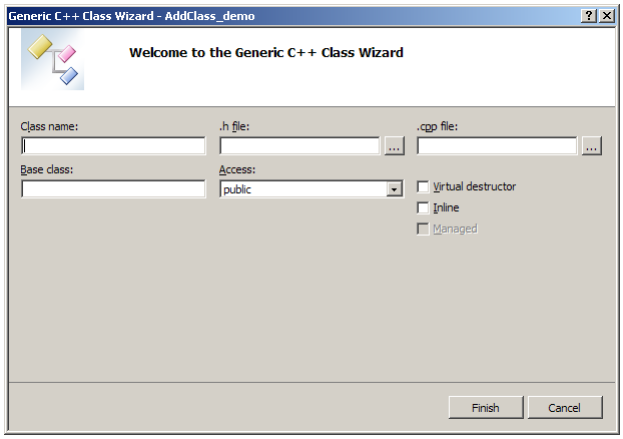
Для цього у вікні Solution Explorer виведемо контекстне меню проекту (клацнувши правою кнопкою мишки по імені проекту). В цьому контекстному меню вибираємо команду Add → Class… :

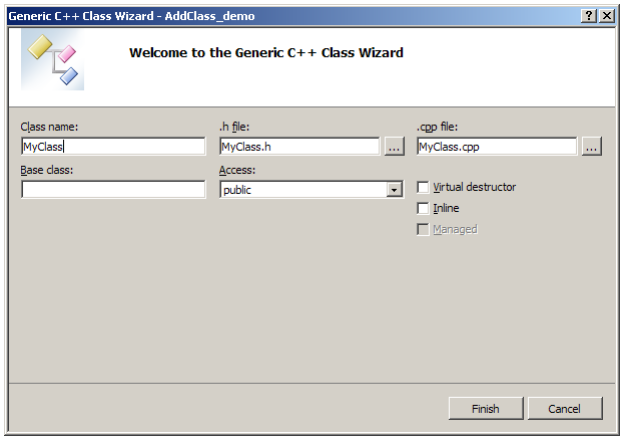


У вікні Add Class вибираємо категорію Categories – C++ та шаблон Templates – C++ Class та натискаємо кнопку Add:

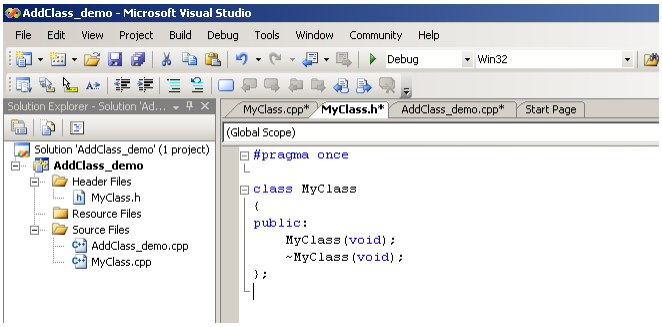


У вікні Generic C++ Class Wizard в полі Class Name слід ввести ім’я класу (наприклад, MyClass) та натиснути кнопку Finish:





Система створить файл заголовку та файл реалізації для класу (у цьому випадку – MyClass):



Підключення класу

В головному файлі проекту слід сказати директиву `#include`, яка буде підключати файл заголовку для класу:

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

using namespace std;

int main()
{
    return 0;
}
```

Лабораторна робота № 1.1. Поля та методи – дії над одним (поточним) об’єктом

Мета роботи

Освоїти використання класів. Навчитися створювати багатомодульні C++-проекти.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття класів та об’єктів.
2) Загальний синтаксис опису класів.
3) Визначення та оголошення класу.
4) Елементи класу: поля та методи.
5) Інкапсуляція[1]: об’єднання опису даних та опису дій над даними в єдине ціле.
6) Роль вказівника this в реалізації інкапсуляції[1].
7) Інкапсуляція[2]: обмеження доступу до даних.
8) Директиви доступу (видимості) елементів класу.
9) Звертання до полів та методів класу.
10) Передавання об’єктів у функції та повернення об’єктів в якості результату.
11) Позначення класів та об’єктів на UML-діаграмах класів.
12) Позначення елементів класів на UML-діаграмах класів.
13) Позначення доступних та приватних елементів класів на UML-діаграмах класів.

Зразки виконання завдання

Приклад 1.

Подається лише умова завдання та текст програми.

Умова завдання

Створити клас Account, який описує банківський рахунок з приватними полями:

no – номер рахунку (тип даних `int`);

name – прізвище (або ім’я) власника рахунку (тип даних `string`);

summa – величина залишку коштів на рахунку (тип даних `double`).

Клас має містити методи доступу (константні методи зчитування та методи запису) для кожного поля та метод Init(…) для ініціалізації всіх полів.

Текст програми

```
////////////////////////////////////////////////////////////////////////
// Account.h
//                                          заголовний файл – визначення класу
#pragma once
#include <string>

using namespace std;

class Account
{
private:
    int no;
    string name;
    double summa;

public:
    int GetNo() const { return no; }
    string GetName() const { return name; }
    double GetSumma() const { return summa; }

    void SetNo(int value) { no = abs(value); }
    void SetName(string value) { name = value; }
    void SetSumma(double value);

    void Init(int no, string name, double summa);
};

////////////////////////////////////////////////////////////////////////
// Account.cpp
//                                  файл реалізації – реалізація методів класу
#include "Account.h"

void Account::SetSumma(double value)
{
    summa = (value >= 0) ? value : 0;
}

void Account::Init(int no, string name, double summa)
{
    SetNo(no);
    SetName(name);
    SetSumma(summa);
}

////////////////////////////////////////////////////////////////////////
// Source.cpp
//                                      головний файл проекту – функція main
#include <iostream>
#include "Account.h"

using namespace std;

int main()
{
    Account a1;
    a1.Init(1, "Вася", 100.0);

    return 0;
}
```

Приклад 2.

Подається лише умова завдання та текст програми.

Умова завдання

Класом-парою називається клас з двома приватними полями, які мають імена first та second. Потрібно реалізувати такий клас. Обов'язково мають бути присутніми:

- методи доступу (константні методи зчитування та методи запису) для кожного поля;
- метод ініціалізації Init( ); метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури Read( );
- метод виведення на екран Display( ).

Реалізувати зовнішню функцію з ім'ям make*Клас*( ), де *Клас* – ім'я класу, об'єкт якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об'єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Реалізувати клас Number2, який описує точку (вектор) на координатній пів-площині. Поле first – ціле число, x-координата точки; поле second – додатне ціле число, y-координата точки. Реалізувати метод multiply( ) – множення вектора на довільне ціле число типу int. Метод має правильно працювати при будь-яких допустимих значеннях first і second.

Текст програми

```
//////////////////////////////////////////////////////////////////////////
// Number2.h
//                                        заголовний файл – визначення класу

#pragma once

class Number2
{
private:
    int first;
    unsigned int second;

public:
    int GetFirst() const { return first; }
    unsigned int GetSecond() const { return second; }
    void SetFirst(int value);
    void SetSecond(unsigned int value);

    bool Init(int x, int y);
    void Display() const;
    void Read();

    void multiply(int N);
};
```

```
//////////////////////////////////////////////////////////////////////////
// Number2.cpp
//                                  файл реалізації – реалізація методів класу

#include "Number2.h"
#include <iostream>

using namespace std;

void Number2::SetFirst(int value)
{
    first = value;
}


void Number2::SetSecond(unsigned int value)
{
    second = value;
}


bool Number2::Init(int x, int y)
{
    first = x;
    if (y >= 0)
    {
        second = y;
        return true;
    }
    else
    {
        second = 0;
        return false;
    }
}

void Number2::Display() const
{
    cout << "first = " << first << "    second = " << second << endl;
}

void Number2::Read()
{
    int x, y;
    cout << "first = ? ";
    cin >> x;
    do {
        cout << "second = ?";
        cin >> y;
    } while (!Init(x, y));
}

void Number2::multiply(int N)
{
    first *= N;
    second *= N;
}


//////////////////////////////////////////////////////////////////////////
```

```
// Source.cpp
//                                        головний файл проекту – функція main

#include <iostream>
#include "Number2.h"

using namespace std;

Number2 makeNumber2(int x, int y)
{
    Number2 nn;
    if (!nn.Init(x, y))
        cout << "wrong argument to Init (second)" << endl;
    return nn;
}

int main()
{
    Number2 n;
    n.Init(10, 10);
    n.Display();
    n.multiply(5);
    n.Display();

    Number2 k;
    k.Read();
    k.Display();
    k.multiply(2);
    k.Display();

    Number2 i;
    int a, b;
    cout << "first = ? ";
    cin >> a;
    cout << "second = ?";
    cin >> b;
    i = makeNumber2(a, b);
    i.Display();

    return 0;
}
```

Приклад 3.

Подається лише умова завдання та текст програми.

Умова завдання

Класом-парою називається клас з двома полями, які мають імена first та second. Потрібно реалізувати такий клас. Обов'язково мають бути присутніми:

- метод ініціалізації Init( ); метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури Read( );
- метод виведення на екран Display( ).

Реалізувати зовнішню функцію з ім'ям make*Клас*( ), де *Клас* – ім'я класу, об'єкт

якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об'єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Реалізувати клас Complex. Поле first – дробове число x, дійсна частина комплексного числа z, $x = \mathrm{Re}\, z$; поле second – дробове число y, мнима частина комплексного числа z, $y = \mathrm{Im}\, z$, ($|x| \le 100$, $|y| \le 100$). Реалізувати метод abs( ) – обчислення модуля комплексного числа, $\mathrm{abs}(z) = \sqrt{(\mathrm{Re}\, z)^2 + (\mathrm{Im}\, z)^2}$.

Текст програми

```
////////////////////////////////////////////////////////////////////////////
// Complex.h
//                заголовний файл – визначення класу

#pragma once

class Complex
{
    double first, second;

public:
    bool Init(double, double);
    void Read();
    void Display() const;

    double Abs() const;
};


////////////////////////////////////////////////////////////////////////////
// Complex.cpp
//                файл реалізації – реалізація методів класу

#include "Complex.h"
#include <iostream>
#include <cmath>

using namespace std;

bool Complex::Init(double x, double y)
{
    if (fabs(x) <= 100 && fabs(y) <= 100)
    {
        first = x;
        second = y;
        return true;
    }
    else
    {
        return false;
    }
}
```

```
void Complex::Read()
{
    double x, y;
    do
    {
        cout << "Input complex value:" << endl;
        cout << "  Re = "; cin >> x;
        cout << "  Im = "; cin >> y;

    } while (!Init(x, y));
}

void Complex::Display() const
{
    cout << endl;
    cout << "  Re = " << first << endl;
    cout << "  Im = " << second << endl;
}

double Complex::Abs() const
{
    return sqrt(first * first + second * second);
}

////////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include "Complex.h"
#include <iostream>

using namespace std;

Complex makeComplex(double x, double y)
{
    Complex z;
    if (!z.Init(x, y))
        cout << "Wrong arguments to Init!" << endl;

    return z;
}

int main()
{
    Complex z;
    z.Read();
    z.Display();
    cout << "Abs(z) = " << z.Abs() << endl << endl;

    double x, y;
    cout << "Input complex value:" << endl << endl;
    cout << "  Re = "; cin >> x;
    cout << "  Im = "; cin >> y;
    z = makeComplex(x, y);
    z.Display();
    cout << "Abs(z) = " << z.Abs() << endl;

    cin.get();
    return 0;
}
```

Варіанти завдань

Класом-парою називається клас з двома приватними полями, які мають імена first та second. Потрібно реалізувати такий клас. Обов'язково мають бути присутніми:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( ); метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури Read( );
- метод виведення на екран Display( ).

Реалізувати зовнішню функцію з ім'ям make*Клас*( ), де *Клас* – ім'я класу, об'єкт якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об'єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Реалізувати клас IntPower. Поле first – дійсне ненульове число; поле second – ціле число, показник степені. Реалізувати метод power( ) – піднесення числа first до степені second. Метод має правильно працювати при будь-яких допустимих значеннях first та second.

Варіант 2.

Реалізувати клас FloatPower. Поле first – дійсне ненульове число; поле second – дійсне число, показник степеня. Реалізувати метод power( ) – піднесення числа first до степеня second. Метод має правильно працювати при будь-яких допустимих значеннях first і second.

Варіант 3.

Реалізувати клас Fraction. Поле first – ціле додатне число, чисельник; поле second – ціле додатне число, знаменник. Реалізувати метод ipart( ) – виділення цілої частини дробу first / second. Метод має перевіряти нерівність знаменника нулю.

Варіант 4.

Реалізувати клас Money. Поле first – ціле додатне число, номінал купюри; номінал

може приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Поле second – ціле додатне число, кількість купюр відповідної вартості. Реалізувати метод summa( ) – обчислення грошової суми.

Варіант 5.

Реалізувати клас Goods. Поле first – дробове додатне число, ціна товару; поле second – ціле додатне число, кількість одиниць товару. Реалізувати метод cost( ) – обчислення вартості товару.

Варіант 6.

Реалізувати клас Product. Поле first – ціле додатне число, калорійність 100 г. продукту; поле second – дійсне додатне число, маса продукту в кілограмах. Реалізувати метод power( ) – обчислення загальної калорійності продукту.

Варіант 7.

Реалізувати клас FloatRange. Поле first – дійсне число, ліва границя діапазону; поле second – дійсне число, права границя діапазону: first < second. Реалізувати метод rangeCheck( ) – перевірку заданого числа на входження до діапазону.

Варіант 8.

Реалізувати клас IntRange. Поле first – ціле число, ліва границя діапазону, включається в діапазон; поле second – ціле число, права границя діапазону, не включається в діапазон: first < second. Пара чисел представляє напів-відкритий інтервал [first, second). Реалізувати метод rangeCheck( ) – перевірку заданого цілого числа на входження до діапазону.

Варіант 9.

Реалізувати клас Time. Поле first – ціле додатне число, години; поле second – ціле додатне число, хвилини. Реалізувати метод minutes( ) – приведення часу з формату (години, хвилини) в хвилини.

Варіант 10.

Реалізувати клас Line. Лінійне рівняння $y = Ax + B$. Поле first – дійсне число, коефіцієнт A ($A \neq 0$); поле second – дійсне число, коефіцієнт B. Реалізувати метод function( ) – обчислення для заданого x значення функції y.

Варіант 11.

Реалізувати клас Line. Лінійне рівняння $y = Ax + B$. Поле first – дійсне число, коефіцієнт A; поле second – дійсне число, коефіцієнт B ($A \neq 0$). Реалізувати метод root( ) – обчислення кореня лінійного рівняння. Метод має перевіряти нерівність коефіцієнта A нулю.

Варіант 12.

Реалізувати клас Point. Поле first – дійсне число, координата x точки на площині; поле second – дійсне число, координата y точки на площині, ($|x| \leq 100$, $|y| \leq 100$). Реалізувати метод distance( ) – обчислення відстані від точки до початку координат.

Варіант 13.

Реалізувати клас Triangle. Поле first – дійсне додатне число, катет a прямокутного трикутника; поле second – дійсне додатне число, катет b прямокутного трикутника. Реалізувати метод hypotenuse( ) – обчислення гіпотенузи.

Варіант 14.

Реалізувати клас Pay. Поле first – дійсне додатне число, оклад; поле second – ціле додатне число, кількість відпрацьованих днів в місяці. Реалізувати метод summa( ) – обчислення нарахованої суми за відповідну кількість днів для заданого місяця за формулою:

оклад / кількість_робочих_днів_у_місяці * кількість_відпрацьованих_днів

Варіант 15.

Реалізувати клас Bill. Поле first – ціле додатне число, тривалість телефонної розмови в хвилинах; поле second – дійсне додатне число, вартість однієї хвилини розмови в гривнях. Реалізувати метод cost( ) – обчислення загальної вартості розмови.

Варіант 16.

Реалізувати клас Number. Поле first – дійсне число, ціла частина числа; поле second – додатне дійсне число, дробова частина числа. Реалізувати метод multiply( ) – множення на довільне дійсне число типу double. Метод має правильно працювати при будь-яких допустимих значеннях first та second.

Варіант 17.

Реалізувати клас Cursor. Поле first – ціле додатне число, горизонтальна координата курсору; поле second – ціле додатне число, вертикальна координата курсору. Реалізувати метод changeX( ) – зміну горизонтальної координати курсору, та метод changeY( ) – зміну

вертикальної координати курсору. Методи мають перевіряти вихід за межі екрану.

Варіант 18.

Реалізувати клас Number. Поле first – ціле число, ціла частина числа; поле second – додатне ціле число, дробова частина числа. Реалізувати метод multiply( ) – множення на довільне ціле число типу int. Метод має правильно працювати при будь-яких допустимих значеннях first і second.

Варіант 19.

Реалізувати клас Combination. Число сполучень по k об'єктів з n об'єктів ($k < n$) обчислюється за формулою

$$C(n, k) = n! / ((n-k)! \times k!)$$

Поле first – ціле додатне число, k; поле second – додатне ціле число, n. Реалізувати метод combination( ) – обчислення $C(n, k)$.

Варіант 20.

Реалізувати клас Progression. Елемент a_j геометричної прогресії обчислюється за формулою:

$$a_j = a_0 r^j, j = 0, 1, 2, \ldots$$

Поле first – дійсне число, перший елемент прогресії a_0; поле second – постійне відношення r. Визначити метод elementJ( ) для обчислення заданого елементу прогресії.

Варіант 21.

Реалізувати клас IntPower. Поле first – дійсне ненульове число; поле second – ціле число, показник степені. Реалізувати метод power( ) – піднесення числа first до степені second. Метод має правильно працювати при будь-яких допустимих значеннях first та second.

Варіант 22.

Реалізувати клас FloatPower. Поле first – дійсне ненульове число; поле second – дійсне число, показник степеня. Реалізувати метод power( ) – піднесення числа first до степеня second. Метод має правильно працювати при будь-яких допустимих значеннях first і second.

Варіант 23.

Реалізувати клас Fraction. Поле first – ціле додатне число, чисельник; поле second –

ціле додатне число, знаменник. Реалізувати метод ipart( ) – виділення цілої частини дробу first / second. Метод має перевіряти нерівність знаменника нулю.

Варіант 24.

Реалізувати клас Money. Поле first – ціле додатне число, номінал купюри; номінал може приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Поле second – ціле додатне число, кількість купюр відповідної вартості. Реалізувати метод summa( ) – обчислення грошової суми.

Варіант 25.

Реалізувати клас Goods. Поле first – дробове додатне число, ціна товару; поле second – ціле додатне число, кількість одиниць товару. Реалізувати метод cost( ) – обчислення вартості товару.

Варіант 26.

Реалізувати клас Product. Поле first – ціле додатне число, калорійність 100 г. продукту; поле second – дійсне додатне число, маса продукту в кілограмах. Реалізувати метод power( ) – обчислення загальної калорійності продукту.

Варіант 27.

Реалізувати клас FloatRange. Поле first – дійсне число, ліва границя діапазону; поле second – дійсне число, права границя діапазону: first < second. Реалізувати метод rangeCheck( ) – перевірку заданого числа на входження до діапазону.

Варіант 28.

Реалізувати клас IntRange. Поле first – ціле число, ліва границя діапазону, включається в діапазон; поле second – ціле число, права границя діапазону, не включається в діапазон: first < second. Пара чисел представляє напів-відкритий інтервал [first, second). Реалізувати метод rangeCheck( ) – перевірку заданого цілого числа на входження до діапазону.

Варіант 29.

Реалізувати клас Time. Поле first – ціле додатне число, години; поле second – ціле додатне число, хвилини. Реалізувати метод minutes( ) – приведення часу з формату (години, хвилини) в хвилини.

Варіант 30.

Реалізувати клас Line. Лінійне рівняння $y = Ax + B$. Поле first – дійсне число, коефіцієнт A ($A \neq 0$); поле second – дійсне число, коефіцієнт B. Реалізувати метод function( ) – обчислення для заданого x значення функції y.

Варіант 31.

Реалізувати клас Line. Лінійне рівняння $y = Ax + B$. Поле first – дійсне число, коефіцієнт A; поле second – дійсне число, коефіцієнт B ($A \neq 0$). Реалізувати метод root( ) – обчислення кореня лінійного рівняння. Метод має перевіряти нерівність коефіцієнта A нулю.

Варіант 32.

Реалізувати клас Point. Поле first – дійсне число, координата x точки на площині; поле second – дійсне число, координата y точки на площині, ($|x| \leq 100$, $|y| \leq 100$). Реалізувати метод distance( ) – обчислення відстані від точки до початку координат.

Варіант 33.

Реалізувати клас Triangle. Поле first – дійсне додатне число, катет a прямокутного трикутника; поле second – дійсне додатне число, катет b прямокутного трикутника. Реалізувати метод hypotenuse( ) – обчислення гіпотенузи.

Варіант 34.

Реалізувати клас Pay. Поле first – дійсне додатне число, оклад; поле second – ціле додатне число, кількість відпрацьованих днів в місяці. Реалізувати метод summa( ) – обчислення нарахованої суми за відповідну кількість днів для заданого місяця за формулою:

оклад / кількість_робочих_днів_у_місяці * кількість_відпрацьованих_днів

Варіант 35.

Реалізувати клас Bill. Поле first – ціле додатне число, тривалість телефонної розмови в хвилинах; поле second – дійсне додатне число, вартість однієї хвилини розмови в гривнях. Реалізувати метод cost( ) – обчислення загальної вартості розмови.

Варіант 36.

Реалізувати клас Number. Поле first – дійсне число, ціла частина числа; поле second – додатне дійсне число, дробова частина числа. Реалізувати метод multiply( ) – множення на довільне дійсне число типу double. Метод має правильно працювати при будь-яких допустимих значеннях first та second.

Варіант 37.

Реалізувати клас Cursor. Поле first – ціле додатне число, горизонтальна координата курсору; поле second – ціле додатне число, вертикальна координата курсору. Реалізувати метод changeX( ) – зміну горизонтальної координати курсору, та метод changeY( ) – зміну вертикальної координати курсору. Методи мають перевіряти вихід за межі екрану.

Варіант 38.

Реалізувати клас Number. Поле first – ціле число, ціла частина числа; поле second – додатне ціле число, дробова частина числа. Реалізувати метод multiply( ) – множення на довільне ціле число типу int. Метод має правильно працювати при будь-яких допустимих значеннях first і second.

Варіант 39.

Реалізувати клас Combination. Число сполучень по k об'єктів з n об'єктів ($k < n$) обчислюється за формулою

$$C(n, k) = n! / ((n-k)! \times k!)$$

Поле first – ціле додатне число, k; поле second – додатне ціле число, n. Реалізувати метод combination( ) – обчислення $C(n, k)$.

Варіант 40.

Реалізувати клас Progression. Елемент a_j геометричної прогресії обчислюється за формулою:

$$a_j = a_0 r^j, j = 0, 1, 2, \ldots$$

Поле first – дійсне число, перший елемент прогресії a_0; поле second – постійне відношення r. Визначити метод elementJ( ) для обчислення заданого елементу прогресії.

Лабораторна робота № 1.2. Оголошення та будова класу

Мета роботи

Освоїти використання класів та об'єктів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об'єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об'єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об'єктів у функції та повернення об'єктів в якості результату.*
11) *Позначення класів та об'єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*

Зразок виконання завдання

Приклад.

Подається лише умова завдання та текст програми.

Умова завдання

Потрібно реалізувати клас з приватними полями, вказаний в завданні свого варіанту.

Обов'язково потрібно реалізувати:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( ); метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури Read( );
- метод виведення на екран Display( ).

Реалізувати зовнішню функцію з ім'ям make*Клас*( ), де *Клас* – ім'я класу, об'єкт

якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об’єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Розробити клас ГазовийЛічильник. У закритій частині визначити поля:

- прізвище,
- номер рахунку,
- кількість кубометрів використаного газу;
- ціна одного кубометра.

Визначити методи:

- встановлення та зчитування значень полів даних,
- знаходження вартості спожитого газу.

Текст програми

