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

Григорович Віктор

Об’єктно-орієнтоване програмування
Опрацювання виняткових ситуацій

Навчальний посібник

2021

Григорович Віктор Геннадійович

Об’єктно-орієнтоване програмування. Опрацювання виняткових ситуацій. 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 «Опрацювання виняткових ситуацій».

Розглядаються наступні роботи лабораторного практикуму:

Лабораторна робота № 5.1.
Класи з опрацюванням виняткових ситуацій

Лабораторна робота № 5.2.
Функції, що генерують виняткові ситуації

Лабораторна робота № 5.3.
Опрацювання виняткових ситуацій

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 «Опрацювання виняткових ситуацій».

Тема 5. Опрацювання виняткових ситуацій

Стисло та головне про опрацювання виняткових ситуацій

Виняткова ситуація або *виняток* – це виникнення непередбаченої або аварійної події, яке може бути породженим некоректним використанням апаратури або програмною помилкою. Наприклад, це ділення на нуль або звертання до відсутньої адреси пам'яті. Зазвичай такі події приводять до завершення виконання програми з системним повідомленням про помилку. C++ дає програмісту можливість відновити та продовжити її виконання.

В C++ вбудовано *механізм опрацювання винятків*, за допомогою якого методи-операції та конструктори класу отримують можливість повідомити клієнтську програму про аварійну ситуацію. Механізм опрацювання винятків – це об'єктно-орієнтовані засоби опрацювання помилок, які виникають при неправильній роботі програми.

Винятки дозволяють логічно розділити основний алгоритм та виокремити опрацювання аварійної ситуації. Це краще структурує програму. Функція, яка виявляє помилку, може не знати, як її виправити, а той код, який використовує цю функцію, може знати, що робити, але не вміти визначити місце виникнення. Це особливо важливо при використанні бібліотечних функцій та при створенні багатомодульних проектів.

Для передавання інформації про помилку у місце виклику функції не потрібно використовувати результат функції, параметри або глобальні змінні – тобто, інтерфейс функції не змінюється. Це особливо важливо для конструкторів, які взагалі не можуть повертати значення.

Ніщо не заважає розглядати в якості винятків не лише помилки, а і нормальні ситуації, що виникають при опрацюванні даних. Проте таке використання не за призначенням механізму винятків не має переваг перед іншими способами опрацювання даних, не покращує структуру та зрозумілість програми.

Принципи опрацювання винятків

Місце, в якому може виникнути помилка, повинно входити в *контрольований* (або ще кажуть *захищений*) *блок* – блок, перед яким записано ключове слово `try`.

Розглянемо, як реалізується опрацювання виняткових ситуацій:

- Опрацювання винятку починається з появлення помилки. Функція, в якій виникла помилка, генерує виняток. Для цього використовується ключове слово `throw` з аргументом, який визначає тип винятку. Аргумент може бути константою, змінною або об'єктом і використовується для передавання інформації про виняток його обробнику.
- Відшуковується відповідний обробник винятку і йому передається управління.
- Якщо обробник винятку не знайдений, викликається стандартна функція `terminate()`, яка викликає функцію `abort()`, що аварійно завершує поточний процес. Можна визначити власну функцію завершення процесу.

При виклику кожної функції в стеку створюється область пам'яті для зберігання локальних змінних та адреси повернення у місце виклику. Термін *стек виклику* означає послідовність викликаних, але ще не завершених функцій. *Розкручування стеку* – це процес звільнення пам'яті від локальних змінних та повернення управління в місце виклику. Коли функція завершується, відбувається природне розкручування стеку. Такий ж механізм використовується і при опрацюванні винятків. Тому, після того, як був зафіксований виняток, виконання програми не може бути продовжене з точки генерування винятку.

Синтаксис та механізм опрацювання винятків

В C++ *виняток – це об'єкт*; при виникненні виняткової ситуації програма повинна генерувати об'єкт-виняток. Генерування об'єкта-винятку і створює виняткову ситуацію.

Загальна схема опрацювання виняткової ситуації наступна: в одній частині програми, де виявлено аварійну ситуацію, породжується виняток; інша частина програми контролює виникнення винятку та опрацьовує його.

Виняток генерується командою `throw`, яка має наступний синтаксис:

```
throw вираз_генерування_винятку;
```

тут *вираз_генерування_винятку* – зазвичай або константа, або змінна деякого типу; допустимо вказувати і вираз. Тип об'єкта-винятку може бути як вбудованим, так і визначатися програмістом.

