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

Григорович Віктор

Об’єктно-орієнтоване програмування
Конструктори та перевантаження операцій

Навчальний посібник

2021

Григорович Віктор Геннадійович

Об’єктно-орієнтоване програмування. Конструктори та перевантаження операцій. 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 «Конструктори та перевантаження операцій».

Розглядаються наступні роботи лабораторного практикуму:

Лабораторна робота № 2.1.
Конструктори та перевантаження операцій для класів з двома полями

Лабораторна робота № 2.2.
Перевантаження операцій

Лабораторна робота № 2.3.
Конструктори та перевантаження операцій для класів

Лабораторна робота № 2.4.
Масиви та константи в класі

Лабораторна робота № 2.5.
Конструктори та перевантаження операцій для класів з композицією

Лабораторна робота № 2.6.
Конструктори та перевантаження операцій для класів з вкладеними класами. Обчислення кількості об’єктів

Лабораторна робота № 2.7.
Конструктори та перевантаження операцій для класів з композицією – складніші завдання

Лабораторна робота № 2.8.
Конструктори та перевантаження операцій для класів з вкладеними класами – складніші завдання. Обчислення кількості об’єктів

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 «Конструктори та перевантаження операцій».

Тема 2. Конструктори та перевантаження операцій

Стисло та головне про конструктори та перевантаження операцій

Конструктори та деструктори

Конструктори

При створенні нового типу даних бажано оголошувати об’єкти цього нового типу аналогічно вбудованим – форми ініціалізації об’єктів нових типів не мають відрізнятися від форм ініціалізації об’єктів вбудованих типів. Таку можливість забезпечують конструктори.

Призначення та особливості конструкторів

Конструктор – це особливий метод, ім’я якого збігається з іменем класу.

Конструктори призначені для створення та ініціалізації об’єктів.

Конструктор може мати будь-яку кількість аргументів будь-якого типу, конструктор може зовсім не мати аргументів; аргументи можуть мати значення за умовчанням; аргументи конструктора зазвичай використовуються для ініціалізації полів об’єкта. Конструктор не повертає результату – не можна писати навіть `void`.

Основні властивості конструкторів:

- Конструктор *не повертає значення*, навіть типу `void`.
- *Не можна отримати вказівника* на конструктор.
- Клас може мати *кілька конструкторів* з різними параметрами для різних способів ініціалізації об’єкта (при цьому використовується механізм перевантаження методів).
- Конструктор, який не має параметрів, називається *конструктором за умовчанням*.
- *Параметри конструктора* можуть бути будь-якого типу, крім того самого класу. Можна задавати значення параметрів за умовчанням, проте їх може містити лише один із конструкторів.
- Якщо програміст не вказав *жодного конструктора*, то компілятор створює його *автоматично*. Такий конструктор викликає конструктори за умовчанням для полів класу та конструктори за умовчанням базових класів (див. тему

Успадковування). У випадку, коли клас містить константи чи посилання, то при спробі створити об'єкт класу буде виведене повідомлення про помилку, оскільки їх необхідно ініціалізувати конкретними значеннями, а конструктор за умовчанням цього робити не вміє.

- *Конструктори не успадковуються*.
- Конструктори не можна описати з модифікаторами `const`, `virtual` та `static`.
- Конструктори глобальних об'єктів викликаються до виклику функції `main()`. Локальні об'єкти створюються, як тільки стає активною область їх дії – відповідний блок. Конструктор викликається і при створенні тимчасового об'єкту (наприклад, при передаванні об'єкту із функції).
- Конструктори викликаються, якщо в програмі є хоч одна із наступних синтаксичних конструкцій:

```
ім'я_класу ім'я_об'єкту;
ім'я_класу ім'я_об'єкту( список_параметрів );
// список параметрів не має бути порожнім

ім'я_класу( список_параметрів );
// створюється безім'янний об'єкт, список параметрів може бути порожнім

ім'я_класу ім'я_об'єкту = вираз;
// створюється безім'янний об'єкт і копіюється у вказаний
```

Конструктор за умовчанням та конструктор ініціалізації

За відсутності в класі явно визначених конструкторів автоматично створюється конструктор без аргументів (конструктор за умовчанням) та конструктор копіювання. Конструктор за умовчанням, який створюється автоматично, має вигляд:

```
клас::клас() {}
```

Якщо явно визначити хоча би один конструктор, який приймає параметр (тобто, хоча би один конструктор ініціалізації), то конструктор за умовчанням автоматично створюватися не буде, його потрібно написати явно. При цьому можна використовувати механізм параметрів із значеннями за умовчанням.

Будь-який конструктор з параметрами не свого класу є конструктором ініціалізації, який призначений для ініціалізації полів класу.

Конструктор ініціалізації, в якого всі параметри мають значення за умовчанням, відіграє роль конструктора без аргументів (конструктора за умовчанням).

Конструктор копіювання

Конструктор копіювання автоматично створюється у вигляді:

```
клас::клас(const клас &r)
{
    *this = r;
}
```

Аргументом конструктора копіювання завжди є об’єкт цього самого класу.

Конструктор копіювання створюється автоматично завжди (навіть якщо визначено конструктори ініціалізації), якщо він не визначений явно.

При явному визначенні конструктора копіювання аргумент має передаватися за посиланням – передавати його за значенням не можна, оскільки при передаванні за значенням виконується копіювання, тобто дія, яку власне і має виконати конструктор копіювання.

В класі може бути кілька конструкторів копіювання, можна визначати кілька аргументів, якщо їм присвоїти значення за умовчанням.

Конструктори та параметри

Конструктор копіювання викликається неявно при передаванні у функцію параметра-об’єкта за значенням і при поверненні значення об’єкта із функції. При інших способах передачі параметру конструктор копіювання не викликається.

Конструктор копіювання викликається для створення прихованої «внутрішньої» копії об’єктів. При передаванні виразу в якості параметру може викликатися конструктор ініціалізації.

Виклик функцій без аргументів у всіх випадках супроводжується викликом конструктора ініціалізації.

Деструктори

Призначення та особливості деструкторів

Деструктори призначені для знищення об’єктів, тобто деструктори – це особливі методи, які використовуються для звільнення пам’яті, виділеної для об’єкта.

Деструктор викликається автоматично, коли об’єкт виходить за межі області видимості:

- для *локальних* об’єктів – при виході із блоку, в якому вони оголошені;
- для *глобальних* – як частина процедури виходу із `main()`;
- для *динамічних об’єктів, визначених через вказівники*, деструктор викликається неявно при використанні операції `delete`.

Автоматичний виклик деструктора об’єкту при виході із області дії вказівника на нього – не виконується!

Особливості деструктора:

- Ім’я деструктора починається з тильди (~), одразу за якою записується ім’я класу.
- Не має аргументів і не повертає результату (навіть `void` вказувати не можна).
- Не може бути оголошений як `const` або `static`.
- Не успадковується.
- Може бути віртуальним.
- Не можна визначити вказівника на деструктор.

Опис деструкторів

Деструктор – це особливий метод, ім’я якого збігається з іменем класу, з єдиною різницею: його першим символом має бути символ ~ (тильда). Деструктор не повертає результату та не приймає аргументів. Деструктор в класі може бути лише один – перевантаження деструкторів заборонене. За відсутності в класі явно визначеного деструктора він створюється автоматично. Деструктор, який створюється автоматично, має такий вигляд:

```
клас::~клас() {}
```

Якщо конструктори класу не вимагають розподілу ресурсів, які необхідно звільняти (наприклад, динамічну пам’ять), або не виконують дій щодо ініціалізації, які при знищенні об’єкта вимагають завершальних дій (наприклад, якщо файл відкривається, то його необхідно закрити), то деструктор можна не створювати.

Перевантаження операцій

В C++ можна перевантажувати вбудовані операції – це один із проявів *поліморфізму*. Перевантаження операцій не є обов'язковою в об'єктно-орієнтованому програмуванні – в мові java воно відсутнє. Проте наявність перевантаження операцій в C++ забезпечує додатковий рівень зручності при використанні нових типів даних.

Особливості перевантаження операцій

При перевантажені операцій слід враховувати наступні обмеження:

1. Заборонено перевантажувати наступні операції:
   - `sizeof()` – визначення розміру аргументу;
   - `.` (крапка) – селектор компонента (поля чи метода) об'єкта;
   - `?:` – умовна операція;
   - `::` – операція доступу до області дії;
   - `.*` – вибір компонента класу через вказівник;
   - `#` та `##` – операції препроцесора.
2. Операції можна перевантажувати лише для нового типу даних – не можна перевантажити операцію для вбудованого типу. В C++ новий тип можна утворити за допомогою конструкцій `enum`, `union`, `struct` та `class`.
3. Не можна змінити пріоритет перевантаженої операції та кількість операндів. Унарна операція має мати один операнд (за винятком перевантаження операцій інкременту `++` та декременту `--`), бінарна – два операнди; не можна використовувати параметри за умовчанням. Єдина операція, яка не має фіксованої кількості операндів, – це операція виклику функції `()`. Операції `+`, `-`, `*`, `&` можна перевантажувати як унарні, так і бінарні.
4. Операції можна перевантажувати або як незалежні зовнішні функції (лише такий спосіб допустимий для `enum`), або як методи класу.

   Чотири операції:

   - `=` – присвоєння;
   - `()` – виклику функції;
   - `[]` – індексування;
   - `->` – доступу за вказівником

   можна перевантажувати лише як методи класу. Ці операції в принципі не можна перевантажити для `enum`.

5. Якщо операція перевантажується як метод класу, то лівим (або єдиним) операндом обов’язково буде об’єкт класу, для якого перевантажується операція.

Рекомендації щодо перевантаження операцій

Форми перевантаження операцій:

Операція	**Рекомендований спосіб перевантаження**
Всі унарні операції	Метод класу
`=  []  ()  ->  ->*`	Обов’язково метод класу
`+=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=`	Метод класу
Решта бінарних операцій	Зовнішня дружня функція

Теоретичні відомості

Конструктори та деструктори

Конструктори

При створенні нового типу даних бажано оголошувати об’єкти цього нового типу аналогічно вбудованим – форми ініціалізації об’єктів нових типів не мають відрізнятися від форм ініціалізації об’єктів вбудованих типів. Таку можливість забезпечують конструктори.

Призначення та особливості конструкторів

Конструктор – це особливий метод, ім’я якого збігається з іменем класу.

Конструктори призначені для створення та ініціалізації об’єктів.

Конструктор може мати будь-яку кількість аргументів будь-якого типу, конструктор може зовсім не мати аргументів; аргументи можуть мати значення за умовчанням; аргументи конструктора зазвичай використовуються для ініціалізації полів об’єкта. Конструктор не повертає результату – не можна писати навіть `void`.

Якщо конструктор реалізується поза визначенням класу, то (як і для інших методів) його ім’я має записуватися з префіксом, який уточнює область дії класу:

```
class A
{
    /* ... */           // поля

public:                 // зазвичай конструктор - доступний
    A(/* параметри */); // оголошення конструктора

    /* ... */           // інші методи
};

A::A(/* параметри */)   // зовнішнє визначення конструктора
{
    /* ... */           // тіло конструктора
}
```

Основні властивості конструкторів:

- Конструктор *не повертає значення*, навіть типу `void`.
- *Не можна отримати вказівника* на конструктор.
- Клас може мати *кілька конструкторів* з різними параметрами для різних способів ініціалізації об’єкта (при цьому використовується механізм перевантаження методів).
- Конструктор, який не має параметрів, називається *конструктором за умовчуванням*.

- *Параметри конструктора* можуть бути будь-якого типу, крім того самого класу. Можна задавати значення параметрів за умовчанням, проте їх може містити лише один із конструкторів.
- Якщо програміст не вказав *жодного конструктора*, то компілятор створює його *автоматично*. Такий конструктор викликає конструктори за умовчанням для полів класу та конструктори за умовчанням базових класів (див. тему Успадковування). У випадку, коли клас містить константи чи посилання, то при спробі створити об'єкт класу буде виведене повідомлення про помилку, оскільки їх необхідно ініціалізувати конкретними значеннями, а конструктор за умовчанням цього робити не вміє.
- *Конструктори не успадковуються*.
- Конструктори не можна описати з модифікаторами `const`, `virtual` та `static`.
- Конструктори глобальних об'єктів викликаються до виклику функції `main()`. Локальні об'єкти створюються, як тільки стає активною область їх дії – відповідний блок. Конструктор викликається і при створенні тимчасового об'єкту (наприклад, при передаванні об'єкту із функції).
- Конструктори викликаються, якщо в програмі є хоч одна із наступних синтаксичних конструкцій:

```
ім'я_класу ім'я_об'єкту;
ім'я_класу ім'я_об'єкту( список_параметрів );
// список параметрів не має бути порожнім

ім'я_класу( список_параметрів );
// створюється безім'янний об'єкт, список параметрів може бути порожнім

ім'я_класу ім'я_об'єкту = вираз;
// створюється безім'янний об'єкт і копіюється у вказаний
```

Конструктори, як і звичайні методи можна перевантажувати. Зазвичай розрізняють три види конструкторів:

- конструктор без аргументів (конструктор за умовчанням);
- конструктор ініціалізації;
- конструктор копіювання (з одним параметром типу посилання на екземпляр класу).

Розглянемо приклад:

```
class Account
{
    char* name;                 // власник
    double summa;               // сума

public:
    Account();                  // конструктор за умовчанням
    Account(char*);             // конструктор ініціалізації
```

```
    Account(double);           // конструктор ініціалізації
    Account(char*, double);    // конструктор ініціалізації
    Account(Account&);         // конструктор копіювання

    /* ... */                  // інші методи
};

Account::Account()             // конструктор за умовчанням
{
    name = new char[100];
    strcpy(name, "");

    summa = 0;
}

Account::Account(char *aName)  // конструктор ініціалізації
{
    name = new char[ strlen(aName) + 1 ];
    strcpy(name, aName);

    summa = 0;
}

Account::Account(double aSumma) // конструктор ініціалізації
{
    name = new char[100];
    strcpy(name, "");

    summa = aSumma;
}

Account::Account(char *aName, double aSumma)  // конструктор ініціалізації
{
    name = new char[ strlen(aName) + 1 ];
    strcpy(name, aName);

    summa = aSumma;
}

Account::Account(Account &account)  // конструктор копіювання
{
    name = new char[ strlen( account.name ) + 1 ];
    strcpy(name, account.name);

    summa = account.summa;
}
```

Перший конструктор (який викликається без параметрів) – це конструктор за умовчанням, наступні три – конструктори ініціалізації; останній – це конструктор копіювання.

Конструктор за умовчанням та конструктор ініціалізації

За відсутності в класі явно визначених конструкторів автоматично створюється конструктор без аргументів (конструктор за умовчанням) та конструктор копіювання. Конструктор за умовчанням, який створюється автоматично, має вигляд:

```
клас::клас() {}
```

Якщо явно визначити хоча би один конструктор, який приймає параметр (тобто, хоча би один конструктор ініціалізації), то конструктор за умовчанням автоматично створюватися не буде, його потрібно написати явно. При цьому можна використовувати механізм параметрів із значеннями за умовчанням.

Попередній приклад можна переписати так:

```
class Account
{
    char* name;                 // власник
    double summa;               // сума

public:
    Account();                  // конструктор за умовчанням
    Account(double);            // конструктор ініціалізації
    Account(char*, double = 0);// конструктор ініціалізації
    Account(Account&);          // конструктор копіювання

    /* ... */                   // інші методи
};

Account::Account()              // конструктор за умовчанням
{
    name = new char[100];
    strcpy(name, "");

    summa = 0;
}

Account::Account(double aSumma) // конструктор ініціалізації
{
    name = new char[ 100 ];
    strcpy(name, "");

    summa = aSumma;
}

Account::Account(char *aName, double aSumma)  // конструктор ініціалізації
{
    name = new char[ strlen(aName) + 1 ];
    strcpy(name, aName);

    summa = aSumma;
}

Account::Account(Account &account)  // конструктор копіювання
{
    name = new char[ strlen( account.name ) + 1 ];
    strcpy(name, account.name);

    summa = account.summa;
}
```

Будь-який конструктор з параметрами не свого класу є конструктором ініціалізації, який призначений для ініціалізації полів класу.

```
class Time
{
    int hours, minutes, seconds;

public:
    Time();                          // конструктор за умовчанням
    Time(int h, int m=0, int s=0)  // конструктор ініціалізації
    {
        hours = h; minutes = m; seconds = s;
    }
};

Time::Time()
{
    hours = minutes = seconds = 0;
}
```

Конструктор ініціалізації забезпечує оголошення та ініціалізацію змінних:

```
Time t(12,59,59), s(13,15), u = 14;
```

Конструктор ініціалізації може викликатися явно:

```
Time r = Time(10);                  // явний виклик конструктора
```

Конструктор ініціалізації, в якому всі параметри мають значення за умовчанням, відіграє роль конструктора без аргументів (конструктора за умовчанням).

Конструктори викликаються і при створенні динамічних змінних:

```
Time *a = new Time;                 // конструктор за умовчанням
Time *b = new Time();               // конструктор за умовчанням
Time *d = new Time(12,35);          // конструктор ініціалізації
Time *f = new Time(*d);             // конструктор копіювання
                                    // (див. наст. параграф)
```

Конструктори забезпечують звичну форму оголошення констант:

```
const Time c1(16,45);            // конструктор ініціалізації
const Time c2 = 15;              // конструктор ініціалізації
const Time c3 = Time(11,25,30);  // явний виклик конструктора
                                 // ініціалізації
```

Конструктор ініціалізації для елементів масиву

Конструктор ініціалізації забезпечує ініціалізацію масиву:

```
Time T[3] = { 10, 15, 23 };
```

Для кожного цілого числа, вказаного в списку ініціалізації масиву, викликається конструктор ініціалізації, тому ініціалізація вказаного масиву являє собою виконання наступних команд:

```
Time T[0] = Time(10);
Time T[1] = Time(15);
Time T[2] = Time(23);
```

Якщо масив не ініціалізовано, то для кожного елемента масиву викликається конструктор без аргументів, якщо він визначений явно чи неявно (або викликається конструктор ініціалізації з аргументами за умовчанням).

Конструктор копіювання

Конструктор копіювання автоматично створюється у вигляді:

```
клас::клас(const клас &r)
{
    *this = r;
}
```

Аргументом конструктора копіювання завжди є об’єкт цього самого класу.

Наступні приклади показують використання конструкторів за умовчанням та копіювання:

```
Time d;         // конструктор за умовчанням
Time t(d);      // конструктор копіювання
Time r = d;     // конструктор копіювання
```

Конструктор копіювання створюється автоматично завжди (навіть якщо визначено конструктори ініціалізації), якщо він не визначений явно.

При явному визначенні конструктора копіювання аргумент має передаватися за посиланням – передавати його за значенням не можна, оскільки при передаванні за значенням виконується копіювання, тобто дія, яку власне і має виконати конструктор копіювання.

В класі може бути кілька конструкторів копіювання, можна визначати кілька аргументів, якщо їм присвоїти значення за умовчанням.

Конструктори та параметри

Конструктор копіювання викликається неявно при передаванні у функцію параметра-об’єкта за значенням і при поверненні значення об’єкта із функції. При інших способах передачі параметру конструктор копіювання не викликається.

Конструктор копіювання викликається для створення прихованої «внутрішньої» копії об’єктів. При передаванні виразу в якості параметру може викликатися конструктор ініціалізації.

Виклик функцій без аргументів у всіх випадках супроводжується викликом конструктора ініціалізації.

Конструктори та константи

Константи в класі

В класі можна визначити цілі константи. До цілих типів відносяться `char`, `short`, `int`, `long` в знаковому та без знаковому варіантах, `bool` та перелічуваний тип `enum`.

Константу в класі можна визначити (тобто, ініціалізувати) одним із двох способів:

- як перелічуваний тип `enum`;
- як статичну константу цілого типу.

```
class Constant
{
public:
    enum Week {MON=1, TUE, WED, THU, FRI, SAT, SUN=0 };

    static const int  c1 = 1;
    static const char c2 = 'x';
    static const Week c3 = SUN;
};
```

Звертання до перелічуваних констант здійснюється або за допомогою імені об’єкта, або за іменем класу:

```
cout << Constant::SAT << endl;  // кваліфікатор - клас
Constant c;
cout << c.SAT << endl;          // кваліфікатор - об'єкт
```

Звертання до статичних констант зазвичай здійснюють за іменем класу:

```
cout << Constant::c1 << endl;
cout << Constant::c2 << endl;
cout << Constant::c3 << endl;
```

Константи всіх інших типів оголошуються як константні поля (без явної ініціалізації).

Ініціалізація константних полів – список ініціалізації конструктора

Лише статичні цілі константи та константи перелічуваного типу можна ініціалізувати при їх визначенні в класі.

Константи інших типів мають бути оголошені як константні поля, ініціалізувати їх має конструктор в *списку ініціалізації конструктора*.

Список ініціалізації вказується одразу після списку параметрів; на початку списку записується двокрапка, яка відокремлює його від списку параметрів; ініціалізатори полів записуються через кому. Константними можуть бути і поля цілого типу:

```
class Constant
{
    const long   c1;              // константне поле цілого типу
    const double c2;              // константне поле дійсного типу

public:
    Constant(const long l = 0, const double d = 0)
        : c2(d), c1(l)            // список ініціалізації
    {}                            // тіло конструктора
};
```

Один ініціалізатор являє собою ім’я поля-константи, за яким в дужках записується вираз ініціалізації.

Правила запису списку ініціалізації:

1. В списку ініціалізації конструктора можна ініціалізувати як константні поля, так і звичайні поля-змінні.
2. Не залежно від порядку запису полів в списку ініціалізації, поля отримують значення в порядку їх визначення в класі.
3. В якості виразу ініціалізації можна вказувати будь-який вираз (не обов’язково константний), значення якого можна привести до типу відповідного поля; для полів-об’єктів виконується виклик конструктора.
4. Для ініціалізації поля вбудованого типу нулем використовується особлива форма ініціалізатора: *`ім'я_поля`* `( )` – вираз ініціалізації не вказується; у випадку такого ініціалізатора для поля-об’єкта викликається конструктор за умовчанням.
5. Замість виразу ініціалізації в дужках можна вказувати список виразів, записаних через кому; необхідно, щоб значення останнього виразу можна було привести до типу відповідного поля.
6. Список ініціалізації виконується до початку виконання тіла конструктора.

Деструктори

Призначення та особливості деструкторів

Деструктори призначені для знищення об'єктів, тобто деструктори – це особливі методи, які використовуються для звільнення пам'яті, виділеної для об'єкта.