```
//////////////////////////////////////////////////////////////////////////
// Gas.h
//                заголовний файл – визначення класу

#pragma once
#include <string>

using namespace std;

class Gas
{
private:
    string name;
    int no;
    double count;
    double price;

public:
    string getName() const { return name; }
    int getNo() const { return no; }
    double getCount() const { return count; }
    double getPrice() const { return price; }

    void setName(string);
    bool setNo(int);
    bool setCount(double);
    bool setPrice(double);

    double getVartist() const { return count * price; }

    bool Init(string name, int no, double count, double price);
    void Read();
    void Display() const;
};
```

```
////////////////////////////////////////////////////////////////////////
// Gas.cpp
//                файл реалізації – реалізація методів класу

#include "Gas.h"
#include <iostream>

using namespace std;

void Gas::setName(string value)
{
    name = value;
}

bool Gas::setNo(int value)
{
    if (value > 0)
    {
        no = value;
        return true;
    }
    else
    {
        no = 0;
        return false;
    }
}

bool Gas::setCount(double value)
{
    if (value > 0)
    {
        count = value;
        return true;
    }
    else
    {
        count = -value;
        return false;
    }
}

bool Gas::setPrice(double value)
{
    if (value > 0)
    {
        price = value;
        return true;
    }
    else
    {
        price = -value;
        return false;
    }
}

bool Gas::Init(string name, int no, double count, double price)
{
    setName(name);

    return setNo(no) && setCount(count) && setPrice(price);
}
```

```cpp
void Gas::Read()
{
    string name;
    int no;
    double count;
    double price;

    cout << " name  = ? "; cin >> name;

    do
    {
        cout << " no    = ? "; cin >> no;
        cout << " count = ? "; cin >> count;
        cout << " price = ? "; cin >> price;
    } while (!Init(name, no, count, price));
}

void Gas::Display() const
{
    cout << " name  = " << name << endl;
    cout << " no    = " << no << endl;
    cout << " count = " << count << endl;
    cout << " price = " << price << endl;
}

//////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include <iostream>
#include "Gas.h"

using namespace std;

int main()
{
    Gas g;
    g.Read();
    g.Display();

    cout << g.getVartist() << endl;

    return 0;
}
```

Варіанти завдань

Потрібно реалізувати клас з приватними полями, вказаний в завданні свого варіанту.

Обов'язково потрібно реалізувати:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( ); метод має контролювати значення аргументів на коректність;
- метод введення з клавіатури Read( );
- метод виведення на екран Display( ).

Реалізувати зовнішню функцію з ім'ям make*Клас*( ), де *Клас* – ім'я класу, об'єкт якого вона створює. Функція має отримувати як аргументи значення для полів класу і повертати об'єкт необхідного класу. При передачі помилкових параметрів слід виводити повідомлення і закінчувати роботу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Варіант 1.

Створити клас Прямокутник. У закритій частині визначити поля:

- висоту і
- ширину.

Методи:

- обчислення площі,
- обчислення периметру,
- встановлення висоти,
- встановлення ширини,
- отримання висоти,
- отримання ширини;
- виведення значень елементів класу на екран.

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 2.*

Створити клас Двох-зв'язний список.

Поля:

- вказівник на початок списку,
- вказівник на кінець списку,

Методи:

- додають елемент до списку,
- видаляють елемент зі списку,
- * сортують список;
- відображають елементи списку від початку,
- відображають елементи списку від кінця,

Знайти заданий елемент у списку.

Варіант 3.*

Створити клас Однозв'язний список.

Поле:

- вказівник на початок списку,

Методи:

- додають елемент до списку,
- видаляють елемент зі списку,
- * сортують список;
- відображають елементи списку від початку.

Видалити заданий елемент списку.

Варіант 4.

Створити клас – Коло. У закритій частині визначити поля:

- x,
- y – координати центру та
- R – радіус.

Методи:

- обчислюють площу,
- довжину кола,
- встановлюють поля і
- повертають їх значення.

Методи встановлення значень полів класу мають перевіряти коректність параметрів, що задаються. Розробити функцію виведення полів даних класу.

Варіант 5.

Створити клас Квадрат з полями у закритій частині:

- x_1,

- y_1,
- x_2,
- y_2 – координати головної діагоналі.

Методи обчислюють:

- довжину сторони квадрата,
- площу,
- периметр,
- встановлюють значення полів і
- повертають їх значення.

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 6.

Створити клас Стек.

Поле:

- вказівник на вершину стеку,

Розробити методи для:

- включення елемента у стек та
- виключення елемента зі стеку.

Визначити функцію, яка визначає кількість елементів у стеку.

Варіант 7.*

Визначити клас Дроби – раціональні числа, які є відношенням двох цілих чисел.

Поля:

- чисельник,
- знаменник.

Напишіть методи для:

- введення та
- виведення дробу (чисельника та знаменника),
- * скорочення та
- обчислення значення дробу.

Варіант 8.

Цифровий лічильник – це змінна цілого типу з обмеженим діапазоном, який скидається у початкове значення, коли її значення досягає визначеного максимуму.

Приклади: цифровий годинник, лічильник електроенергії.

Опишіть клас Лічильник.

Забезпечте можливість:

- встановлення максимального і
- мінімального значень,
- збільшення значень лічильника на 1,
- повернення поточного значення.

Варіант 9.

Створити клас Вектор цілих елементів, який має у закритій частині:

- вказівник на int – зв'язок з динамічним масивом,
- кількість елементів і
- змінну стану.

У змінній стану встановлювати код помилки, коли не вистачає пам'яті, або відбувається вихід за межі масиву.

Визначити методи:

- виділення пам'яті для елементів,
- присвоєння елементові масиву деякого значення,
- отримання значення деякого елементу масиву;
- виведення масиву;
- визначення евклідової норми вектора.

Перевірити роботу цього класу.

Варіант 10.

Створити клас Матриця цілих елементів, який у закритій частині містить поля:

- вказівник на вказівник на int – зв'язок з динамічним двох-вимірним масивом,
- кількість рядків,
- кількість стовпців та
- змінну стану.

У змінній стану встановлювати код помилки, коли не вистачає пам'яті, або відбувається вихід за межі масиву, невідповідності розмірностей масивів.

Визначити методи для:

- повернення значення елемента, який має індекси (i, j);
- виведення матриці;
- множення матриці на число.

Перевірити роботу цього класу.

Варіант 11.

Створити клас Дата з полями у закритій частині:

- день (1-31),
- місяць (1-12),
- рік (ціле число).

Методи:

- встановлення дня,
- встановлення місяця і
- встановлення року;
- отримання значення дня,
- отримання значення місяця і
- отримання значення року;
- виведення по шаблону: «12 лютого 2007 року» і
- виведення по шаблону: «12.02.2007».

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 12.

Створити клас Час з полями у закритій частині:

- година (0-23),
- хвилини (0-59),
- секунди (0-59).

Методи:

- встановлення години,
- хвилини і
- секунди
- отримання години,
- хвилини і
- секунди,
- виведення по шаблону: «16 годин 18 хвилин 3 секунди» і
- виведення по шаблону: «4 p.m. 18 хвилин 3 секунди».

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 13.

Розробити клас Квадратна матриця. У закритій частині визначити поля:

- порядок матриці та
- вказівник на її початок в області динамічної пам’яті.

Розробити методи класу, які виконують операції:

- встановлення та
- виведення значень елементів матриці,
- визначення сліду матриці (суми елементів головної діагоналі),
- суми елементів вище та нижче головної діагоналі.

Варіант 14.

Створити клас Матриця, який у закритій частині містить поля:

- вказівник на вказівник на float,
- кількість рядків та
- стовпців.

Визначити методи:

- встановлення та
- виведення елементів матриці,
- пошуку мінімального та
- максимального елементів,
- суми елементів матриці.

Варіант 15.

Розробити клас Рядок символів. У закритій частині визначити поле:

- вказівник на символьний тип (на початок рядка).

Визначити методи:

- введення-виведення рядка,
- виведення символу у вказаній позиції,
- перевірки входження заданого символу у рядок.

Варіант 16.

Розробити клас Студент. У закритій частині визначити поля:

- прізвище,
- номер залікової книжки,
- оцінки з предметів.

Визначити методи:

- встановлення та читання значень полів даних,

- знаходження середнього балу,
- визначення кількості незадовільних оцінок.

Варіант 17.

Розробити клас Книга. У закритій частині визначити поля:

- автор,
- назва,
- видавництво,
- рік видання.

Визначити методи:

- встановлення та читання значень полів даних,
- визначення відповідності книги пошуковим критеріям.

Варіант 18.

Розробити клас Динамічний вектор, який має у закритій частині поля:

- вказівник на float,
- кількість елементів і
- змінну стану.

У змінній стану встановлювати код помилки, коли не вистачає пам'яті, або відбувається вихід за межі масиву.

Визначити методи:

- метод ініціалізації Init( ) без параметрів – виділяє місце для одного елемента та ініціалізує його нулем.
- метод ініціалізації Init( ) з одним параметром (кількістю елементів) виділяє місце та ініціалізує масив індексом елемента,
- метод ініціалізації Init( ) із двома параметрами виділяє місце для масиву (кількість елементів задається першим аргументом) та ініціалізує другим аргументом;
- введення-виведення елементів масиву.
- пошуку мінімального та максимального елементів,
- сортування по зростанню та по спаданню значень елементів векторів.

Передбачити можливість підрахунку числа об'єктів створеного класу.

Варіант 19.*

Створіть клас Великі цілі числа, які не можуть бути зображені значеннями вбудованих типів. Методи мають:

- вводити,
- виводити,
- * додавати,
- * віднімати,
- * множити,
- * ділити та
- порівнювати великі числа.

Варіант 20.*

Розробити клас Множина цілих чисел. У закритій частині визначити поле:

- вказівник на цілий тип (на елементи множини).

Визначити методи:

- створення множини,
- виведення вмісту множини,
- * об’єднання,
- * різниці та
- * перетину множин.

Варіант 21.

Створити клас Triangle для представлення трикутника. Поля даних:

- *a*,
- *b*,
- *c* – сторони;
- *A*,
- *B*,
- *C* – протилежні кути.

– мають включати допустимі набори значень кутів та сторін.

Потрібно реалізувати операції:

- перевірки допустимості значень полів даних,
- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,

- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Варіант 22.

Реалізувати клас Account, що є банківським рахунком. У класі має бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Варіант 23.

Реалізувати клас Bankomat, що моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Сума грошей представляється полями-номіналами 10 – 500:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод toString( ) має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Варіант 24.

Створити клас Goods (товари). У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Варіант 25.

Створити клас Payment (зарплата). У класі мають бути представлені поля:

- прізвище-ім’я-побаткові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Варіант 26.

Реалізувати клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Варіант 27.*

Створити клас Date для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день).

Варіант 28.*

Створити клас Time для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Обов’язковими операціями є:

- * додавання часу і заданої кількості секунд,
- * віднімання з часу заданої кількості секунд,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини).

Варіант 29.

Створити клас ModelWindow для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об’єкту.

Варіант 30.

Створити клас Angle для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- grades
- minutes

Обов’язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону 0° – 360°,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса.

Варіант 31.

Створити клас Point для роботи з точками на площині. Координати точки – декартові.

Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- перетворення у полярні координати.

Варіант 32.

Створити клас Прямокутник. У закритій частині визначити поля:

- висоту і

- ширину.

Методи:

- обчислення площі,
- обчислення периметру,
- встановлення висоти,
- встановлення ширини,
- отримання висоти,
- отримання ширини;
- виведення значень елементів класу на екран.

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 33.*

Створити клас Двох-зв'язний список.

Поля:

- вказівник на початок списку,
- вказівник на кінець списку,

Методи:

- додають елемент до списку,
- видаляють елемент зі списку,
- * сортують список;
- відображають елементи списку від початку,
- відображають елементи списку від кінця,

Знайти заданий елемент у списку.

Варіант 34.*

Створити клас Однозв'язний список.

Поле:

- вказівник на початок списку,

Методи:

- додають елемент до списку,
- видаляють елемент зі списку,
- * сортують список;
- відображають елементи списку від початку.

Видалити заданий елемент списку.

Варіант 35.

Створити клас – Коло. У закритій частині визначити поля:

- x,
- y – координати центру та
- R – радіус.

Методи:

- обчислюють площу,
- довжину кола,
- встановлюють поля і
- повертають їх значення.

Методи встановлення значень полів класу мають перевіряти коректність параметрів, що задаються. Розробити функцію виведення полів даних класу.

Варіант 36.

Створити клас Квадрат з полями у закритій частині:

- x_1,
- y_1,
- x_2,
- y_2 – координати головної діагоналі.

Методи обчислюють:

- довжину сторони квадрата,
- площу,
- периметр,
- встановлюють значення полів і
- повертають їх значення.

Методи встановлення полів класу мають перевіряти коректність параметрів, що задаються.

Варіант 37.

Створити клас Стек.

Поле:

- вказівник на вершину стеку,

Розробити методи для:

- включення елемента у стек та
- виключення елемента зі стеку.

Визначити функцію, яка визначає кількість елементів у стеку.

Варіант 38.*

Визначити клас Дроби – раціональні числа, які є відношенням двох цілих чисел.

Поля:

- чисельник,
- знаменник.

Напишіть методи для:

- введення та
- виведення дробу (чисельника та знаменника),
- * скорочення та
- обчислення значення дробу.

Варіант 39.

Цифровий лічильник – це змінна цілого типу з обмеженим діапазоном, який скидається у початкове значення, коли її значення досягає визначеного максимуму.

Приклади: цифровий годинник, лічильник електроенергії.

Опишіть клас Лічильник.

Забезпечте можливість:

- встановлення максимального і
- мінімального значень,
- збільшення значень лічильника на 1,
- повернення поточного значення.

Варіант 40.

Створити клас Вектор цілих елементів, який має у закритій частині:

- вказівник на int – зв’язок з динамічним масивом,
- кількість елементів і
- змінну стану.

У змінній стану встановлювати код помилки, коли не вистачає пам’яті, або відбувається вихід за межі масиву.

Визначити методи:

- виділення пам’яті для елементів,
- присвоєння елементові масиву деякого значення,
- отримання значення деякого елементу масиву;

- виведення масиву;
- визначення евклідової норми вектора.

Перевірити роботу цього класу.

Лабораторна робота № 1.3. Об’єкти – параметри методів (дії над кількома об’єктами)

Мета роботи

Освоїти використання класів та об’єктів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об’єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об’єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об’єктів у функції та повернення об’єктів в якості результату.*
11) *Позначення класів та об’єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*

Зразки виконання завдання

Подається лише умова завдання та текст програми.

У всіх завданнях, крім вказаних в завданні операцій, обов’язково мають бути реалізовані наступні методи:

- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Варіант 01.

Умова завдання

Створити клас Number для роботи з дійсними числами. Число має бути представлене полем x дійсного типу. Тобто, клас Number – це оболонка для стандартного типу double, кожний об'єкт класу Number містить поле (змінну) типу double.

Реалізувати арифметичну операцію додавання двох об'єктів класу Number.

Реалізувати метод перетворення до літерного рядка toString( ).

Аналіз та пояснення різних способів реалізації операції додавання

В першому наближенні визначення класу Number матиме вигляд:

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }
};
```

Кожний об'єкт класу Number буде мати таку будову (припустимо, що створено два об'єкти a та b):

```
Number a, b;
```

– кожний об'єкт містить свою комірку пам'яті, виділену для поля x.