Перевірка виникнення винятку здійснюється командою оголошення *контрольованого* (його ще називають *захищеного*) блоку `try{}`, одразу після якої записуються один або кілька блоків опрацювання `catch...` (англ. *ловити*, *пастка* – тому їх ще називають *секції-пастки*).

Якщо при виконанні команд контрольованого блоку (блоку try) не було помилок (не було виняткової ситуації, тобто не був згенерований виняток), то всі команди try-блоку виконуються в штатному режимі. Тоді команди catch-блоку пропускаються і управління передається першій команді, записаній після catch-блоку.

Якщо ж при виконанні try-блоку виникла помилка і був згенерований виняток, то управління одразу ж передається підходящому обробнику (тобто, catch-блоку).

Контрольований (захищений блок)

Ключове слово `try` (англ. *пробувати, намагатися*) використовується для позначення *контрольованого* (іншими словами, *захищеного*) блоку – фрагменту коду, в якому може генеруватися виняток. Блок записується всередині фігурних дужок.

Загальний *синтаксис контрольованого блоку*:

```
try                                 // контрольований блок
{
    ...                             // захищені команди
}
```

Контрольований блок має всі характеристики звичайного блоку: змінні, оголошені в ньому, є локальними і не доступні поза цим блоком.

Всі функції, які прямо або опосередковано викликаються із блоку `try`, також йому належать.

Генерування виняткiв

Генерування (*породження*) *винятку* виконується командою

```
throw [вираз];
```

Ключове слово `throw` (англ. *кидати*) використовується разом з виразом-аргументом, або без нього (квадратні дужки – не елемент синтаксису, а позначення необов'язковості).

Тип виразу, вказаного після `throw`, визначає тип породженого винятку.

При генеруванні винятків виконання поточного блоку припиняється і відбувається пошук відповідного обробника і передача йому управління. Після того, як був згенерований виняток і управління передано в блок опрацювання, повернутися назад у контрольований блок – неможливо! Зазвичай, виняток генерується не безпосередньо у блоці `try`, а у функціях, що прямо або опосередковано входять до нього.

Не завжди виняток, який виник у внутрішньому блоці, може бути правильно опрацьований. В такому випадку використовуються вкладені блоки `try`, і виняток передається на зовнішній рівень за допомогою команди `throw` без параметрів.

Перехоплення та опрацювання винятків

Виняток потрібно перехопити та опрацювати.

Після того, як виняток був згенерований, управління із захищеного блоку `try` передається в блоки опрацювання винятків `catch`.

Безпосередньо після блоку `try{}` обов'язково слід записати один або кілька блоків `catch`, які називаються *обробниками винятків* або *блоками опрацювання винятків*.

Обробники винятків починаються з ключового слова `catch`, за яким в круглих дужках записується *специфікація обробника винятку*, потім – *блок команд опрацювання винятку*:

```
catch (специфікація_обробника_винятку)  // заголовок блоку опрацювання винятку
{
    ...                                 // блок команд опрацювання винятку
}
```

Специфікація обробника винятку може мати наступні три форми:

```
(тип ім'я)
(тип)
(...)
```

тип повинен бути одним із допустимих типів винятків – або вбудованим, або визначеним програмістом.

Перші дві форми із наведених специфікацій обробників винятків призначені для опрацювання конкретного типу винятку. Якщо ж замість специфікації обробника винятку записано три крапки, то такий обробник перехоплює всі винятки. Тому секцію-пастку з параметром – трьома крапками слід вказувати в списку catch-блоків останньою.

Теоретичні відомості

Виняткова ситуація або *виняток* – це виникнення непередбаченої або аварійної події, яке може бути породженим некоректним використанням апаратури або програмною помилкою. Наприклад, це ділення на нуль або звертання до відсутньої адреси пам'яті. Зазвичай такі події приводять до завершення виконання програми з системним повідомленням про помилку. C++ дає програмісту можливість відновити та продовжити її виконання.

Зауваження: винятки C++ не підтримують опрацювання асинхронних подій, таких, як апаратні помилки (зокрема, ділення на нуль) та опрацювання переривань (наприклад, натискання клавіш Ctrl+C). Винятки C++ призначені для опрацювання подій, які виникають в результаті роботи самої програми та вказуються явно. Винятки виникають тоді, коли деяка частина програми не змогла виконати свого призначення. При цьому інша частина програми може спробувати зробити щось інше.