Деструктор викликається автоматично, коли об'єкт виходить за межі області видимості:

- для *локальних* об'єктів – при виході із блоку, в якому вони оголошені;
- для *глобальних* – як частина процедури виходу із `main()`;
- для *динамічних об'єктів, визначених через вказівники*, деструктор викликається неявно при використанні операції `delete`.

Автоматичний виклик деструктора об'єкту при виході із області дії вказівника на нього – не виконується!

Особливості деструктора:

- Ім'я деструктора починається з тильди (~), одразу за якою записується ім'я класу.
- Не має аргументів і не повертає результату (навіть `void` вказувати не можна).
- Не може бути оголошений як `const` або `static`.
- Не успадковується.
- Може бути віртуальним.
- Не можна визначити вказівника на деструктор.

Деструктор можна викликати явно за допомогою повністю уточненого імені, наприклад:

```
Account *pa;
/* ... */
pa->~Account();
```

Це може стати потрібним для об'єктів, яким за допомогою перевантаженої операції `new` виділялася конкретна адреса пам'яті. Без необхідності явно викликати деструктор об'єкту не рекомендується.

Опис деструкторів

Деструктор – це особливий метод, ім'я якого збігається з іменем класу, з єдиною різницею: його першим символом має бути символ ~ (тильда). Деструктор не повертає результату та не приймає аргументів. Деструктор в класі може бути лише один – перевантаження деструкторів заборонене. За відсутності в класі явно визначеного

деструктора він створюється автоматично. Деструктор, який створюється автоматично, має такий вигляд:

```
клас::~клас() {}
```

Якщо конструктори класу не вимагають розподілу ресурсів, які необхідно звільняти (наприклад, динамічну пам'ять), або не виконують дій щодо ініціалізації, які при знищенні об'єкта вимагають завершальних дій (наприклад, якщо файл відкривається, то його необхідно закрити), то деструктор можна не створювати.

Для вище наведеного класу `Account` деструктор буде наступним:

```
class Account
{
    char* name;                  // власник
    double summa;                // сума

public:
    Account();                   // конструктор за умовчанням
    Account(double);             // конструктор ініціалізації
    Account(char*, double = 0);  // конструктор ініціалізації
    Account(Account&);           // конструктор копіювання

    ~Account();                  // деструктор

    /* ... */                    // інші методи
};

Account::Account()               // конструктор за умовчанням
{
    name = new char[100];
    strcpy(name, "");

    summa = 0;
}

Account::Account(double aSumma)  // конструктор ініціалізації
{
    name = new char[ 100 ];
    strcpy(name, "");

    summa = aSumma;
}

Account::Account(char *aName, double aSumma)  // конструктор ініціалізації
{
    name = new char[ strlen(aName) + 1 ];
    strcpy(name, aName);

    summa = aSumma;
}

Account::Account(Account &account)            // конструктор копіювання
{
    name = new char[ strlen( account.name ) + 1 ];
    strcpy(name, account.name);

    summa = account.summa;
}
```

```
Account::~Account()                          // деструктор
{
    delete [] name;  // звільнили пам’ять, виділену в конструкторі
}
```

Використання конструкторів та деструкторів

Обчислення кількості наявних об’єктів певного класу

Статичне поле `count` містить лічильник наявних екземплярів, який ініціалізується нульовим значенням. Конструктор збільшує значення лічильника, а деструктор – зменшує:

```
class Object
{
    static unsigned int count;

public:
    static unsigned int Count()
    {
        return count;
    }

    Object()
    {
        count++;
    }

    ~Object()
    {
        count--;
    }
};

unsigned int Object::count = 0;

int main()
{
    Object a;
    cout << Object::Count() << endl;
    {
        Object b;
        cout << Object::Count() << endl;
    }
    cout << Object::Count() << endl;

    return 0;
}
```

Результат:

```
1
2
1
```

Недоступні конструктори. Патерн Singleton

Оголошення конструкторів закритими (не в секції `public`) унеможливлює створення об’єктів звичним способом, – це використовується в патерні Singleton (*одиночка*).

Патерни – це найбільш загальні шаблони проектування, кожний з них призначений для вирішення певної типової задачі.

Патерн Singleton гарантує, що клас буде мати лише один єдиний екземпляр і надає глобальну точку доступу до цього єдиного екземпляру:

```
class Singleton
{
public:
    static Singleton* getInstance();

private:
    Singleton();                    // закритий конструктор
    static Singleton* instance;
};

Singleton* Singleton::instance = 0;  // ініціалізація статичного поля

Singleton::Singleton() { /*...*/ }   // конструктор

Singleton* Singleton::getInstance()
{
    if (instance == 0)
        instance = new Singleton();

    return instance;
}
```

Доступ до класу здійснюється через відкритий статичний метод `getInstance()`, який повертає значення приватного статичного поля `instance` (тип цього поля – вказівник на екземпляр класу).

Поле ініціалізується нулем. Метод `getInstance()` перевіряє значення поля і, якщо воно дорівнює нулю, створює екземпляр класу `Singleton` та присвоює полю `instance` значення вказівника на цей екземпляр. Незалежно від початкового значення поля (нуль чи ні), метод повертає вказівник на екземпляр класу.

При цьому конструктор – закритий, оголошений як не доступний, і якщо буде спроба створити об’єкт класу `Singleton` безпосередньо, звичним способом, – то отримаємо помилку на етапі компіляції. Це гарантує, що буде створено єдиний екземпляр класу `Singleton`.

У випадку, коли потрібно створити ієрархію класів, які мають мати один єдиний екземпляр, то конструктор слід оголосити захищеним, записавши його в секції `protected`: це дасть можливість виклику конструктора похідними класами (див. Тему 3. Успадкування).

Перевантаження операцій

В C++ можна перевантажувати вбудовані операції – це один із проявів *поліморфізму*. Перевантаження операцій не є обов'язковою в об'єктно-орієнтованому програмуванні – в мові java воно відсутнє. Проте наявність перевантаження операцій в C++ забезпечує додатковий рівень зручності при використанні нових типів даних.

Особливості перевантаження операцій

При перевантажені операцій слід враховувати наступні обмеження:

1. Заборонено перевантажувати наступні операції:
   - `sizeof()` – визначення розміру аргументу;
   - `.` (крапка) – селектор компонента (поля чи метода) об'єкта;
   - `?:` – умовна операція;
   - `::` – операція доступу до області дії;
   - `.*` – вибір компонента класу через вказівник;
   - `#` та `##` – операції препроцесора.
2. Операції можна перевантажувати лише для нового типу даних – не можна перевантажити операцію для вбудованого типу. В C++ новий тип можна утворити за допомогою конструкцій `enum`, `union`, `struct` та `class`.
3. Не можна змінити пріоритет перевантаженої операції та кількість операндів. Унарна операція має мати один операнд (за винятком перевантаження операцій інкременту `++` та декременту `--`), бінарна – два операнди; не можна використовувати параметри за умовчанням. Єдина операція, яка не має фіксованої кількості операндів, – це операція виклику функції `()`. Операції `+`, `-`, `*`, `&` можна перевантажувати як унарні, так і бінарні.
4. Операції можна перевантажувати або як незалежні зовнішні функції (лише такий спосіб допустимий для `enum`), або як методи класу.

   Чотири операції:

   - `=` – присвоєння;
   - `()` – виклику функції;
   - `[]` – індексування;
   - `->` – доступу за вказівником

   можна перевантажувати лише як методи класу. Ці операції в принципі не можна перевантажити для `enum`.

5. Якщо операція перевантажується як метод класу, то лівим (або єдиним) операндом обов'язково буде об'єкт класу, для якого перевантажується операція.

Прототип функції-операції має наступний вигляд:

```
тип operator ☺ ( список_параметрів );
```

де ☺ – позначає знак операції. Слово `operator` є зарезервованим і може використовуватися лише при визначенні або функціональній формі виклику операції.

Способи перевантаження операцій

В загальному випадку в C++ є два способи перевантаження операцій:

- за допомогою зовнішніх дружніх функцій;
- за допомогою методів класу.

Розглянемо особливості цих способів на прикладі:

Умова прикладу на реалізацію операції додавання

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.

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;
```

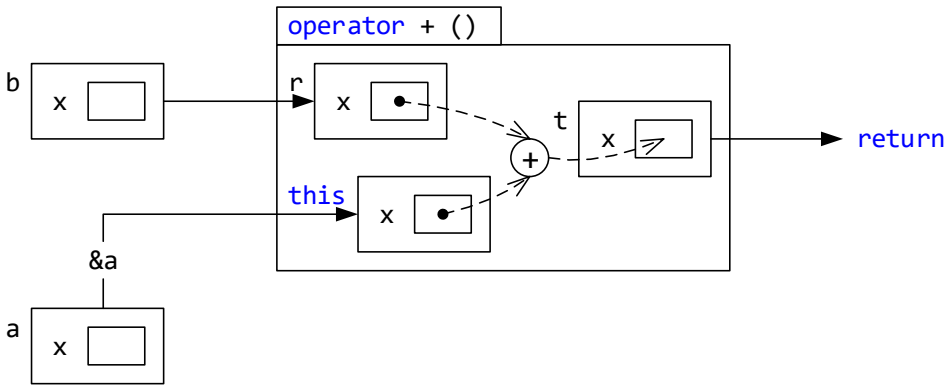
– кожний об’єкт містить свою комірку пам’яті, виділену для поля x.

Реалізація бінарної операції звичайним методом з одним параметром

Реалізуємо операцію методом `operator + ()`, 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 operator + (Number r);
};

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

Діаграма виконання методу `operator + ()` буде мати наступний вигляд:



Використовувати цей метод можна так:

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

    c = a.operator + (b);    // функціональна форма
                             // або
    c = a + b;               // операторна форма
```

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

    return 0;
}
```

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

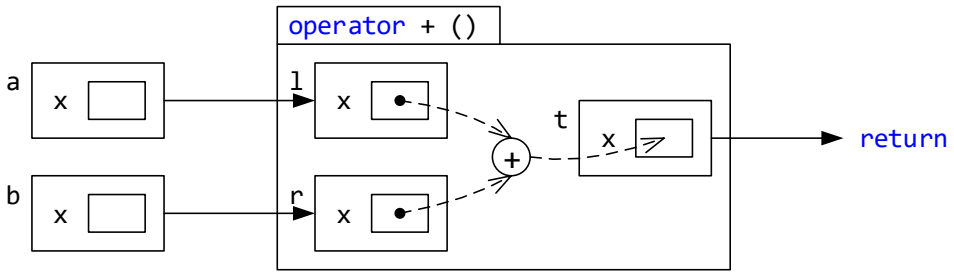
Проте є одна проблема: в такій реалізації операції додавання її лівий та правий операнди – нерівнозначні. Перший (лівий) операнд – це поточний об’єкт, що викликає метод `operator + ()`. Другий (правий) операнд – це параметр метода `operator + ()`. Завжди поточний об’єкт, який викликає метод, буде більш 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 бінарної операції дружньою функцією з двома параметрами

Перепишемо наш приклад, але тепер реалізуємо операцію додавання за допомогою дружньої функції `operator + ()`, 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 operator + (Number l, Number r);
};

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

В цій версії діаграма виконання дружньої функції `operator + ()` буде мати наступний вигляд:



Приклад використання цієї дружньої функції:

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

    c = operator + (a, b);    // функціональна форма
                              // або
    c = a + b;                // операторна форма

    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.

Перевантаження операцій за допомогою дружніх функцій

Прототип бінарної операції, яка перевантажується за допомогою зовнішньої незалежної функції, має наступний вигляд:

```
тип operator ☺ ( параметр_1, параметр_2 );
```

де ☺ – позначає знак операції.

Для забезпечення комутативності бінарної операції з параметрами-операндами різних типів, зазвичай потрібно реалізувати дві функції-операції:

```
тип operator ☺ ( параметр_1, параметр_2 );
тип operator ☺ ( параметр_2, параметр_1 );
```

Хоча би один із параметрів має бути нового типу. Параметри можна передавати будь-яким доступним способом.

Виклик операції виконується одним із двох способів:

- операторна інфіксна форма: `параметр_1 ☺ параметр_2`
- функціональна форма: `operator ☺ ( параметр_1, параметр_2 )`

Прототип унарної операції, яка перевантажується незалежною зовнішньою функцією, відрізняється лише кількістю параметрів:

```
тип operator ☺ ( параметр );
```

Параметр має бути нового типу.

Викликати таку перевантажену операцію можна теж двома способами:

- операторна префіксна форма: ☺ *параметр*
- функціональна форма: `operator` ☺ (*параметр*)

Операції інкременту та декременту мають дві форми: префіксну та постфіксну. При визначенні постфіксної форми операції потрібно оголосити додатковий параметр типу `int`:

```
тип operator ☺ ( параметр, int );
```

Цей параметр не використовується в тілі функції. Операторна постфіксна форма виклику такої операції: *параметр* ☺ ; при функціональній формі виклику необхідно вказати другий фіктивний аргумент, наприклад:

```
operator ☺ ( параметр, 0 );
```

При перевантаженні операцій зовнішніми для класу функціями поля мають бути відкриті, або клас має надати методи доступу для отримання та зміни значень полів, або ці функції-операції мають бути оголошені в класі як дружні.

Приклад:

```
class Fraction
{
    int num, denum;               // поля (чисельник, знаменник) - приховані

public:
    void reduce() {/*...*/}
    friend Fraction operator + (const Fraction &L, const Fraction &R);
};

Fraction operator + (const Fraction &L, const Fraction &R)
{
    Fraction t;                   // (a,b) + (c,d) = (ad + bc, bd)
    t.denum = L.denum * R.denum;  // знаменник результату
    t.num   = L.num * R.denum
            + R.num * L.denum;    // чисельник результату
    t.reduce();                   // скорочення
    return t;
}
```

Перевантаження операцій за допомогою методів класу

Для методів-операцій один параметр-операнд – поточний об’єкт – вже за умовчанням визначений.

Унарні операції виконуються для поточного об’єкту, який і буде єдиним аргументом. Тому унарні операції, які перевантажуються за допомогою методів класу, не мають параметрів. Прототип унарної операції, реалізованої як метод класу, має наступний вигляд:

```
тип operator ☺ ();
```

де ☺ – позначає знак операції. Результат операції може бути будь-якого типу, в тому числі, і цього ж самого класу. Операція може повертати значення, посилання або вказівник на об’єкт. Можна вказувати `void` в якості типу результату.

Постфіксні операції інкременту та декременту є винятком з цього правила. Для того, щоб відрізнити постфіксну операцію від префіксної, в постфіксній формі слід задати фіктивний параметр типу `int`, який реально не використовується.

Префіксна форма операцій інкременту та декременту має мати прототип

```
тип& operator ☺ ();
```

– операція має повертати посилання на об’єкт!

Постфіксна форма операцій інкременту та декременту має мати прототип

```
тип  operator ☺ ();
```

– операція має повертати значення об’єкту!

Тут *тип* – це ім’я класу, в якому визначаємо операцію інкременту чи декременту.

Прототип бінарної операції має один аргумент і виглядає так:

```
тип operator ☺ ( параметр );
```

Параметри можна передавати будь-яким потрібним способом: за значенням, за посиланням або за вказівником. Метод-операція може повертати значення, вказівник або посилання на об’єкт, в тому числі і свого класу. Можна вказувати `void` в якості типу результату.

При реалізації метода-операції поза класом слід (як і для інших методів) в заголовку вказати префікс – ім’я класу та операцію доступу до області дії:

```
тип клас::operator ☺ ()            // унарна операція
{ /* ... */ }

тип клас::operator ☺ ( параметр )  // бінарна операція
{ /* ... */ }
```

Виклик унарної операції для об’єкта має вигляд:

```
☺ об'єкт
об'єкт ☺
```

– в залежності від того, який вид операції використовується – префіксний чи постфіксний.

Функціональний виклик для префіксної операції має вигляд

```
об'єкт.operator ☺ ()
```

Для постфіксної операції у функціональній формі слід вказати фіктивний аргумент:

```
об'єкт.operator ☺ ( 0 )
```

Функціональна форма виклику бінарної операції, реалізованої як метод класу, виглядає так:

```
об'єкт.operator ☺ ( аргумент )
```

Операторна інфіксна форма:

```
об'єкт  ☺ аргумент
```

– скорочення функціональної.

Операції присвоєння є бінарними, тому мають мати один аргумент.

Якщо операція присвоєння `operator =` не реалізована в класі явно, вона створюється автоматично за умовчанням. Для будь-якого класу операція присвоєння, яка створюється автоматично, має прототип:

```
клас& operator = ( const клас &r );
```

Можна реалізувати операцію присвоєння з параметром та результатом не свого класу; можна передавати параметр будь-яким способом; повертати можна не посилання, а значення чи вказівник – все залежить від потреб задачі.

Інші операції з присвоєнням автоматично не створюються. Зазвичай операції з присвоєнням мають повертати посилання на поточний об’єкт. Будь-який метод, який повертає неконстантне посилання, можна використовувати багатократно, формуючи ланцюжки операцій:

```
class Money
{
    double summa;

public:
    Money& operator +=(const Money &r);
};

Money& Money::operator +=(const Money &r)
{
    summa += r.summa;
    return *this;
}
```

```
int main()
{
    Money a, b, c;
    /* ... */
    a += b += c;

    return 0;
}
```

Повторне використання можна явно заборонити, якщо повертати константне посилання, наприклад:

```
const Money& Money::operator +=(const Money &r)
```

Модифікатор `const` забороняє модифікацію результату.

Приклади та рекомендації щодо перевантаження операцій

Форми перевантаження операцій:

Операція	**Рекомендований спосіб перевантаження**
Всі унарні операції	Метод класу
`=  []  ()  ->  ->*`	Обов'язково метод класу
`+=  -=  *=  /=  %=  &=  |=  ^=  <<=  >>=`	Метод класу
Решта бінарних операцій	Зовнішня дружня функція

Перевантаження унарних операцій

Унарна операція, реалізована як *метод* класу, має бути нестатичним методом без параметрів, – операндом буде об'єкт, який викликає цю операцію.

Якщо унарна операція реалізована як *зовнішня функція*, вона має приймати один параметр типу класу. Якщо таку функцію не оголошувати в класі як дружню, необхідно забезпечити доступність полів (наприклад за допомогою відповідних методів доступу).

Перевантаження операцій інкременту та декременту

Операції постфіксного інкременту та декременту мають мати перший параметр типу `int`. Він використовується лише для того, щоб відрізнити їх від префіксної форми.

Для збереження стандартної семантики цих операцій слід реалізувати наступне:

- операції префіксного інкременту та декременту мають повертати посилання на модифікований поточний об'єкт – аргумент операції;
- операції постфіксного інкременту та декременту мають повертати значення копії ще не модифікованого об'єкту і модифікувати відповідне поле поточного об'єкта – аргумента операції.

Розглянемо приклад класу `A`, що інкапсулює ціле число, яке представлене полем `x`:

```
#include <iostream>

using namespace std;

class A
{
    int x;

public:
    A(int value = 0) : x(value) {}  // конструктор ініціалізації
    A(const A& a) { x = a.x; }      // конструктор копіювання

    A& operator --();
    A& operator ++();
    A operator --(int);
    A operator ++(int);

    friend ostream& operator <<(ostream& out, const A& a);
    friend istream& operator >>(istream& in, A& a);
};

A& A::operator --()
{
    --x;            // модифікували поточний об'єкт
    return *this;   // повернули модифікований об'єкт
}

A& A::operator ++()
{
    ++x;            // модифікували поточний об'єкт
    return *this;   // повернули модифікований об'єкт
}

A A::operator --(int)
{
    A a(*this);     // створили копію
    x--;            // модифікували поточний об'єкт
    return a;       // повернули копію
}

A A::operator ++(int)
{
    A a(*this);     // створили копію
    x++;            // модифікували поточний об'єкт
    return a;       // повернули копію
}

ostream& operator <<(ostream& out, const A& a)
{
    return out << a.x;
}

istream& operator >>(istream& in, A& a)
{
    cout << ".x = "; in >> a.x;
    return in;
}
```

```
int main()
{
    A a(1);

    cout << "++a   : " << ++a << endl;
    cout << "  a   : " << a << endl;
    cout << "  a++ : " << a++ << endl;
    cout << "  a   : " << a << endl;

    system("pause");
    return 0;
}
```

Вивід:

```
++a   : 2
  a   : 2
  a++ : 2
  a   : 3
```

Перевантаження бінарних операцій

Бінарна операція, реалізована як *метод* класу, має бути нестатичним методом з одним параметром, – першим операндом буде об'єкт, який викликає цю операцію.

Якщо бінарна операція реалізована як *зовнішня дружня функція*, вона має приймати два параметри типу класу. Якщо таку функцію не оголошувати в класі як дружню, необхідно забезпечити доступність полів (наприклад за допомогою відповідних методів доступу).

Перевантаження операції присвоєння

Операція присвоєння визначена за умовчанням для будь-якого класу як по елементне копіювання. Ця операція викликається щоразу, коли одному вже створеному об'єкту присвоюється значення іншого.

Якщо клас містить поля, пам'ять для яких виділяється динамічно, необхідно визначити власну операцію присвоєння.

Для того, щоб зберегти семантику присвоєння, метод-операція має повертати посилання на об'єкт, для якого вона викликана, і приймати в якості параметру єдиний аргумент – константне посилання на об'єкт, що присвоюється.

Результатом має бути *l*-value – те, що можна записати ліворуч від знаку присвоєння:

```
class A
{
    int x;

public:
    A(int value = 0) : x(value) {}  // конструктор ініціалізації
    A(const A& a) { x = a.x; }      // конструктор копіювання

    A& operator =(const A& r);      // присвоєння
};
```

```
A& A::operator =(const A& r)
{
    x = r.x;                        // змінили поле поточного об’єкта
    return *this;                   // повернули посилання на
}                                   // поточний об’єкт
```

Ще приклад:

```
class Account
{
    char* name;                 // власник
    double summa;               // сума

public:
    /* ... */
    Account& operator =(const Account& r);
};

Account& Account::operator =(const Account& r)
{
    // перевірка на самоприсвоєння
    if (&r == this) return *this;

    if (name) delete[] name;
    if (r.name)
    {
        name = new char[strlen(r.name) + 1];
        strcpy_s(name, strlen(r.name) + 1, r.name);
    }
    else
        name = 0;

    summa = r.summa;
    return *this;
}
```

Зазвичай операції з присвоєнням мають повертати посилання на поточний об’єкт. Будь-який метод, який повертає неконстантне посилання, можна використовувати багатократно, формуючи ланцюжки операцій:

```
Account A(10), B, C;
C = B = A;
```

Повторне використання можна явно заборонити, якщо повертати константне посилання, наприклад:

```
const Account& operator =(const Account &r);
```

Модифікатор `const` забороняє модифікацію результату.

Операцію присвоєння можна визначити лише як метод класу. Вона не успадковується.

Перевантаження операцій вводу / виводу

Це – бінарні операції, лівим операндом яких є об'єкт-потік вводу / виводу, а правим – об'єкт, який необхідно отримати чи помістити у цей потік. Результатом має бути посилання на потік, щоб можна було організувати ланцюжки операцій, як для стандартних типів:

```
#include <iostream>

using namespace std;

class Account
{
    char* name;               // власник
    double summa;             // сума

public:
    /* ... */
    friend ostream& operator <<(ostream& out, const Account& a);
    friend istream& operator >>(istream& in, Account& a);
};

ostream& operator <<(ostream& out, const Account& a)
{
    return out << "name = " << a.name << "    summa = " << a.summa;
}

istream& operator >>(istream& in, Account& a)
{
    cout << "name:  "; in.getline(a.name, strlen(a.name), '\n');
    cout << "summa: "; in >> a.summa;
    return in;
}

int main()
{
    Account A, B, C;

    cin >> A >> B >> C;
    cout << A << B << C << endl;

    return 0;
}
```

Перевантаження операції приведення типу

Для перетворення об'єкта до іншого типу в класі можна визначити метод-операцію перетворення типу, прототип якої – наступний:

```
operator mun ();
operator mun () const;
```

mun – може бути вбудованим типом, типом класу або іменем `typedef`. На місці *muny* не може бути записаний тип масиву або функції.

Тип результату та параметри вказувати не потрібно – інакше це буде помилкою.

Операція перетворення типу *обов'язково* має бути методом. В оголошенні операції приведення типу не можна вказувати ні тип результату, ні список параметрів – це буде помилкою.

Бажано цей метод описувати як константний – тоді його можна буде викликати для константних об'єктів.

Конструктор ініціалізації виконує перетворення типу *за умовчанням* із типу аргументу до типу класу. Операція приведення типу здійснює приведення типу *за умовчанням* із типу класу до вказаного типу. Таким чином, операція перетворення типу – обернена до дії конструктора ініціалізації:

Конструктор ініціалізації: *значення_типу* ⟶ *об'єкт*

Операція перетворення типу: *об'єкт* ⟶ *значення_типу*

```
class Account
{
    string name;              // власник
    double summa;             // сума

public:
    /* ... */

    operator double() const;
    operator string() const;
};

Account::operator double() const
{
    return summa;
}

Account::operator string() const
{
    return name;
}

int main()
{
    Account A;
    /* ... */

    cout << double(A) << endl;
    cout << string(A) << endl;

    return 0;
}
```

Заборона неявного приведення типу: директива `explicit`

Щоб уникнути помилок неявного перетворення типу об'єкта, краще ці неявні перетворення зовсім заборонити – тоді клієнт буде вимушений використовувати конструкції явного перетворення типу.

Якщо не визначати операції перетворення типу, то це унеможливить перетворення *об'єкт → значення_типу*

C++ дозволяє також заборонити неявні перетворення *значення_типу → об'єкт*, які виконує конструктор ініціалізації. Для цього використовується ключове слово `explicit` в заголовку конструктора ініціалізації при визначенні класу:

```
class Account
{
    char* name;                 // власник
    double summa;               // сума

public:
    explicit Account(const double sum = 0);
    /* ... */
};

Account::Account(const double sum)
{
    summa = sum;
}
```

Тепер допускається лише явний виклик конструктора як при оголошенні з ініціалізацією, так і передаванні параметрів:

```
Account f(Account a)  // приклад функції
{
    return a;
}

int main()
{
    Account A = f(Account(2.15));
    Account B = Account(12.65);

    return 0;
}
```

При спробі не вказувати праворуч від операції присвоєння тип `Account` виникають помилки компіляції:

```
    Account C = 15.5;
```

```
— error C2440: 'initializing' : cannot convert from 'double' to 'Account'. Constructor for class 'Account' is declared 'explicit'
```

Проте наступна команда – не забороняється, бо це – явний виклик конструктора ініціалізації:

```
    Account C(15.5);
```

Явні перетворення типу в C++

Приведення типу в стилі C

Операцію можна записати у двох формах:

```
тип (вираз)
(тип) вираз
```

Результатом буде вираз заданого типу, наприклад:

```
int a = 2;
double b = 3.14;

cout << double (a) << (int) b << endl;
```

Змінна `a` перетворюється до дійсного типу, а змінна `b` – до цілого з відкиданням дробової частини. В обох випадках форма внутрішнього представлення результату операції перетворення інша, чим початкового значення.

Така форма явного приведення типу є джерелом можливих помилок, оскільки вся відповідальність за його результат покладається на програміста. Тому в C++ введені операції, які дозволяють більш ретельно контролювати перетворення типу, насамперед – шляхом більш чіткого синтаксису, який дозволяє програмісту явно висловити свої наміри стосовно перетворення типу.

Вище наведене перетворення в стилі C залишене в C++ лише для сумісності із попередніми версіями C-програм, використовувати його в C++ програмах *не рекомендується*.

Операція const_cast

Операція використовується для того, щоб зняти модифікатор `const`. Зазвичай, вона використовується, коли у функцію передається вказівник на константу, а відповідний параметр оголошено без модифікатора `const`. Формат операції:

```
const_cast <тип> (вираз)
```

тип – визначає той тип, до якого приводиться *вираз*, цей тип має бути таким ж, як і тип виразу, за винятком модифікатора `const`:

```
const int *k = 0;
int *n = const_cast <int*> (k);
```

Необхідність такої операції викликана тим, що не завжди параметри, які не змінюються в тілі функції, описуються з модифікатором `const`, хоча це і рекомендується. Правила C++ забороняють передавання вказівника на константу в якості аргументу функції, яка приймає звичайний вказівник. Операція `const_cast` дозволяє обійти це обмеження, причому функція не може змінювати значення, на яке вказує переданий вказівник, бо результат роботи програми буде непередбачуваним:

```
void Print(int* p)
{
    cout << *p << endl;            // значення *p не змінюється
}

int main()
{
    int* r;

    Print(r);                      // все вірно: типи аргументу
                                   // і параметра збігаються
    const int* q;

    Print(q);                      // помилка, оскільки q оголошено
                                   // як вказівник на константу
    Print(const_cast<int*>(q));    // все вірно

    return 0;
}
```

Звичайно, більш правильно буде не змушувати інших використовувати `const_cast` для передачі значення вказівника на константу параметру – звичайному вказівнику, а одразу при розробленні функції використовувати модифікатор `const` для параметру, який в тілі функції не змінюється:

```
void Print(const int* p)           // параметр - вказівник на константу
{
    cout << *p << endl;            // значення *p не змінюється
}

int main()
{
    int* r;

    Print(r);                      // все вірно: вказівник на константу
                                   // може отримати значення звичайного
                                   // вказівника

    const int* q;

    Print(q);                      // все вірно: типи аргументу
                                   // і параметра збігаються

    return 0;
}
```

Див. також «Тема 3. Успадковування. Явні перетворення типів в мові C++. Перетворення `const_cast<>`».

Операція dynamic_cast

Операція використовується для перетворення вказівників на об’єкти споріднених в одній ієрархії класів, зазвичай – вказівника на об’єкт базового класу до вказівника на об’єкт похідного класу. Це – так зване перетворення, що *понижує* (рівень ієрархії) – downcast. Можливе ще перетворення із похідного до базового класу – яке *підвищує* (upcast), а також

перетворення між похідними класами одного базового чи між базовими класами одного похідного – *перехресні* перетворення (crosscast).

Більш детально цю операцію розглянемо при вивченні теми «Успадковування» – див. «Тема 3. Успадковування. Явні перетворення типів в мові C++. Перетворення типів споріднених класів ієрархії `dynamic_cast<>`».

Операція static_cast

Операція `static_cast` використовується для перетворення типу на етапі компіляції між:

- цілими типами;
- цілими та дійсними типами;
- цілими та перелічуваними типами;
- вказівниками та посиланнями на об'єкти однієї ієрархії, за умови що таке перетворення однозначне і не пов'язане з перетворенням, яке понижує віртуальний базовий клас (цей випадок буде детально розглядатися при вивченні теми «Успадковування»).

Формат операції:

```
static_cast <тип> (вираз)
```

тип – визначає той тип, до якого приводиться *вираз*.

Результат операції має вказаний тип, який може бути посиланням, вказівником, арифметичним чи перелічуваним типом.

При виконанні операції внутрішнє представлення даних може бути модифіковане, хоча числове значення залишається незмінним. Наприклад:

```
double d = 100;
int i = static_cast <int>( d );  // ціле і дійсне мають різне
                                 // внутрішнє представлення
```

Такі перетворення використовують для відключення повідомлень компілятора про можливу втрату даних при присвоєнні дійсного числа цілій змінній.

Результат операції залишається на совісті програміста.

Операція reinterpret_cast

Ця операція використовується для перетворення типів, які не пов'язані між собою, наприклад – вказівників до цілих та навпаки, а також – вказівників типу `void*` до вказівників на конкретний тип. При цьому внутрішнє представлення даних залишається незмінним, змінюється лише «точка зору компілятора» на дані, тобто, спосіб інтерпретації даних.

Формат операції:

```
reinterpret_cast <тип> (вираз)
```

тип – визначає той тип, до якого приводиться *вираз*.

Результат операції має вказаний тип, який може бути посиланням, вказівником, цілим або дійсним типом.

Приклад:

```
char *p = reinterpret_cast <char*>( malloc(100) );
long  l = reinterpret_cast <long> ( p );
```

Відмінності між `static_cast` та `reinterpret_cast`: перше перетворення змінює внутрішнє представлення даних, не змінюючи його значення; друге – змінює тлумачення даних (тобто, їх значення), не змінюючи внутрішнього представлення. Це дозволяє компілятору виконувати мінімальну перевірку при використанні `static_cast`.

Результат операції залишається на совісті програміста.

Перевантаження операції виклику функції

Клас, в якому визначено операцію виклику функції `()`, називається *функціональним класом* або *функтором*. Більш детально функтори розглядаються в Темі 6. Контейнери та шаблони.

Такий клас може і не мати полів чи інших методів:

```
class Is_Greater
{
public:
    bool operator ()(int a, int b)
    {
        return a > b;
    }
};
```

Операцію `operator ()` можна визначати лише як метод класу. Операція `operator ()` не може бути статичним методом. Операція `operator ()` може бути віртуальним методом.

Приклад використання:

```
Is_Greater x;
cout << x(1, 5) << endl;             // результат 0
cout << Is_Greater()(5, 1) << endl;  // результат 1
```

В першій команді виведення вираз `x(1,5)` означає `x.operator()(1,5)`.

В другій команді виведення вираз `Is_Greater()` використовується для виклику конструктора за умовчанням класу `Is_Greater`, результатом якого буде об'єкт цього класу, для якого викликається визначена в класі операція `()` з двома аргументами.

Перевантаження операції індексування

Операція індексування `[]` зазвичай перевантажується, коли клас містить набір значень, для яких індексування має зміст (тобто, коли клас містить поле-масив). Операція індексування має повертати посилання на елемент відповідного поля-масиву.

Операцію `[]` можна визначити лише як метод класу.

Отже, за наявності в класі поля-масиву, буде зручно перевантажити операцію індексування `[]`. Це – бінарна операція: її лівий операнд – поточний об'єкт, а правий – аргумент методу-операції. Вираз *об'єкт*`[` *індекс* `]` трактується як

об'єкт`.operator [](` *індекс* `)`

Операція `[]` (як і операція присвоєння) має повертати посилання, оскільки вираз *ім'я*`[` *індекс* `]` може бути записаний як ліворуч, так і праворуч знаку операції присвоєння.

Тип індексу може не збігатися з типом результуючого посилання; тип результату не обов'язково має бути типом класу, що визначається; тип індексу не обов'язково має бути цілим.

Зазвичай реалізують два методи: константний і неконстантний.

```
mun1& operator [](const mun2 &індекс);
const mun1& operator [](const mun2 &індекс) const;
```

Неконстантний метод спрацьовує, коли вираз *ім'я*`[` *індекс* `]` записують ліворуч від знаку операції присвоєння. Константний метод викликається, наприклад, для параметрів, які передаються за константним посиланням.

Перевантаження операцій `new` та `delete`

Для реалізації альтернативних варіантів управління пам'яттю можна визначити власні варіанти операцій `new` та `new []` для виділення динамічної пам'яті об'єкту та масиву об'єктів відповідно, а також операції `delete` та `delete []` для її звільнення.

Ці методи-операції мають задовольняти правилам:

- їм не потрібно передавати параметр типу класу;
- першим параметром для методів-операцій `new` та `new []` має передаватися розмір об'єкту типу `size_t` (це тип результату операції `sizeof()`), при виклику він передається неявним способом;
- вони мають бути визначені з типом результату `void*`, навіть якщо `return` повертає вказівник на інші типи (зазвичай – на клас);
- операція `delete` має мати тип результату `void` і перший аргумент типу `void*`;
- операції виділення та звільнення пам'яті мають бути статичними методами класу.

Приклад:

```
class Account
{
    char* name;               // власник
    double summa;             // сума

public:
    /* ... оголошення інших методів ... */

    static void* operator new(size_t size);
    static void  operator delete(void* objToDelete, size_t size);
};


void* Account::operator new(size_t size)
{
    // делегуємо спробу виділити невірну кількість пам'яті
    // стандартній операції new
    if (size != sizeof(Account)) return ::operator new(size);

    Account* p;
    /* ... своя реалізація виділення пам'яті ... */

    return p;
}

void  Account::operator delete(void* objToDelete, size_t size)
{
    if (objToDelete == 0) return;

    // делегуємо спробу звільнити невірну кількість пам'яті
    // стандартній операції delete
    if (size != sizeof(Account))
    {
        ::operator delete(objToDelete);
        return;
    }

    /* ... своя реалізація звільнення пам'яті... */
}
```

Масиви та класи

Визначивши деякий клас, ми можемо оголошувати масиви об’єктів цього типу. При цьому, для створення кожного елемента масиву викликається конструктор без аргументів (конструктор за умовчанням). Якщо в класі є конструктор ініціалізації, то елементи масиву можна ініціалізувати звичайним способом: для кожного елемента буде викликано конструктор ініціалізації (див. приклад в параграфі «Конструктор ініціалізації для елементів масиву»).

Поля-масиви в класі

В класі можна оголосити поле-масив.

Кількість елементів масиву можна вказати за допомогою явної константи чи константним виразом, що містить статичну чи перелічувану константу, причому константи мають бути визначені раніше масиву. Вказувати кількість елементів для поля-масиву – обов’язково:

```
class Array
{
    static const int k = 10;
    enum { n = 10 };

    int m0[10];
    int m1[k];
    int m2[n];

public:
    Array()
    {
        for (int i = 0; i < k; i++)
            m0[i] = m1[i] = m2[i] = 0;
    }
};
```

– оголошено три поля-масиви: `m0`, `m1` та `m2`.

Не можна вказувати кількість елементів як значення іншого поля, навіть якщо це поле – константне і визначається списком ініціалізації конструктора:

```
const int K;  // помилка: K не ім'я типу,
              // статична чи перелічувана константа
int m[K];
```

У наведеному прикладі конструктор присвоює всім елементам всіх трьох масивів нульові значення. Цей спосіб – в тілі конструктора – найпростіший і дозволяє присвоїти елементам масиву будь-які значення.

Якщо необхідно ініціалізувати масиви нульовими значеннями, це можна зробити в списку ініціалізації конструктора:

```
class Array
{
    static const int k = 10;
    enum { n = 10 };

    int m0[10];
    int m1[k];
    int m2[n];

public:
    Array()
        : m0(), m1(), m2() {}
};
```

Для масиву об’єктів деякого класу це означає виклик конструктора за умовчанням (без аргументів) для кожного елемента масиву.

В списку ініціалізації конструктора можлива лише така ініціалізація масиву, ніякого значення в дужках вказувати не можна.

Для константних масивів вбудованих типів – ще складніше: константи не можна ініціалізувати в тілі конструктора, значення їм можна присвоїти лише в списку ініціалізації:

```
class Array
{
    static const int k = 10;

    const int m3[k];

public:
    Array()
        : m3() {}
};
```

При використанні динамічних масивів в класі оголошується поле-вказівник на масив та поле – розмір масиву. Конструктор має виділяти пам’ять для масиву, а деструктор – звільняти її:

```
class Array
{
    int size;
    int* m;

public:
    Array(int n = 1)
        : size(n)
    {
        m = new int[size];

        for (int i = 0; i < size; i++)
            m[i] = 0;
    }
```

```
    ~Array()
    {
        delete[] m;
    }
};

int main()
{
    Array a;      // конструктор за умовчанням, size = 0
    Array b(10);  // конструктор ініціалізації, size = 10

    return 0;
}
```

Реалізація класу з полем-масивом

При реалізації класу з полем-масивом рекомендується перевизначити операції індексування та вводу-виводу. Конструктор копіювання та перевизначена операція присвоєння мають забезпечити копіювання кількості елементів та по-елементне копіювання масиву (див. наступний параграф).

Статичні поля-масиви

Статичні поля-масиви (як і статичні поля інших типів) існують в єдиному екземплярі на весь клас. Тобто, кількість створених екземплярів статичних полів не залежить від кількості створених об’єктів. Навіть якщо об’єктів цього класу немає, пам’ять для статичних полів – виділяється (бо вона виділяється в області, відведеній для класу, а не для об’єктів).

Це дозволить економити пам’ять: якщо потрібно використовувати одне єдине поле-масив для всього класу (для всіх об’єктів), то його слід оголосити статичним полем.

Динамічне виділення пам’яті

Головне правило: пам’ять, виділену в конструкторі, слід звільнити в деструкторі.

Клас – оболонка для динамічного масиву

Зауваження. Наступний приклад виконується не для всіх версій Microsoft Visual Studio: наступний конструктор за умовчанням

```
Array(int n = 0)
    : size( n )
{
    m = new int[size];

    for (int i=0; i<size; i++)
        m[i] = 0;
}
```

не виділяє пам’яті для масиву при `n = 0`, що в деяких версіях приводить до помилки при виконанні команди

```
delete [] m;
```

при виконанні операції присвоєння.

Щоб виправити зазначену помилку, слід в конструкторі присвоювати за умовчанням кількості елементів масиву значення 1:

```
Array(int n = 1)
    : size( n )
{
    m = new int[size];

    for (int i=0; i<size; i++)
        m[i] = 0;
}
```

Приклад:

```
#include <iostream>

using namespace std;

class Array
{
    int size;
    int* m;

public:
    Array(int n = 0)
        : size(n)
    {
        m = new int[size];

        for (int i = 0; i < size; i++)
            m[i] = 0;
    }
```

```
    Array(const Array& a)
        : size(a.size)
    {
        m = new int[size];

        for (int i = 0; i < size; i++)
            m[i] = a.m[i];
    }

    ~Array()
    {
        delete[] m;
    }

    Array& operator = (const Array& a)
    {
        if (&a == this) return *this;  // самоприсвоєння

        size = a.size;

        delete[] m;
        m = new int[size];

        for (int i = 0; i < size; i++)
            m[i] = a.m[i];

        return *this;
    }

    int& operator [] (const int i)
    {
        return m[i];
    }

    const int& operator [] (const int i) const
    {
        return m[i];
    }

    friend ostream& operator << (ostream& out, const Array& a);
    friend istream& operator >> (istream& in, Array& a);
};

ostream& operator << (ostream& out, const Array& a)
{
    for (int i = 0; i < a.size; i++)
        out << "array[ " << i << " ] = " << a[i] << endl;
    out << endl;

    return out;
}

istream& operator >> (istream& in, Array& a)
{
    for (int i = 0; i < a.size; i++)
    {
        cout << "array[ " << i << " ] = ? "; in >> a[i];
    }
    cout << endl;

    return in;
}
```

```
int main()
{
    Array a;                    // конструктор за умовчанням, size = 0
    Array b(10);                // конструктор ініціалізації, size = 10

    cin >> b;                   // використовується операція вводу

    cout << b[2] << endl;       // використовується операція індексування
                                // Без неї слід було би зробити поля доступними і
    cout << b.m[2] << endl;     // використовувати цей синтаксис

    cout << "b:" << endl << b; // використовується операція вbводу

    Array c = b;                // конструктор копіювання, size = b.size

    cout << "c:" << endl << c;

    Array d;                    // конструктор за умовчанням, size = 0
    d = c;                      // операція присвоєння, size = c.size
    cout << "d:" << endl << d;

    return 0;
}
```

Клас – оболонка для матриці

Операція індексування – це метод, який приймає лише один параметр, тому можна перевантажити лише індексування для класів, що містять одновимірні масиви («одновимірне» індексування).

Для перевантаження індексування для класів, що містять багатовимірні масиви (наприклад, матрицю) слід це реалізувати «каскадом»:

Спочатку описуємо клас «рядок матриці», в ньому перевантажуємо одновимірну операцію індексування.

Потім – реалізуємо клас «матриця» як одновимірний масив, елементи якого – типу «рядок матриці», в цьому класі реалізуємо свою одновимірну операцію індексування:

```
#include <iostream>
using namespace std;

class Matrix
{
    class Row;
    int R, C;
    Row* m;

    // внутрішній клас представляє рядок матриці
    class Row
    {
    public:
        int* v;

        Row(int C = 1)
        {
            v = new int[C];
```

```
            for (int j = 0; j < C; j++)
                v[j] = 0;
        }

        // деструктор не потрібний: якщо його написати, то рядки матриці
        // будуть видалятися до того, як заповниться вся матриця
        int& operator [] (int j)
        {
            return v[j];
        }
    };

public:
    Matrix(int R = 1, int C = 1)
    {
        this->R = R < 1 ? 1 : R;
        this->C = C < 1 ? 1 : C;

        m = new Row[this->R];
        for (int i = 0; i < this->R; i++)
        {
            // створення рядків матриці.
            // якщо підклас рядків матриці буде мати деструктор,
            // то рядки будуть видалятися в кожній ітерації циклу

            m[i] = Row(this->C);
        }
    }

    Row& operator [] (int i)
    {
        return m[i];
    }

    int getC() const { return C; }

    int getR() const { return R; }

    ~Matrix()
    {
        for (int i = 0; i < R; i++)
        {
            if (m[i].v != nullptr)
                delete[](m[i].v); // видалення рядків матриці
        }

        // видалення масиву вказівників на рядки матриці
        if (m != nullptr)
            delete[] m;
    }
};

int main()
{
    Matrix mas(2, 3);
    for (int i = 0; i < mas.getR(); i++)
    {
        for (int k = 0; k < mas.getC(); k++)
        {
            cout << mas[i][k] << "\t";
        }
        cout << endl;
```