Реалізація звичайним методом з двома параметрами

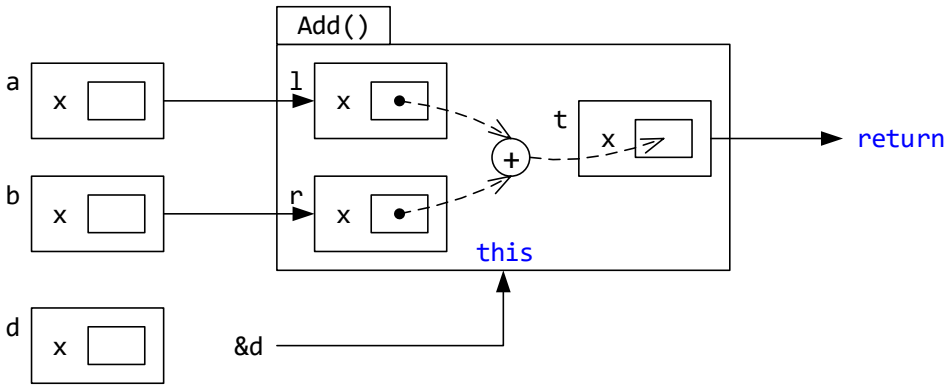
Реалізуємо операцію додавання двох об'єктів класу Number.

Мабуть, найперше, що хочеться зробити – написати звичайний метод, який буде приймати два параметри класу Number та повертати результат типу Number (добавлені рядки виділено сірим фоном):

```
class Number
{
    double x;
```

```
public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    Number Add(Number l, Number r);
};

Number Number::Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

Проте слід пам'ятати, що кожний звичайний метод, крім явно вказаних параметрів, приймає ще один параметр – вказівник `this`, який отримує значення адреси того об'єкта, що викликає цей метод. Таким чином, метод Add() буде отримувати три параметри:



Використовувати таку операцію можна, наприклад, так;

```
int main()
{
    Number a, b, c, d;
    a.setX(1);
    b.setX(2);

    c = d.Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до звичайного методу використовуємо префікс, який містить ім'я об'єкта.

Як бачимо, метод Add() насправді приймає три параметри: два – явно описані в заголовку методу, та параметр `this` – неявний, він містить адресу об'єкта, що викликає цей

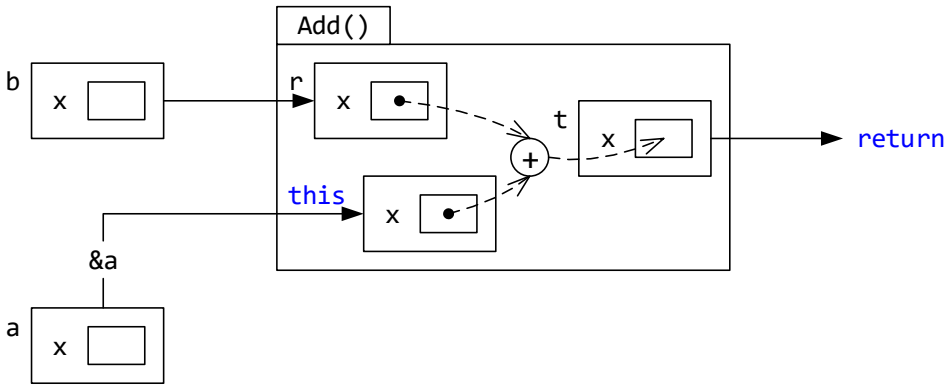
метод (адресу d в нашому прикладі). Також видно, що параметр this ніде в методі не використовується, тому один з цих трьох параметрів – зайвий. Тому цей спосіб, хоча і вірний з точки зору синтаксису, не правильний з точки зору ефективного використання параметрів.

Реалізація звичайним методом з одним параметром

Для того, щоб позбутися зайвих параметрів, перепишемо приклад, використовуючи метод Add(), який приймає лише один параметр (змінені рядки виділено сірим фоном):

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    Number Add(Number r);
};

Number Number::Add(Number r)
{
    Number t;
    t.x = this->x + r.x;
    return t;
}
```

В цій версії діаграма виконання методу Add() буде мати наступний вигляд:



Використовувати цей метод можна так:

```
int main()
{
    Number a, b, c;
    a.setX(1);
    b.setX(2);
```

```
    c = a.Add(b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до звичайного методу використовуємо префікс, який містить ім'я об'єкта.

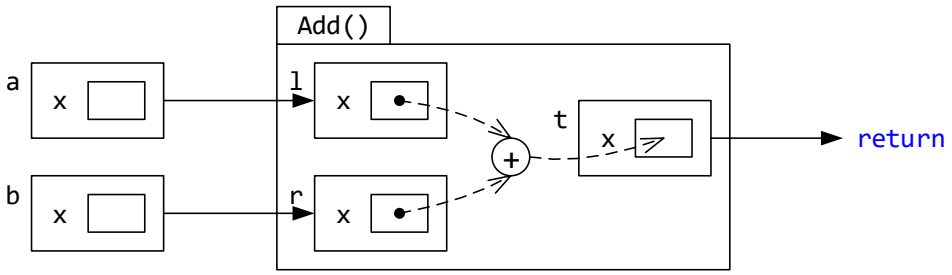
Як бачимо, ми позбулися зайвих параметрів. Проте є ще одна проблема: в такій реалізації операції додавання її лівий та правий операнди – нерівнозначні. Перший (лівий) операнд – це поточний об'єкт, що викликає метод Add(). Другий (правий) операнд – це параметр метода Add(). Завжди поточний об'єкт, який викликає метод, буде більш важливий за параметр цього методу! Але в операції додавання обидва операнди – рівносильні, однакові за важливістю. Тому цей спосіб, хоча і вірний з точки зору синтаксису та правильний з точки зору відсутності параметрів, які не використовуються, – не правильний концептуально: якщо в предметній області (в математиці) для операції додавання лівий та правий операнди – рівносильні, то і в програмній реалізації слід зберегти таку рівносильність.

Реалізація дружньою функцією з двома параметрами

Перепишемо наш приклад, але тепер реалізуємо операцію додавання за допомогою дружньої функції Add(), яка приймає два параметри (змінені рядки виділено сірим фоном):

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    friend Number Add(Number l, Number r);
};

Number Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

В цій версії діаграма виконання дружньої функції Add() буде мати наступний вигляд:



Приклад використання цієї дружньої функції:

```
int main()
{
    Number a, b, c;
    a.setX(1);
    b.setX(2);

    c = Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

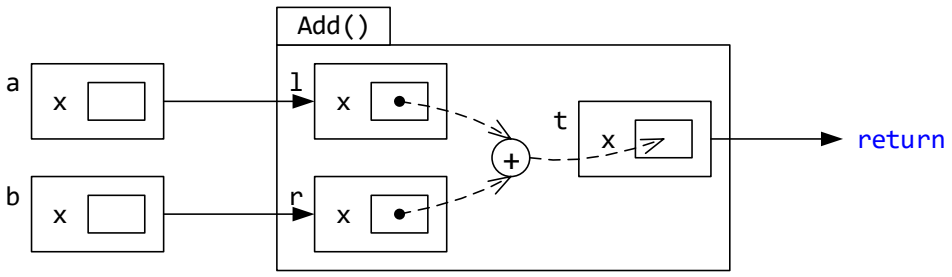
Цей спосіб, вірний як з точки зору синтаксису так і правильний з точки зору відсутності параметрів, що не використовуються. Також цей спосіб правильний концептуально: операція додавання – симетрична, бо її лівий та правий операнди – рівносильні.

Реалізація статичним методом з двома параметрами

Ще один можливий спосіб реалізації операції додавання – використовувати статичний метод (він не приймає параметра `this`). Змінені рядки виділено сірим фоном:

```
class Number
{
    double x;
public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    static Number Add(Number l, Number r);
};

Number Number::Add(Number l, Number r)
{
    Number t;
    t.x = l.x + r.x;
    return t;
}
```

Діаграма виконання статичного методу Add() нічим не буде відрізнятися від діаграми використання дружньої функції:



Приклад використання такого статичного методу:

```
int main()
{
    Number a, b, c;
    a.setX(1);
    b.setX(2);

    c = Number::Add(a, b);

    cout << c.getX() << endl;

    return 0;
}
```

– для звертання до статичного методу використовуємо префікс, який містить ім'я класу.

Реалізація методу перетворення до літерного рядка toString( )

Повернемося до початкового визначення класу Number, яке мало вигляд:

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }
};
```

– це дозволить не захаращувати код реалізацією операції додавання та сконцентруватися на поясненні методу перетворення до літерного рядка toString().

Добавимо метод toString() (нові рядки виділено сірим фоном):

```
#include <string>        // підключаємо бібліотеку, яка описує літерні рядки
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки

using namespace std;
```

```
class Number
{
    double x;

public:
    double getX() const
    {
        return x;
    }

    void setX(double value)
    {
        x = value;
    }

    string toString() const;
};
```

```
string Number::toString() const
{
    stringstream sout;            // створили об'єкт класу "літерний потік"

    sout << "x = " << x << endl;  // направили в літерний потік виведення
                                  // даних про об'єкт

    return sout.str();            // метод str() перетвоює літерний потік
                                  // до літерного рядка
}
```

Пояснення методу перетворення до літерного рядка toString( )

Наведений спосіб реалізації методу toString() використовує літерні потоки.

Потік введення / виведення – це послідовність символів, які пересилаються із консолі в пам'ять (введення) чи з пам'яті на консоль (виведення). Потоки введення / виведення ми використовували, коли реалізовували ефективне потокове введення / виведення даних.

Літерний потік – це послідовність символів, які пересилаються із однієї області пам'яті в іншу.

Для використання літерних потоків необхідно підключити бібліотеку <sstream> (подібно до того, як для використання потоків введення / виведення необхідно підключити бібліотеку <iostream>).

Крім того, бажано добавити директиву використання стандартного простору імен, яка дозволить не вказувати префікс std:: перед іменем кожного ресурсу з цього простору, який ми використовуємо в програмі:

```
#include <string>        // підключаємо бібліотеку, яка описує літерні рядки
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки

using namespace std;
```

Метод перетворення до літерного рядка не міняє стану об’єкта (не змінює його полів). Тому стилістично правильно визначити його як константний метод:

```
class Number
{
    ...

    string toString() const;
};
```

Розглянемо реалізацію цього методу:

```
string Number::toString() const
{
    stringstream sout;              // створили об'єкт класу "літерний потік"

    sout << "x = " << x << endl;    // направили в літерний потік виведення
                                    // даних про об'єкт

    return sout.str();              // метод str() перетвоює літерний потік
                                    // до літерного рядка
}
```

Спочатку оголошуємо змінну sout – об’єкт класу stringstream. Клас stringstream описує тип «літерні потоки», а об’єкт sout (String OUTput) названий за аналогією з іменем cout – об’єкт, який відповідає за консольне виведення (Console OUTput).

Потім ми направляємо до літерного потоку ту інформацію, яку бажаємо перетворити до літерного представлення. Запис у літерний потік відбувається аналогічно до запису у потік виведення: всі дані автоматично перетворюються до літерного представлення (числа подаються не у машинних кодах, а як групи символів, що містять зображення відповідних чисел).

Нарешті, при поверненні результату, для об’єкта sout викликаємо метод str() – цей метод на основі літерного потоку формує літерний рядок, який буде містити символи, що зображують всі ті дані, що були направлені у літерний потік.

Значно гірший спосіб реалізації методу toString() наведено в наступному прикладі:

Варіант 02.

Умова завдання

Реалізувати клас Complex. Поле re – дійсне число x, дійсна частина комплексного числа z, x = Re z; поле im – дійсне число y, мнима частина комплексного числа z, y = Im z, ($|x| \le 100$, $|y| \le 100$).

- Реалізувати метод Abs( ) – обчислення модуля комплексного числа, $\text{abs}(z) = \sqrt{(\text{Re}\, z)^2 + (\text{Im}\, z)^2}$.

- Реалізувати метод AbsToNumeral( ) – перетворення цілої частини в межах до 1000 модуля комплексного числа – результату методу Abs( ) – до числівника (літерного рядка, який містить значення числа прописом): якщо результат методу Abs( ) більший чи рівний 1000, то результатом методу AbsToNumeral( ) має бути літерний рядок «тисяча або більше», в усіх інших випадках – відповідний числівник, наприклад, якщо Abs( ) → 378,14 ,– то AbsToNumeral( ) → «триста сімдесят вісім»

Текст програми

```
////////////////////////////////////////////////////////////////////////////
// Complex.h
//                заголовний файл – визначення класу

#pragma once

class Complex
{
    double re, im;

public:
    double GetRe() const { return re; }
    double GetIm() const { return im; }
    void SetRe(double value) { re = value; }
    void SetIm(double value) { im = value; }

    void   Init(double, double);
    void   Read();
    void   Display() const;
    const char* toString() const;

    double Abs() const;
    const char* AbsToNumeral() const;
};


////////////////////////////////////////////////////////////////////////////
// Complex.cpp
//                файл реалізації – реалізація методів класу

#include "Complex.h"

#include <iostream>
#include <cmath>
#include <string>
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки

using namespace std;

void Complex::Display() const
{
    cout << endl;
    cout << "  Re = " << re << endl;
    cout << "  Im = " << im << endl;
}
```

```
void Complex::Init(double x, double y)
{
    re = x;
    im = y;
}

void Complex::Read()
{
    double x, y;

    cout << "Input complex value:" << endl;
    cout << "  Re = "; cin >> x;
    cout << "  Im = "; cin >> y;

    Init(x, y);
}

const char* Complex::toString() const
{
    stringstream sout;                          // створили об'єкт "літерний потік"

    sout << re << "   +   i * " << im << endl;  // направили в літерний потік
                                                // виведення даних про об'єкт

    return sout.str().c_str();                  // str() перетворює літерний потік
                                                // до літерного рядка
}

double Complex::Abs() const
{
    return sqrt(re * re + im * im);
}

const char* Complex::AbsToNumeral() const
{
    const char* _centuries[11] = { "",              "sto",
                                "dvisti",       "trysta",
                                "4onyrysta",    "p’jatsot",
                                "6istsot",      "simsot",
                                "visimsot",     "dev’jatsot",
                                "tysia4a abo >" };

    const char* _decades[10] = { "",              "",
                                "dvadciat’",    "trydciat’",
                                "sorok",        "p’jatdesiat",
                                "6istdesiat",   "simdesiat",
                                "visimdesiat",  "dev’janosto" };

    const char* _digits[20] = { "",             "odyn",
                                "dva",          "try",
                                "4otyry",       "p’jat’",
                                "6ist’",        "sim",
                                "visim",        "dev’jat’",
                                "desiat’",      "odynadciad’",
                                "dvanadciad’",  "trynadciad’",
                                "4otyrnadciad’", "p’jatnadciad’",
                                "6istnadciad’",  "simnadciad’",
                                "visimnadciad’", "dev’jatnadciad’" };


    if (Abs() >= 1000)
        return _centuries[10];
```

```
    int abs = floor(Abs());
    int cen = abs / 100;
    abs = abs % 100;
    int dec = abs / 10;

    int dig;
    if (dec == 0 || dec == 1)
        dig = abs % 20;
    else
        dig = abs % 10;

    char s[100] = "";
    strcat_s(s, _centuries[cen]);
    strcat_s(s, "  ");
    strcat_s(s, _decades[dec]);
    strcat_s(s, "  ");
    strcat_s(s, _digits[dig]);

    return s;
}

//////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include "Complex.h"

using namespace std;

int main()
{
    Complex z;
    z.Read();
    z.Display();
    cout << "Abs(z) = " << z.Abs() << endl << endl;

    char s[100];
    strcpy(s, z.toString());
    cout << s << endl << endl;

    strcpy(s, z.AbsToNumeral());
    cout << s << endl << endl;

    cin.get();
    return 0;
}
```

Варіанти завдань

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

Варіанти завдань наступні:

Варіант 1.

Комплексне число представляються парою дійсних чисел (x, y), де поля

- x – дійсна частина,
- y – мнима частина.

Реалізувати клас Complex для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання add() $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$;
- множення mul() $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$;
- порівняння equ() $(x_1, y_1) = (x_2, y_2)$, якщо $(x_1 = x_2)$ і $(y_1 = y_2)$.

Варіант 2.

Створити клас Vector3D, що задається трійкою координат. Поля

- x
- y
- z

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

Варіант 4.

Створити клас Point для роботи з точками на площині. Координати точки – декартові.

Поля:

- *x*
- *y*

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані між двома точками.

Варіант 5.*

Раціональний (нескоротний) дріб представляється парою цілих чисел (*a*, *b*), де поля:

- *a* – чисельник,
- *b* – знаменник.