Наприклад, розглянемо функцію, яка повинна ділити два числа:

```
double div(double x, double y)
{
    return x / y;
}
```

Якщо другий параметр дорівнює нулю, – виникне помилка «ділення на нуль» і програма аварійно завершить роботу.

Можна добавити перевірку умови і виводити повідомлення про помилку:

```
double div(double x, double y)
{
    if (y != 0)
        return x / y;
    else
    {
        cerr << "div: y==0" << endl;  // вивід в потік повідомлень
        return 0;                     // про помилки
    }
}
```

Але таке рішення – не найкраще: по-перше, це занадто ускладнило алгоритм – добавили конструкцію розгалуження; по-друге, незрозуміло, чому у випадку помилки функція повертає нуль (щось вона повинна повернути, але чому саме нуль)?

Ще одна проблема: перевантажені операції та конструктори, на відміну від звичайних функцій та методів, не можуть ні мати додаткових параметрів для сигналізації про помилки, ні повертати код помилки. В C++ вбудовано *механізм опрацювання винятків*, за допомогою якого методи-операції та конструктори класу отримують можливість повідомити клієнтську

програму про аварійну ситуацію. Механізм опрацювання винятків – це об’єктно-орієнтовані засоби опрацювання помилок, які виникають при неправильній роботі програми.

Винятки дозволяють логічно розділити основний алгоритм та виокремити опрацювання аварійної ситуації. Це краще структурує програму. Але головна причина полягає в тому, що функція, яка виявляє помилку, може не знати, як її виправити, а той код, який використовує цю функцію, може знати, що робити, але не вміти визначити місце виникнення. Це особливо важливо при використанні бібліотечних функцій та при створенні багатомодульних проектів.

Інша перевага винятків: для передавання інформації про помилку у місце виклику функції не потрібно використовувати результат функції, параметри або глобальні змінні – тобто, інтерфейс функції не змінюється. Це особливо важливо для конструкторів, які взагалі не можуть повертати значення.

Крім того, ніщо не заважає розглядати в якості винятків не лише помилки, а і нормальні ситуації, що виникають при опрацюванні даних. Проте таке використання не за призначенням механізму винятків не має переваг перед іншими способами опрацювання даних, не покращує структуру та зрозумілість програми.

Принципи опрацювання винятків

Місце, в якому може виникнути помилка, повинно входити в *контрольований* (або ще кажуть *захищений*) *блок* – блок, перед яким записано ключове слово `try`.

Розглянемо, як реалізується опрацювання виняткових ситуацій:

- Опрацювання винятку починається з появлення помилки. Функція, в якій виникла помилка, генерує виняток. Для цього використовується ключове слово `throw` з аргументом, який визначає тип винятку. Аргумент може бути константою, змінною або об'єктом і використовується для передавання інформації про виняток його обробнику.
- Відшуковується відповідний обробник винятку і йому передається управління.
- Якщо обробник винятку не знайдений, викликається стандартна функція `terminate()`, яка викликає функцію `abort()`, що аварійно завершує поточний процес. Можна визначити власну функцію завершення процесу.

При виклику кожної функції в стеку створюється область пам'яті для зберігання локальних змінних та адреси повернення у місце виклику. Термін *стек виклику* означає послідовність викликаних, але ще не завершених функцій. *Розкручування стеку* – це процес звільнення пам'яті від локальних змінних та повернення управління в місце виклику. Коли функція завершується, відбувається природне розкручування стеку. Такий ж механізм використовується і при опрацюванні винятків. Тому, після того, як був зафіксований виняток, виконання програми не може бути продовжене з точки генерування винятку.

Синтаксис та механізм опрацювання винятків

В C++ *виняток – це об'єкт*; при виникненні виняткової ситуації програма повинна генерувати об'єкт-виняток. Генерування об'єкта-винятку і створює виняткову ситуацію.

Загальна схема опрацювання виняткової ситуації наступна: в одній частині програми, де виявлено аварійну ситуацію, породжується виняток; інша частина програми контролює виникнення винятку та опрацьовує його.