```
    }

    system("pause");
}
```

Клас – оболонка для стеку

Реалізуємо клас-оболонку для стеку.

Зауважимо, що це не лише далеко не найкращий спосіб (бо в цьому класі дії з окремим елементом змішані з діями над усім стеком), він ще і приводить до помилок в деструкторі при знищенні об'єктів цього класу:

```
#include <iostream>

using namespace std;

class Stack
{
    Stack* link;
    int    data;

public:
    Stack() : link() {}             // конструктор - ініціалізує
                                    // вказівник link нулем
    ~Stack()
    {
        while (link)                // поки існують
            Pop();                  // знищуємо елементи
    }

    Stack* top() const              // повертає вказівник на вершину
    {
        return link;
    }

    void Push(const int value)
    {
        Stack* tmp = new Stack();
        tmp->data = value;
        tmp->link = link;
        link = tmp;
    }

    int Pop()
    {
        Stack* tmp = link->link;
        int value = data;
        delete link;                // тут викликається деструктор
        link = tmp;
        return value;
    }

    unsigned int Count() const
    {
        Stack* tmp = link;
        int count = 0;
        while (tmp)
        {
            count++;
```

```
            tmp = tmp->link;
        }
        return count;
    }

    friend ostream& operator << (ostream& out, const Stack& s);
    friend istream& operator >> (istream& in, Stack& s);
};

ostream& operator << (ostream& out, const Stack& s)
{
    Stack* tmp = s.top();
    while (tmp)
    {
        out << tmp->data << "  ";
        tmp = tmp->link;
    }
    out << endl;

    return out;
}

istream& operator >> (istream& in, Stack& s)
{
    int value;
    cout << "value = "; in >> value;
    s.Push(value);

    return in;
}

int main()
{
    {
        Stack s;
        cin >> s >> s >> s;
        cout << s << endl;
        cout << s.Count() << endl;
    }                                   // помилка звільнення
                                        // пам'яті при знищенні
                                        // локальної змінної

    return 0;
}
```

Причина помилки: деструктор в циклі викликає метод `Pop()` для знищення кожного елементу, а в тілі метода `Pop()` викликається операція `delete` для звільнення пам’яті, виділеної поточному елементу. Виконання цієї операції `delete` приводить до автоматичного виклику деструктора стеку.

Виправимо помилку: замість визначення власного деструктора будемо використовувати деструктор, який автоматично створить компілятор. Тепер для знищення стеку користувач сам мусить в циклі викликати метод `Pop()`:

```
#include <iostream>

using namespace std;
```

```
class Stack
{
    Stack* next;
    int    data;

public:
    Stack() : next() {}             // конструктор - ініціалізує
                                    // вказівник next нулем
    Stack* Top() const
    {                               // повертає вказівник на вершину
        return next;
    }

    void Push(const int value)
    {
        Stack* tmp = new Stack();
        tmp->data = value;
        tmp->next = next;
        next = tmp;
    }

    int Pop()
    {
        Stack* tmp = next->next;
        int value = data;
        delete next;                // тут викликається деструктор
        next = tmp;
        return value;
    }

    unsigned int Count() const
    {
        Stack* tmp = next;
        int count = 0;
        while (tmp)
        {
            count++;
            tmp = tmp->next;
        }
        return count;
    }

    friend ostream& operator << (ostream& out, const Stack& s);
    friend istream& operator >> (istream& in, Stack& s);
};

ostream& operator << (ostream& out, const Stack& s)
{
    Stack* tmp = s.Top();
    while (tmp)
    {
        out << tmp->data << "  ";
        tmp = tmp->next;
    }
    out << endl;

    return out;
}

istream& operator >> (istream& in, Stack& s)
{
    int value;
```

```
    cout << "value = "; in >> value;
    s.Push(value);

    return in;
}

int main()
{
    {
        Stack s;
        cin >> s >> s >> s;
        cout << s << endl;
        cout << s.Count() << endl;

        while (s.Top())
            s.Pop();
    }                           // звільнення пам'яті при знищенні
                                // локальної змінної виконується вірно
    return 0;
}
```

Нарешті, наведемо лад із поняттями Стек та Елемент_стеку. Щоб відобразити різницю між ними в нашому класі, інкапсулюємо елемент стеку в окрему структуру (той же клас, лише з відкритими елементами).

Реалізуємо наступні дії із стеком:

- добавлення елемента у стек – `Push()`;
- вилучення елемента із стеку – `Pop()`;
- отримання значення елемента з вершини стеку – `Top()`;
- перевірка, чи стек порожній – `Empty()`,

а також операції вводу-виводу та обчислення кількості елементів.

Для забезпечення універсальності нашого стеку опишемо тип даних як `void*`:

```
struct Elem
{
    Elem* next;
    void* data;
    Elem(Elem* n, void* d)        // конструктор елемента стеку
        : data(d), next(n) {}
};
```

Оскільки при цьому лише користувач (програма, яка використовує стек) знає справжній тип інформаційного поля, то нехай він і звільняє пам’ять, виділену для стеку, за допомогою метода `Pop()`. Отже, деструктор створювати не будемо, обійдемося автоматично створеним деструктором за умовчанням.

Конструктор ініціалізації теж не потрібний – оскільки на початку стек не містить жодного елемента. Присвоювати та копіювати стеки зазвичай теж не потрібно. Тому нам вистачить одного конструктора без аргументів, функція якого – вказівнику на вершину стеку присвоїти нульове значення. Нульове значення вказівника на вершину – признак відсутності елементів в стеку.

```
#include <iostream>

using namespace std;

class Stack                             // універсальний стек
{
    struct Elem                         // елемент стеку
    {
        Elem* next;
        void* data;
        Elem(Elem* n, void* d)          // конструктор елемента стеку
            : data(d), next(n) {}
    };

    Elem* head;                         // вершина стеку
    Stack(const Stack&);                // закрили копіювання
    Stack& operator = (const Stack&);   // закрили присвоєння

public:
    Stack() : head(0) {}                // конструктор - ініціалізує
                                        // вказівник на вершину нулем

    void* Top() const                   // значення з вершини стеку
    {
        return Empty() ? 0 : head->data;
    }

    bool Empty() const                  // стек порожній, якщо
    {                                   // вершина == 0
        return head == 0;
    }

    void Push(void* d)                  // добавити елемент
    {
        head = new Elem(head, d);
    }

    void* Pop()                         // вилучити елемент
    {
        if (Empty()) return 0;          // якщо стек - порожній,
                                        // то нічого не робити

        void* top = head->data;         // зберегли для повернення
        Elem* old = head;               // запам'ятали вказівник
        head = head->next;              // пересунули вершину
        delete old;                     // звільнили пам'ять
        return top;                     // повернули значення
    }                                   // елемента

    unsigned int Count() const
    {
        Elem* tmp = head;
        int count = 0;
        while (tmp)
        {
            count++;
            tmp = tmp->next;
        }
        return count;
    }

    friend ostream& operator << (ostream& out, const Stack& s);
```

```
    friend istream& operator >> (istream& in, Stack& s);
};


ostream& operator << (ostream& out, const Stack& s)
{
    Stack::Elem* tmp = s.head;
    while (tmp)
    {
        out << *(int*)tmp->data << " ";  // трактуємо data як int*
        tmp = tmp->next;
    }
    out << endl;

    return out;
}

istream& operator >> (istream& in, Stack& s)
{
    int value;
    cout << "value = "; in >> value;
    s.Push(new int(value));              // заносимо вказівник на int

    return in;
}

int main()
{
    {
        Stack s;
        cin >> s >> s >> s;
        cout << s << endl;
        cout << s.Count() << endl;

        while (s.Top())
            s.Pop();
    }

    return 0;
}
```

Лабораторний практикум

Оформлення звіту про виконання лабораторних робіт

Вимоги до оформлення звіту про виконання лабораторних робіт №№ 2.1–2.8

Звіт про виконання лабораторних робіт №№ 2.1–2.8 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 №№ 2.1–2.8

ЗВІТ

про виконання лабораторної роботи № < *номер* >

« *назва теми лабораторної роботи* »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента(ки) групи КН-27

< *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:

…

Лабораторна робота № 2.1. Конструктори та перевантаження операцій для класів з двома полями

Мета роботи

Освоїти використання конструкторів та перевантаження операцій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій.

У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і

- декременту в обох формах (префіксній та постфіксній).

При цьому префіксні операції інкременту, декременту модифікують поле first, а постфіксні – поле second.

Обчислити значення виразу $y = a\ x^2 + b\ x + c$ для комплексних коефіцієнтів a, b, c у комплексній точці x.

Текст програми

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

using namespace std;

int main() {
	Complex a(1, 1);
	Complex b(1, 1);
	Complex c(1, 1);

	cout << a++ << endl;
	cout << a   << endl;
	cout << ++a << endl;
	cout << a   << endl;

	Complex x;
	cout << "Input complex number ->" << endl;
	cin >> x;

	Complex y; y = a * (x ^ 2) + b * x + c;
	cout << "Result: y = a*(x^2) + b*x + c = " << endl;
	cout << y;

	cin.get();
	return 0;
}

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

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

using namespace std;

class Complex
{
	double re, im;
public:
	Complex();
	Complex(double, double);
	Complex(const Complex&);
	~Complex();
```

```
	void SetRe(double);
	void SetIm(double);
	double GetRe() const;
	double GetIm() const;

	Complex& operator = (const Complex&);
	operator string() const;

	friend Complex operator + (const Complex&, const Complex&);
	friend Complex operator - (const Complex&, const Complex&);
	friend Complex operator * (const Complex&, const Complex&);
	friend Complex operator / (const Complex&, const Complex&);
	friend Complex operator ^ (const Complex&, const unsigned);

	friend ostream& operator << (ostream&, const Complex&);
	friend istream& operator >> (istream&, Complex&);

	Complex& operator ++();
	Complex& operator --();
	Complex operator ++(int);
	Complex operator --(int);
};

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

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

Complex::Complex()
{
	re = 0;
	im = 0;
}

Complex::Complex(double r = 0, double i = 0)
{
	re = r;
	im = i;
}

Complex::Complex(const Complex& r)
{
	re = r.re;
	im = r.im;
}

Complex::~Complex()
{ }

void Complex::SetRe(double r)
{
	re = r;
}

void Complex::SetIm(double i)
{
	im = i;
}
```

```
double Complex::GetRe() const
{
	return re;
}

double Complex::GetIm() const
{
	return im;
}

Complex& Complex::operator = (const Complex& r)
{
	re = r.re;
	im = r.im;

	return *this;
}

Complex::operator string () const
{
	stringstream ss;
	ss << " Re = " << re << endl;
	ss << " Im = " << im << endl;

	return ss.str();
}

Complex operator + (const Complex& x, const Complex& y)
{
	return Complex(x.re + y.re, x.im + y.im);
}

Complex operator - (const Complex& x, const Complex& y)
{
	return Complex(x.re - y.re, x.im - y.im);
}

Complex operator * (const Complex& x, const Complex& y)
{
	return Complex(x.re * y.re - x.im * y.im,
		x.re * y.im + x.im * y.re);
}

Complex operator / (const Complex& x, const Complex& y)
{
	double r1 = x.re;
	double i1 = x.im;
	double r2 = y.re;
	double i2 = y.im;

	return Complex((r1 * r2 - i1 * i2) / (r2 * r2 + i2 * i2),
		(-r1 * i2 + i1 * r2) / (r2 * r2 + i2 * i2));
}

Complex operator ^ (const Complex& x, const unsigned n)
{
	Complex y(1, 0);
	for (unsigned i = 1; i <= n; i++)
		y = y * x;
	return y;
}
```

```
ostream& operator << (ostream& out, const Complex& r)
{
	out << string(r);
	return out;
}

istream& operator >> (istream& in, Complex& r)
{
	cout << " Re = "; in >> r.re;
	cout << " Im = "; in >> r.im;
	cout << endl;
	return in;
}

Complex& Complex::operator ++()
{
	re++;
	return *this;
}

Complex& Complex::operator --()
{
	re--;
	return *this;
}

Complex Complex::operator ++(int)
{
	Complex t(*this);
	im++;
	return t;
}

Complex Complex::operator --(int)
{
	Complex t(*this);
	im--;
	return t;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій.

У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,

- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній).

При цьому префіксні операції інкременту, декременту модифікують поле first, а постфіксні – поле second.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

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

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.1 (Класи з двома полями), реалізувавши для кожного класу вказані конструктори та операції. Функції введення / виведення оформити як дружні.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

Лабораторна робота № 1.1:

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

- 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 відповідного номіналу. 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 певну 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 відповідного номіналу. 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 певну 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 № 2.2. Перевантаження операцій

Мета роботи

Освоїти використання конструкторів та перевантаження операцій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

Для всіх варіантів клас має мати конструктор за умовчанням, конструктор ініціалізації та конструктор копіювання; слід перевантажити операції введення-виведення.

Створити клас *цілі числа*. Визначити операції: `+` додавання, `-` віднімання, `*` множення, `/` ділення, `%` залишок від ціло-чисельного ділення, `[]` перевірка числа на парність, `^` піднесення до степеню, `~` перевірка, чи число є простим. Визначити потокові операції введення-виведення.

Текст програми

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

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

using namespace std;

int main()
{
    Integer a, b;
    cout << "a = ? "; cin >> a;
```

```
    cout << "b = ? "; cin >> b;
    cout << endl;

    cout << "a + b = " << a + b;
    cout << "a - b = " << a - b;
    cout << "a * b = " << a * b;
    cout << "a / b = " << a / b;
    cout << "a % b = " << a % b;
    cout << "a ^ b = " << (a ^ b) << endl;

    if (~a)
        cout << "a - prime number" << endl;
    else
        cout << "a - not prime number" << endl;

    if (a[2])
        cout << "a - even number" << endl;
    else
        cout << "a - not even number" << endl;

    return 0;
}

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

#pragma once
#include <iostream>

using namespace std;

class Integer
{
    int x;

public:
    int GetX() const { return x; }
    void SetX(int value) { x = value; }

    Integer();
    Integer(int);
    Integer(const Integer&);

    friend Integer operator + (Integer&, Integer&);
    friend Integer operator - (Integer&, Integer&);
    friend Integer operator * (Integer&, Integer&);
    friend Integer operator / (Integer&, Integer&);
    friend Integer operator % (Integer&, Integer&);
    friend Integer operator ^ (Integer&, Integer&);

    bool operator [] (int);
    bool operator ~ ();

    friend ostream& operator << (ostream&, const Integer&);
    friend istream& operator >> (istream&, Integer&);
};

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

```cpp
#include "Integer.h"
#include <iostream>

using namespace std;

Integer::Integer()
{
    x = 0;
}

Integer::Integer(int y)
{
    x = y;
}

Integer::Integer(const Integer& r)
{
    x = r.x;
}

Integer operator + (Integer& a, Integer& b)
{
    Integer t(0);
    t.x = a.x + b.x;
    return t;
}

Integer operator - (Integer& a, Integer& b)
{
    Integer t(0);
    t.x = a.x - b.x;
    return t;
}

Integer operator * (Integer& a, Integer& b)
{
    Integer t(0);
    t.x = a.x * b.x;
    return t;
}

Integer operator / (Integer& a, Integer& b)
{
    if (b.x)
    {
        Integer t(0);
        t.x = a.x / b.x;
        return t;
    }
    else
    {
        cerr << "Error!" << endl;
        return -1;
    }
}

Integer operator % (Integer& a, Integer& b)
{
    if (a.x)
    {
        Integer t(0);
        t.x = a.x / b.x;
```

```
        return t;
    }
    else
    {
        cerr << "Error!" << endl;
        return -1;
    }
}

Integer operator ^ (Integer& a, Integer& b)
{
    Integer t(0);

    if (a.x == 0)
        return t;    // 0 в будь-якій степені = 0
    else
        t.x = 1;     // початкове значення = 1

    if (b.x == 0)    // будь-яке число в степені 0
        return t;    // = 1

    if (b.x > 0)
    {
        for (int i = 1; i <= b.x; i++)
            t.x *= a.x;
    }
    if (b.x < 0)
    {
        for (int i = 1; i <= b.x; i++)
            t.x /= a.x;
    }
    return t;
}

bool Integer::operator [] (int i)
{
    return x % i == 0;
}

bool Integer::operator ~ ()
{
    int n = 0;
    for (int i = 2; i < x / 2; i++)
        if (x % i == 0)
            n++;

    return n == 0;
}

ostream& operator << (ostream& out, const Integer& a)
{
    out << a.x << endl;
    return out;
}

istream& operator >> (istream& in, Integer& a)
{
    in >> a.x;
    return in;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій.

У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

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

Варіант 1.

Створити клас *цілі числа*. Визначити операцію ++, як метод класу, унарну операцію - та бінарну операцію +, як дружні функції. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 2.

Створити клас *цілі числа*. Визначити бінарну операцію +, як метод класу та унарну операцію - як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 3.*

Створити клас *вектор*, який містить вказівник на int – вказівник на елементи вектора, та кількість елементів вектора. Клас має мати конструктор за умовчанням, конструктор з одним і двома параметрами, конструктор копіювання і деструктор. Визначити операції +, -, * – як дружні функції, =, +=, -=, *=, [ ] – як методи класу. Перевантажити потокові операції введення і виведення елементів вектора.

Варіант 4.*

Створити клас *матриця*, який містить вказівник на вказівник на int – вказівник на елементи матриці, та кількість рядків і стовпчиків. Визначити конструктор за умовчанням, конструктор з одним і з двома параметрами, конструктор копіювання, деструктор. Визначити операції =, +, -, *, +=, -=, *= над об'єктами цього класу. Перевантажити потокові операції введення і виведення елементів матриці.

Варіант 5.

Створити клас *цілі числа*. Визначити унарну операцію - як метод класу та операцію ++ як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 6.

Створити клас *координати точки*. Визначити операцію + як метод класу та операцію - як дружню функцію, ці операції мають реалізувати додавання і віднімання координат двох точок, координати точки з числом. Визначити операцію присвоєння = та операції порівняння ==, != координат. Визначити потокові операції введення-виведення.

Варіант 7.

Створити клас *дійсні числа*. Визначити операцію -- як метод класу та бінарну операцію - як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 8.*

Створити клас *вектор*, який містить вказівник на float – вказівник на елементи вектора та розмірність вектора. Клас має мати конструктор за умовчанням, конструктор з одним і двома параметрами, конструктор копіювання і деструктор. Визначити операції +, -,

* – як дружні функції, =, +=, -=, *=, [ ] – як методи класу. Перевантажити потокові операції введення-виведення.

Варіант 9.*

Створити клас *матриця*, який містить вказівник на вказівник на float – вказівник на елементи матриці, кількість рядків і стовпців. Визначити конструктор за умовчанням, конструктор з одним і з двома параметрами, конструктор копіювання, деструктор. Визначити операції =, +, -, +=, -=, *= над об'єктами цього класу Перевантажити потокові операції введення-виведення.

Варіант 10.*

Створити клас *черга*. Перевантажити операцію ++ як метод класу, та -- як дружню функцію. Операція ++ додає елемент у чергу, а -- вилучає елемент із черги. Операція ! перевіряє чергу на наявність елементів. Розробити потокові операції формування та виведення вмісту черги.

Варіант 11.*

Створити клас *стек*. Перевантажити операції + – як метод класу та * – як дружню функцію. Операція + записує елемент у стек, а операція * множить значення, розміщене у вершині стеку, на число. Розробіть операції присвоєння стеків =, перевірки на рівність == або нерівність !=, потокового введення-виведення стеків, додавання += елемента у стек.

Варіант 12.*

Створити клас *двонапрямлений список*, у якому визначені операції: + – додає елемент в кінець списку, += – додає елемент на початок списку, - – видаляє зазначений елемент зі списку, = – присвоєння списків, ==, !=, >, <, >=, <= – порівняння списків, [ ] – одержання елемента списку, ++ – встановлення вказівника на наступний елемент, -- – встановлення вказівника на попередній елемент, ( ) – вивести частину списку між заданими елементами.

Варіант 13.*

Створити клас *однонапрямлений список*, у якому визначені операції: + – додає елемент в кінець списку, += – додає елемент на початок списку, - – видаляє зазначений елемент зі списку, = – присвоєння списків, ==, !=, >, <, >=, <= – порівняння списків, [ ] – одержання елемента списку, ++ – встановлення вказівника на наступний елемент, -- – встановлення вказівника на попередній елемент, ( ) – вивести частину списку між заданими

елементами.

Варіант 14.

Визначити клас *комплексне число*, перевантаживши операції +, -, ++, --, +=, -=, *, /, *=, /= та операції потокового введення-виведення.

Варіант 15.

Створити клас *комплексне число*. Перевантажити операції порівняння комплексних чисел !=, ==, >, <, >=, <= та операції потокового введення-виведення.

Варіант 16.*

Створити клас *рядок символів*. Перевантажити операції потокового введення-виведення та операції: + конкатенації рядків, - видалення підрядка із рядка, * пошуку позиції входження підрядка у рядок.

Варіант 17.*

Створити клас *рядок символів*. Перевантажити операції потокового введення-виведення та операції порівняння рядків: =, !=, >, >=, <, <=.

Варіант 18.*

Створити клас *множина цілих чисел*. Перевантажити операції: + об'єднання множин, - різниці множин, * перетину множин та операції потокового введення-виведення множини.

Варіант 19.*

Створити клас *множина цілих чисел*. Перевантажити операції: <= перевірка на входження «є підмножиною», >= перевірка на охоплення «є над-множиною», == перевірка тотожності множин, != перевірка нетотожності множин, ^ перевірка входження елемента у множину та операції потокового введення-виведення множини.

Варіант 20.

Створити клас *дійсні числа*. Визначити операції: + додавання, - віднімання, * множення, / ділення, () дробова частина числа, [] ціла частина числа. Визначити потокові операції введення-виведення.

Варіант 21.

Створити клас *цілі числа*. Визначити операції: + додавання, - віднімання, * множення, / ділення, % залишок від ціло-чисельного ділення, [ ] перевірка числа на парність, ^ піднесення до степеня, ~ перевірка, чи число є простим. Визначити потокові операції введення-виведення.

Варіант 22.

Створити клас *цілі числа*. Визначити операцію ++, як метод класу, унарну операцію - та бінарну операцію +, як дружні функції. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 23.

Створити клас *цілі числа*. Визначити бінарну операцію +, як метод класу та унарну операцію - як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 24.*

Створити клас *вектор*, який містить вказівник на int – вказівник на елементи вектора, та кількість елементів вектора. Клас має мати конструктор за умовчанням, конструктор з одним і двома параметрами, конструктор копіювання і деструктор. Визначити операції +, -, * як дружні функції, =, +=, -=, *=, [] як методи класу. Перевантажити потокові операції введення і виведення елементів вектора.