Створити клас Rational для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є даний об'єкт):

- обчислення значення value(), *a* / *b*;

```
double Rational::value()
{
   return 1.*a/b;
```

```
}

Rational z;
...
double x = z.value();
```

Бінарні операції (перший аргумент – даний об'єкт, другий аргумент – об'єкт-параметр):

- додавання add(), $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$;
- віднімання sub(), $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$;
- множення mul(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational
{
private:
    int a, b;
public:
    /* ... */
    Rational add(Rational& r);
};

Rational Rational::add(Rational& r)
{
    Rational tmp;

    tmp.a = a * r.b  +  b * r.a;
    tmp.b = b * r.b;

    return tmp;
}
```

Використання додавання як методу класу:

```
    Rational z1, z2, z3;
    /* ... */
    z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational
{
private:
    int a, b;
public:
    /* ... */
    friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r)
```

```
{
    Rational tmp;

    tmp.a = l.a * r.b  +  l.b * r.a;
    tmp.b = l.b * r.b;

    return tmp;
}
```

Використання додавання як дружньої функції:

```
Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);
```

Варіант 6.

Реалізувати клас FuzzyNumber для роботи з нечіткими числами, які представляються трійками чисел $(x - l, x, x + r)$. Поля:

- x
- l
- r

Для чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

  $A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$;

- множення

  $A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$.

Пояснення FuzzyNumber

Нечіткі числа подаються трійками $(x - l, x, x + r)$, де

x – координата центру,

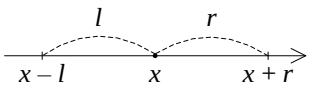
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

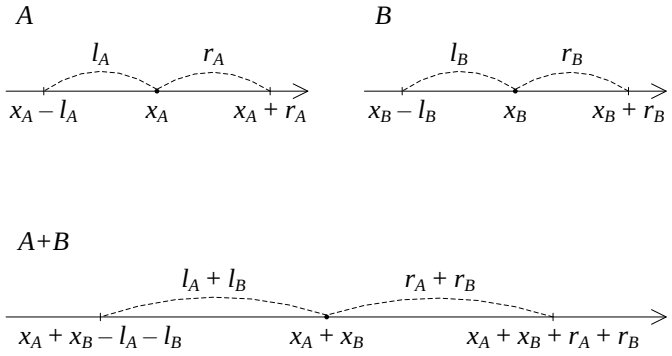
r – відстань від правої границі до центру:



Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

Додавання двох нечітких чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ описується формулою

$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$



Тобто, для суми двох нечітких чисел $A+B$:

$x_A + x_B$ – координата центру,

$x_A + x_B - l_A - l_B$ – координата лівої границі,

$x_A + x_B + r_A + r_B$ – координата правої границі.

Відповідно,

$l_A + l_B$ – відстань від лівої границі до центру,

$r_A + r_B$ – відстань від правої границі до центру.

Таким чином, якщо об'єкт-число A містить поля $\{x_A, l_A, r_A\}$, а об'єкт-число B – поля $\{x_B, l_B, r_B\}$, то об'єкт-сума містить поля $\{x_A + x_B, l_A + l_B, r_A + r_B\}$.

Варіант 7.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.

- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 8.*

Створити клас Fraction для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

* має бути реалізована можливість опрацювання чисел з різною кількістю цифр в дробових частинах.

Варіант 9.

Створити клас BitString для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу long. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- not,
- and,
- or.

Варіант 10.*

Створити клас LongLong для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - * додавання,
  - * множення;
- операції порівняння:
  - менше, не менше, більше.

Варіант 11.

Створити клас Vector2D, що задається парою координат. Поля

- *x*
- *y*

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 12.

Комплексне число представляються парою дійсних чисел (*x*, *y*), де поля

- *x* – дійсна частина,
- *y* – мнима частина.

Реалізувати клас Complex для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання sub() $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$;
- ділення div() $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$;
- комплексно спряжене число conj() $\text{conj}(x, y) = (x, -y)$.

Варіант 13.

Створити клас Vector3D, що задається трійкою координат. Поля

- *x*

- *y*
- *z*

Обов’язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 14.

Створити клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

Варіант 15.

Створити клас Point для роботи з точками на площині. Координати точки – декартові.

Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Варіант 16.*

Раціональний (нескоротний) дріб представляється парою цілих чисел (*a*, *b*), де поля:

- *a* – чисельник,
- *b* – знаменник.

Створити клас Rational для роботи з раціональними дробами. Обов’язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є сам цей об’єкт):

- обчислення значення value(), a / b;

```
double Rational::value(){
   return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

бінарні операції (перший аргумент – сам цей об’єкт, другий аргумент – об’єкт-параметр):

- ділення div(), $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot b_2, a_2 \cdot b_1)$;
- порівняння «чи рівне» equal();
- порівняння «чи більше» great();
- порівняння «чи менше» less().

* має бути реалізована приватна функція скорочення дробу Reduce(), яка обов’язково викликається при виконанні арифметичних операцій.

Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational {
private:
    int a, b;
public:
    /* ... */
    Rational add(Rational& r);
};

Rational Rational::add(Rational& r) {
    Rational tmp;

    tmp.a = a * r.b  +  b * r.a;
    tmp.b = b * r.b;

    return tmp;
}
```

Використання додавання як методу класу:

```
    Rational z1, z2, z3;
    /* ... */
    z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational {
private:
    int a, b;
public:
```

```
    /* ... */
    friend Rational add(Rational& l, Rational& r);
};

Rational add(Rational& l, Rational& r) {
    Rational tmp;

    tmp.a = l.a * r.b  +  l.b * r.a;
    tmp.b = l.b * r.b;

    return tmp;
}
```

Використання додавання як дружньої функції:

```
Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);
```

Варіант 17.

Реалізувати клас FuzzyNumber для роботи з нечіткими числами, які представляються трійками чисел $(x - l, x, x + r)$. Поля:

- x
- l
- r

Для чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- віднімання

  $A - B = (x_A - x_B - l_A - l_B, x_A - x_B, x_A - x_B + r_A + r_B)$;

- зворотне число

  $1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0$;

- ділення

  $A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0$.

Пояснення FuzzyNumber

Нечіткі числа подаються трійками $(x - l, x, x + r)$, де

x – координата центру,

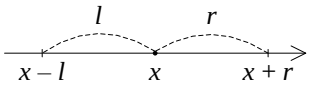
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

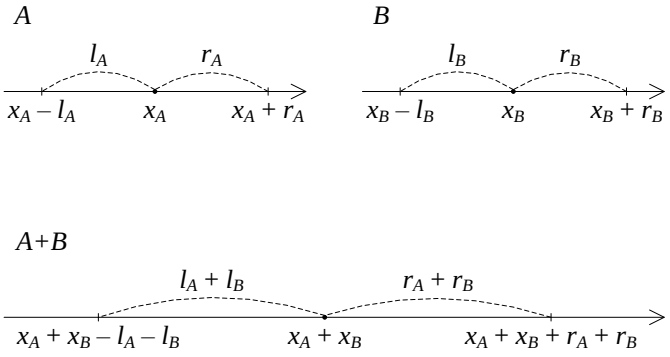
r – відстань від правої границі до центру:



Клас FuzzyNumber містить три поля: {*x*, *l*, *r*}.

Додавання двох нечітких чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ описується формулою

$A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$



Тобто, для суми двох нечітких чисел *A*+*B*:

$x_A + x_B$ – координата центру,

$x_A + x_B - l_A - l_B$ – координата лівої границі,

$x_A + x_B + r_A + r_B$ – координата правої границі.

Відповідно,

$l_A + l_B$ – відстань від лівої границі до центру,

$r_A + r_B$ – відстань від правої границі до центру.

Таким чином, якщо об'єкт-число *A* містить поля $\{x_A, l_A, r_A\}$, а об'єкт-число *B* – поля $\{x_B, l_B, r_B\}$, то об'єкт-сума містить поля $\{x_A + x_B, l_A + l_B, r_A + r_B\}$.

Варіант 18.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.

- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- ділення сум,
- ділення суми на дробове число,
- операції порівняння сум.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 19.

Створити клас Fraction для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- віднімання,
- операції порівняння.

Варіант 20.*

Створити клас BitString для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу long. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- xor,
- * зсув ліворуч shiftLeft на задану кількість бітів,
- * зсув праворуч shiftRight на задану кількість бітів.

Варіант 21.*

Створити клас LongLong для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції (без присвоєння):
    - * віднімання,
    - * ділення;
- операції порівняння:
    - не більше, дорівнює, не дорівнює.

Варіант 22.

Створити клас VectorN, що задається групою *N* дійсних чисел – координат вектора. Поля

- *N* – розмірність вектора,
- *a* – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 23.

Створити клас VectorN, що задається групою *N* дійсних чисел – координат вектора. Поля

- *N* – розмірність вектора,
- *a* – масив дійсних чисел, який реалізує вектор.

Обов'язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 24.

Створити клас Matrix – реалізує матрицю цілих елементів, який містить закриті поля:

- *m* – двовимірний масив,
- *R* – кількість рядків,

- C – кількість стовпців.

Визначити методи для:

- повернення значення елемента, який має індекси (i, j);
- виведення матриці;
- додавання матриць;
- віднімання матриць;
- множення матриць;
- множення матриці на число.

Варіант 25.

Розробити клас CharLine – реалізує рядок N символів. У закритій частині визначити поля:

- N – довжина рядка (кількість символів);
- s – масив, який вміщує N символів.

Визначити методи:

- введення-виведення рядка,
- виведення символу у вказаній позиції,
- перевірки входження заданого символу у рядок.
- конкатенації,
- порівняння рядків,
- перевірки входження під-рядка у рядок.

Варіант 26.

Комплексне число представляються парою дійсних чисел (x, y), де поля

- x – дійсна частина,
- y – мнима частина.

Реалізувати клас Complex для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- додавання add() $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$;
- множення mul() $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + x_2 \cdot y_1)$;
- порівняння equ() $(x_1, y_1) = (x_2, y_2)$, якщо $(x_1 = x_2)$ і $(y_1 = y_2)$.

Варіант 27.

Створити клас Vector3D, що задається трійкою координат. Поля

- x
- y

- *z*

Обов’язково мають бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 28.

Створити клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- додавання сум,
- ділення сум,
- ділення суми на дробове число.

Варіант 29.

Створити клас Point для роботи з точками на площині. Координати точки – декартові.

Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані між двома точками.

Варіант 30.*

Раціональний (нескоротний) дріб представляється парою цілих чисел (*a*, *b*), де поля:

- *a* – чисельник,
- *b* – знаменник.

Створити клас Rational для роботи з раціональними дробами. Обов’язково мають бути реалізовані наступні методи:

Унарна операція (аргументом є даний об’єкт):

- обчислення значення value(), *a* / *b*;

```
double Rational::value()
{
   return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

Бінарні операції (перший аргумент – даний об'єкт, другий аргумент – об'єкт-параметр):

- додавання add(), $(a_1, b_1) + (a_2, b_2) = (a_1 \cdot b_2 + a_2 \cdot b_1, b_1 \cdot b_2)$;
- віднімання sub(), $(a_1, b_1) - (a_2, b_2) = (a_1 \cdot b_2 - a_2 \cdot b_1, b_1 \cdot b_2)$;
- множення mul(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Пояснення Rational

Реалізація додавання за допомогою методу класу:

```
class Rational
{
private:
    int a, b;
public:
    /* ... */
    Rational add(Rational& r);
};

Rational Rational::add(Rational& r)
{
    Rational tmp;

    tmp.a = a * r.b  +  b * r.a;
    tmp.b = b * r.b;

    return tmp;
}
```

Використання додавання як методу класу:

```
    Rational z1, z2, z3;
    /* ... */
    z3 = z1.add(z2);
```

Реалізація додавання за допомогою дружньої функції:

```
class Rational
{
private:
    int a, b;
public:
    /* ... */
    friend Rational add(Rational& l, Rational& r);
```

```
};

Rational add(Rational& l, Rational& r)
{
    Rational tmp;

    tmp.a = l.a * r.b  +  l.b * r.a;
    tmp.b = l.b * r.b;

    return tmp;
}
```

Використання додавання як дружньої функції:

```
Rational z1, z2, z3;
/* ... */
z3 = add(z1, z2);
```

Варіант 31.

Реалізувати клас FuzzyNumber для роботи з нечіткими числами, які представляються трійками чисел $(x - l, x, x + r)$. Поля:

- x
- l
- r

Для чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

  $A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$;

- множення

  $A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$.

Пояснення FuzzyNumber

Нечіткі числа подаються трійками $(x - l, x, x + r)$, де

x – координата центру,

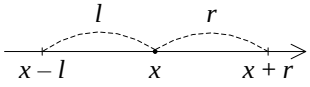
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

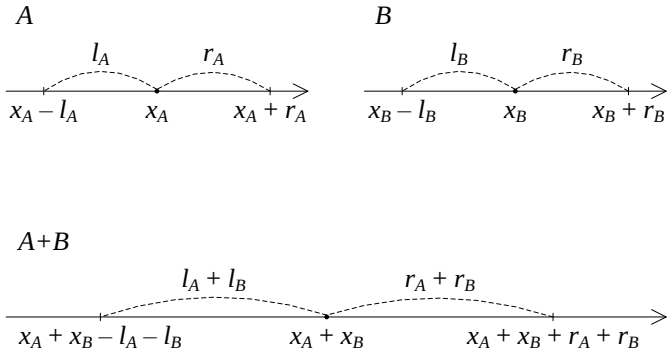
r – відстань від правої границі до центру:



Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

Додавання двох нечітких чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ описується формулою

$$A + B = (x_A + x_B - l_A - l_B,\ x_A + x_B,\ x_A + x_B + r_A + r_B)$$



Тобто, для суми двох нечітких чисел $A+B$:

$x_A + x_B$ – координата центру,

$x_A + x_B - l_A - l_B$ – координата лівої границі,

$x_A + x_B + r_A + r_B$ – координата правої границі.

Відповідно,

$l_A + l_B$ – відстань від лівої границі до центру,

$r_A + r_B$ – відстань від правої границі до центру.

Таким чином, якщо об'єкт-число A містить поля $\{x_A, l_A, r_A\}$, а об'єкт-число B – поля $\{x_B, l_B, r_B\}$, то об'єкт-сума містить поля $\{x_A + x_B, l_A + l_B, r_A + r_B\}$.

Варіант 32.

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок).

Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.

- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- множення суми на дробове число.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 33.*

Створити клас Fraction для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати методи – арифметичні операції:

- додавання,
- множення.

* має бути реалізована можливість опрацювання чисел з різною кількістю цифр в дробових частинах.

Варіант 34.

Створити клас BitString для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу long. Мають бути визначені методи, які реалізують всі традиційні операції для роботи з бітами:

- not,
- and,
- or.

Варіант 35.*

Створити клас LongLong для роботи з 64-розрядними цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Мають бути реалізовані методи, які представляють:

- арифметичні операції, присутні в мові програмування (без присвоєння):
  - * додавання,
  - * множення;
- операції порівняння:
  - менше, не менше, більше.

Варіант 36.

Створити клас Vector2D, що задається парою координат. Поля

- *x*
- *y*

Обов'язково мають бути реалізовані:

- скалярний добуток векторів,
- множення на скаляр,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 37.

Комплексне число представляються парою дійсних чисел (*x*, *y*), де поля

- *x* – дійсна частина,
- *y* – мнима частина.

Реалізувати клас Complex для роботи з комплексними числами. Обов'язково мають бути реалізовані методи:

- віднімання sub() $(x_1, y_1) - (x_2, y_2) = (x_1 - x_2, y_1 - y_2)$;
- ділення div() $(x_1, y_1) / (x_2, y_2) = (x_1 \cdot x_2 + y_1 \cdot y_2, x_2 \cdot y_1 - x_1 \cdot y_2) / (x_2^2 + y_2^2)$;
- комплексно спряжене число conj() $\text{conj}(x, y) = (x, -y)$.

Варіант 38.

Створити клас Vector3D, що задається трійкою координат. Поля

- *x*

- *y*
- *z*

Обов’язково мають бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 39.

Створити клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати методи:

- віднімання сум,
- множення на дробове число,
- операції порівняння сум.

Варіант 40.

Створити клас Point для роботи з точками на площині. Координати точки – декартові.

Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Лабораторна робота № 1.4. Використання класів

Мета роботи

Освоїти використання вкладених та дружніх класів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об'єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об'єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об'єктів у функції та повернення об'єктів в якості результату.*
11) *Позначення класів та об'єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*
14) *Поняття агрегування та композиції – в термінах предметної області.*
15) *Реалізація агрегування та композиції – в термінах програмування.*
16) *Позначення агрегування та композиції на UML-діаграмах класів.*
17) Поняття вкладеного класу.
18) Поняття та визначення дружнього класу.
19) Визначення вкладеного класу всередині класу-контейнера.
20) Визначення вкладеного класу поза класом-контейнером.
21) Доступ до елементів вкладеного класу із методів класу-контейнера.
22) Доступ до елементів класу-контейнера із вкладеного класу.

Варіанти завдань

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як клас із закритими полями, де операції реалізуються як методи класу.

Визначення класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

Зауваження щодо реалізації множин:

Множина – це сукупність даних, для яких ідентифікується лише факт входження елемента у цю сукупність; це означає, що ніякої інформації про кількість однакових елементів ми не зберігаємо – тобто, дублікати не допускаються.

Множина представляється масивом, кожний елемент масиву набуває значень лише 0 або 1: 0 означає, що відповідного елемента немає у множині, 1 – що відповідний елемент у множині є. Незалежно від кількості елементів у множині, масив завжди має стільки елементів, скільки максимально може бути елементів у множині. Фактична кількість елементів у множині = кількості 1 у масиві.

Включення (добавлення) елемента у множину = записати 1 у відповідну комірку масиву. Вилучення (видалення) елемента із множини = записати 0 у відповідну комірку масиву.

Об'єднання 2-х множин = виконати операцію "або" над елементами 2-х масивів у відповідних позиціях, і т.д.

Варіанти завдань наступні:

Варіант 1.**

Описати клас, який реалізує стек. Написати програму, використовуючи цей клас для моделювання Т-подібного сортувального вузла на залізниці. Програма має розділяти на два напрями потяг, який складається із вагонів двох типів (на кожний напрям формується потяг із вагонів одного типу). Передбачити можливість формування потягу із файлу* та з клавіатури.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 2.***

Описати клас, який реалізує бінарне дерево, яке має можливість додавання нових елементів, видалення існуючих, пошуку елемента за ключем, а також послідовного доступу до всіх елементів.

Написати програму, яка використовує цей клас для представлення англо-українського словника. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класу. Передбачити можливість формування словника із файлу* та з клавіатури.

Варіант 3.

Побудувати систему класів для опису плоских геометричних фігур: круга, квадрата, прямокутника. Передбачити методи для створення об’єктів, переміщення на площині, зміни розмірів і обертання на заданий кут.

Написати програму, яка демонструє роботу з цими класами. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класів.

Варіант 4.

Описати клас, який містить інформацію про поштову адресу організації. Передбачити можливість роздільної зміни складових частин адреси, створення і ліквідації об’єктів цього класу.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класу.

Варіант 5.

Описати клас для представлення комплексних чисел. Забезпечити виконання операцій додавання, віднімання і множення комплексних чисел.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 6.

Описати клас для об’єктів-векторів, які задаються координатами кінців в трьохвимірному просторі. Забезпечити методи додавання і віднімання векторів з отриманням нового вектору (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 7.