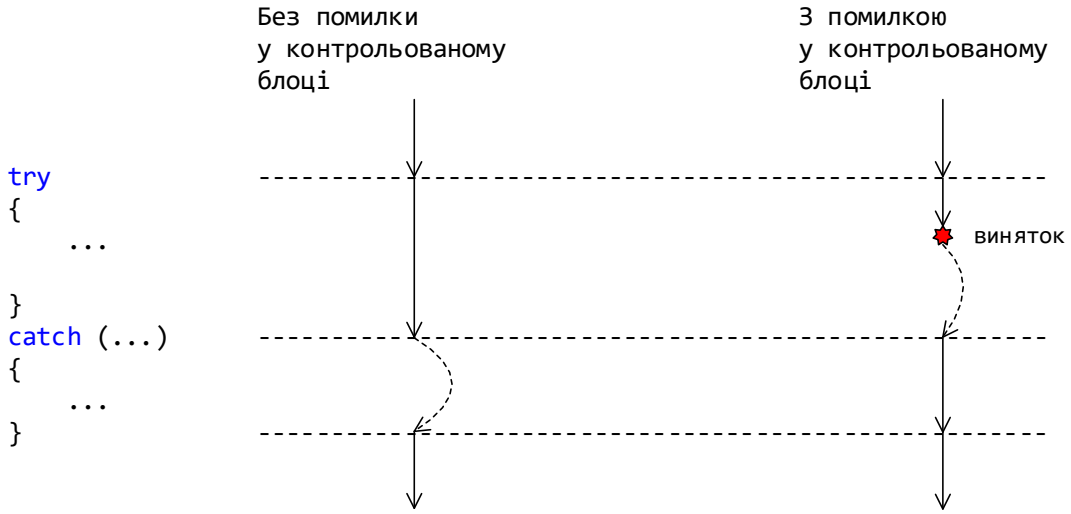
Виняток генерується командою `throw`, яка має наступний синтаксис:

```
throw вираз_генерування_винятку;
```

тут *вираз_генерування_винятку* – зазвичай або константа, або змінна деякого типу; допустимо вказувати і вираз. Тип об'єкта-винятку може бути як вбудованим, так і визначатися програмістом.

Перевірка виникнення винятку здійснюється командою оголошення *контрольованого* (його ще називають *захищеного*) блоку `try{}`, одразу після якої записуються один або кілька блоків опрацювання `catch...` (англ. *ловити*, *пастка* – тому їх ще називають *секції-пастки*).

Діаграми виконання блоку try-catch:



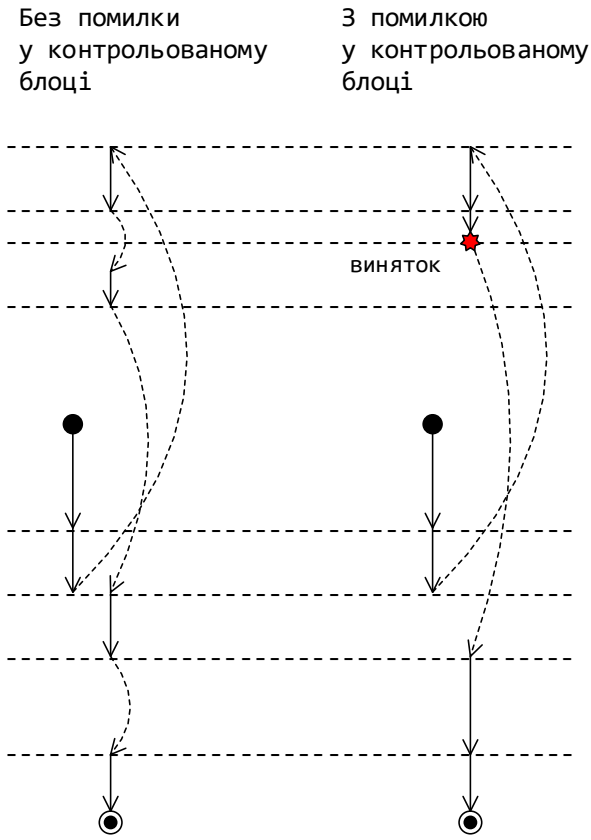
Якщо при виконанні команд контрольованого блоку (блоку try) не виникло помилок (не було виняткової ситуації, тобто не був згенерований виняток), то всі команди try-блоку виконуються в штатному режимі. Тоді команди catch-блоку пропускаються і управління передається першій команді, записаній після catch-блоку.

Якщо ж при виконанні try-блоку виникла помилка і був згенерований виняток, то управління одразу ж передається підходящому обробнику (тобто, catch-блоку).