Варіант 25.*

Створити клас *матриця*, який містить вказівник на вказівник на int – вказівник на елементи матриці, та кількість рядків і стовпчиків. Визначити конструктор за умовчанням, конструктор з одним і з двома параметрами, конструктор копіювання, деструктор. Визначити операції =, +, -, *, +=, -=, *= над об’єктами цього класу. Перевантажити потокові операції введення і виведення елементів матриці.

Варіант 26.

Створити клас *цілі числа*. Визначити унарну операцію - як метод класу та операцію ++ як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації,

конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 27.

Створити клас *координати точки*. Визначити операцію + як метод класу та операцію - як дружню функцію, ці операції мають реалізувати додавання і віднімання координат двох точок, координати точки з числом. Визначити операцію присвоєння = та операції порівняння ==, != координат. Визначити потокові операції введення-виведення.

Варіант 28.

Створити клас *дійсні числа*. Визначити операцію -- як метод класу та бінарну операцію - як дружню функцію. Клас має мати конструктор за умовчанням, конструктор ініціалізації, конструктор копіювання і деструктор. Визначити потокові операції введення-виведення.

Варіант 29.*

Створити клас *вектор*, який містить вказівник на float – вказівник на елементи вектора та розмірність вектора. Клас має мати конструктор за умовчанням, конструктор з одним і двома параметрами, конструктор копіювання і деструктор. Визначити операції + - * як дружні функції, = += -= *= [] як методи класу. Перевантажити потокові операції введення-виведення.

Варіант 30.*

Створити клас *матриця*, який містить вказівник на вказівник на float – вказівник на елементи матриці, кількість рядків і стовпців. Визначити конструктор за умовчанням, конструктор з одним і з двома параметрами, конструктор копіювання, деструктор. Визначити операції = + - += -= *= над об’єктами цього класу Перевантажити потокові операції введення-виведення.

Варіант 31.*

Створити клас *черга*. Перевантажити операцію ++ як метод класу, та -- як дружню функцію. Операція ++ додає елемент у чергу, а -- вилучає елемент із черги. Операція ! перевіряє чергу на наявність елементів. Розробити потокові операції формування та виведення вмісту черги.

Варіант 32.*

Створити клас *стек*. Перевантажити операції + як метод класу та * як дружню

функцію. Операція + записує елемент у стек, а операція * множить значення, розміщене у вершині стеку, на число. Розробіть операції присвоєння стеків =, перевірки на рівність == або нерівність !=, потокового введення-виведення стеків, додавання += елемента у стек.

Варіант 33.*

Створити клас *двонапрямлений список*, у якому визначені операції: + додає елемент в кінець списку, += додає елемент на початок списку, - видаляє зазначений елемент зі списку, = присвоєння списків, ==, !=, >, <, >=, <= порівняння списків, [] одержання елемента списку, ++ встановлення вказівника на наступний елемент, -- встановлення вказівника на попередній елемент, () вивести частину списку між заданими елементами.

Варіант 34.*

Створити клас *однонапрямлений список*, у якому визначені операції: + додає елемент в кінець списку, += додає елемент на початок списку, - видаляє зазначений елемент зі списку, = присвоєння списків, ==, !=, >, <, >=, <= порівняння списків, [] одержання елемента списку, ++ встановлення вказівника на наступний елемент, -- встановлення вказівника на попередній елемент, () вивести частину списку між заданими елементами.

Варіант 35.

Визначити клас *комплексне число*, перевантаживши операції +, -, ++, --, +=, -=, *, /, *=, /= та операції потокового введення-виведення.

Варіант 36.

Створити клас *комплексне число*. Перевантажити операції порівняння комплексних чисел !=, ==, >, <, >=, <= та операції потокового введення-виведення.

Варіант 37.*

Створити клас *рядок символів*. Перевантажити операції потокового введення-виведення та операції: + конкатенації рядків, - видалення підрядка із рядка, * пошуку позиції входження підрядка у рядок.

Варіант 38.*

Створити клас *рядок символів*. Перевантажити операції потокового введення-виведення та операції порівняння рядків: =, !=, >, >=, <, <=.

Варіант 39.*

Створити клас *множина цілих чисел*. Перевантажити операції: + об’єднання множин, - різниці множин, * перетину множин та операції потокового введення-виведення множини.

Варіант 40.*

Створити клас *множина цілих чисел*. Перевантажити операції: <= перевірка на входження «є підмножиною», >= перевірка на охоплення «є над-множиною», == перевірка тотожності множин, != перевірка нетотожності множин, ^ перевірка входження елемента у множину та операції потокового введення-виведення множини.

Лабораторна робота № 2.3. Конструктори та перевантаження операцій для класів

Мета роботи

Освоїти використання конструкторів та перевантаження операцій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – зміст цих операцій

уточнено далі (див. Варіанти завдань).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

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

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

- година,
- хвилина і
- секунда.

Префіксні операції інкременту, декременту модифікують годину, а постфіксні – хвилину.

Текст програми

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

#include "Time.h"

using namespace std;

int main()
{
    Time t1, t2(23, 59, 14);
    cout << "time t1:" << endl;
    cin >> t1;

    cout << t1 << endl << endl;

    Time t3;
    cout << "t2 = " << t2 << endl;
    t3 = ++t2;
    cout << "t3 = ++t2:  \n" << "t2 = " << t2 << "t3 = " << t3 << endl;
    t3 = --t2;
    cout << "t3 = --t2:  \n" << "t2 = " << t2 << "t3 = " << t3 << endl;
    t3 = t2++;
    cout << "t3 = t2++:  \n" << "t2 = " << t2 << "t3 = " << t3 << endl;
    t3 = t2--;
    cout << "t3 = t2--:  \n" << "t2 = " << t2 << "t3 = " << t3 << endl;

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

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

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

using namespace std;

class Time
{
    unsigned int hh, mm, ss;

public:
    unsigned GetH() const { return hh; }
    unsigned GetM() const { return mm; }
    unsigned GetS() const { return ss; }
    void SetH(unsigned value);
    void SetM(unsigned value);
    void SetS(unsigned value);

    Time();
    Time(unsigned int, unsigned int, unsigned int);
    Time(const Time&);

    operator string() const;

    Time& operator ++();
    Time& operator --();
    Time operator ++(int);
    Time operator --(int);

    friend ostream& operator <<(ostream&, const Time&);
    friend istream& operator >>(istream&, Time&);
};

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

#include "Time.h"

#include <iostream>
#include <iomanip>
#include <cmath>
#include <stdlib.h>
#include <sstream>

using namespace std;

void Time::SetH(unsigned value)
{
    if (value >= 0 && value < 24)
    {
        hh = value;
    }
    else
    {
        cout << "Error in value of hours!" << endl;
```

```
    }
}

void Time::SetM(unsigned value)
{
    if (value >= 0 && value < 60)
    {
        mm = value;
    }
    else
    {
        cout << "Error in value of minutes!" << endl;
    }
}

void Time::SetS(unsigned value)
{
    if (value >= 0 && value < 60)
    {
        ss = value;
    }
    else
    {
        cout << "Error in value of seconds!" << endl;
    }
}

Time::Time()
    : hh(0), mm(0), ss(0)
{}

Time::Time(unsigned int h, unsigned int m, unsigned int s)
{
    SetH(h); SetM(m); SetS(s);
}

Time::Time(const Time& t)
{
    *this = t;
}

Time::operator string() const
{
    stringstream sout;
    sout << setfill('0') << setw(2) << hh << ":"
                         << setw(2) << mm << ":" << setw(2) << ss;

    return sout.str();
}

Time& Time::operator ++()
{
    if (hh == 23)
        hh = 0;
    else
        ++hh;

    return *this;
}

Time& Time::operator --()
{
```

```
    if (hh == 0)
        hh = 23;
    else
        --hh;

    return *this;
}

Time Time::operator ++(int)
{
    Time tmp = *this;
    if (mm == 59)
    {
        mm = 0;
        ++* this;
    }
    else
        ++mm;

    return tmp;
}

Time Time::operator --(int)
{
    Time tmp = *this;
    if (mm == 0)
    {
        mm = 59;
        --* this;
    }
    else
        --mm;

    return tmp;
}

ostream& operator <<(ostream& out, const Time& a)
{
    out << string(a) << endl;
    return out;
}

istream& operator >>(istream& in, Time& a)
{
    cout << "Hour  ";
    in >> a.hh;
    cout << "Min  ";
    in >> a.mm;
    cout << "Sec  ";
    in >> a.ss;

    return in;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – зміст цих операцій визначити самостійно.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

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

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.3. «Об'єкти – параметри методів (дії над кількома об'єктами)» як незалежні класи з конструкторами і перевантаженням операцій.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

Лабораторна робота № 1.3:

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 – різними конструкторами. 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() $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + a_2 \cdot y_1)$;
- 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 поточний об'єкт):

- обчислення значення value(), *a* / *b*;

```
double Rational::value(){
   return 1.*a/b;
}

Rational z;
...
double x = z.value();
```

бінарні операції (перший аргумент – поточний 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,

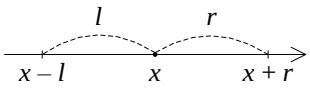
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

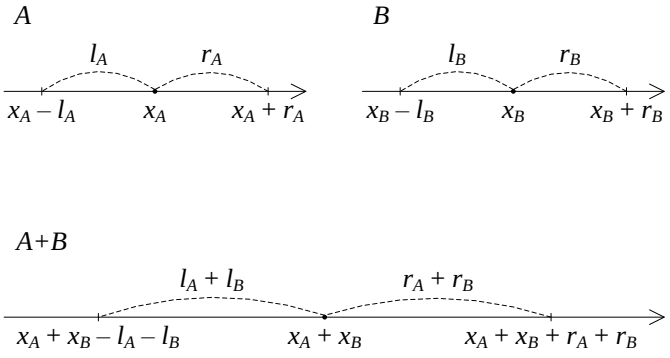
r – відстань від правої границі до центру:



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.

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 поточний об'єкт):

- обчислення значення value(), *a* / *b*;

```
double Rational::value(){
   return 1.*a/b;
}

Rational z;
...
```

```
double x = z.value();
```

бінарні операції (перший аргумент – поточний 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,

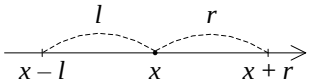
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

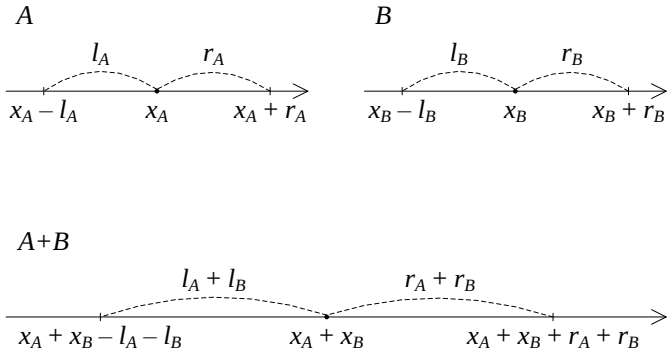
r – відстань від правої границі до центру:



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() $(x_1, y_1) \times (x_2, y_2) = (x_1 \cdot x_2 - y_1 \cdot y_2, x_1 \cdot y_2 + a_2 \cdot y_1)$;
- 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 27.

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 поточний об'єкт):

- обчислення значення value(), *a* / *b*;

```
double Rational::value(){
   return 1.*a/b;
}

Rational z;
```

```
...
double x = z.value();
```

бінарні операції (перший аргумент – поточний 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,

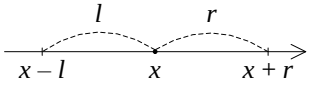
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

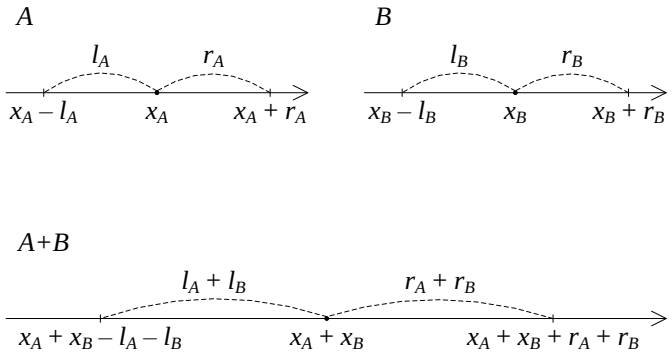
r – відстань від правої границі до центру:



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.

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 № 2.4. Масиви та константи в класі

Мета роботи

Освоїти використання масивів та констант в класі.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються

як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Додатково до потрібних в завданнях операцій слід перевантажити операцію:

- індексування [ ].

Максимально можливий розмір масиву задати константою. У окремому полі size має зберігатися максимальна для цього об'єкту кількість елементів масиву; реалізувати метод size( ), що повертає встановлену довжину. Якщо кількість елементів масиву змінюється під час роботи, визначити в класі поле count. Початкові значення size і count встановлюються конструктором.

У тих завданнях, де можливо, реалізувати конструктор ініціалізації літерним рядком.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання

Навантаження викладача за навчальний рік – це список дисциплін, що викладаються ним протягом року. Одна дисципліна представляється інформаційною структурою з полями: назва дисципліни, семестр проведення, кількість студентів, кількість аудиторних годин лекцій, кількість аудиторних годин практики, вид контролю (залік або іспит). Реалізувати клас Workload, що моделює бланк призначеного викладачеві навантаження. Клас містить прізвище викладача, дату затвердження, список дисциплін, що викладаються, обсяг повного навантаження в годинах і в ставках. Дисципліни в списку не мають повторюватися. Обсяг в ставках обчислюється як результат від ділення обсягу в годинах на середню річну ставку, однакову для всіх викладачів кафедри. Елемент списку дисциплін, що викладаються, містить дисципліну, кількість годин, що виділяється на залік (0,2 год. на одного студента) або іспит (0,3 год. на студента), суму годин по дисципліні. Реалізувати додавання і видалення дисциплін; обчислення сумарного навантаження в годинах і ставках. Має здійснюватися контроль за перевищення навантаження – не більш, ніж півтори ставки.

Текст програми

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

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

```cpp
using namespace std;

int menu()
{
    int i;
    cout << endl;
    cout << "Enter your choise:" << endl << endl;
    cout << " 1 - input data" << endl;
    cout << " 2 - print data" << endl;
    cout << " 3 - add discipline" << endl;
    cout << " 4 - delete discipline" << endl;
    cout << " 0 - exit" << endl << endl;
    cin >> i;
    return i;
}

int main()
{
    Workload a;
    sDiscipline sd;
    int no;

    int i;
    do
        switch (i = menu())
        {
        case 1:
            cin >> a;
            break;
        case 2:
            cout << a;
            break;
        case 3:
            cout << "Enter discipline to add:" << endl;
            cin >> sd;
            a.AddDiscipline(sd);
            break;
        case 4:
            cout << "Enter # discipline to delete:" << endl;
            cin >> no;
            a.DelDiscipline(no);
            break;
        case 0:
            break;
        default:
            cout << "Incorrect value!" << endl << endl;
        }
    while (i != 0);

    return 0;
}



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

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

```
using namespace std;

class Discipline
{
    string       name;   // назва
    unsigned int noSem,  // № семестру
                 kStud;  // к-ть студентів
    unsigned int lec,    // годин лекцій
                 prac;   // годин практичних
    bool         exam;   // форма контролю (екзамен = true)

public:
    Discipline();
    Discipline(string,
        unsigned int,
        unsigned int,
        unsigned int,
        unsigned int,
        bool);
    Discipline(const Discipline&);

    bool isExam();
    string getName();
    int getNoSem();
    int getKStud();
    int getLec();
    int getPrac();

    void setNoSem(int value);
    void setKStud(int value);

    Discipline& operator ++ ();
    Discipline& operator -- ();
    Discipline operator ++ (int);
    Discipline operator -- (int);

    Discipline& operator = (const Discipline&);
    friend bool operator == (Discipline&, Discipline&);
    operator string () const;
    unsigned int Summa();

    friend ostream& operator << (ostream&, const Discipline&);
    friend istream& operator >> (istream&, Discipline&);
};

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

#include "Discipline.h"
#include <string>
#include <iostream>
#include <sstream>
using namespace std;

Discipline::Discipline()
{
    name = "";     // назва
    noSem = 0;     // № семестру
    kStud = 0;     // к-ть студентів
    lec = 0;       // годин лекцій
    prac = 0;      // годин практичних
```

```
    exam = false;  // форма контролю (екзамен = true)
}

Discipline::Discipline(string name,
    unsigned int noSem,
    unsigned int kStud,
    unsigned int lec,
    unsigned int prac,
    bool exam)
{
    this->name = name;   // назва
    this->noSem = noSem; // № семестру
    this->kStud = kStud; // к-ть студентів
    this->lec = lec;     // годин лекцій
    this->prac = prac;   // годин практичних
    this->exam = exam;   // форма контролю (екзамен = true)
}

Discipline::Discipline(const Discipline& d)
{
    this->name = d.name;   // назва
    this->noSem = d.noSem; // № семестру
    this->kStud = d.kStud; // к-ть студентів
    this->lec = d.lec;     // годин лекцій
    this->prac = d.prac;   // годин практичних
    this->exam = d.exam;   // форма контролю (екзамен = true)
}

bool Discipline::isExam()
{
    return exam;
}

string Discipline::getName()
{
    return name;
}

int Discipline::getNoSem()
{
    return noSem;
}

int Discipline::getKStud()
{
    return kStud;
}

int Discipline::getLec()
{
    return lec;
}

int Discipline::getPrac()
{
    return prac;
}

void Discipline::setNoSem(int value)
{
    noSem = value;
}
```

```
void Discipline::setKStud(int value)
{
    kStud = value;
}

Discipline& Discipline::operator ++ ()
{
    noSem++;
    return *this;
}

Discipline& Discipline::operator -- ()
{
    noSem--;
    return *this;
}

Discipline Discipline::operator ++ (int)
{
    Discipline t(*this);
    kStud++;
    return t;
}

Discipline Discipline::operator -- (int)
{
    Discipline t(*this);
    kStud--;
    return t;
}


Discipline& Discipline::operator = (const Discipline& d)
{
    this->name = d.name;   // назва
    this->noSem = d.noSem; // № семестру
    this->kStud = d.kStud; // к-ть студентів
    this->lec = d.lec;     // годин лекцій
    this->prac = d.prac;   // годин практичних
    this->exam = d.exam;   // форма контролю (екзамен = true)

    return *this;
}

bool operator == (Discipline& l, Discipline& r)
{
    return l.name == r.name;
}


Discipline::operator string () const
{
    stringstream ss;
    ss  << "  name:  " << name
        << "  noSem: " << noSem
        << "  kStud: " << kStud
        << "  lek:   " << lec
        << "  prac:  " << prac
        << "  exam:  " << exam;
    return ss.str();
```

```
}

unsigned int Discipline::Summa()
{
    return lec + prac;
}

ostream& operator << (ostream& out, const Discipline& d)
{
    out << string(d);
    return out;
}

istream& operator >> (istream& in, Discipline& d)
{
    cout << "  name:       "; in >> d.name;
    cout << "  noSem:      "; in >> d.noSem;
    cout << "  kStud:      "; in >> d.kStud;
    cout << "  lek:        "; in >> d.lec;
    cout << "  prak:       "; in >> d.prac;
    cout << "  exam (0/1): "; in >> d.exam;

    return in;
}

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

#pragma once

#include "Discipline.h"
#include <iostream>
#include <string>

using namespace std;

// допоміжний клас, всі елементи якого - доступні

struct sDiscipline
{
    Discipline discipline;
    double     examTime;
    double     amount;

    sDiscipline();
    sDiscipline(Discipline);
    sDiscipline(const sDiscipline&);

    sDiscipline& operator ++ ();
    sDiscipline& operator -- ();
    sDiscipline operator ++ (int);
    sDiscipline operator -- (int);

    operator string () const;
    sDiscipline& operator = (const sDiscipline&);
    friend bool operator == (sDiscipline&, sDiscipline&);
    double Summa();

    friend ostream& operator << (ostream&, const sDiscipline&);
    friend istream& operator >> (istream&, sDiscipline&);
};
```

```
class Workload
{
    string name;                // прізвище
    string date;                // дата затвердження

    int size;                   // максимальна кількість елементів масиву
    int count;                  // кількість елементів масиву
    sDiscipline* disciplines;   // вказівник на масив дисциплін

    double totalAmount;         // обсяг навантаження в годинах
    double salaryRate;          // доля ставки зарплати

    static double avgAmount;    // середнє навантаження на ставку

public:
    Workload();
    Workload(string, string);
    Workload(const Workload&);
    ~Workload();

    int getSize() const;
    int getCount() const;
    void setSize(int value);

    void AddDiscipline(sDiscipline&);        // добавити дисципліну
    sDiscipline DelDiscipline();             // вилучити дисципліну
    sDiscipline DelDiscipline(int);          // вилучити дисципліну
    sDiscipline DelDiscipline(string);       // вилучити дисципліну
    sDiscipline DelDiscipline(sDiscipline&); // вилучити дисципліну

    Workload& operator = (const Workload&);
    sDiscipline& operator [] (int);
    operator string () const;

    Workload& operator ++ ();
    Workload& operator -- ();
    Workload operator ++ (int);
    Workload operator -- (int);
    double Summa();             // сумарне навантаження в годинах
    double SalaryRate();        // сумарне навантаження в ставках

    friend ostream& operator << (ostream&, const Workload&);
    friend istream& operator >> (istream&, Workload&);

    bool CheckWorkload();       // перевірка навантаження
};                              // - чи не перевищує 1,5 ставки