Описати клас прямокутників зі сторонами, паралельними осям координат. Передбачити можливість переміщення прямокутників на площині, зміни розмірів, побудову найменшого прямокутника, який містить два заданих прямокутника, і прямокутника, який є спільною частиною (перетином) двох прямокутників.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 8.*

Описати клас для визначення одномірних масивів цілих чисел (векторів). Передбачити можливість звертання до окремого елементу масиву з контролем виходу за межі масиву, можливість визначення довільних меж (границь) індексів при створенні об’єкту, можливість виконання по-елементного додавання і віднімання масивів з однаковими межами індексів, множення і ділення всіх елементів масиву на скаляр, виводу на екран елемента масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 9.*

Описати клас для визначення одномірних масивів літерних рядків фіксованої довжини. Передбачити можливість звернення до окремих рядків масиву за індексом, контроль виходу за межі масиву, виконання по-елементного з’єднання (конкатенації) двох масивів з виключенням елементів, що повторюються, вивід на екран елемента масиву по заданому індексу і всього масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 10.*

Описати клас многочленів від одної змінної, які задаються степенем многочлена n і масивом коефіцієнтів $a_0, a_1,\ldots, a_n$. Передбачити методи для обчислення значення многочлена для заданого аргументу x, методи додавання, віднімання і множення з отриманням нового об'єкта-многочлена, вивід на екран опису многочлена у вигляді літерного рядка виду:

$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$

– де замість $n, a_0, a_1,\ldots, a_n$ – будуть вказані конкретні значення.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 11.*

Описати клас одновимірних масивів літерних рядків, кожний рядок задається довжиною і вказівником на виділену для нього пам'ять. Передбачити можливість звертання до окремих рядків масиву по індексу, контроль виходу за межі масивів, виконання по-елементної конкатенації двох масивів з утворенням нового масиву; злиття двох масивів з виключенням елементів, які повторюються; вивід на екран елементу масиву із заданим індексом та всього масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 12.*

Описати клас, який забезпечує представлення матриці довільного розміру з можливістю зміни кількості рядків і стовпчиків, виводу на екран під-матриці будь-якого розміру і всієї матриці.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 13.

Написати клас для ефективної роботи з літерними рядками, який дозволяє форматувати і порівнювати рядки, зберігати в рядках числові значення і отримувати їх.

Для цього необхідно реалізувати методи:

- присвоєння і конкатенації;
- порівняння;
- перетворення вмісту рядка в число цілого чи дійсного типу;
- перетворення числа цілого чи дійсного типу у літерний рядок;

- форматний вивід рядків.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 14.*

Описати клас «домашня бібліотека». Передбачити можливість роботи з довільною кількістю книг, пошуку книги за вказаною ознакою (наприклад, по автору або по року видання), добавлення книг в бібліотеку, видалення книг з бібліотеки, сортування книг по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 15.*

Описати клас «записна книжка». Передбачити можливість роботи з довільною кількістю записів, пошуку запису за вказаною ознакою (наприклад, по прізвищу, даті народження або номеру телефону), додавання і видалення записів, сортування по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 16.*

Описати клас «студентська група». Передбачити можливість роботи із змінною кількістю студентів, пошуку студента за вказаною ознакою (наприклад, по прізвищу, даті народження або номеру телефону), додавання і видалення записів, сортування по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 17.*

Описати клас, який реалізує тип даних «дійсна матриця» і роботу з ними. Клас має реалізувати наступні дії над матрицями:

- додавання, віднімання, множення (+, –, *, /) на іншу матрицю;
- додавання, віднімання, множення, ділення (+, –, *, /) на число;
- порівняння на рівність/нерівність;
- методи, які реалізують перевірку типу матриці (квадратна, діагональна, нульова, одинична, симетрична, верхня трикутна, нижня трикутна);

- ввід/вивід в стандартні потоки.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 18.*

Описати клас «множина», який дозволяє виконувати основні операції – добавлення і вилучення елемента, перетин, об'єднання і різниця множин.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 19.**

Описати клас, який реалізує стек. Написати програму, яка використовує цей клас для пошуку проходу по лабіринту.

Лабіринт представляється у вигляді матриці, яка складається з квадратів. Кожний квадрат або відкритий, або закритий. Вхід в закритий квадрат заборонений. Якщо квадрат відкритий, то вхід в нього можливий зі сторони, але не з кута. Кожний квадрат визначається його координатами в матриці. Після знаходження проходу програма друкує знайдений шлях в вигляді координат квадратів.

Варіант 20.*

Описати клас «предметний покажчик». Кожний компонент покажчика містить слово і номери сторінок, на котрих це слово зустрічається. Кількість номерів сторінок, які стосуються одного слова, від одного до десяти. Передбачити можливість формування покажчика з клавіатури, виводу покажчика, виводу номера сторінок для заданого слова, видалення елемента із покажчика.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 21.**

Описати клас, який реалізує стек. Написати програму, використовуючи цей клас для моделювання Т-подібного сортувального вузла на залізниці. Програма має розділяти на два напрями потяг, який складається із вагонів двох типів (на кожний напрям формується потяг із вагонів одного типу). Передбачити можливість формування потягу із файлу* та з клавіатури.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 22.***

Описати клас, який реалізує бінарне дерево, яке має можливість додавання нових елементів, видалення існуючих, пошуку елемента за ключем, а також послідовного доступу до всіх елементів.

Написати програму, яка використовує цей клас для представлення англо-українського словника. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класу. Передбачити можливість формування словника із файлу* та з клавіатури.

Варіант 23.

Побудувати систему класів для опису плоских геометричних фігур: круга, квадрата, прямокутника. Передбачити методи для створення об’єктів, переміщення на площині, зміни розмірів і обертання на заданий кут.

Написати програму, яка демонструє роботу з цими класами. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класів.

Варіант 24.

Описати клас, який містить інформацію про поштову адресу організації. Передбачити можливість роздільної зміни складових частин адреси, створення і ліквідації об’єктів цього класу.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснювати перевірку всіх методів класу.

Варіант 25.

Описати клас для представлення комплексних чисел. Забезпечити виконання операцій додавання, віднімання і множення комплексних чисел.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 26.

Описати клас для об’єктів-векторів, які задаються координатами кінців в трьохвимірному просторі. Забезпечити методи додавання і віднімання векторів з отриманням нового вектору (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 27.

Описати клас прямокутників зі сторонами, паралельними осям координат. Передбачити можливість переміщення прямокутників на площині, зміни розмірів, побудову найменшого прямокутника, який містить два заданих прямокутника, і прямокутника, який є спільною частиною (перетином) двох прямокутників.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 28.*

Описати клас для визначення одномірних масивів цілих чисел (векторів). Передбачити можливість звертання до окремого елементу масиву з контролем виходу за межі масиву, можливість визначення довільних меж (границь) індексів при створенні об’єкту, можливість виконання по-елементного додавання і віднімання масивів з однаковими межами індексів, множення і ділення всіх елементів масиву на скаляр, виводу на екран елемента масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 29.*

Описати клас для визначення одномірних масивів літерних рядків фіксованої довжини. Передбачити можливість звернення до окремих рядків масиву за індексом, контроль виходу за межі масиву, виконання по-елементного з’єднання (конкатенації) двох масивів з виключенням елементів, що повторюються, вивід на екран елемента масиву по заданому індексу і всього масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 30.*

Описати клас многочленів від одної змінної, які задаються степенем многочлена n і масивом коефіцієнтів $a_0, a_1,\ldots, a_n$. Передбачити методи для обчислення значення многочлена для заданого аргументу x, методи додавання, віднімання і множення з отриманням нового об’єкта-многочлена, вивід на екран опису многочлена у вигляді літерного рядка виду:

$P(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \ldots + a_n \cdot x^n$

– де замість $n, a_0, a_1,\ldots, a_n$ – будуть вказані конкретні значення.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 31.*

Описати клас одновимірних масивів літерних рядків, кожний рядок задається довжиною і вказівником на виділену для нього пам'ять. Передбачити можливість звертання до окремих рядків масиву по індексу, контроль виходу за межі масивів, виконання поелементної конкатенації двох масивів з утворенням нового масиву; злиття двох масивів з виключенням елементів, які повторюються; вивід на екран елементу масиву із заданим індексом та всього масиву.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 32.*

Описати клас, який забезпечує представлення матриці довільного розміру з можливістю зміни кількості рядків і стовпчиків, виводу на екран під-матриці будь-якого розміру і всієї матриці.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 33.

Написати клас для ефективної роботи з літерними рядками, який дозволяє форматувати і порівнювати рядки, зберігати в рядках числові значення і отримувати їх.

Для цього необхідно реалізувати методи:

- присвоєння і конкатенації;
- порівняння;
- перетворення вмісту рядка в число цілого чи дійсного типу;
- перетворення числа цілого чи дійсного типу у літерний рядок;
- форматний вивід рядків.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 34.*

Описати клас «домашня бібліотека». Передбачити можливість роботи з довільною кількістю книг, пошуку книги за вказаною ознакою (наприклад, по автору або по року

видання), добавлення книг в бібліотеку, видалення книг з бібліотеки, сортування книг по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 35.*

Описати клас «записна книжка». Передбачити можливість роботи з довільною кількістю записів, пошуку запису за вказаною ознакою (наприклад, по прізвищу, даті народження або номеру телефону), додавання і видалення записів, сортування по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 36.*

Описати клас «студентська група». Передбачити можливість роботи із змінною кількістю студентів, пошуку студента за вказаною ознакою (наприклад, по прізвищу, даті народження або номеру телефону), додавання і видалення записів, сортування по різним полям.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 37.*

Описати клас, який реалізує тип даних «дійсна матриця» і роботу з ними. Клас має реалізувати наступні дії над матрицями:

- додавання, віднімання, множення (+, –, *, /) на іншу матрицю;
- додавання, віднімання, множення, ділення (+, –, *, /) на число;
- порівняння на рівність/нерівність;
- методи, які реалізують перевірку типу матриці (квадратна, діагональна, нульова, одинична, симетрична, верхня трикутна, нижня трикутна);
- ввід/вивід в стандартні потоки.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 38.*

Описати клас «множина», який дозволяє виконувати основні операції – добавлення і вилучення елемента, перетин, об’єднання і різниця множин.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Варіант 39.**

Описати клас, який реалізує стек. Написати програму, яка використовує цей клас для пошуку проходу по лабіринту.

Лабіринт представляється у вигляді матриці, яка складається з квадратів. Кожний квадрат або відкритий, або закритий. Вхід в закритий квадрат заборонений. Якщо квадрат відкритий, то вхід в нього можливий зі сторони, але не з кута. Кожний квадрат визначається його координатами в матриці. Після знаходження проходу програма друкує знайдений шлях в вигляді координат квадратів.

Варіант 40.*

Описати клас «предметний покажчик». Кожний компонент покажчика містить слово і номери сторінок, на котрих це слово зустрічається. Кількість номерів сторінок, які стосуються одного слова, від одного до десяти. Передбачити можливість формування покажчика з клавіатури, виводу покажчика, виводу номера сторінок для заданого слова, видалення елемента із покажчика.

Написати програму, яка демонструє роботу з цим класом. Програма має містити меню, яке дозволяє здійснити перевірку всіх методів класу.

Лабораторна робота № 1.5. Композиція класів та об’єктів

Мета роботи

Освоїти використання композитних класів та об’єктів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об’єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об’єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об’єктів у функції та повернення об’єктів в якості результату.*
11) *Позначення класів та об’єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*
14) Поняття агрегування та композиції – в термінах предметної області.
15) Реалізація агрегування та композиції – в термінах програмування.
16) Позначення агрегування та композиції на UML-діаграмах класів.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

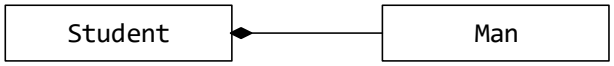
Умова варіанту:

Створити клас Man (Людина) з полями: ім’я, вік. Визначити методи пере-присвоєння імені, зміни віку.

Створити клас-контейнер Student (Студент), що має поля «людина» та «спеціальність». Визначити методи пере-присвоєння та зміни спеціальності.

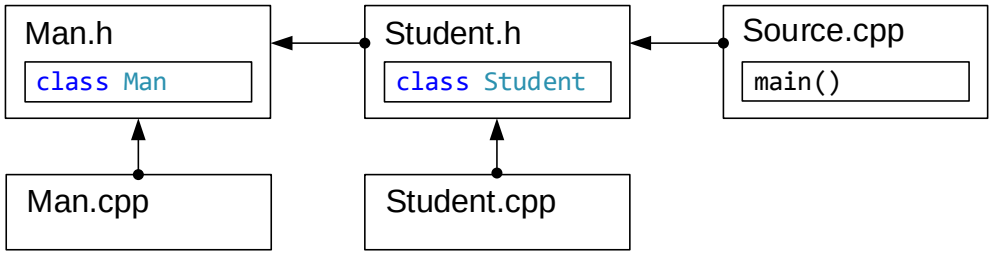
Семантика зв'язку композиції Студент – Людина: студент містить людину, – тобто, вважаємо, що десь глибоко всередині кожного студента схована людина (схована – бо всі поля традиційно робимо прихованими).

UML-діаграма класів

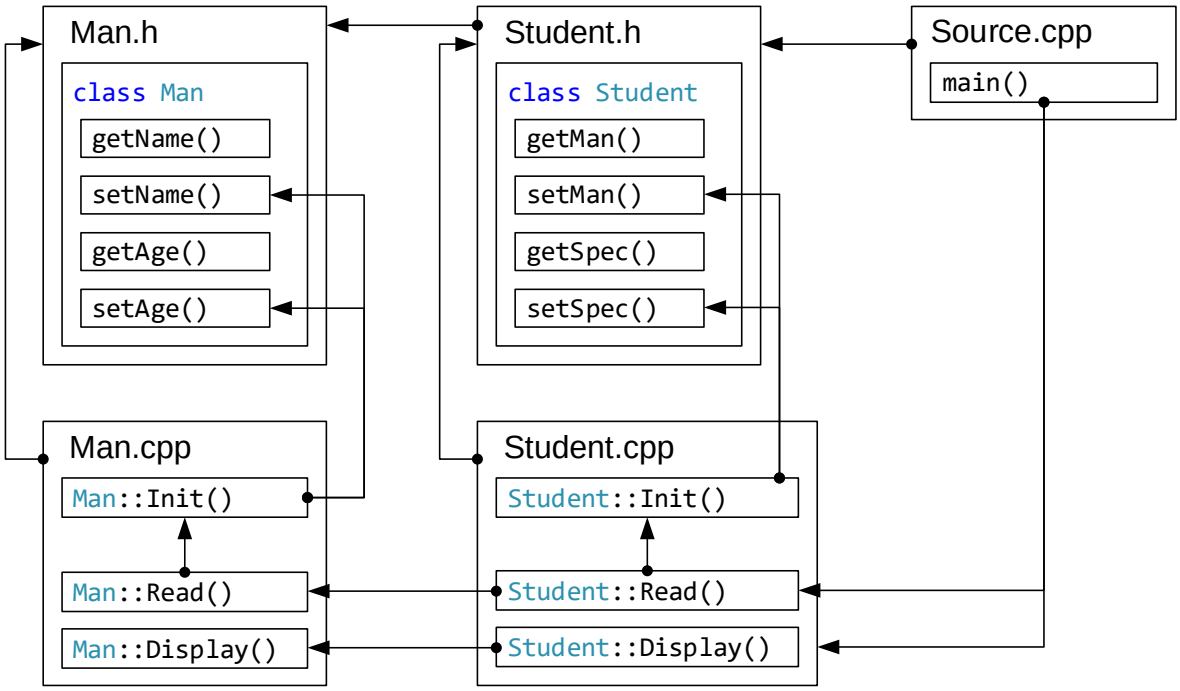


Структурна схема

Скорочений варіант:



Повний варіант:



Текст програми

```
////////////////////////////////////////////////////////////////////////
// Man.h
//                заголовний файл – визначення класу

#pragma once
```

```cpp
#include <string>

using namespace std;

class Man
{
private:
    string name;
    int age;

public:
    string getName() const { return name; }
    int getAge() const { return age; }

    void setName(string name) { this->name = name; }
    void setAge(int age) { this->age = age; }

    void Init(string name, int age);
    void Display() const;
    void Read();
};

////////////////////////////////////////////////////////////////////////////
// Man.cpp
//                файл реалізації – реалізація методів класу

#include "Man.h"
#include <iostream>

using namespace std;

void Man::Init(string name, int age)
{
    setName(name);
    setAge(age);
}

void Man::Display() const
{
    cout << endl;
    cout << "name = " << name << endl;
    cout << "age  = " << age << endl;
}

void Man::Read()
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age  = ? "; cin >> age;
    Init(name, age);
}

////////////////////////////////////////////////////////////////////////////
// Student.h
//                заголовний файл – визначення класу

#pragma once
#include <string>
#include "Man.h"
```

```
using namespace std;

class Student
{
private:
    string spec;
    Man man;

public:
    string getSpec() const { return spec; }
    Man getMan() const { return man; }

    void setSpec(string spec) { this->spec = spec; }
    void setMan(Man man) { this->man = man; }

    void Init(string spec, Man man);
    void Display() const;
    void Read();
};

//////////////////////////////////////////////////////////////////////////
// Student.cpp
//                файл реалізації – реалізація методів класу

#include "Student.h"
#include <iostream>

using namespace std;

void Student::Init(string spec, Man man)
{
    setSpec(spec);
    setMan(man);
}

void Student::Display() const
{
    cout << endl;
    cout << "man = " << endl;
    man.Display();                          // використовуємо делегування
    cout << "spec  = " << spec << endl;
}

void Student::Read()
{
    string spec;
    Man m;
    cout << endl;
    cout << "Man  = ? " << endl;
    m.Read();                               // використовуємо делегування
    cout << "spec = ? "; cin >> spec;
    Init(spec, m);
}

//////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include "Student.h"

using namespace std;
```

```
int main()
{
    Student s;
    s.Read();
    s.Display();

    return 0;
}
```

Варіанти завдань

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як класи із закритими полями, де операції реалізуються як методи класу.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Варіанти завдань наступні:

Варіант 1.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 2.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 3.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 4.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 5.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 6.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 7.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 8.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни

сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 9.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 10.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 11.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 12.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 13.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на n днів.

Варіант 14.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь заданого степеня, і метод для обчислення числа π в заданій степені.

Варіант 15.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на n секунд і хвилин.

Варіант 16.

Створити клас Pair (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) – (c, d) = (a – c, b – d).$$

Створити клас-контейнер Rational, що містить поле «пара цілих чисел»; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) – (c, d) = (ad – bc, bd).$$

Варіант 17.

Створити клас Pair (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Complex, що містить поле «пара чисел» – пара чисел описує дійсну і мниму частини. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

Варіант 18.*

Створити клас Pair (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Long, що містить поле «пара цілих чисел» – пара чисел описує старшу і молодшу частини числа. Перевизначити операцію додавання і визначити методи множення і віднімання.

Варіант 19.

Створити клас Triad (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити клас-контейнер Vector3D, що містить поле «трійка чисел». Вектор задається трійкою координат, які описуються полем – трійкою чисел. Мають бути реалізовані: операція додавання векторів, скалярний добуток векторів.

Варіант 20.

Створити клас Pair (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Money, що містить поле «пара цілих чисел» – пара чисел описує гривні і копійки. Перевизначити операцію додавання і визначити методи віднімання і ділення грошових сум.

Варіант 21.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Bus (автобус), що містить поля «машина» та «кількість пасажирських місць». Визначити функції пере-присвоєння марки та зміни кількості пасажирських місць.

Варіант 22.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар: пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Rational (дріб), що містить поле «пара чисел» – пара чисел описує чисельник і знаменник. Визначити повний набір методів порівняння.

Варіант 23.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Solution (розчин), що містить поля «рідина» та «відносна кількість речовини». Визначити методи пере-присвоєння та зміни відносної кількості речовини.