Приклад

```
#include <iostream>
#include <Windows.h>                    // забезпечення відображення кирилиці

using namespace std;

class EDivideByZero {};                 // наш клас винятків

double div(double x, double y)
{
    if (y == 0)
        throw EDivideByZero();          // генеруємо об'єкт-виняток
    return x / y;
}

int main()
{
    SetConsoleCP(1251);                 // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251);           // забезпечення відображення кирилиці

    double x, y;
    cout << "x = "; cin >> x;
    cout << "y = "; cin >> y;

    try                                 // контрольований блок
    {
        cout << div(x, y) << endl;
    }
    catch (EDivideByZero)               // блок опрацювання винятку
    {
        cerr << "div: y==0" << endl;
    }

    return 0;
}
```

```
double div(double x, double y)
{
    if (y == 0)
        throw EDivideByZero();

    return x / y;
}

int main()
{
    ...

    try
    {
        cout << div(x, y) << endl;
    }
    catch (EDivideByZero)
    {
        cerr << "div: y==0" << endl;
    }

    return 0;
}
```



Оскільки винятки C++ не підтримують опрацювання асинхронних подій, таких, як апаратні помилки (зокрема, ділення на нуль) та опрацювання переривань (наприклад, натискання клавіш Ctrl+C), то потрібно в програмі вказати команду `throw` для програмного генерування винятку. В наведеному прикладі для ідентифікації винятку визначено клас `EDivideByZero`, і в команді `throw` створюється безіменний об'єкт цього класу:

```
throw EDivideByZero();
```

Виняток можна ідентифікувати і вбудованим типом, наприклад:

```
#include <iostream>
#include <Windows.h>                // забезпечення відображення кирилиці

using namespace std;

double div(double x, double y)
{
    if (y == 0)
        throw 1;                    // генеруємо виняток вбудованого типу
    return x / y;
}

int main() {
    SetConsoleCP(1251);             // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251);       // забезпечення відображення кирилиці

    double x, y;
    cout << "x = "; cin >> x;
    cout << "y = "; cin >> y;
```

```
    try                             // контрольований блок
    {
        cout << div(x, y) << endl;
    }
    catch (int)                     // блок опрацювання винятку
    {
        cerr << "div: y==0" << endl;
    }

    return 0;
}
```

Виняток також можна ідентифікувати динамічним об’єктом, наприклад:

```
#include <iostream>
#include <Windows.h>                // забезпечення відображення кирилиці

using namespace std;

class EDivideByZero {};             // наш клас винятків

double div(double x, double y)
{
    if (y == 0)
        throw new EDivideByZero();  // генеруємо динамічний об'єкт-виняток
    return x / y;
}

int main()
{
    SetConsoleCP(1251);             // забезпечення відображення кирилиці
    SetConsoleOutputCP(1251);       // забезпечення відображення кирилиці

    double x, y;
    cout << "x = "; cin >> x;
    cout << "y = "; cin >> y;

    try                             // контрольований блок
    {
        cout << div(x, y) << endl;
    }
    catch (EDivideByZero*)          // блок опрацювання винятку
    {
        cerr << "div: y==0" << endl;
    }

    return 0;
}
```

Контрольований (захищений блок)

Ключове слово `try` (англ. *пробувати*, *намагатися*) використовується для позначення *контрольованого* (іншими словами, *захищеного*) блоку – фрагменту коду, в якому може генеруватися виняток. Блок записується всередині фігурних дужок.

Загальний *синтаксис контрольованого блоку*:

```
try                             // контрольований блок
{
    ...                         // захищені команди
}
```

Контрольований блок має всі характеристики звичайного блоку: змінні, оголошені в ньому, є локальними і не доступні поза цим блоком.

Всі функції, які прямо або опосередковано викликаються із блоку `try`, також йому належать.

Генерування виняткiв

Генерування (*породження*) *винятку* виконується командою

```
throw [вираз];
```

Ключове слово `throw` (англ. *кидати*) використовується разом з виразом-аргументом, або без нього (квадратні дужки – не елемент синтаксису, а позначення необов'язковості).