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

#include "Workload.h"
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

double Workload::avgAmount = 600;
```

```
// sDiscipline

sDiscipline::sDiscipline()
{
    examTime = 0;
    amount = 0;
}

sDiscipline::sDiscipline(Discipline d)
{
    discipline = d;
    examTime = discipline.isExam()
        ? discipline.getKStud() * 0.3
        : discipline.getKStud() * 0.2;
    amount = examTime + discipline.getLec() + discipline.getPrac();
}

sDiscipline::sDiscipline(const sDiscipline& sd)
{
    this->discipline = sd.discipline;
    this->examTime = sd.examTime;
    this->amount = sd.amount;
}

sDiscipline& sDiscipline::operator ++ ()
{
    int noSem = discipline.getNoSem();
    noSem++;
    discipline.setNoSem(noSem);
    return *this;
}

sDiscipline& sDiscipline::operator -- ()
{
    int noSem = discipline.getNoSem();
    noSem--;
    discipline.setNoSem(noSem);
    return *this;
}

sDiscipline sDiscipline::operator ++ (int)
{
    sDiscipline st(*this);

    int kStud = discipline.getKStud();
    kStud++;
    discipline.setKStud(kStud);
    return st;
}

sDiscipline sDiscipline::operator -- (int)
{
    sDiscipline st(*this);

    int kStud = discipline.getKStud();
    kStud--;
    discipline.setKStud(kStud);
    return st;
}

bool operator == (sDiscipline& l, sDiscipline& r)
{
```

```
    return l.discipline == r.discipline;
}

sDiscipline::operator string () const
{
    stringstream ss;
    ss  << "  discipline:  " << discipline
        << "  examTime:    " << examTime
        << "  amount:      " << amount;
    return ss.str();
}

sDiscipline& sDiscipline::operator = (const sDiscipline& sd)
{
    this->discipline = sd.discipline;
    this->examTime = sd.examTime;
    this->amount = sd.amount;

    return *this;
}

double sDiscipline::Summa()
{
    examTime = discipline.isExam()
        ? discipline.getKStud() * 0.3
        : discipline.getKStud() * 0.2;
    amount = examTime + discipline.getLec() + discipline.getPrac();
    return amount;
}

ostream& operator << (ostream& out, const sDiscipline& sd)
{
    out << string(sd);
    return out;
}

istream& operator >> (istream& in, sDiscipline& sd)
{
    cout << "discipline:" << endl;
    in >> sd.discipline;
    return in;
}

// Workload

Workload::Workload()
{
    name = "";
    date = "";

    size = 0;
    count = 0;
    disciplines = new sDiscipline[size];

    totalAmount = 0;
    salaryRate = 0;
}

Workload::Workload(string name, string date)
{
    this->name = name;
    this->date = date;
```

```
    size = 0;
    count = 0;
    disciplines = new sDiscipline[size];

    totalAmount = 0;
    salaryRate = 0;
}

Workload::Workload(const Workload& w)
{
    name = w.name;
    date = w.date;

    size = w.size;
    count = w.count;
    disciplines = new sDiscipline[size];

    for (int i = 0; i < count; i++)
        disciplines[i] = w.disciplines[i];

    totalAmount = w.totalAmount;
    salaryRate = w.salaryRate;
}

Workload::~Workload()
{
    delete[] disciplines;
}

int Workload::getSize() const
{
    return size;
}


int Workload::getCount() const
{
    return count;
}

void Workload::setSize(int value)
{
    sDiscipline* t = new sDiscipline[value];

    int minSize = (size < value) ? size : value;

    for (int i = 0; i < minSize; i++)
        t[i] = disciplines[i];

    delete[] disciplines;

    size = value;
    disciplines = t;
}

void Workload::AddDiscipline(sDiscipline& d) // добавити дисципліну
{
    if (count == size)
        setSize(size + 1);

    disciplines[count] = d;
```

```
    count++;

    Summa();
    SalaryRate();
}

sDiscipline Workload::DelDiscipline()        // вилучити дисципліну
{
    if (count > 0)
    {
        sDiscipline t = disciplines[count - 1];
        count--;

        Summa();
        SalaryRate();

        return t;
    }
    else
    {
        cerr << "Index out of range!" << endl;
        return disciplines[0];
    }
}

sDiscipline Workload::DelDiscipline(int index)      // вилучити дисципліну
{
    if (index >= 0 && index < count)
    {
        sDiscipline t = disciplines[index];
        for (int i = index + 1; i < count; i++)
            disciplines[i - 1] = disciplines[i];
        count--;

        Summa();
        SalaryRate();

        return t;
    }
    else
    {
        cerr << "Index out of range!" << endl;
        return disciplines[0];
    }
}

sDiscipline Workload::DelDiscipline(string name)    // вилучити дисципліну
{
    int index = -1;
    for (int i = 0; i < count; i++)
        if (disciplines[i].discipline.getName() == name)
        {
            index = i;
            break;
        }
    return DelDiscipline(index);
}

sDiscipline Workload::DelDiscipline(sDiscipline& d) // вилучити дисципліну
{
    int index = -1;
    for (int i = 0; i < count; i++)
```

```
        if (disciplines[i] == d)
        {
            index = i;
            break;
        }
    return DelDiscipline(index);
}

Workload& Workload::operator = (const Workload& w)
{
    name = w.name;
    date = w.date;

    delete[] disciplines;

    size = w.size;
    count = w.count;
    disciplines = new sDiscipline[size];

    for (int i = 0; i < count; i++)
        disciplines[i] = w.disciplines[i];

    totalAmount = w.totalAmount;
    salaryRate = w.salaryRate;

    return *this;
}

sDiscipline& Workload::operator [] (int i)
{
    return disciplines[i];
}


Workload::operator string () const
{
    stringstream ss;
    ss  << " teacher name:         " << name << endl
        << " beginning work date : " << date << endl << endl;
    for (int i = 0; i < count; i++)
    {
        ss << "discipline #" << i << ":" << endl;
        ss << disciplines[i] << endl;
    }
    ss << endl << endl;

    return ss.str();
}


Workload& Workload::operator ++ ()
{
    for (int i = 0; i < count; i++)
        ++disciplines[i];
    return *this;
}

Workload& Workload::operator -- ()
{
    for (int i = 0; i < count; i++)
        --disciplines[i];
    return *this;
```

```
}

Workload Workload::operator ++ (int)
{
    Workload t(*this);
    for (int i = 0; i < count; i++)
        disciplines[i] ++;
    return t;
}

Workload Workload::operator -- (int)
{
    Workload t(*this);
    for (int i = 0; i < count; i++)
        disciplines[i] --;
    return t;
}

double Workload::Summa()              // сумарне навантаження в годинах
{
    totalAmount = 0;
    for (int i = 0; i < count; i++)
        totalAmount += disciplines[i].Summa();
    return totalAmount;
}

double Workload::SalaryRate()         // сумарне навантаження в ставках
{
    return salaryRate = totalAmount / avgAmount;
}

ostream& operator << (ostream& out, const Workload& w)
{
    out << string(w);
    return out;
}

istream& operator >> (istream& in, Workload& w)
{
    cout << endl;
    cout << " teacher name:          "; in >> w.name;
    cout << " beginning work date : "; in >> w.date;

    sDiscipline sd;
    char c;
    do
    {
        cout << endl; in >> sd;
        w.AddDiscipline(sd);

        cout << endl << "continue? (Y/N) "; cin >> c;
    } while (c == 'y' || c == 'Y');