Варіант 24.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Ellipse (еліпс), що містить поле «пара чисел» – пара чисел описує пів-осі. Визначити методи обчислення периметру та площі еліпсу.

Варіант 25.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «курс». Визначити методи пере-присвоєння та збільшення курсу.

Варіант 26.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 27.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 28.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 29.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 30.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 31.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 32.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 33.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 35.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 35.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 36.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 37.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 38.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

Варіант 39.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь заданого степеня, і метод для обчислення числа π в заданій степені.

Варіант 40.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

Лабораторна робота № 1.6. Вкладені класи

Мета роботи

Освоїти використання вкладених та дружніх класів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об'єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об'єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об'єктів у функції та повернення об'єктів в якості результату.*
11) *Позначення класів та об'єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*
14) *Поняття агрегування та композиції – в термінах предметної області.*
15) *Реалізація агрегування та композиції – в термінах програмування.*
16) *Позначення агрегування та композиції на UML-діаграмах класів.*
17) Поняття вкладеного класу.
18) Поняття та визначення дружнього класу.
19) Визначення вкладеного класу всередині класу-контейнера.
20) Визначення вкладеного класу поза класом-контейнером.
21) Доступ до елементів вкладеного класу із методів класу-контейнера.
22) Доступ до елементів класу-контейнера із вкладеного класу.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

Виконати завдання свого варіанту Лабораторної роботи № 1.5, використовуючи конструкцію вкладеного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Умова варіанту:

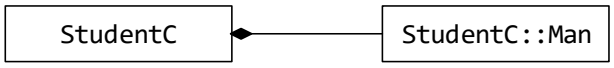
Створити допоміжний клас Man (людина) з полями: ім'я, вік. Визначити методи пере-присвоєння імені, зміни віку.

Створити основний клас-контейнер StudentC (студент), що має поля «людина» та «спеціальність». Визначити методи пере-присвоєння та зміни спеціальності.

Семантика зв'язку композиції Студент – Людина:

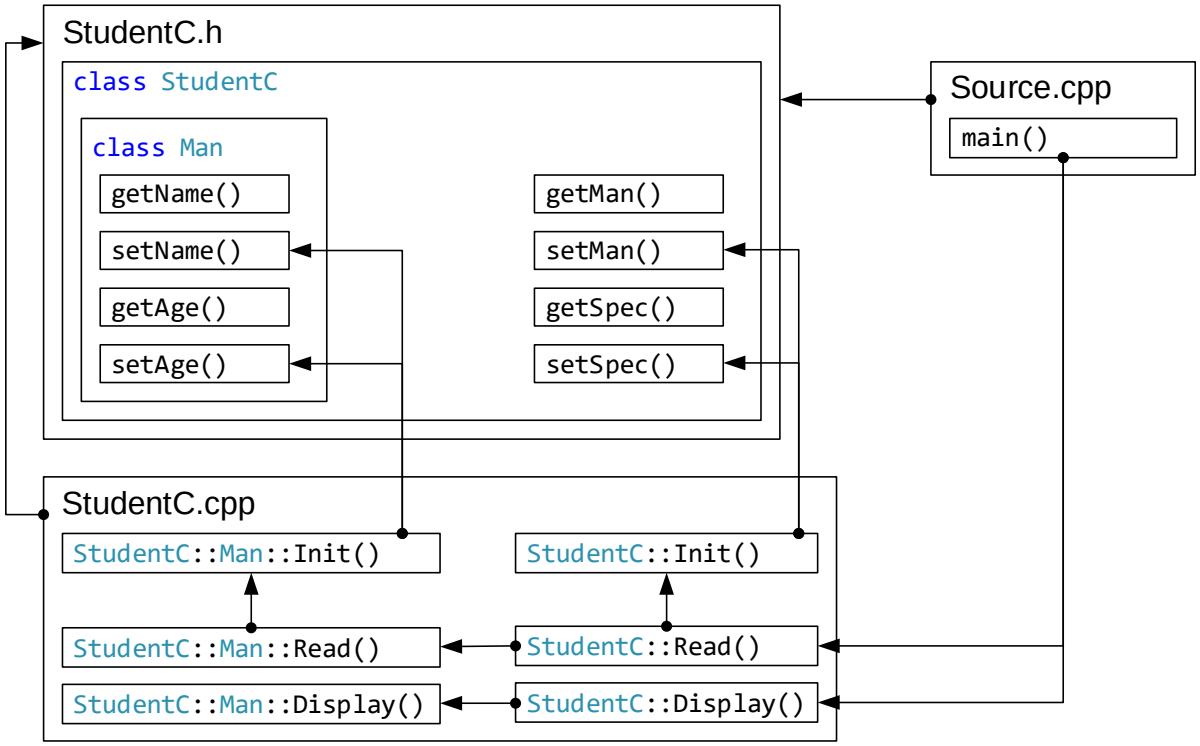
1) студент містить людину, – тобто, вважаємо, що десь глибоко всередині кожного студента схована людина (схована – бо всі поля традиційно робимо прихованими).
2) поняття «людина» існує лише в контексті поняття «студент», – тобто, не може бути інших людей, крім тих, які є глибоко всередині студентів.

UML-діаграма класів

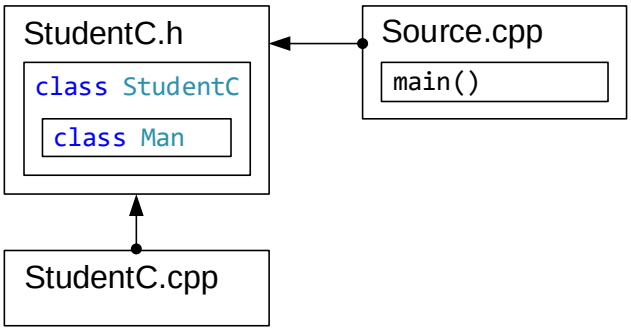


Структурна схема

Повний варіант:



Скорочений варіант:



Текст програми

```
////////////////////////////////////////////////////////////////////////////
// StudentC.h
//                заголовний файл – визначення класу

#pragma once
#include <string>

using namespace std;

class StudentC
{
private:
    string spec;

    class Man
    {
    private:
        string name;
        int age;

    public:
        string getName() const { return name; }
        int getAge() const { return age; }

        void setName(string name) { this->name = name; }
        void setAge(int age) { this->age = age; }

        void Init(string name, int age);
        void Display() const;
        void Read();
    };

    Man man;

public:
    string getSpec() const { return spec; }
    StudentC::Man getMan() const { return man; }

    void setSpec(string spec) { this->spec = spec; }
    void setMan(StudentC::Man man) { this->man = man; }

    void Init(string spec, StudentC::Man man);
    void Display() const;
    void Read();
};
```

```
//////////////////////////////////////////////////////////////////////////
// StudentC.cpp
//                файл реалізації – реалізація методів класу

#include "StudentC.h"
#include <iostream>

using namespace std;

void StudentC::Init(string spec, StudentC::Man man)
{
    setSpec(spec);
    setMan(man);
}

void StudentC::Display() const
{
    cout << endl;
    cout << "man = " << endl;
    man.Display();                          // використовуємо делегування
    cout << "spec  = " << spec << endl;
}

void StudentC::Read()
{
    string spec;
    StudentC::Man m;
    cout << endl;
    cout << "Man  = ? " << endl;
    m.Read();                               // використовуємо делегування
    cout << "spec = ? "; cin >> spec;
    Init(spec, m);
}

void StudentC::Man::Init(string name, int age)
{
    setName(name);
    setAge(age);
}

void StudentC::Man::Display() const
{
    cout << endl;
    cout << "name = " << name << endl;
    cout << "age  = " << age << endl;
}

void StudentC::Man::Read()
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age  = ? "; cin >> age;
    Init(name, age);
}

//////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include "StudentC.h"
```

```
using namespace std;

int main()
{
    StudentC s;
    s.Read();
    s.Display();

    return 0;
}
```

Варіанти завдань

Виконати завдання свого варіанту Лабораторної роботи № 1.5, використовуючи конструкцію вкладеного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Лабораторна робота № 1.5:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як класи із закритими полями, де операції реалізуються як методи класу.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння

марки та зміни вантажопідйомності.

Варіант 2.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 3.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 4.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 5.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 6.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 7.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни

сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 8.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 9.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 10.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 11.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 12.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи

додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 13.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

Варіант 14.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь заданого степеня, і метод для обчислення числа π в заданій степені.

Варіант 15.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

Варіант 16.

Створити клас Pair (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) - (c, d) = (a - c, b - d).$$

Створити клас-контейнер Rational, що містить поле «пара цілих чисел»; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) - (c, d) = (ad - bc, bd).$$

Варіант 17.

Створити клас Pair (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Complex, що містить поле «пара чисел» – пара чисел описує дійсну і мниму частини. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

Варіант 18.*

Створити клас Pair (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Long, що містить поле «пара цілих чисел» – пара чисел описує старшу і молодшу частини числа. Перевизначити операцію додавання і визначити методи множення і віднімання.

Варіант 19.

Створити клас Triad (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити клас-контейнер Vector3D, що містить поле «трійка чисел». Вектор задається трійкою координат, які описуються полем – трійкою чисел. Мають бути реалізовані: операція додавання векторів, скалярний добуток векторів.

Варіант 20.

Створити клас Pair (пара цілих чисел); визначити метод множення на число і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Money, що містить поле «пара цілих чисел» – пара чисел описує гривні і копійки. Перевизначити операцію додавання і визначити методи віднімання і ділення грошових сум.

Варіант 21.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни

потужності.

Створити клас-контейнер Bus (автобус), що містить поля «машина» та «кількість пасажирських місць». Визначити функції пере-присвоєння марки та зміни кількості пасажирських місць.

Варіант 22.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:
пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Rational (дріб), що містить поле «пара чисел» – пара чисел описує чисельник і знаменник. Визначити повний набір методів порівняння.

Варіант 23.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Solution (розчин), що містить поля «рідина» та «відносна кількість речовини». Визначити методи пере-присвоєння та зміни відносної кількості речовини.

Варіант 24.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Ellipse (еліпс), що містить поле «пара чисел» – пара чисел описує пів-осі. Визначити методи обчислення периметру та площі еліпсу.

Варіант 25.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «курс». Визначити методи пере-присвоєння та збільшення курсу.

Варіант 26.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і

характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 27.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 28.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 29.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 30.

Створити клас Man (людина) з полями: ім’я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 31.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 32.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 33.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 34.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 35.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 36.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 37.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 38.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

Варіант 39.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь заданого степеня, і метод для обчислення числа π в заданій степені.

Варіант 40.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

Лабораторна робота № 1.7. Композиція класів та об’єктів – складніші завдання

Мета роботи

Освоїти використання композитних класів та об’єктів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об’єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об’єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об’єктів у функції та повернення об’єктів в якості результату.*
11) *Позначення класів та об’єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*
14) Поняття агрегування та композиції – в термінах предметної області.
15) Реалізація агрегування та композиції – в термінах програмування.
16) Позначення агрегування та композиції на UML-діаграмах класів.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є основним, всі решту – допоміжні. Допоміжні класи мають бути визначені як незалежні. Об’єкти допоміжних класів мають використовуватися як поля основного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Реалізувати клас ComplexPoint (точка на комплексній площині). Поле name – символьного типу – літера, що означає назву точки. Поле complex – комплексне число –

реалізувати за допомогою класу Complex із завдання демонстраційного прикладу Лабораторної роботи № 1.3. Реалізувати методи роботи з назвою точки SetName( ), GetName( ) та методи роботи з комплексним числом – вказані у завданні демонстраційного прикладу Лабораторної роботи № 1.3.

Текст програми

```
////////////////////////////////////////////////////////////////////////////
// Complex.h
//                заголовний файл – визначення класу

#pragma once

class Complex
{
    double re, im;

public:
    double GetRe() const { return re; }
    double GetIm() const { return im; }
    void SetRe(double value) { re = value; }
    void SetIm(double value) { im = value; }

    void   Init(double, double);
    void   Read();
    void   Display();
    const char* toString();

    double Abs();
    const char* AbsToNumeral();
};

////////////////////////////////////////////////////////////////////////////
// Complex.cpp
//                файл реалізації – реалізація методів класу

#include "Complex.h"

#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <string>
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки
using namespace std;

void Complex::Init(double x, double y)
{
    re = x;
    im = y;
}

void Complex::Read()
{
    double x, y;

    cout << "Input complex value:" << endl;
    cout << "  Re = "; cin >> x;
    cout << "  Im = "; cin >> y;
```

```
    Init(x, y);
}

void Complex::Display()
{
    cout << endl;
    cout << "  Re = " << re << endl;
    cout << "  Im = " << im << endl;
}

const char* Complex::toString()
{
    stringstream sout;                          // створили об'єкт "літерний потік"

    sout << re << "   +   i * " << im << endl;  // направили в літерний потік
                                                // виведення даних про об'єкт

    return sout.str().c_str();                  // str() перетворює літерний потік
                                                // до літерного рядка
}

double Complex::Abs()
{
    return sqrt(re * re + im * im);
}

const char* Complex::AbsToNumeral()
{
    const char* _centuries[11] = { "",              "sto",
                                "dvisti",       "trysta",
                                "4onyrysta",    "p'jatsot",
                                "6istsot",      "simsot",
                                "visimsot",     "dev'jatsot",
                                "tysia4a abo >" };

    const char* _decades[10] = { "",              "",
                                "dvadciat'",    "trydciat'",
                                "sorok",        "p'jatdesiat",
                                "6istdesiat",   "simdesiat",
                                "visimdesiat",  "dev'janosto" };

    const char* _digits[20] = { "",             "odyn",
                                "dva",          "try",
                                "4otyry",       "p'jat'",
                                "6ist'",        "sim",
                                "visim",        "dev'jat'",
                                "desiat'",      "odynadciad'",
                                "dvanadciad'",  "trynadciad'",
                                "4otyrnadciad'", "p'jatnadciad'",
                                "6istnadciad'",  "simnadciad'",
                                "visimnadciad'", "dev'jatnadciad'" };

    if (Abs() >= 1000)
        return _centuries[10];

    int abs = floor(Abs());
    int cen = abs / 100;
    abs = abs % 100;
    int dec = abs / 10;

    int dig;
```

```
    if (dec == 0 || dec == 1)
        dig = abs % 20;
    else
        dig = abs % 10;

    char s[100] = "";
    strcat_s(s, _centuries[cen]);
    strcat_s(s, "  ");
    strcat_s(s, _decades[dec]);
    strcat_s(s, "  ");
    strcat_s(s, _digits[dig]);

    return s;
}

////////////////////////////////////////////////////////////////////////////
// ComplexPoint.h
//                заголовний файл – визначення класу

#pragma once
#include "Complex.h"

class ComplexPoint
{
    char    name;
    Complex complex;

public:
    char   GetName() const { return name; }
    void   SetName(char value) { name = value; }

    Complex GetComplex() const { return complex; }
    void SetComplex(Complex value)
    {
        complex.SetRe(value.GetRe());
        complex.SetIm(value.GetIm());
    }

    void   Init(double, double);
    void   Read();
    void   Display();
    const char* toString();

    double Abs();
    const char* AbsToNumeral();
};

////////////////////////////////////////////////////////////////////////////
// ComplexPoint.cpp
//                файл реалізації – реалізація методів класу

#include "ComplexPoint.h"

void ComplexPoint::Init(double x, double y)
{
    complex.Init(x, y);                 // використовуємо делегування
}

void ComplexPoint::Read()
{
    complex.Read();                     // використовуємо делегування
}
```

```cpp
void ComplexPoint::Display()
{
    complex.Display();                    // використовуємо делегування
}

const char* ComplexPoint::toString()
{
    return complex.toString();            // використовуємо делегування
}

double ComplexPoint::Abs()
{
    return complex.Abs();                 // використовуємо делегування
}

const char* ComplexPoint::AbsToNumeral()
{
    return complex.AbsToNumeral();        // використовуємо делегування
}

//////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include "complexPoint.h"
#include <iostream>

using namespace std;

int main()
{
    ComplexPoint z;

    char c;
    cout << " Name = "; cin >> c;
    z.SetName(c);
    z.Read();

    cout << endl << "Output:" << endl;
    cout << " Name = " << z.GetName() << endl;
    z.Display();
    cout << "Abs(z) = " << z.Abs() << endl << endl;

    char s[100];
    strcpy_s(s, z.toString());
    cout << s << endl << endl;

    strcpy_s(s, z.AbsToNumeral());
    cout << s << endl << endl;

    cin.get();
    return 0;
}
```

Варіанти завдань

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як класи із закритими полями, де операції реалізуються як методи класу.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,

- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 2.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,

- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев’ятсот дев’яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої

частини комою.

Варіант 3.

Реалізувати клас Calculator з повним набором арифметичних операцій, використовуючи клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 4.

Реалізувати клас Bankomat:

Клас Bankomat, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод toString( ) має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки

представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 5*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу DigitString,

- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас DigitString, а для представлення дробової частини без-знакове коротке ціле.

Клас DigitString – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 6*.

Реалізувати клас Calculator з повним набором арифметичних операцій, на основі класу Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас LongLong, а для представлення дробової частини додатне дробове число типу double:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 7.

Реалізувати клас Triangle:

Клас Triangle – для представлення трикутника. Поля даних:

- *a*,
- *b*,
- *c* – сторони;
- *A*,
- *B*,
- *C* – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас Angle:

Клас Angle – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов’язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону 0° – 360°,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

Варіант 8.

Реалізувати клас Goods:

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

Варіант 9.

Реалізувати клас Triangle з полями – координатами вершин:

Клас Triangle – для представлення трикутника. Поля даних:

- P_1,

- P_2,
- P_3 – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- *get_a( ),*
- *get_b( ),*
- *get_c( )* – обчислення довжин сторін;
- *get_A( ),*
- *get_B( ),*
- *get_C( )* – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас Point:

Клас Point – для роботи з точками на площині. Координати точки – декартові. Поля:

- *x*
- *y*

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

Варіант 10.

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побаткові,
- ставка,
- рік поступлення на роботу,

- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас Money:

Клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 11.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене

вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 12.

Реалізувати клас ModelWindow, додавши поле для курсору:

Створити клас ModelWindow для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,

- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об’єкту.

Для представлення поля курсору використовуйте клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Варіант 13*.

Реалізувати клас Set (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас BitString:

Клас BitString – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу unsigned long. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- and,
- or,
- xor,
- not.
- * зсув ліворуч shiftLeft та
- * зсув праворуч shiftRight на задану кількість бітів.

Клас Set (множина) має забезпечувати операції:

- включення елементу в множину,
- виключення елементу з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

Варіант 14*.

Реалізувати клас Rational:

Раціональний (нескоротний) дріб представляється парою цілих чисел (a, b), де поля:

- a – чисельник,
- b – знаменник.

Клас Rational – для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні операції:

Припустимо, що (a, b) – перше число = $^{a}/_{b}$ – перший об'єкт; (c, d) – друге число = $^{c}/_{d}$ – другий об'єкт.

- * додавання add(), $(a, b) + (c, d) = (ad + bc, bd) = {}^{(ad + bc)}/_{(bd)}$;
- * віднімання sub(), $(a, b) – (c, d) = (ad – bc, bd)$;
- * множення mul(), $(a, b) \times (c, d) = (ac, bd)$;
- * ділення div(), $(a, b) / (c, d) = (ad, bc)$;
- порівняння equal(), great(), less().

Має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 15*.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- * додавання,
- * віднімання,
- * ділення сум,
- * ділення суми на дробове число,
- * множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 16*.

Реалізувати клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- * зміни координат курсору,
- зміни виду курсору,

- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас LongLong:

Клас LongLong для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 17.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев’ятсот дев’яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу

unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

Варіант 18.*

Реалізувати клас Goods, додавши поле – дату надходження товару на склад.

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Використовувати клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

Варіант 19.*

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,

- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу Date.

Варіант 20.*

Реалізувати клас Bill, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,

- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас Time:

Клас Time – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- * обчислення різниці між двома моментами часу в секундах,
- * додавання часу і заданої кількості секунд,
- * віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод toString( ) має видавати суму в гривнях.

Варіант 21.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і

- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 22.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати

наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,

- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 23.

Реалізувати клас Calculator з повним набором арифметичних операцій, використовуючи клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 24.

Реалізувати клас Bankomat:

Клас Bankomat, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод toString( ) має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 25*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу DigitString,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас DigitString, а для представлення дробової частини без-знакове коротке ціле.

Клас DigitString – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 26*.

Реалізувати клас Calculator з повним набором арифметичних операцій, на основі класу Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас LongLong, а для

представлення дробової частини додатне дробове число типу double:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 27.

Реалізувати клас Triangle:

Клас Triangle – для представлення трикутника. Поля даних:

- *a*,
- *b*,
- *c* – сторони;
- *A*,
- *B*,
- *C* – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас Angle:

Клас Angle – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов’язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону 0° – 360°,

- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

Варіант 28.

Реалізувати клас Goods:

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

Варіант 29.

Реалізувати клас Triangle з полями – координатами вершин:

Клас Triangle – для представлення трикутника. Поля даних:

- P_1,
- P_2,
- P_3 – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- *get_a( ),*
- *get_b( ),*
- *get_c( )* – обчислення довжин сторін;
- *get_A( ),*
- *get_B( ),*
- *get_C( )* – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас Point:

Клас Point – для роботи з точками на площині. Координати точки – декартові. Поля:

- *x*
- *y*

Обов'язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

Варіант 30.

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім’я-побаткові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас Money:

Клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 31.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 32.

Реалізувати клас ModelWindow, додавши поле для курсору:

Створити клас ModelWindow для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,

- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об’єкту.

Для представлення поля курсору використовуйте клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Варіант 33*.

Реалізувати клас Set (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас BitString:

Клас BitString – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу unsigned long. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- and,
- or,
- xor,
- not.

- * зсув ліворуч shiftLeft та
- * зсув праворуч shiftRight на задану кількість бітів.

Клас Set (множина) має забезпечувати операції:

- включення елементу в множину,
- виключення елементу з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

Варіант 34*.

Реалізувати клас Rational:

Раціональний (нескоротний) дріб представляється парою цілих чисел (a, b), де поля:

- a – чисельник,
- b – знаменник.

Клас Rational – для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні операції:

Припустимо, що (a, b) – перше число = $^{a}/_{b}$ – перший об'єкт; (c, d) – друге число = $^{c}/_{d}$ – другий об'єкт.

- * додавання add(), (a, b) + (c, d) = ($ad + bc$, bd) = $^{(ad + bc)}/_{(bd)}$;
- * віднімання sub(), (a, b) – (c, d) = ($ad – bc$, bd);
- * множення mul(), (a, b) × (c, d) = (ac, bd);
- * ділення div(), (a, b) / (c, d) = (ad, bc);
- порівняння equal(), great(), less().

Має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 35*.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- * додавання,
- * віднімання,
- * ділення сум,
- * ділення суми на дробове число,
- * множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 36*.

Реалізувати клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- * зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас LongLong:

Клас LongLong для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 37.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев’ятсот дев’яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

Варіант 38.*

Реалізувати клас Goods, додавши поле – дату надходження товару на склад.

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.

- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Використовувати клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

Варіант 39.*

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу Date.

Варіант 40.*

Реалізувати клас Bill, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас Time:

Клас Time – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- * обчислення різниці між двома моментами часу в секундах,
- * додавання часу і заданої кількості секунд,
- * віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод toString( ) має видавати суму в гривнях.

Лабораторна робота № 1.8. Вкладені класи – складніші завдання

Мета роботи

Освоїти використання вкладених та дружніх класів.

Питання, які необхідно вивчити та пояснити на захисті

1) *Поняття класів та об'єктів.*
2) *Загальний синтаксис опису класів.*
3) *Визначення та оголошення класу.*
4) *Елементи класу: поля та методи.*
5) *Інкапсуляція[1]: об'єднання опису даних та опису дій над даними в єдине ціле.*
6) *Роль вказівника* this *в реалізації інкапсуляції[1].*
7) *Інкапсуляція[2]: обмеження доступу до даних.*
8) *Директиви доступу (видимості) елементів класу.*
9) *Звертання до полів та методів класу.*
10) *Передавання об'єктів у функції та повернення об'єктів в якості результату.*
11) *Позначення класів та об'єктів на UML-діаграмах класів.*
12) *Позначення елементів класів на UML-діаграмах класів.*
13) *Позначення доступних та приватних елементів класів на UML-діаграмах класів.*
14) *Поняття агрегування та композиції – в термінах предметної області.*
15) *Реалізація агрегування та композиції – в термінах програмування.*
16) *Позначення агрегування та композиції на UML-діаграмах класів.*
17) Поняття вкладеного класу.
18) Поняття та визначення дружнього класу.
19) Визначення вкладеного класу всередині класу-контейнера.
20) Визначення вкладеного класу поза класом-контейнером.
21) Доступ до елементів вкладеного класу із методів класу-контейнера.
22) Доступ до елементів класу-контейнера із вкладеного класу.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

Виконати завдання свого варіанту Лабораторної роботи № 1.7, використовуючи конструкцію вкладеного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Реалізувати клас ComplexPoint (точка на комплексній площині). Поле name – символьного типу – літера, що означає назву точки. Поле complex – комплексне число – реалізувати за допомогою класу Complex із завдання демонстраційного прикладу Лабораторної роботи № 1.3. Реалізувати методи роботи з назвою точки SetName(), GetName() та методи роботи з комплексним числом – вказані у завданні демонстраційного прикладу Лабораторної роботи № 1.3.

Текст програми

```
////////////////////////////////////////////////////////////////////////
// ComplexPoint.h
//                заголовний файл – визначення класу

#pragma once

class ComplexPoint
{
    char    name;

public:
    class Complex
    {
        double re, im;

    public:
        double GetRe() const { return re; }
        double GetIm() const { return im; }
        void SetRe(double value) { re = value; }
        void SetIm(double value) { im = value; }

        void   Init(double, double);
        void   Read();
        void   Display();
        const char* toString();

        double Abs();
        const char* AbsToNumeral();
    };

    char   GetName() const { return name; }
    void   SetName(char value) { name = value; }

    Complex GetComplex() const { return complex; }
    void SetComplex(Complex value)
    {
        complex.SetRe(value.GetRe());
        complex.SetIm(value.GetIm());
    }
```

```
    void   Init(double, double);
    void   Read();
    void   Display();
    const char* toString();

    double Abs();
    const char* AbsToNumeral();

private:
    Complex complex;
};

//////////////////////////////////////////////////////////////////////////
// ComplexPoint.cpp
//                файл реалізації – реалізація методів класу

#include "ComplexPoint.h"

#include <iostream>
#include <cmath>
#include <string>
#include <sstream>       // підключаємо бібліотеку, яка описує літерні потоки

using namespace std;

//////////////////////////////////////////////////////////
// class ComplexPoint

void ComplexPoint::Init(double x, double y)
{
    complex.Init(x, y);                  // використовуємо делегування
}

void ComplexPoint::Read()
{
    complex.Read();                      // використовуємо делегування
}

void ComplexPoint::Display()
{
    complex.Display();                   // використовуємо делегування
}

const char* ComplexPoint::toString()
{
    return complex.toString();           // використовуємо делегування
}

double ComplexPoint::Abs()
{
    return complex.Abs();                // використовуємо делегування
}

const char* ComplexPoint::AbsToNumeral()
{
    return complex.AbsToNumeral();       // використовуємо делегування
}


//////////////////////////////////////////////////////////
// class ComplexPoint::Complex
```

```
void ComplexPoint::Complex::Init(double x, double y)
{
    re = x;
    im = y;
}

void ComplexPoint::Complex::Display()
{
    cout << endl;
    cout << "  Re = " << re << endl;
    cout << "  Im = " << im << endl;
}

void ComplexPoint::Complex::Read()
{
    double x, y;

    cout << "Input complex value:" << endl;
    cout << "  Re = "; cin >> x;
    cout << "  Im = "; cin >> y;

    Init(x, y);
}

const char* ComplexPoint::Complex::toString()
{
    stringstream sout;                          // створили об'єкт "літерний потік"

    sout << re << "   +   i * " << im << endl;  // направили в літерний потік
                                                // виведення даних про об'єкт

    return sout.str().c_str();                  // str() перетворює літерний потік
                                                // до літерного рядка
}

double ComplexPoint::Complex::Abs()
{
    return sqrt(re * re + im * im);
}

const char* ComplexPoint::Complex::AbsToNumeral()
{
    const char* _centuries[11] = { "",              "sto",
                                   "dvisti",        "trysta",
                                   "4onyrysta",     "p'jatsot",
                                   "6istsot",       "simsot",
                                   "visimsot",      "dev'jatsot",
                                   "tysia4a abo >" };

    const char* _decades[10] = { "",              "",
                                 "dvadciat'",     "trydciat'",
                                 "sorok",         "p'jatdesiat",
                                 "6istdesiat",    "simdesiat",
                                 "visimdesiat",   "dev'janosto" };

    const char* _digits[20] = { "",             "odyn",
                                "dva",          "try",
                                "4otyry",       "p'jat'",
                                "6ist'",        "sim",
                                "visim",        "dev'jat'",
                                "desiat'",      "odynadciad'",
                                "dvanadciad'",  "trynadciad',
```

```
                                "4otyrnadciad'", "p'jatnadciad'",
                                "6istnadciad'",  "simnadciad'",
                                "visimnadciad'", "dev'jatnadciad'" };


    if (Abs() >= 1000)
        return _centuries[10];

    int abs = floor(Abs());
    int cen = abs / 100;
    abs = abs % 100;
    int dec = abs / 10;

    int dig;
    if (dec == 0 || dec == 1)
        dig = abs % 20;
    else
        dig = abs % 10;

    char s[100] = "";
    strcat_s(s, _centuries[cen]);
    strcat_s(s, "  ");
    strcat_s(s, _decades[dec]);
    strcat_s(s, "  ");
    strcat_s(s, _digits[dig]);

    return s;
}

////////////////////////////////////////////////////////////////////////////
// Source.cpp
//                головний файл проекту – функція main

#include "ComplexPoint.h"
#include <iostream>

using namespace std;

int main()
{
    ComplexPoint z;

    char c;
    cout << " Name = "; cin >> c;
    z.SetName(c);
    z.Read();

    cout << endl << "Output:" << endl;
    cout << " Name = " << z.GetName() << endl;
    z.Display();
    cout << "Abs(z) = " << z.Abs() << endl << endl;

    char s[100];
    strcpy_s(s, z.toString());
    cout << s << endl << endl;

    strcpy_s(s, z.AbsToNumeral());
    cout << s << endl << endl;

    cin.get();
    return 0;
}
```

Варіанти завдань

Виконати завдання свого варіанту Лабораторної роботи № 1.7, використовуючи конструкцію вкладеного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Лабораторна робота № 1.7:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково мають бути реалізовані наступні методи:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання мають бути реалізовані як класи із закритими полями, де операції реалізуються як методи класу.

Визначення кожного класу та реалізацію його методів слід розмістити в окремих модулях.

Для демонстрації роботи з об'єктами нового типу у всіх завданнях потрібно написати головну функцію. У програмі обов'язково мають бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма має демонструвати використання всіх функцій і методів.

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є «контейнером», всі решту – описують поля, які містяться в «контейнері». Класи, що описують поля класу-«контейнера», мають бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 2.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,

- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 3.

Реалізувати клас Calculator з повним набором арифметичних операцій, використовуючи клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 4.

Реалізувати клас Bankomat:

Клас Bankomat, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається: вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод toString( ) має перетворити у літерний рядок суму грошей, що залишилася в

банкоматі.

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 5*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу DigitString,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас DigitString, а для представлення дробової частини без-знакове коротке ціле.

Клас DigitString – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 6*.

Реалізувати клас Calculator з повним набором арифметичних операцій, на основі класу Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- * додавання,
- * віднімання,
- * множення,
- операції порівняння.

Для представлення цілої частини використовувати клас LongLong, а для

представлення дробової частини додатне дробове число типу double:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 7.

Реалізувати клас Triangle:

Клас Triangle – для представлення трикутника. Поля даних:

- *a*,
- *b*,
- *c* – сторони;
- *A*,
- *B*,
- *C* – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас Angle:

Клас Angle – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов’язково мають бути реалізовані:

- переведення в радіани,
- приведення до діапазону 0° – 360°,

- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

Варіант 8.

Реалізувати клас Goods:

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення терміну придатності.

Варіант 9.

Реалізувати клас Triangle з полями – координатами вершин:

Клас Triangle – для представлення трикутника. Поля даних:

- P_1,
- P_2,
- P_3 – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- *get_a( ),*
- *get_b( ),*
- *get_c( )* – обчислення довжин сторін;
- *get_A( ),*
- *get_B( ),*
- *get_C( )* – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас Point:

Клас Point – для роботи з точками на площині. Координати точки – декартові. Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

Варіант 10.

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім’я-побаткові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас Money:

Клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 11.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 12.

Реалізувати клас ModelWindow, додавши поле для курсору:

Створити клас ModelWindow для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,
- колір вікна,
- стан «видиме / невидиме»,

- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об’єкту.

Для представлення поля курсору використовуйте клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Варіант 13*.

Реалізувати клас Set (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас BitString:

Клас BitString – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу unsigned long. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- and,
- or,
- xor,
- not.

- * зсув ліворуч shiftLeft та
- * зсув праворуч shiftRight на задану кількість бітів.

Клас Set (множина) має забезпечувати операції:

- включення елементу в множину,
- виключення елементу з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

Варіант 14*.

Реалізувати клас Rational:

Раціональний (нескоротний) дріб представляється парою цілих чисел (a, b), де поля:

- a – чисельник,
- b – знаменник.

Клас Rational – для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні операції:

Припустимо, що (a, b) – перше число = $^{a}/_{b}$ – перший об'єкт; (c, d) – друге число = $^{c}/_{d}$ – другий об'єкт.

- * додавання add(), (a, b) + (c, d) = ($ad + bc$, bd) = $^{(ad + bc)}/_{(bd)}$;
- * віднімання sub(), (a, b) – (c, d) = ($ad – bc$, bd);
- * множення mul(), (a, b) × (c, d) = (ac, bd);
- * ділення div(), (a, b) / (c, d) = (ad, bc);
- порівняння equal(), great(), less().

Має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 15*.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- * додавання,
- * віднімання,
- * ділення сум,
- * ділення суми на дробове число,
- * множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 16*.

Реалізувати клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- * зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас LongLong:

Клас LongLong для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 17.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев’ятсот дев’яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

Варіант 18.*

Реалізувати клас Goods, додавши поле – дату надходження товару на склад.

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.

- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Використовувати клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

Варіант 19.*

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу Date.

Варіант 20.*

Реалізувати клас Bill, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас Time:

Клас Time – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- * обчислення різниці між двома моментами часу в секундах,
- * додавання часу і заданої кількості секунд,
- * віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод toString( ) має видавати суму в гривнях.

Варіант 21.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,

- номер рахунку,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Варіант 22.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути чотири поля:

- прізвище власника,
- номер рахунку,
- відсоток нарахування і

- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28 → «одна тисяча дев'ятсот дев'яносто дві грн. 28 коп.»).

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою.

Варіант 23.

Реалізувати клас Calculator з повним набором арифметичних операцій, використовуючи клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 24.

Реалізувати клас Bankomat:

Клас Bankomat, – моделює роботу банкомату. У класі мають міститися поля для зберігання:

- ідентифікаційного номера банкомату,
- інформації про поточну суму грошей, що залишилася у банкоматі,
- мінімальній і
- максимальній сумах, які дозволяється зняти клієнтові в один день.

Реалізувати:

- метод ініціалізації банкомату,
- метод завантаження купюр в банкомат,
- метод зняття певної суми грошей.

Метод зняття грошей має виконувати перевірку на коректність суми, що знімається:

вона не може бути менше мінімального значення і не може перевищувати максимальне значення.

- метод toString( ) має перетворити у літерний рядок суму грошей, що залишилася в банкоматі.

Для представлення суми використовувати клас Money:

Номінали гривень можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500. Копійки представити як 0.01 (1 копійка), 0.02 (2 копійки), 0.05 (5 копійок), 0.1 (10 копійок), 0.25 (25 копійок), 0.5 (50 копійок). Створити клас Money для роботи з грошовими сумами. Сума має бути представлена полями-номіналами, значеннями яких має бути кількість купюр відповідного номіналу. Поля:

- кількість банкнот по 500 грн.
- кількість банкнот по 200 грн.
- кількість банкнот по 100 грн.
- кількість банкнот по 50 грн.
- кількість банкнот по 20 грн.
- кількість банкнот по 10 грн.
- кількість банкнот по 5 грн.
- кількість банкнот по 2 грн.
- кількість банкнот по 1 грн.
- кількість монет по 50 коп.
- кількість монет по 25 коп.
- кількість монет по 10 коп.
- кількість монет по 5 коп.
- кількість монет по 2 коп.
- кількість монет по 1 коп.

Реалізувати:

- додавання сум,
- віднімання сум,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої

частини комою.

Варіант 25*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – класу DigitString,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Для представлення цілої частини використовувати клас DigitString, а для представлення дробової частини без-знакове коротке ціле.

Клас DigitString – для роботи з цілими числами. Число має бути представлене символами-цифрами, які утворюють літерний рядок.

Мають бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 26*.

Реалізувати клас Calculator з повним набором арифметичних операцій, на основі класу Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Для представлення цілої частини використовувати клас LongLong, а для представлення дробової частини додатне дробове число типу double:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 27.

Реалізувати клас Triangle:

Клас Triangle – для представлення трикутника. Поля даних:

- *a*,
- *b*,
- *c* – сторони;
- *A*,
- *B*,
- *C* – протилежні кути.

– мають включати кути і сторони. Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).

Для представлення кутів використовувати клас Angle:

Клас Angle – для роботи з кутами на площині, що задаються величиною в градусах і хвилинах. Поля:

- *grades*
- *minutes*

Обов’язково мають бути реалізовані:

- переведення в радіани,

- приведення до діапазону 0° – 360°,
- збільшення кута на задану величину,
- зменшення кута на задану величину,
- отримання синуса,
- порівняння кутів.