    return in;
}

bool Workload::CheckWorkload()        // перевірка навантаження
{
    return salaryRate <= 1.5;
}
```

Варіанти завдань

Частина 1.

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок,
- індексування.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

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

Варіант 1.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- додавання векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 2.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- віднімання векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 3.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- множення вектора на скаляр;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 4.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- скалярного добутку векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 5.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 6.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 7.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (корінь із суми квадратів елементів).

Варіант 8.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- добутку матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 9.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 10.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 11.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (корінь із суми квадратів елементів).

Варіант 12.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- додавання векторів;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 13.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- віднімання векторів;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 14.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- множення вектора на скаляр;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 15.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- скалярного добутку векторів;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 16.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;

- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 17.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 18.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (модуль різниці максимального та мінімального елементів).

Варіант 19.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- добутку матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 20.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 21.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 22.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (модуль різниці максимального та мінімального елементів).

Варіант 23.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- додавання векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 24.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- віднімання векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 25.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- множення вектора на скаляр;
- порівняння векторів;

- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 26.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- скалярного добутку векторів;
- порівняння векторів;
- обчислення норми вектора (корінь із суми квадратів елементів).

Варіант 27.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 28.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 29.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (корінь із суми квадратів елементів).

Варіант 30.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- добутку матриць;
- порівняння матриць;

- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 31.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 32.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (корінь із суми квадратів елементів).

Варіант 33.*

Описати клас, який представляє матрицю із K×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (корінь із суми квадратів елементів).

Варіант 34.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- додавання векторів;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 35.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- віднімання векторів;

- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 36.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- множення вектора на скаляр;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 37.

Описати клас, який представляє вектор із N цілих чисел.

Реалізувати операції

- скалярного добутку векторів;
- порівняння векторів;
- обчислення норми вектора (модуль різниці максимального та мінімального елементів).

Варіант 38.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- додавання матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 39.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- віднімання матриць;
- порівняння матриць;
- обчислення норми матриці (модуль різниці максимального та мінімального елементів).

Варіант 40.*

Описати клас, який представляє матрицю із N×N цілих чисел.

Реалізувати операції

- множення матриці на скаляр;
- порівняння матриць;
- обчислення норми матриць (модуль різниці максимального та мінімального елементів).

Частина 2.

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Додатково до потрібних в завданнях операцій слід перевантажити операцію:

- індексування [ ].

Максимально можливий розмір масиву задати константою. У окремому полі size має зберігатися максимальна для цього об'єкту кількість елементів масиву; реалізувати метод size( ), що повертає встановлену довжину. Якщо кількість елементів масиву змінюється під час роботи, визначити в класі поле count. Початкові значення size і count встановлюються конструктором.

У тих завданнях, де можливо, реалізувати конструктор ініціалізації літерним рядком.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

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.

Варіант 1.

Створити клас BitString для роботи з бітовими рядками довжиною не більше ніж 100 біт. Бітовий рядок має бути представлений масивом типу unsigned char, кожен елемент якого приймає значення 0 або 1. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Мають бути реалізовані всі традиційні операції для роботи з бітовими рядками: and, or, xor, not. Реалізувати зсув ліворуч та зсув праворуч на задану кількість бітів.

Варіант 2.

Створити клас Decimal для роботи з без-знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожний з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Варіант 3.

Створити клас Hex для роботи з без-знаковими цілими шістнадцятковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожний з яких є шістнадцятковою цифрою. Молодша цифра має менший індекс. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Варіант 4.

Створити клас Money для роботи з грошовими сумами. Сума має бути представлена

масивом, кожен елемент якого – десяткова цифра. Максимальна довжина масиву – 100 цифр, реальна довжина задається конструктором. Молодший індекс відповідає молодшій цифрі грошової суми. Молодші дві цифри – копійки.

Варіант 5.

Створити клас Polinom для роботи з многочленами до 100-ої степені. Коефіцієнти мають бути представлені масивом з 100 елементів-коефіцієнтів. Молодша степінь має менший індекс (нульова степінь – нульовий індекс). Розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції та операції порівняння, обчислення значення поліному для заданого значення *x,* диференціювання, інтегрування.

Варіант 6.

Створити клас Fraction для роботи з без-знаковими дробовими десятковими числами. Число має бути представлене двома масивами типу unsigned char: ціла і дробова частина, кожен елемент – десяткова цифра. Для цілої частини молодша цифра має менший індекс, для дробової частини старша цифра має менший індекс (десяті – в нульовому елементі, соті – в першому, і т. д.). Реальний розмір масивів задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції додавання, віднімання та множення, а також операції порівняння.

Варіант 7.

Реалізувати клас Rational, використовуючи два масиви з 100 елементів типу unsigned char для представлення чисельника і знаменника. Кожен елемент є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації.

Варіант 8.

Реалізувати клас Money, використовуючи для представлення суми грошей масив структур. Структура має два поля: номінал купюри і кількість купюр цього номіналу. Номінали представити як переліковий тип nominal. Елемент масиву структур з меншим індексом містить менший номінал.

Варіант 9.

Реалізувати клас Date, використовуючи для представлення місяців масив структур. Структура має два поля: назва місяця (літерний рядок) та кількість днів в місяці. Індексом в масиві місяців є переліковий тип month. Реалізувати два варіанти класу: із звичайним

масивом і статичним масивом місяців.

Варіант 10.

Реалізувати клас Set (множина) не більше ніж з 256 елементів-символів. Потрібно забезпечити включення елементу у множину, вилучення елементу із множини, об'єднання, перетин і різницю множин, обчислення кількості елементів в множині, перевірку наявності елемента у множині, перевірку входження однієї множини у іншу.

Варіант 11.

Створити клас String для роботи з літерними рядками, аналогічними рядкам Turbo Pascal (рядок представляється як масив символів із 256 байт, елементи з 1 по 255 представляють значущі символи, фактична довжина – в нульовому байті). Максимальний розмір рядка має задаватися користувачем. Обов'язково мають бути реалізовані: визначення довжини рядка, пошук під-рядка у рядку, видалення під-рядка з рядка, вставка під-рядка у рядок, зчеплення двох рядків.

Варіант 12.

Створити клас Octal для роботи з без-знаковими цілими вісімковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожен елемент якого є вісімковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ для цілих чисел, та операції порівняння.

Варіант 13.*

Картка іноземного слова є структурою, що містить іноземне слово та його переклад. Для моделювання електронного словника іноземних слів реалізувати клас Dictionary. Цей клас має поле – назву словника і містить масив структур WordCard, що є картками іноземного слова. Назва словника задається при створенні нового словника, але має бути надана можливість його зміни під час роботи. Картки додаються в словник і видаляються з нього. Реалізувати пошук певного слова як окремий метод. Аргументом операції індексування має бути іноземне слово. У словнику не має бути карток-дублікатів.

Реалізувати операції об'єднання, перетину і обчислення різниці словників. При реалізації має створюватися новий словник, а початкові словники не мають змінюватися. При об'єднанні новий словник має містити без повторень всі слова, що містяться в обох словниках-операндах. При перетині новий словник має складатися тільки з тих слів, які є в обох словниках-

операндах. При обчисленні різниці новий словник має містити слова першого словника-операнда, які відсутні в другому.

Варіант 14.*

Одне тестове питання є структурою Task з наступними полями: текст питання; п’ять варіантів відповіді; номер правильної відповіді; бали за правильну відповідь, що нараховуються. Для моделювання набору тестових питань реалізувати клас TestContent, що містить масив тестових питань. Реалізувати методи додавання і видалення тестових питань, а також метод доступу до тестового завдання за його порядковим номером у списку. У масиві не має бути питань, що повторюються.

Реалізувати операцію злиття двох тестових наборів, операцію перетину і операцію обчислення різниці.

При реалізації має створюватися новий тестовий набір, а початкові тестові набори не мають змінюватися. При об’єднанні новий тестовий набір має містити без повторень всі елементи, що містяться в обох тестових наборах-операндах. При перетині новий тестовий набір має складатися тільки з тих елементів, які є в обох тестових наборах-операндах. При обчисленні різниці новий тестовий набір має містити елементи першого тестового набору, які відсутні у другому.

Додатково реалізувати операцію генерації конкретного об’єкту Test (об’ємом не більш K питань) з об’єкту типу TestContent.

Варіант 15.*

Картка персони містить прізвище і дату народження. Реалізувати клас ListPerson для роботи з картотекою персоналій. Клас має містити масив карток персон. Реалізувати методи додавання і видалення карток персон, а також метод доступу до картки за прізвищем. Прізвища в масиві мають бути унікальні.

Реалізувати операції об’єднання двох картотек, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картотека, а початкові картотеки не мають змінюватися. При об’єднанні нова картотека має містити без повторень всі елементи, що містяться в обох картотеках-операндах. При перетині нова картотека має складатися тільки з тих елементів, які є в обох картотеках-операндах. При обчисленні різниці нова картотека має містити елементи першої картотеки-операнді, які відсутні в другій.

Реалізувати метод, що видає за прізвищем знак зодіаку. Для цього в класі має бути оголошений масив структур Zodiac з полями: назва знаку зодіаку, дата початку і дата закінчення періоду. Індексом в масиві має бути переліковий тип zodiac. Реалізувати два варіанти класу: із

звичайним масивом і статичним масивом Zodiac.

Варіант 16.*

Товарний чек містить список товарів, куплених покупцем в магазині. Один елемент списку є парою: товар-сума. Товар – це клас Goods з полями коду і найменування товару, ціни за одиницю товару, кількості одиниць товару, що купуються. У класі мають бути реалізовані методи доступу до полів для отримання і зміни інформації, а також метод обчислення суми оплати за товар. Для моделювання товарного чеку реалізувати клас Receipt, полями якого є номер товарного чека, дата і час його створення, список товарів, що купуються. У класі Receipt реалізувати методи додавання, зміни і видалення запису про товар, що купується, метод пошуку інформації про певний вид товару за його кодом, а також метод підрахунку загальної суми, на яку були здійснені покупки. Методи додавання і зміни приймають як аргумент об'єкт класу Goods. Метод пошуку повертає об'єкт класу Goods як результат.

Варіант 17.*

Інформаційний запис про книгу в бібліотеці містить наступні поля: автор, назва, рік видання, видавництво, ціна. Для моделювання облікової картки абонента реалізувати клас Subscriber, що містить прізвище абонента, його бібліотечний номер і список взятих в бібліотеці книг. Один елемент списку складається з інформаційного запису про книгу, дати видачі, необхідної дати повернення і признаку повернення. Реалізувати методи додавання книг в список і видалення книг з нього; метод пошуку книг, що підлягають поверненню; методи пошуку за автором, видавництвом та роком видання; метод обчислення вартості всіх книг, що підлягають поверненню.

Реалізувати операцію злиття двох облікових карток, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картка, а початкові картки не мають змінюватися. При об'єднанні нова картка має містити без повторень всі елементи, що містяться в обох картках-операндах. При перетині нова картка має складатися тільки з тих елементів, які є в обох картках-операндах. При обчисленні різниці нова картка має містити елементи першої картки, які відсутні у другій.

Реалізувати операцію генерації конкретного об'єкту Debt (борг), що містить список книг, що підлягають поверненню, з об'єкту типа Subscriber.

Варіант 18.*

Інформаційний запис про файл в каталозі містить поля: ім'я файлу, розширення, дату і час створення, атрибути «тільки зчитування», «прихований», «системний», розмір файлу на

диску. Для моделювання каталогу реалізувати клас Directory, що містить назву батьківського каталогу, кількість файлів в каталозі, список файлів в каталозі. Один елемент списку включає інформаційний запис про файл, дату останньої зміни, признак виділення і признак видалення. Реалізувати методи додавання файлів в каталог і видалення файлів з нього; метод пошуку файлу по імені, по розширенню, по даті створення; метод обчислення повного об'єму каталогу.

Реалізувати операцію об'єднання і операцію перетину каталогів.

При реалізації має створюватися новий каталог, а початкові каталоги не мають змінюватися. При об'єднанні новий каталог має містити без повторень всі елементи, що містяться в обох каталогах-операндах. При перетині новий каталог має складатися тільки з тих елементів, які є в обох каталогах-операндах. При обчисленні різниці новий каталог має містити елементи першого каталога, які відсутні у другому.

Реалізувати операцію генерації конкретного об'єкту Group (група), що містить список файлів, з об'єкту типа Directory. Має бути можливість вибирати групу файлів за ознакою видалення, по атрибутах, по даті створення (до або після), за об'ємом (менше або більше).

Варіант 19.*

Навчальний план спеціальності є списком дисциплін, які студент має вивчити за час навчання. Одна дисципліна є структурою з полями: номер дисципліни в плані; тип дисципліни (нормативна, за вибором навчального закладу, за вибором студента); назва дисципліни; семестр, в якому дисципліна вивчається; вид підсумкового контролю (залік або іспит); загальна кількість годин, необхідна для вивчення дисципліни; кількість аудиторних годин, яка складається з лекційних годин і годин практики. Реалізувати клас PlanEducation для моделювання навчального плану спеціальності. Клас має містити код і назву спеціальності, дату затвердження, загальну кількість годин спеціальності за стандартом і список дисциплін. Один елемент списку дисциплін має містити запис про дисципліну, кількість годин для самостійної роботи (різниця між загальною кількістю годин і аудиторними годинами), ознака наявності курсової роботи, яка виконується з цієї дисципліни. Реалізувати методи додавання і видалення дисциплін; метод пошуку дисципліни по семестру, за типом дисципліни, по виду підсумкового контролю; метод обчислення сумарної кількості годин всіх дисциплін; метод обчислення кількості іспитів і заліків по семестрах.

Реалізувати операцію об'єднання, операцію віднімання навчальних планів і операцію перетину навчальних планів.

При реалізації має створюватися новий навчальний план, а початкові навчальні плани не мають змінюватися. При об'єднанні новий навчальний план має містити без повторень всі

елементи, що містяться в обох навчальних планах-операндах. При перетині новий навчальний план має складатися тільки з тих елементів, які є в обох навчальних планах-операндах. При відніманні новий навчальний план має містити елементи першого навчального плану, які відсутні у другому.

Реалізувати операцію генерації конкретного об'єкту Group (група дисциплін), що містить список дисциплін, з об'єкту типу PlanEducation. Має бути можливість вибирати групу дисциплін за типом, по семестру, по вигляду підсумкового контролю, по наявності курсової роботи. Має здійснюватися контроль за сумарною кількістю годин, за кількістю іспитів в семестрі (не більше п'яти і не меншого трьох).

Варіант 20.*

Навантаження викладача за навчальний рік – це список дисциплін, що викладаються ним протягом року. Одна дисципліна представляється інформаційною структурою з полями: назва дисципліни, семестр проведення, кількість студентів, кількість аудиторних годин лекцій, кількість аудиторних годин практики, вид контролю (залік або іспит). Реалізувати клас WorkTeacher, що моделює бланк призначеного викладачеві навантаження. Клас містить прізвище викладача, дату затвердження, список дисциплін, що викладаються, об'єм повного навантаження в годинах і в ставках. Дисципліни в списку не мають повторюватися. Об'єм в ставках обчислюється як результат від ділення об'єму в годинах на середню річну ставку, однакову для всіх викладачів кафедри. Елемент списку дисциплін, що викладаються, містить дисципліну, кількість годин, що виділяється на залік (0,2 год. на одного студента) або іспит (0,3 год. на студента), суму годин по дисципліні. Реалізувати додавання і видалення дисциплін; обчислення сумарного навантаження в годинах і ставках. Має здійснюватися контроль за перевищення навантаження – не більш, ніж півтори ставки.

Варіант 21.*

Один запис в списку запланованих справ є структурою DailyItem, яка містить час початку і закінчення роботи, опис і признак виконання. Реалізувати клас DailySchedule, що представляє собою план робіт на день. Реалізувати методи додавання, видалення і зміни планованої роботи. При додаванні перевіряти коректність часових рамок (вони не мають перетинатися з вже запланованими заходами). Реалізувати метод пошуку вільного проміжку часу. Умова пошуку задає розмір шуканого інтервалу, а також часові рамки, в які він має потрапляти. Метод пошуку повертає структуру DailyItem з порожнім описом виду робіт. Реалізувати операцію генерації об'єкту Redo (ще раз), що містить список справ, не виконаних протягом дня, з об'єкту типа DailySchedule.

Варіант 22.*

Прайс-лист комп'ютерної фірми включає список моделей комп'ютерів, що продаються. Одна позиція списку (Model) містить марку комп'ютера, тип процесора, частоту роботи процесора, об'єм пам'яті, об'єм жорсткого диску, об'єм пам'яті відеокарти, ціну комп'ютера в умовних одиницях і кількість екземплярів, що є в наявності. Реалізувати клас PriceList, полями якого є дата його створення, номінал умовної одиниці в гривнях і список моделей комп'ютерів, що продаються. У списку не ає бути двох моделей однакової марки. У класі PriceList реалізувати методи додавання, зміни і видалення запису про модель, метод пошуку інформації про модель по марці комп'ютера, за об'ємом пам'яті, диска і відеокарти (рівно або не менше заданого), а також метод підрахунку загальної суми.

Реалізувати методи об'єднання і перетину прайс-листів.

При реалізації має створюватися новий прайс-лист, а початкові прайс-листи не мають змінюватися. При об'єднанні новий прайс-лист має містити без повторень всі елементи, що містяться в обох прайс-листах - операндах. При перетині новий прайс-лист має складатися тільки з тих елементів, які є в обох прайс-листах - операндах. При відніманні новий прайс-лист амє містити елементи першого прайс-листа, які відсутні в другому.

Методи додавання і зміни приймають як вхідний параметр об'єкт класу Model. Метод пошуку повертає об'єкт класу Model як результат.

Варіант 23.

Створити клас BitString для роботи з бітовими рядками довжиною не більше ніж 100 біт. Бітовий рядок має бути представлений масивом типу unsigned char, кожен елемент якого приймає значення 0 або 1. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Мають бути реалізовані всі традиційні операції для роботи з бітовими рядками: and, or, xor, not. Реалізувати зсув ліворуч та зсув праворуч на задану кількість бітів.

Варіант 24.

Створити клас Decimal для роботи з без-знаковими цілими десятковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожний з яких є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Варіант 25.

Створити клас Hex для роботи з без-знаковими цілими шістнадцятковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожний з яких є шістнадцятковою цифрою. Молодша цифра має менший індекс. Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ операціям для цілих чисел та операції порівняння.

Варіант 26.

Створити клас Money для роботи з грошовими сумами. Сума має бути представлена масивом, кожен елемент якого – десяткова цифра. Максимальна довжина масиву – 100 цифр, реальна довжина задається конструктором. Молодший індекс відповідає молодшій цифрі грошової суми. Молодші дві цифри – копійки.

Варіант 27.

Створити клас Polinom для роботи з многочленами до 100-ої степені. Коефіцієнти мають бути представлені масивом з 100 елементів-коефіцієнтів. Молодша степінь має менший індекс (нульова степінь – нульовий індекс). Розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції та операції порівняння, обчислення значення поліному для заданого значення *x,* диференціювання, інтегрування.

Варіант 28.

Створити клас Fraction для роботи з без-знаковими дробовими десятковими числами. Число має бути представлене двома масивами типу unsigned char: ціла і дробова частина, кожен елемент – десяткова цифра. Для цілої частини молодша цифра має менший індекс, для дробової частини старша цифра має менший індекс (десяті – в нульовому елементі, соті – в першому, і т. д.). Реальний розмір масивів задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції додавання, віднімання та множення, а також операції порівняння.

Варіант 29.

Реалізувати клас Rational, використовуючи два масиви з 100 елементів типу unsigned char для представлення чисельника і знаменника. Кожен елемент є десятковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації.

Варіант 30.

Реалізувати клас Money, використовуючи для представлення суми грошей масив структур. Структура має два поля: номінал купюри і кількість купюр цього номіналу. Номінали представити як переліковий тип nominal. Елемент масиву структур з меншим індексом містить менший номінал.

Варіант 31.

Реалізувати клас Date, використовуючи для представлення місяців масив структур. Структура має два поля: назва місяця (літерний рядок) та кількість днів в місяці. Індексом в масиві місяців є переліковий тип month. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом місяців.

Варіант 32.

Реалізувати клас Set (множина) не більше ніж з 256 елементів-символів. Потрібно забезпечити включення елементу у множину, вилучення елементу із множини, об'єднання, перетин і різницю множин, обчислення кількості елементів в множині, перевірку наявності елемента у множині, перевірку входження однієї множини у іншу.

Варіант 33.

Створити клас String для роботи з літерними рядками, аналогічними рядкам Turbo Pascal (рядок представляється як масив символів із 256 байт, елементи з 1 по 255 представляють значущі символи, фактична довжина – в нульовому байті). Максимальний розмір рядка має задаватися користувачем. Обов'язково мають бути реалізовані: визначення довжини рядка, пошук під-рядка у рядку, видалення під-рядка з рядка, вставка під-рядка у рядок, зчеплення двох рядків.

Варіант 34.

Створити клас Octal для роботи з без-знаковими цілими вісімковими числами, використовуючи для представлення числа масив з 100 елементів типу unsigned char, кожен елемент якого є вісімковою цифрою. Молодша цифра має менший індекс (одиниці – в нульовому елементі масиву). Реальний розмір масиву задається як аргумент конструктора ініціалізації. Реалізувати арифметичні операції, аналогічні вбудованим в C++ для цілих чисел, та операції порівняння.

Варіант 35.*

Картка іноземного слова є структурою, що містить іноземне слово та його переклад.

Для моделювання електронного словника іноземних слів реалізувати клас Dictionary. Цей клас має поле – назву словника і містить масив структур WordCard, що є картками іноземного слова. Назва словника задається при створенні нового словника, але має бути надана можливість його зміни під час роботи. Картки додаються в словник і видаляються з нього. Реалізувати пошук певного слова як окремий метод. Аргументом операції індексування має бути іноземне слово. У словнику не має бути карток-дублікатів.

Реалізувати операції об'єднання, перетину і обчислення різниці словників. При реалізації має створюватися новий словник, а початкові словники не мають змінюватися. При об'єднанні новий словник має містити без повторень всі слова, що містяться в обох словниках-операндах. При перетині новий словник має складатися тільки з тих слів, які є в обох словниках-операндах. При обчисленні різниці новий словник має містити слова першого словника-операнда, які відсутні в другому.

Варіант 36.*

Одне тестове питання є структурою Task з наступними полями: текст питання; п'ять варіантів відповіді; номер правильної відповіді; бали за правильну відповідь, що нараховуються. Для моделювання набору тестових питань реалізувати клас TestContent, що містить масив тестових питань. Реалізувати методи додавання і видалення тестових питань, а також метод доступу до тестового завдання за його порядковим номером у списку. У масиві не має бути питань, що повторюються.

Реалізувати операцію злиття двох тестових наборів, операцію перетину і операцію обчислення різниці.

При реалізації має створюватися новий тестовий набір, а початкові тестові набори не мають змінюватися. При об'єднанні новий тестовий набір має містити без повторень всі елементи, що містяться в обох тестових наборах-операндах. При перетині новий тестовий набір має складатися тільки з тих елементів, які є в обох тестових наборах-операндах. При обчисленні різниці новий тестовий набір має містити елементи першого тестового набору, які відсутні у другому.

Додатково реалізувати операцію генерації конкретного об'єкту Test (об'ємом не більш K питань) з об'єкту типу TestContent.

Варіант 37.*

Картка персони містить прізвище і дату народження. Реалізувати клас ListPerson для роботи з картотекою персоналій. Клас має містити масив карток персон. Реалізувати методи додавання і видалення карток персон, а також метод доступу до картки за прізвищем. Прізвища

в масиві мають бути унікальні.

Реалізувати операції об’єднання двох картотек, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картотека, а початкові картотеки не мають змінюватися. При об’єднанні нова картотека має містити без повторень всі елементи, що містяться в обох картотеках-операндах. При перетині нова картотека має складатися тільки з тих елементів, які є в обох картотеках-операндах. При обчисленні різниці нова картотека має містити елементи першої картотеки-операнді, які відсутні в другій.

Реалізувати метод, що видає за прізвищем знак зодіаку. Для цього в класі має бути оголошений масив структур Zodiac з полями: назва знаку зодіаку, дата початку і дата закінчення періоду. Індексом в масиві має бути переліковий тип zodiac. Реалізувати два варіанти класу: із звичайним масивом і статичним масивом Zodiac.

Варіант 38.*

Товарний чек містить список товарів, куплених покупцем в магазині. Один елемент списку є парою: товар-сума. Товар – це клас Goods з полями коду і найменування товару, ціни за одиницю товару, кількості одиниць товару, що купуються. У класі мають бути реалізовані методи доступу до полів для отримання і зміни інформації, а також метод обчислення суми оплати за товар. Для моделювання товарного чеку реалізувати клас Receipt, полями якого є номер товарного чека, дата і час його створення, список товарів, що купуються. У класі Receipt реалізувати методи додавання, зміни і видалення запису про товар, що купується, метод пошуку інформації про певний вид товару за його кодом, а також метод підрахунку загальної суми, на яку були здійснені покупки. Методи додавання і зміни приймають як аргумент об’єкт класу Goods. Метод пошуку повертає об’єкт класу Goods як результат.

Варіант 39.*

Інформаційний запис про книгу в бібліотеці містить наступні поля: автор, назва, рік видання, видавництво, ціна. Для моделювання облікової картки абонента реалізувати клас Subscriber, що містить прізвище абонента, його бібліотечний номер і список взятих в бібліотеці книг. Один елемент списку складається з інформаційного запису про книгу, дати видачі, необхідної дати повернення і признаку повернення. Реалізувати методи додавання книг в список і видалення книг з нього; метод пошуку книг, що підлягають поверненню; методи пошуку за автором, видавництвом та роком видання; метод обчислення вартості всіх книг, що підлягають поверненню.

Реалізувати операцію злиття двох облікових карток, операцію перетину і обчислення різниці.

При реалізації має створюватися нова картка, а початкові картки не мають змінюватися. При об’єднанні нова картка має містити без повторень всі елементи, що містяться в обох картках-операндах. При перетині нова картка має складатися тільки з тих елементів, які є в обох картках-операндах. При обчисленні різниці нова картка має містити елементи першої картки, які відсутні у другій.

Реалізувати операцію генерації конкретного об’єкту Debt (борг), що містить список книг, що підлягають поверненню, з об’єкту типа Subscriber.

Варіант 40.*

Інформаційний запис про файл в каталозі містить поля: ім’я файлу, розширення, дату і час створення, атрибути «тільки зчитування», «прихований», «системний», розмір файлу на диску. Для моделювання каталогу реалізувати клас Directory, що містить назву батьківського каталогу, кількість файлів в каталозі, список файлів в каталозі. Один елемент списку включає інформаційний запис про файл, дату останньої зміни, признак виділення і признак видалення. Реалізувати методи додавання файлів в каталог і видалення файлів з нього; метод пошуку файлу по імені, по розширенню, по даті створення; метод обчислення повного об’єму каталогу.

Реалізувати операцію об’єднання і операцію перетину каталогів.

При реалізації має створюватися новий каталог, а початкові каталоги не мають змінюватися. При об’єднанні новий каталог має містити без повторень всі елементи, що містяться в обох каталогах-операндах. При перетині новий каталог має складатися тільки з тих елементів, які є в обох каталогах-операндах. При обчисленні різниці новий каталог має містити елементи першого каталога, які відсутні у другому.

Реалізувати операцію генерації конкретного об’єкту Group (група), що містить список файлів, з об’єкту типа Directory. Має бути можливість вибирати групу файлів за ознакою видалення, по атрибутах, по даті створення (до або після), за об’ємом (менше або більше).

Лабораторна робота № 2.5. Конструктори та перевантаження операцій для класів з композицією

Мета роботи

Освоїти використання конструкторів та перевантаження операцій для класів з композицією.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.

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.

Створити допоміжний клас Man (людина) з полями: ім’я, вік. Визначити методи пере-присвоєння імені, зміни віку.

Створити основний клас-контейнер Student (студент), що має поле «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).

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому

вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Текст програми

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

#include "Student.h"

using namespace std;

int main()
{
    Student s1;

    Student s4("Kuzia", 19, "KN");
    cout << s4 << endl;

    s1 = ++s4;
    cout << s1 << endl;
    cout << s4 << endl;
```

```
    s1 = s4++;
    cout << s1 << endl;
    cout << s4 << endl;

    return 0;
}

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

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

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();
    Man(const string name);
    Man(const int age);
    Man(const string name, const int age);
    Man(const Man& m);

    Man& operator = (const Man& m);

    friend ostream& operator << (ostream& out, const Man& m);
    friend istream& operator >> (istream& in, Man& m);

    operator string () const;

    Man& operator ++ ();
    Man& operator -- ();
    Man operator ++ (int);
    Man operator -- (int);

    ~Man(void);
};


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

#include "Man.h"
#include <sstream>
```

```
void Man::Display() const
{
    cout << "name = " << name << endl;
    cout << "age  = " << age << endl;
}

void Man::Init(string name, int age)
{
    setName(name);
    setAge(age);
}

void Man::Read()
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age  = ? "; cin >> age;
    Init(name, age);
}

Man::Man()
    : name(""), age(0)
{}

Man::Man(const string name)
    : name(name), age(0)
{}

Man::Man(const int age)
    : name(""), age(age)
{}

Man::Man(const string name, const int age)
    : name(name), age(age)
{}

Man::Man(const Man& m)
    : name(m.name), age(m.age)
{}

Man& Man::operator = (const Man& m)
{
    this->name = m.name;
    this->age = m.age;

    return *this;
}

ostream& operator << (ostream& out, const Man& m)
{
    out << string(m);
    return out;
}

istream& operator >> (istream& in, Man& m)
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; in >> name;
```

```
    cout << "age  = ? "; in >> age;
    m.setName(name);
    m.setAge(age);

    return in;
}

Man::operator string () const
{
    stringstream ss;
    ss << endl;
    ss << "name = " << name << endl;
    ss << "age  = " << age << endl;

    return ss.str();
}

Man& Man::operator ++ ()
{
    ++age;
    return *this;
}

Man& Man::operator -- ()
{
    --age;
    return *this;
}

Man Man::operator ++ (int)
{
    Man t(*this);
    age++;
    return t;
}

Man Man::operator -- (int)
{
    Man t(*this);
    age--;
    return t;
}

Man::~Man(void)
{}

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

#pragma once
#include "Man.h"

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(const string name = "", const int age = 0, const string spec = "");
    Student(const Student& s);

    Student& operator = (const Student& s);

    friend ostream& operator << (ostream& out, const Student& s);
    friend istream& operator >> (istream& in, Student& s);

    operator string () const;

    Student& operator ++ ();
    Student& operator -- ();
    Student operator ++ (int);
    Student operator -- (int);

    ~Student(void);
};

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

#include "Man.h"
#include "Student.h"
#include <sstream>

void Student::Init(string spec, Man man)
{
    setSpec(spec);
    setMan(man);
}

void Student::Display() const
{
    cout << endl;
    cout << "man = " << endl;
    man.Display();
    cout << "spec  = " << spec << endl;
}

void Student::Read()
{
    string spec;
    Man m;
    cout << endl;
    cout << "Man  = ? " << endl;
    m.Read();
    cout << "spec = ? "; cin >> spec;
    Init(spec, m);
}

Student::Student(const string name, const int age, const string spec)
    : man(name, age), spec(spec)
{}
```

```
Student::Student(const Student& s)
{
    man = s.man;
    spec = s.spec;
}
Student& Student::operator = (const Student& s)
{
    man = s.man;
    spec = s.spec;
    return *this;
}

ostream& operator << (ostream& out, const Student& s)
{
    out << string(s);
    return out;
}

istream& operator >> (istream& in, Student& s)
{
    string spec;
    cout << endl;
    cout << "man = ? "; in >> s.man;
    cout << "spec  = ? "; in >> spec;
    s.setSpec(spec);

    return in;
}

Student::operator string () const
{
    stringstream ss;
    ss << "spec  = " << spec << endl;

    return string(man) + ss.str();
}

Student& Student::operator ++ ()
{
    ++man;
    return *this;
}

Student& Student::operator -- ()
{
    --man;
    return *this;
}

Student Student::operator ++ (int)
{
    Student s(*this);
    man++;
    return s;
}

Student Student::operator -- (int)
{
    Student s(*this);
    man--;
    return s;
}
```

```
Student::~Student(void)
{}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.5 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

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.

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 довільного степеня, і метод для обчислення числа π в певній 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 довільного степеня, і метод для обчислення числа π в певній 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 № 2.6. Конструктори та перевантаження операцій для класів з вкладеними класами. Обчислення кількості об’єктів

Мета роботи

Освоїти використання конструкторів та перевантаження операцій для вкладених класів.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.
9) Обчислення кількості наявних об’єктів.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.6 (Вкладені класи) з конструкторами і перевантаженням операцій (тобто, Лабораторної роботи № 2.5), використовуючи конструкцію вкладеного класу замість композиції. Реалізувати обчислення кількості об'єктів внутрішнього та зовнішнього класів.

Створити допоміжний клас Man (людина) з полями: ім'я, вік. Визначити методи пере-присвоєння імені, зміни віку.

Створити основний клас-контейнер Student (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.

Текст програми

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

#include "Student.h"

using namespace std;

int main()
{
    Student s1;
```

```
    cout << "Student : " << Student::getCounter() << endl;
    cout << "Student::Man : " << Student::Man::getCounter() << endl;

    Student::Man m1;
    cout << "Student::Man +1 : " << Student::Man::getCounter() << endl;

    {
        Student s4("Kuzia", 19, "KN");
        cout << "Student local : " << Student::getCounter() << endl;
        cout << "Student::Man local : " << Student::Man::getCounter() << endl;

        Student::Man m1;
        cout << "Student::Man local +1 : " << Student::Man::getCounter() << endl;
    }

    cout << "Student : " << Student::getCounter() << endl;
    cout << "Student::Man : " << Student::Man::getCounter() << endl;

    return 0;
}

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

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

using namespace std;

class Student
{
private:
    string spec;
    static int counter;

public:
    class Man
    {
    private:
        string name;
        int age;
        static int counter;

    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(const string name);
        Man(const int age);
        Man(const string name, const int age);
        Man(const Man& m);
        ~Man(void);
```

```
        Man& operator = (const Man& m);
        friend ostream& operator << (ostream& out, const Man& m);
        friend istream& operator >> (istream& in, Man& m);

        operator string () const;

        Man& operator ++ ();
        Man& operator -- ();
        Man operator ++ (int);
        Man operator -- (int);

        static int getCounter();
    };

    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(const string name = "", const int age = 0, const string spec = "");
    Student(const Student& s);
    ~Student(void);

    Student& operator = (const Student& s);
    friend ostream& operator << (ostream& out, const Student& s);
    friend istream& operator >> (istream& in, Student& s);

    operator string () const;

    Student& operator ++ ();
    Student& operator -- ();
    Student operator ++ (int);
    Student operator -- (int);

    static int getCounter();

private:
    Man man;
};

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

#include "Student.h"
#include <sstream>

////////////////////////////////////////////////////////
// class Student

int Student::counter = 0;

void Student::Init(string spec, Man man)
{
    setSpec(spec);
    setMan(man);
}
```

```
void Student::Display() const
{
    cout << endl;
    cout << "man = " << endl;
    man.Display();
    cout << "spec  = " << spec << endl;
}

void Student::Read()
{
    string spec;
    Man m;
    cout << endl;
    cout << "Man  = ? " << endl;
    m.Read();
    cout << "spec = ? "; cin >> spec;
    Init(spec, m);
}

Student::Student(const string name, const int age, const string spec)
    : man(name, age), spec(spec)
{
    Student::counter++;
}

Student::Student(const Student& s)
{
    man = s.man;
    spec = s.spec;
    Student::counter++;
}

Student::~Student(void)
{
    Student::counter--;
}

Student& Student::operator = (const Student& s)
{
    man = s.man;
    spec = s.spec;

    return *this;
}

Student::operator string () const
{
    stringstream ss;
    ss << "man = " << man << endl;
    ss << "spec  = " << spec << endl;

    return ss.str();
}


ostream& operator << (ostream& out, const Student& s)
{
    out << string(s);

    return out;
}
```

```
istream& operator >> (istream& in, Student& s)
{
    string spec;
    cout << endl;
    cout << "man = ? "; in >> s.man;
    cout << "spec  = ? "; in >> spec;
    s.setSpec(spec);

    return in;
}

Student Student::operator ++ (int)
{
    Student s(*this);
    man++;
    return s;
}

Student Student::operator -- (int)
{
    Student s(*this);
    man--;
    return s;
}

Student& Student::operator ++ ()
{
    ++man;
    return *this;
}

Student& Student::operator -- ()
{
    --man;
    return *this;
}

int Student::getCounter()
{
    return Student::counter;
}

////////////////////////////////////////////////////////
// class Man

int Student::Man::counter = 0;

void Student::Man::Init(string name, int age)
{
    setName(name);
    setAge(age);
}

void Student::Man::Display() const
{
    cout << "name = " << name << endl;
    cout << "age  = " << age << endl;
}

void Student::Man::Read()
{
    string name;
```

```
    int age;
    cout << endl;
    cout << "name = ? "; cin >> name;
    cout << "age  = ? "; cin >> age;
    Init(name, age);
}

Student::Man::Man()
    : name(""), age(0)
{
    Student::Man::counter++;
}

Student::Man::Man(const string name)
    : name(name), age(0)
{
    Student::Man::counter++;
}

Student::Man::Man(const int age)
    : name(""), age(age)
{
    Student::Man::counter++;
}

Student::Man::Man(const string name, const int age)
    : name(name), age(age)
{
    Student::Man::counter++;
}

Student::Man::Man(const Student::Man& m)
    : name(m.name), age(m.age)
{
    Student::Man::counter++;
}

Student::Man::~Man(void)
{
    Student::Man::counter--;
}

Student::Man& Student::Man::operator = (const Student::Man& m)
{
    this->name = m.name;
    this->age = m.age;
    return *this;
}

Student::Man::operator string () const
{
    stringstream ss;
    ss << "name = " << name << endl;
    ss << "age  = " << age << endl;
    return ss.str();
}

ostream& operator << (ostream& out, const Student::Man& m)
{
    out << string(m);
    return out;
}
```

```
istream& operator >> (istream& in, Student::Man& m)
{
    string name;
    int age;
    cout << endl;
    cout << "name = ? "; in >> name;
    cout << "age  = ? "; in >> age;
    m.setName(name);
    m.setAge(age);

    return in;
}


Student::Man& Student::Man::operator ++ ()
{
    ++age;
    return *this;
}

Student::Man& Student::Man::operator -- ()
{
    --age;
    return *this;
}

Student::Man Student::Man::operator ++ (int)
{
    Student::Man t(*this);
    age++;
    return t;
}

Student::Man Student::Man::operator -- (int)
{
    Student::Man t(*this);
    age--;
    return t;
}

int Student::Man::getCounter()
{
    return Student::Man::counter;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.6 (Вкладені класи) з конструкторами і перевантаженням операцій (тобто, Лабораторної роботи № 2.5), використовуючи конструкцію вкладеного класу замість композиції. Реалізувати обчислення кількості об'єктів внутрішнього та зовнішнього класів.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

Лабораторна робота № 1.6:

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.

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 довільного степеня, і метод для обчислення числа π в певній 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 довільного степеня, і метод для обчислення числа π в певній 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 № 2.7. Конструктори та перевантаження операцій для класів з композицією – складніші завдання

Мета роботи

Освоїти використання конструкторів та перевантаження операцій для класів з композицією.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.

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 № 1.7. Реалізувати методи роботи з назвою точки SetName(), GetName() та методи роботи з комплексним числом – вказані у завданні демонстраційного прикладу Лабораторної роботи № 1.7.

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Текст програми

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

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

using namespace std;

int main()
{
    ComplexPoint z1('a', 0, 2), z2;

    cout << "Input 2-nd point: " << endl;
    cin >> z2;
    cout << endl;

    cout << "z1: " << z1 << endl;
    cout << "z2: " << z2 << endl;
    cout << endl;
```

```
    z2 = ++z1;
    cout << "z2 = ++z1:  " << "z1 = " << z1 << " ,   z2 = " << z2 << endl;
    z2 = --z1;
    cout << "z2 = --z1:  " << "z1 = " << z1 << " ,   z2 = " << z2 << endl;
    z2 = z1++;
    cout << "z2 = z1++:  " << "z1 = " << z1 << " ,   z2 = " << z2 << endl;
    z2 = z1--;
    cout << "z2 = z1--:  " << "z1 = " << z1 << " ,   z2 = " << z2 << endl;

    cout << "Abs(z1) = " << z1.Abs() << endl << endl;

    cout << z1 << endl << endl;

    char s[100];
    strcpy_s(s, z1.AbsToNumeral());
    cout << s << endl << endl;

    cin.get();
    return 0;
}

//////////////////////////////////////////////////////////////////////////
// Complex.h
//                заголовочний файл – визначення класу

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

using namespace std;

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; }

    Complex();
    Complex(double, double);
    Complex(const Complex&);

    double Abs();
    const char* AbsToNumeral();

    Complex& operator =(const Complex&);
    operator string() const;

    Complex& operator ++();
    Complex& operator --();
    Complex operator ++(int);
    Complex operator --(int);

    friend ostream& operator <<(ostream&, const Complex&);
    friend istream& operator >>(istream&, Complex&);
};

//////////////////////////////////////////////////////////////////////////
// Complex.cpp
```

```
//                файл реалізації – реалізація методів класу

#include "Complex.h"

#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <string>
#include <sstream>

using namespace std;

Complex::Complex()
{}

Complex::Complex(double x, double y)
{
    re = x;
    im = y;
}

Complex::Complex(const Complex& a)
{
    *this = a;
}

double Complex::Abs()
{
    return sqrt(re * re + im * im);
}

Complex& Complex::operator =(const Complex& r)
{
    re = r.re;
    im = r.im;
    return *this;
}

Complex::operator string() const
{
    stringstream ss;
    ss << "(" << re << ", " << im << ")";

    return ss.str();
}

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;
}

Complex& Complex::operator ++()
{
    ++re;
    return *this;
}

Complex& Complex::operator --()
{
    --re;
    return *this;
}

Complex Complex::operator ++(int)
{
    Complex tmp = *this;
    ++im;
    return tmp;
}

Complex Complex::operator --(int)
{
    Complex tmp = *this;
    --im;
    return tmp;
}

ostream& operator <<(ostream& out, const Complex& x)
{
```

```
    out << string(x);
    return out;
}

istream& operator >>(istream& in, Complex& x)
{
    cout << "  Re = "; in >> x.re;
    cout << "  Im = "; in >> x.im;
    return in;
}


//////////////////////////////////////////////////////////////////////////
// ComplexPoint.h
//                заголовочний файл – визначення класу

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

using namespace std;

class ComplexPoint
{
    char    name;
    Complex complex;

public:
    char GetName() const { return name; }
    Complex GetComplex() const { return complex; }
    void SetName(char value) { name = value; }
    void SetComplex(Complex& value) { complex = value; }

    ComplexPoint();
    ComplexPoint(char, double, double);
    ComplexPoint(const ComplexPoint&);

    double Abs();
    const char* AbsToNumeral();

    ComplexPoint& operator =(const ComplexPoint&);
    operator string() const;

    ComplexPoint& operator ++();
    ComplexPoint& operator --();
    ComplexPoint operator ++(int);
    ComplexPoint operator --(int);

    friend ostream& operator <<(ostream&, const ComplexPoint&);
    friend istream& operator >>(istream&, ComplexPoint&);
};

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

#include <iostream>
#include <sstream>
#include "ComplexPoint.h"

using namespace std;
```

```
ComplexPoint::ComplexPoint()
{}

ComplexPoint::ComplexPoint(char a, double x, double y)
    : name(a), complex(x, y)
{}

ComplexPoint::ComplexPoint(const ComplexPoint& a)
{
    *this = a;
}

double ComplexPoint::Abs()
{
    return complex.Abs();
}

const char* ComplexPoint::AbsToNumeral()
{
    return complex.AbsToNumeral();
}

ComplexPoint& ComplexPoint::operator =(const ComplexPoint& r)
{
    name = r.name;
    complex = r.complex;

    return *this;
}

ComplexPoint::operator string() const
{
    stringstream ss;
    ss << name << " " << complex;

    return ss.str();
}


ComplexPoint& ComplexPoint::operator ++()
{
    ++complex;
    return *this;
}

ComplexPoint& ComplexPoint::operator --()
{
    --complex;
    return *this;
}

ComplexPoint ComplexPoint::operator ++(int)
{
    ComplexPoint tmp = *this;
    this->complex++;
    return tmp;
}

ComplexPoint ComplexPoint::operator --(int)
{
    ComplexPoint tmp = *this;
    this->complex--;
```

```
    return tmp;
}

ostream& operator <<(ostream& out, const ComplexPoint& x)
{
    out << string(x);
    return out;
}

istream& operator >>(istream& in, ComplexPoint& x)
{
    cout << "  name: "; in >> x.name;
    in >> x.complex;
    return in;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.7 (Композиція класів та об'єктів) з конструкторами і перевантаженням операцій.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

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 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.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлена 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 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.»).

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 має бути меншою мінімального значення і не має 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-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.

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 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.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлена 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 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.»).

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 має бути меншою мінімального значення і не має 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-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.

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(), $(a, b) \times (c, d) = (ac, bd)$;
- * 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 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 № 2.8. Конструктори та перевантаження операцій для класів з вкладеними класами – складніші завдання. Обчислення кількості об’єктів

Мета роботи

Освоїти використання конструкторів та перевантаження операцій для вкладених класів.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення конструктора.
2) Загальний синтаксис конструктора.
3) Види конструкторів.
4) Загальний синтаксис конструктора за умовчанням.
5) Загальний синтаксис конструктора ініціалізації.
6) Загальний синтаксис конструктора копіювання.
7) Перевантаження операцій присвоєння, вводу / виводу, приведення типу.
8) Перевантаження операцій інкременту та декременту.
9) Обчислення кількості наявних об’єктів.

Зразок виконання завдання

Подається лише умова завдання та текст програми.

Умова завдання

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов’язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.8 (Вкладені класи) з конструкторами і перевантаженням операцій (тобто, Лабораторної роботи № 2.7), використовуючи конструкцію вкладеного класу замість композиції. Реалізувати обчислення кількості об'єктів внутрішнього та зовнішнього класів.

Текст програми

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

#include "ComplexPoint.h"

using namespace std;

int main()
{
    ComplexPoint p1;
    cout << "ComplexPoint : " << ComplexPoint::getCounter() << endl;
    cout << "ComplexPoint::Complex : " << ComplexPoint::Complex::getCounter() << endl;

    ComplexPoint::Complex c1;
    cout << "ComplexPoint::Complex +1 : "
         << ComplexPoint::Complex::getCounter() << endl;

    {
        ComplexPoint p4('P', 20, 21);
        cout << "ComplexPoint local : " << ComplexPoint::getCounter() << endl;
        cout << "ComplexPoint::Complex local : "
             << ComplexPoint::Complex::getCounter() << endl;

        ComplexPoint::Complex m1;
        cout << "ComplexPoint::Complex local +1 : "
             << ComplexPoint::Complex::getCounter() << endl;
    }
```

```
    cout << "ComplexPoint : " << ComplexPoint::getCounter() << endl;
    cout << "ComplexPoint::Complex : " << ComplexPoint::Complex::getCounter() << endl;

    return 0;
}

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

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

using namespace std;

class ComplexPoint
{
    char    name;
    static int counter;

public:
    class Complex
    {
        double re, im;
        static int counter;

    public:
        double GetRe() const { return re; }
        double GetIm() const { return im; }
        void SetRe(double value) { re = value; }
        void SetIm(double value) { im = value; }

        Complex();
        Complex(double, double);
        Complex(const Complex&);
        ~Complex();

        double Abs();
        const char* AbsToNumeral();

        Complex& operator =(const Complex&);
        operator string() const;

        Complex& operator ++();
        Complex& operator --();
        Complex operator ++(int);
        Complex operator --(int);

        friend ostream& operator <<(ostream&, const Complex&);
        friend istream& operator >>(istream&, Complex&);

        static int getCounter();
    };

    char GetName() const { return name; }
    Complex GetComplex() const { return complex; }
    void SetName(char value) { name = value; }
    void SetComplex(Complex& value) { complex = value; }

    ComplexPoint();
```

```
    ComplexPoint(char, double, double);
    ComplexPoint(const ComplexPoint&);
    ~ComplexPoint();

    double Abs();
    const char* AbsToNumeral();

    ComplexPoint& operator =(const ComplexPoint&);
    operator string() const;

    ComplexPoint& operator ++();
    ComplexPoint& operator --();
    ComplexPoint operator ++(int);
    ComplexPoint operator --(int);

    friend ostream& operator <<(ostream&, const ComplexPoint&);
    friend istream& operator >>(istream&, ComplexPoint&);

    static int getCounter();

private:
    Complex complex;
};

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

#include <iostream>
#include <cmath>
#include <stdlib.h>
#include <string>
#include <sstream>
#include "ComplexPoint.h"

using namespace std;

//////////////////////////////////////////////////////////
// class ComplexPoint

int ComplexPoint::counter = 0;

ComplexPoint::ComplexPoint()
{
    counter++;
}

ComplexPoint::ComplexPoint(char a, double x, double y)
    : name(a), complex(x, y)
{
    counter++;
}

ComplexPoint::ComplexPoint(const ComplexPoint& a)
{
    counter++;
    *this = a;
}

ComplexPoint::~ComplexPoint()
{
    counter--;
```

```
}

double ComplexPoint::Abs()
{
    return complex.Abs();
}

const char* ComplexPoint::AbsToNumeral()
{
    return complex.AbsToNumeral();
}

ComplexPoint& ComplexPoint::operator =(const ComplexPoint& r)
{
    name = r.name;
    complex = r.complex;

    return *this;
}

ComplexPoint::operator string() const
{
    stringstream ss;
    ss << name << " " << complex;

    return ss.str();
}

ComplexPoint& ComplexPoint::operator ++()
{
    ++complex;
    return *this;
}

ComplexPoint& ComplexPoint::operator --()
{
    --complex;
    return *this;
}

ComplexPoint ComplexPoint::operator ++(int)
{
    ComplexPoint tmp = *this;
    this->complex++;
    return tmp;
}

ComplexPoint ComplexPoint::operator --(int)
{
    ComplexPoint tmp = *this;
    this->complex--;
    return tmp;
}

ostream& operator <<(ostream& out, const ComplexPoint& x)
{
    out << string(x);
    return out;
}

istream& operator >>(istream& in, ComplexPoint& x)
{
```

```
    cout << "  name: "; in >> x.name;
    in >> x.complex;
    return in;
}


int ComplexPoint::getCounter()
{
    return counter;
}

////////////////////////////////////////////////////////
// class ComplexPoint

int ComplexPoint::Complex::counter = 0;

ComplexPoint::Complex::Complex()
{
    counter++;
}

ComplexPoint::Complex::Complex(double x, double y)
{
    re = x;
    im = y;
    counter++;
}

ComplexPoint::Complex::Complex(const ComplexPoint::Complex& a)
{
    *this = a;
    counter++;
}

ComplexPoint::Complex::~Complex()
{
    counter--;
}

double ComplexPoint::Complex::Abs()
{
    return sqrt(re * re + im * im);
}

ComplexPoint::Complex& ComplexPoint::Complex::operator =(const ComplexPoint::Complex&
r)
{
    re = r.re;
    im = r.im;
    return *this;
}

ComplexPoint::Complex::operator string() const
{
    stringstream ss;
    ss << "(" << re << ", " << im << ")";

    return ss.str();
}

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;
}


ComplexPoint::Complex& ComplexPoint::Complex::operator ++()
{
    ++re;
    return *this;
}

ComplexPoint::Complex& ComplexPoint::Complex::operator --()
{
    --re;
    return *this;
}

ComplexPoint::Complex ComplexPoint::Complex::operator ++(int)
```

```
{
    ComplexPoint::Complex tmp = *this;
    ++im;
    return tmp;
}

ComplexPoint::Complex ComplexPoint::Complex::operator --(int)
{
    Complex tmp = *this;
    --im;
    return tmp;
}

ostream& operator <<(ostream& out, const ComplexPoint::Complex& x)
{
    out << string(x);
    return out;
}

istream& operator >>(istream& in, ComplexPoint::Complex& x)
{
    cout << "  Re = "; in >> x.re;
    cout << "  Im = "; in >> x.im;
    return in;
}

int ComplexPoint::Complex::getCounter()
{
    return counter;
}
```