Варіант 28.

Реалізувати клас Goods:

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної, по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,
- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Реалізувати метод уцінки товару, зменшуючи ціну на 1% за кожен день прострочення

терміну придатності.

Варіант 29.

Реалізувати клас Triangle з полями – координатами вершин:

Клас Triangle – для представлення трикутника. Поля даних:

- P_1,
- P_2,
- P_3 – точки (вершини трикутника),

Потрібно реалізувати операції:

- отримання полів даних,
- зміни полів даних,
- обчислення площі,
- обчислення периметру,
- обчислення висот,
- визначення виду трикутника (рівносторонній, рівнобедрений або прямокутний).
- *get_a( ),*
- *get_b( ),*
- *get_c( )* – обчислення довжин сторін;
- *get_A( ),*
- *get_B( ),*
- *get_C( )* – обчислення величин протилежних кутів.

Для представлення координат вершин використовуйте клас Point:

Клас Point – для роботи з точками на площині. Координати точки – декартові. Поля:

- *x*
- *y*

Обов’язково мають бути реалізовані:

- переміщення точки по осі X,
- переміщення по осі Y,
- визначення відстані до початку координат,
- відстані між двома точками,
- перетворення у полярні координати,
- порівняння на рівність та нерівність.

Варіант 30.

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім’я-побаткові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,
- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Для представлення полів нарахувань і утримань використовувати клас Money:

Клас Money для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,

- множення на дробове число,
- операції порівняння.

Варіант 31.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене вкладеним об'єктом класу Fraction.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- додавання,
- віднімання,
- ділення сум,
- ділення суми на дробове число,
- множення на дробове число,
- операції порівняння.

Для представлення величини грошової суми використовувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число має бути представлене двома полями:

- ціла частина – довге ціле із знаком,
- дробова частина – без-знакове коротке ціле.

Реалізувати арифметичні операції:

- додавання,
- віднімання,
- множення,
- операції порівняння.

Варіант 32.

Реалізувати клас ModelWindow, додавши поле для курсору:

Створити клас ModelWindow для роботи з моделями екранних вікон. В якості полів задаються:

- заголовок вікна,
- координати лівого верхнього кута,
- розмір по горизонталі,
- розмір по вертикалі,

- колір вікна,
- стан «видиме / невидиме»,
- стан «з рамкою / без рамки».

Координати і розміри вказуються в цілих числах. Реалізувати операції:

- пересування вікна по горизонталі,
- пересування вікна по вертикалі;
- зміна висоти і/або ширини вікна;
- зміна кольору;
- встановлення стану,
- отримання значення стану.

Операції пересування і зміни розміру мають здійснювати перевірку на перетин меж екрану. Функція виводу на екран має змінювати стан полів об’єкту.

Для представлення поля курсору використовуйте клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,
- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Варіант 33*.

Реалізувати клас Set (множина) не більше ніж з 64 елементів цілих чисел, використовуючи клас BitString:

Клас BitString – для роботи з 64-бітовими рядками. Бітовий рядок має бути представлений двома полями типу unsigned long. Мають бути реалізовані всі традиційні операції для роботи з бітами:

- and,
- or,

- xor,
- not.
- * зсув ліворуч shiftLeft та
- * зсув праворуч shiftRight на задану кількість бітів.

Клас Set (множина) має забезпечувати операції:

- включення елементу в множину,
- виключення елементу з множини,
- об'єднання,
- перетин і
- різницю множин,
- обчислення кількості елементів в множині.

Варіант 34*.

Реалізувати клас Rational:

Раціональний (нескоротний) дріб представляється парою цілих чисел (a, b), де поля:

- a – чисельник,
- b – знаменник.

Клас Rational – для роботи з раціональними дробами. Обов'язково мають бути реалізовані наступні операції:

Припустимо, що (a, b) – перше число = $^{a}/_{b}$ – перший об'єкт; (c, d) – друге число = $^{c}/_{d}$ – другий об'єкт.

- * додавання add(), (a, b) + (c, d) = ($ad + bc$, bd) = $^{(ad + bc)}/_{(bd)}$;
- * віднімання sub(), (a, b) – (c, d) = ($ad – bc$, bd);
- * множення mul(), (a, b) × (c, d) = (ac, bd);
- * ділення div(), (a, b) / (c, d) = (ad, bc);
- порівняння equal(), great(), less().

Має бути реалізована приватна функція скорочення дробу Reduce(), яка обов'язково викликається при виконанні арифметичних операцій.

Для представлення чисельника і знаменника використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,

- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 35*.

Реалізувати клас Money:

Клас Money – для роботи з грошовими сумами. Число має бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран має бути відокремлена від цілої частини комою. Реалізувати операції:

- * додавання,
- * віднімання,
- * ділення сум,
- * ділення суми на дробове число,
- * множення на дробове число,
- операції порівняння.

Для представлення гривень використовувати клас LongLong:

Клас LongLong – для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 36*.

Реалізувати клас Cursor:

Клас Cursor. Полями є:

- *x*
- *y* – координати курсору по горизонталі і вертикалі – цілі додатні числа,
- вид курсору – горизонтальний або вертикальний,

- розмір курсору – ціле число від 1 до 15.

Реалізувати методи:

- * зміни координат курсору,
- зміни виду курсору,
- зміни розміру курсору,
- метод гасіння і
- метод відновлення курсору.

Для представлення координат використовувати клас LongLong:

Клас LongLong для роботи з 64 бітовими цілими числами. Число має бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Мають бути реалізовані:

- * всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 37.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі мають бути поля:

- прізвище власника,
- номер рахунку,
- дата відкриття,
- відсоток нарахування і
- сума в гривнях.

Відкриття нового рахунку виконується операцією ініціалізації. Необхідно виконувати наступні операції:

- змінити власника рахунку,
- зняти деяку суму грошей з рахунку,
- покласти гроші на рахунок,
- нарахувати відсотки,
- перевести суму в долари,
- перевести суму в євро,
- отримати суму прописом (число перетворити у літерний рядок, наприклад 1992,28

→ «одна тисяча дев’ятсот дев’яносто дві грн. 28 коп.»).

Додати поле – дату відкриття рахунку, використовуючи клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Додати метод, що обчислює кількість днів, що пройшли з початку відкриття рахунку, і що додає по 0,01 % до відсотку нарахування за кожен день.

Варіант 38.*

Реалізувати клас Goods, додавши поле – дату надходження товару на склад.

Клас Goods – товари. У класі мають бути представлені поля:

- найменування товару,
- дата оформлення,
- ціна товару,
- кількість одиниць товару,
- номер накладної,
- по якій товар поступив на склад.

Реалізувати методи:

- зміни ціни товару,

- зміни кількості товару (збільшення і зменшення),
- обчислення вартості товару.
- Метод toString( ) має повертати у вигляді літерного рядка вартість товару.

Використовувати клас Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов'язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Реалізувати метод, що обчислює термін зберігання товару.

Варіант 39.*

Реалізувати клас Payment:

Клас Payment – зарплата. У класі мають бути представлені поля:

- прізвище-ім'я-побатькові,
- ставка,
- рік поступлення на роботу,
- відсоток надбавки,
- прибутковий податок,
- кількість відпрацьованих днів в місяці,
- кількість робочих днів в місяці,

- нарахована і
- утримана суми.

Реалізувати методи:

- обчислення нарахованої суми,
- обчислення утриманої суми,
- обчислення суми, що видається на руки,
- обчислення стажу.

Стаж обчислюється як повна кількість років, що пройшли від року прийому на роботу, до поточного року. Нарахування є сумою, нарахованою за відпрацьовані дні, і надбавки, тобто долі від першої суми. Утриманнями є відрахування до пенсійного фонду (1% від нарахованої суми) і прибутковий податок. Прибутковий податок складає 13% від нарахованої суми без відрахувань в пенсійний фонд.

Замість поля-року використовувати поле-дату класу Date:

Клас Date – для роботи з датами у форматі «рік.місяць.день» з трьома полями типу unsigned int:

- рік,
- місяць і
- номер дня.

Клас має включати не менше трьох функцій ініціалізації:

- числами,
- літерним рядком виду «рік.місяць.день» (наприклад, «2004.08.31») і
- датою.

Обов’язковими операціями є:

- * обчислення дати через задану кількість днів,
- * віднімання заданої кількості днів з дати,
- визначення, чи рік – високосний,
- присвоєння,
- отримання окремих частин (рік, місяць, день),
- порівняння дат (рівно, до, після),
- * обчислення кількості днів між датами.

Стаж слід обчислювати, використовуючи методи класу Date.

Варіант 40.*

Реалізувати клас Bill, що є разовим платежем за телефонну розмову. Клас має включати поля:

- прізвище платника,
- номер телефону,
- тариф за хвилину розмови,
- знижка (у відсотках),
- час початку розмови,
- час закінчення розмови,
- сума до оплати.

Для представлення часу використовуйте клас Time:

Клас Time – для роботи з часом у форматі «година:хвилина:секунда» з трьома полями типу unsigned int:

- година,
- хвилина і
- секунда.

Клас має включати не менше чотирьох функцій ініціалізації:

- числами,
- літерним рядком (наприклад, «23:59:59»),
- секундами від початку доби і
- часом.

Реалізувати методи:

- * обчислення різниці між двома моментами часу в секундах,
- * додавання часу і заданої кількості секунд,
- * віднімання з часу заданої кількості секунд,
- порівняння моментів часу,
- переведення в секунди,
- переведення в хвилини (з округленням до цілої хвилини);
- отримання і зміни значень полів. Час розмови, який підлягає оплаті, обчислюється в хвилинах; неповна хвилина вважається за повну;
- метод toString( ) має видавати суму в гривнях.

Питання та завдання для контролю знань

Класи та об’єкти

1. Поняття класів та об’єктів.
2. Загальний синтаксис опису класів.
3. Визначення та оголошення класу.
4. Чим відрізняється клас від об’єкту?
5. Позначення класів та об’єктів на UML-діаграмах класів.
6. Чи можна оголошувати масив об’єктів?
7. Чи можна оголошувати масив класів?
8. Чи дозволяється оголошувати вказівник на об’єкт?
9. Чи дозволяється оголошувати вказівник на клас?
10. Чи можна сумістити визначення класу з оголошенням об’єкту?
11. Поясніть різницю між визначенням класу і оголошенням класу.
12. Поясніть, чим відрізняються наступні оголошення та ініціалізація вказівника на об’єкт:

```
TClass *p = new TClass;
TClass *p = new TClass();
```

13. Чи є структура класом?
14. Чим клас відрізняється від структури?
15. Які ключові слова в C++ позначають клас?

Елементи класу: поля та методи. Доступ до елементів класу

16. Елементи класу: поля та методи.
17. Інкапсуляція[1]: об’єднання опису даних та опису дій над даними в єдине ціле.
18. Роль вказівника this в реалізації інкапсуляції[1].
19. Інкапсуляція[2]: обмеження доступу до даних.
20. Директиви доступу (видимості) елементів класу.
21. Звертання до полів та методів класу.
22. Позначення елементів класів на UML-діаграмах класів.
23. Позначення доступних та приватних елементів класів на UML-діаграмах класів.
24. Поясніть принцип інкапсуляції.
25. Для чого потрібні ключові слова public і private?
26. Чи можна використовувати ключові слова public і private в структурі?
27. Чи існують обмеження на використання ключових слів public і private в класі?
28. Чи існують обмеження на використання ключових слів public і private в структурі?

29. Чи обов’язково робити поля класу приватними? Як ініціалізувати приватні поля класу?
30. Що таке «метод»?
31. Як викликається метод?
32. Чи може метод бути приватним?
33. Як визначити метод безпосередньо всередині класу?
34. Як визначити метод поза класом?
35. Поясніть, що розуміється під інтерфейсом класу.
36. Що позначається ключовим словом this?
37. Для чого може використовуватися конструкція *this?
38. Чи дозволяється всередині методу оголошувати об’єкти «свого» класу?
39. Як присвоювати об’єктам, оголошеним всередині методу «свого» класу початкове значення?
40. Чи дозволяється параметрам методів присвоювати значення за умовчанням?
41. Поясніть, чому звичайні методи, що реалізують бінарні операції (наприклад, додавання), мають мати один параметр.
42. Який принцип об’єктно-орієнтованого програмування проявляється в перевантаженні методів?
43. Передавання об’єктів у функції та повернення об’єктів в якості результату.
44. Визначити клас C, який містить поле x цілого типу та метод Twice, який подвоює значення поля.
45. Визначити клас C, який містить закрите поле x цілого типу та відкриті методи зчитування / запису значення поля.
46. Скільки місця в пам’яті займає об’єкт класу? Як це взнати?
47. Який розмір «порожнього» об’єкту?
48. Чи впливають методи на розмір об’єкту?
49. Чи однаковий розмір класу і аналогічної структури?
50. Що таке вирівнювання і від чого воно залежить?
51. Чи впливає вирівнювання на розмір класу?
52. Покажіть, як здійснити вирівнювання полів класу по межі двох байтів.
53. Поясніть призначення директиви #pragma pack.

Вкладені класи

54. Поняття вкладеного класу.
55. Визначення вкладеного класу всередині класу-контейнера.
56. Визначення вкладеного класу поза класом-контейнером.
57. Доступ до елементів вкладеного класу із методів класу-контейнера.
58. Доступ до елементів класу-контейнера із вкладеного класу.

Дружні функції та дружні класи

59. Поняття та визначення дружньої функції.
60. Поняття та визначення дружнього класу.
61. Інтерфейс класу. Розширення інтерфейсу класу за допомогою дружніх функцій та дружніх класів.
62. Дано визначення класу:

```
class C1 {
   .......
};
```

Визначити клас C2, який буде мати доступ до якостей і характеристик класу C1; при цьому слід використовувати зовнішні дружні класи.

63. Дано визначення класу:

```
class C1 {
   .......
};
```

Визначити клас C2, який буде мати доступ до якостей і характеристик класу C1; при цьому слід використовувати вкладені дружні класи.

Композиція та агрегування

64. Як називається використання об’єкту одного класу в якості поля другого класу?
65. Що таке композиція?
66. Поняття агрегування та композиції – в термінах предметної області.
67. Реалізація агрегування та композиції – в термінах програмування.
68. Позначення агрегування та композиції на UML-діаграмах класів.
69. Дано визначення класу:

```
class C1 {
   .......
};
```

Визначити клас C2, який буде мати доступ до відкритих якостей і характеристик класу C1; при цьому слід використовувати композицію: клас C1 входить до складу класу C2.

70. Дано визначення класу:

```
class C1 {
   .......
};
```

Визначити клас C2, який буде мати доступ до відкритих якостей і характеристик класу C1; при цьому слід використовувати агрегування: клас C1 входить до складу класу C2.

Умова для завдань №№ 71-72

Дано визначення класів:

```
class C1 {
};

class C2 {
          C1  f1; // агреговане поле об’єктового типу
};
```

71. Нарисувати UML-діаграму класів C1 та C2.

72. Зв’язок між класами C1 та C2 називається: (відмітити правильні відповіді)

А) композиція
Б) успадковування
В) агрегування
Г) поліморфізм
Д) інкапсуляція
Е) інша відповідь:_____

Умова для завдань №№ 73-74

Дано визначення класів:

```
class C1 {
};

class C2 {
          C1 *f1; // агреговане поле об’єктового типу
};
```

73. Нарисувати UML-діаграму класів C1 та C2.

74. Зв’язок між класами C1 та C2 називається: (відмітити правильні відповіді)

А) композиція
Б) успадковування
В) агрегування
Г) поліморфізм
Д) інкапсуляція
Е) інша відповідь:_____

75. Дано визначення класів:

```
class C1 {
          C2 *f2; // агреговане поле об’єктового типу
};

class C2 {
          C1 *f1; // агреговане поле об’єктового типу
};
```

Виправити помилку у визначенні цих класів (потрібно зберегти структуру об’єктів). Відповідь пояснити.

Предметний покажчик

Література

Основна

1. Павловская Т.А. С/С++. Программирование на языке высокого уровня СПб.: Питер, 2007. – 461 с.
2. Павловская Т.А., Щупак Ю.А. С/С++. Объектно-ориентированное программирование: Практикум СПб.: Питер, 2005. – 265 с.
3. Дейтел Х.М., Дейтел П.Дж. Как программировать на С++ М.: Бином-Пресс, 2005. – 1248 с.
4. Уэллин С. Как не надо программировать на С++ СПб.: Питер, 2004. – 240 с.
5. Хортон А. Visual C++ 2005: Базовый курс М.: Вильямс, 2007. – 1152 с.
6. Солтер Н.А., Клеппер С.Дж. С++ для профессионалов. М.: Вильямс, 2006. – 912 с.
7. Лафоре Р. Объектно-ориентированное программирование в С++ СПб.: Питер, 2006. – 928 с.
8. Лаптев В.В. С++. Объектно-ориентированное программирование СПБ.: Питер, 2008. – 464 с.
9. Лаптев В.В., Морозов А.В., Бокова А.В. С++. Объектно-ориентированное программирование. Задачи и упражнения СПБ.: Питер, 2008. – 464 с.

Додаткова

10. Прата С. Язык программирования С++. Лекции и упражнения СПб.: ДиаСофт, 2003. – 1104 с.
11. Мейн М., Савитч У. Структуры данных и другие объекты в С++ М.: Вильямс, 2002. – 832 с.
12. Саттер Г. Решение сложных задач на С++ М.: Вильямс, 2003. – 400 с.
13. Чепмен Д. Освой самостоятельно Visual C++ .NET за 21 день М: Вильямс, 2002. – 720 с.
14. Мартынов Н.Н. Программирование для Windows на С/С++ М.: Бином-Пресс, 2004. – 528 с.
15. Паппас К., Мюррей У. Эффективная работа: Visual C++ .NET СПб.: Питер, 2002. – 816 с. М.: Вильямс, 2001. – 832 с.
16. Грэхем И. Объектно-ориентированные методы. Принципы и практика М.: Вильямс, 2004. – 880 с.
17. Элиенс А. Принципы объектно-ориентированной разработки программ М.: Вильямс, 2002. – 496 с.

18. Ларман К. Применение UML и шаблонов проектирования М.: Вильямс, 2002. – 624 с.
19. Шилэт Г. Полный справочник по C. 4-е издание. М.-СПб.-К: Вильямс, 2002.
20. Прата С. Язык программирования C. Лекции и упражнения. М.: ДиаСофтЮП, 2002.
21. Александреску А. Современное проектирование на C++ М.: Вильямс, 2002.
22. Браунси К. Основные концепции структур данных и реализация в C++ М.: Вильямс, 2002.
23. Подбельский В.В. Язык СИ++. Учебное пособие. М.: Финансы и статистика, 2003.
24. Павловская Т.А., Щупак Ю.А. C/C++. Программирование на языке высокого уровня. СПб.: Питер, 2002.
25. Савитч У. Язык C++. Объектно-ориентированного программирования. М.-СПб.-К.: Вильямс, 2001.