Варіанти завдань

В кожній лабораторній роботі цієї теми потрібно реалізувати в тому або іншому вигляді визначення нового класу. У всіх завданнях необхідно реалізувати:

- конструктор ініціалізації (один або декілька),
- конструктор без аргументів і
- конструктор копіювання.

Вказані в завданні операції реалізуються за допомогою перевантаження підходящих операцій. У всіх завданнях обов'язково мають бути реалізовані відповідні операції:

- присвоєння,
- введення з клавіатури,
- виводу на екран,
- приведення типу – перетворення у літерний рядок.

Також треба реалізувати операції

- інкременту в обох формах (префіксній та постфіксній) і
- декременту в обох формах (префіксній та постфіксній), – для числових полів (наприклад: так, як вказано у варіантах завдань Лабораторної роботи № 2.3).

Перевантаження операцій виконується таким чином: підходящі операції реалізуються як методи класу, а інші – як зовнішні дружні функції.

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. Вона має виводити на екран розмір класу в режимі #pragma pack(1) і без нього.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Завдання наступне:

Виконати завдання свого варіанту Лабораторної роботи № 1.8 (Вкладені класи) з конструкторами і перевантаженням операцій (тобто, Лабораторної роботи № 2.7), використовуючи конструкцію вкладеного класу замість композиції. Реалізувати обчислення кількості об'єктів внутрішнього та зовнішнього класів.

Метод Init() стане конструкторами, методи Read() та Display() – операціями вводу / виводу.

Лабораторна робота № 1.8:

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 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.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлена 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 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.»).

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 має бути меншою мінімального значення і не має 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 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.»).

Для представлення суми використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Сума має бути представлена 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 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.»).

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 має бути меншою мінімального значення і не має 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

Конструктори та деструктори

1. Дайте визначення конструктора.
2. Яке призначення конструктора?
3. Перерахуєте відмінності конструктора від методу.
4. Скільки конструкторів може бути в класі?
5. Чи допускається перевантаження конструкторів?
6. Які види конструкторів створюються за умовчанням?
7. Чи може конструктор бути приватним?
8. Які наслідки має оголошення конструктора приватним?
9. Приведіть декілька випадків, коли конструктор викликається неявно.
10. Як ініціалізувати динамічну змінну об’єктового типу?
11. Що таке деструктор?
12. Чи може деструктор мати параметри?
13. Чи допускається перевантаження деструкторів?

Константи в класі

14. Навіщо потрібні константні методи?
15. Чим відрізняється визначення константного методу від визначення звичайного?
16. Чи може константний метод викликатися для об’єктів-змінних?
17. Чи може звичайний метод викликатися для об’єктів-констант?
18. Як оголосити константу в класі?
19. Чи можна оголосити дробову константу?
20. Яким чином дозволяється ініціалізувати константні поля в класі?
21. У якому порядку ініціалізуються поля в класі?
22. Чи збігається цей порядок з порядком перерахування ініціалізаторів в списку ініціалізації конструктора?
23. Які конструкції C++ дозволяється використовувати в списку ініціалізації в якості ініціалізуючих виразів?
24. Чи впливає наявність цілочисельних констант-полів на розмір класу?
25. Поясніть, що таке «ініціалізація нулем».

Перевантаження операцій

26. Які операції не можна перевантажувати?
27. Як ви думаєте, чому ці операції не можна перевантажувати?
28. Чи можна перевантажувати операції для вбудованих типів даних?

29. Чи можна при перевантаженні змінити пріоритет операції?
30. Чи можна визначити нову операцію?
31. Чи можна перевантажувати операцію «кома»?
32. Чим відрізняється функціональна форма виклику бінарної операції від інфіксної (операційної) форми?
33. Перерахуєте особливості перевантаження операцій як методів класу.
34. Чим відрізняється перевантаження зовнішньою функцією від перевантаження як методу класу?
35. Який результат мають повертати операції з присвоєнням?
36. Поясніть, що таке *l*-значення.
37. Як відрізняються перевантажена префіксна і постфіксна операції інкремента і декремента?
38. Що означає вираз `*this`?
39. У яких випадках використовується вираз `*this`?
40. Які операції не рекомендується перевантажувати як методи класу?
41. Чому ці операції не рекомендується перевантажувати як методи класу?
42. Поясніть, чому при перевантаженні операцій не можна задавати параметри за умовчанням?
43. Які операції дозволяється перевантажувати тільки як методи класу?
44. Перерахуйте всі можливі способи задання явного перетворення типів.
45. Дайте визначення дружньої функції.
46. Як оголошується дружня функція?
47. Як визначається дружня функція?
48. Поясніть, яку операцію виконує конструкція `static_cast<>`?
49. Який вид конструктора фактично є конструктором перетворення типів?
50. Чим відрізняються операції `reinterpret_cast<>` і `static_cast<>`?
51. Для чого потрібні функції перетворення?
52. Як оголосити функцію перетворення в класі?
53. Поясніть призначення операції `const_cast<>`.
54. Як заборонити неявне перетворення типу, що виконується конструктором ініціалізації?
55. Які проблеми можуть виникнути при визначенні функцій перетворення?
56. Для чого служить ключове слово `explicit`?
57. Як за допомогою функції перетворення і конструктора ініціалізації визначити «нові» операції для вбудованих типів даних.

Масиви в класі

58. Чи можна оголосити в класі поле-масив?
59. Яким чином задається розмір поля-масиву?

60. Яким чином ініціалізується поле-масив?
61. Який результат має повертати перевантажена операція індексування?
62. Чому саме такий результат має повертати перевантажена операція індексування?
63. Чому операцію індексування перевантажують в двох варіантах?
64. Поясніть проблеми, що виникають у разі аварійного завершення роботи конструктора.

Статичні поля в класі

65. Як визначаються статичні поля класу?
66. Які елементи класу можна оголошувати статичними?
67. Чи можна оголосити в класі статичну константу?
68. Чи можна оголосити в класі константний статичний масив?
69. Які статичні поля можна ініціалізувати безпосередньо в класі?
70. Як визначаються статичні поля?
71. У який момент роботи програми виконується ініціалізація статичних полів?
72. Скільки місця в класі займають статичні поля?
73. Чим відрізняється статичний метод від звичайного?
74. Які методи класу не можуть бути статичними?
75. Чи можна статичний метод викликати для об'єкту?
76. Які варіанти застосування статичних полів ви можете привести?
77. Яким чином застосовуються статичні методи?

Використання конструкторів та деструкторів

Умова для завдань №№ 78-81

Дано визначення класів:

```
class C1 {
};

class C2 {
          C1 f1;
};
```

78. Нарисувати UML-діаграму класів `C1` та `C2`.
79. Зв'язок між класами `C1` та `C2` називається: (відмітити правильні відповіді)

А) композиція	Г) поліморфізм
Б) успадковування	Д) інкапсуляція
В) агрегування	Е) інша відповідь:_____

80. Яка саме команда має бути в тілі конструктора класу `C2` для того щоб правильно створювати об'єкти цього класу? (Написати фрагмент повного опису конструктора)
81. Яка саме команда має бути в тілі деструктора класу `C2` для того, щоб правильно знищувати об'єкти цього класу? (Написати фрагмент повного опису деструктора)

Умова для завдань №№ 82-85

Дано визначення класів:

```
class C1 {
};

class C2 {
          C1 *f1;
};
```

82. Нарисувати UML-діаграму класів `C1` та `C2`.
83. Зв'язок між класами `C1` та `C2` називається: (відмітити правильні відповіді)

А) композиція	Г) поліморфізм
Б) успадковування	Д) інкапсуляція
В) агрегування	Е) інша відповідь:_____

84. Яка саме команда має бути в тілі конструктора класу `C2` для того щоб правильно створювати об'єкти цього класу? (Написати фрагмент повного опису конструктора)
85. Яка саме команда має бути в тілі деструктора класу `C2` для того, щоб правильно знищувати об'єкти цього класу? (Написати фрагмент повного опису деструктора)

Перевантаження операцій

86. Визначити клас `C`, який містить поле x цілого типу та перевантажує операцію додавання об'єкту класу `C` і цілого числа. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c;
int i;
... c+i ...
```

87. Визначити клас `C`, який містить поле x цілого типу та перевантажує операцію додавання цілого числа і об'єкту класу `C`. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c;
int i;
... i+c ...
```

88. Визначити клас `C`, який містить поле x цілого типу та перевантажує операцію додавання двох об'єктів класу `C`. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c1, c2;
int i;
... c1+c2 ...
```

89. Визначити клас `C`, який містить поле x цілого типу та перевантажує операцію префіксного інкременту. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c;
... ++c ...
```

90. Визначити клас `C`, який містить поле x цілого типу та перевантажує операцію постфіксного інкременту. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c;
... c++ ...
```

91. Визначити клас `C`, який містить поле `x` цілого типу та перевантажує операцію присвоєння. Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c1, c2;
c1 = c2;
```

92. Визначити клас `C`, який містить поле `x` цілого типу та перевантажує операцію збільшення з присвоєнням (`+=`). Записати визначення класу і повний опис методу, що перевантажує вказану операцію.

```
// використання:
C c1, c2;
c1 += c2;
```

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 С++ 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 С++ .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 С++ .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.