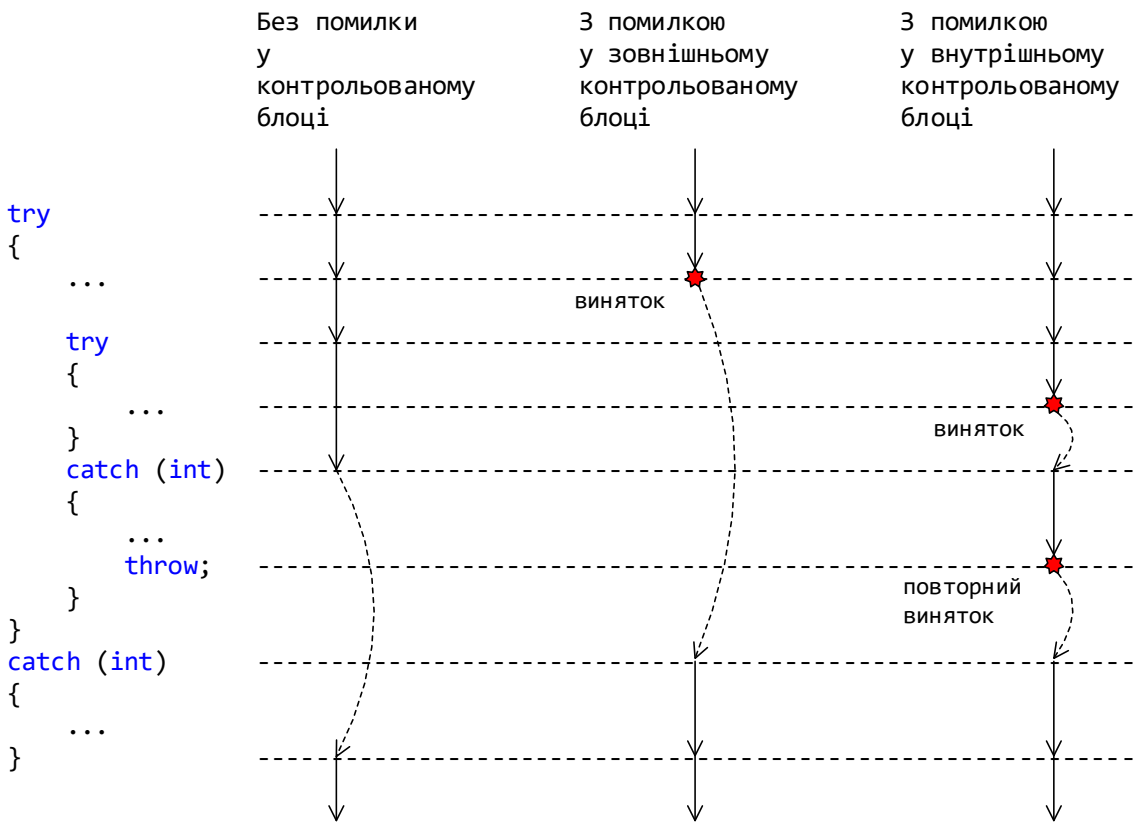
Тип виразу, вказаного після `throw`, визначає тип породженого винятку.

При генеруванні винятків виконання поточного блоку припиняється і відбувається пошук відповідного обробника і передача йому управління. Після того, як був згенерований виняток і управління передано в блок опрацювання, повернутися назад у контрольований блок – неможливо! Зазвичай, виняток генерується не безпосередньо у блоці `try`, а у функціях, що прямо або опосередковано входять до нього.

Не завжди виняток, який виник у внутрішньому блоці, може бути правильно опрацьований. В такому випадку використовуються вкладені блоки `try`, і виняток передається на зовнішній рівень за допомогою команди `throw` без параметрів:

```
try                         // зовнішній захищений блок
{
    ...                     // захищені команди зовнішнього блоку

    try                     // внутрішній захищений блок
    {
        ...                 // захищені команди внутрішнього блоку
    }
    catch (int)             // внутрішній блок опрацювання винятку
    {
        ...                 // команди внутрішнього блоку опрацювання винятку
        throw;              // передаємо управління зовнішньому блоку catch
    }

}
catch (int)                 // зовнішній блок опрацювання винятку
{
    ...                     // команди зовнішнього блоку опрацювання винятку
}
```

Діаграми виконання блоків try-catch:



Для представлення винятків часто використовують порожні класи, наприклад:

```
class EDivideByZero{};      // порожній клас винятків
class ENegativeArgument{};  // порожній клас винятків
```

Тоді генерувати винятки можна, створюючи безіменні об’єкти відповідних класів:

```
throw EDivideByZero();      // генеруємо безіменний виняток
```

Тут в якості виразу генерування винятку використовується явний виклик конструктора без аргументів – така форма генерування винятків найчастіше використовується. Об’єкт-виняток може бути і динамічним, наприклад:

```
throw new EDivideByZero();  // генеруємо безіменний динамічний виняток
```

Якщо в блок опрацювання винятку слід передати конкретну інформацію про те, що привело до помилки, це можна зробити за допомогою об’єкта-винятку відповідного класу:

```
class EDivideByZero         // непорожній клас винятків
{
public:
    string what;
};

EDivideByZero e;            // створили об'єкт
e.what = "division by zero";

throw e;                    // генеруємо об'єкт-виняток
```

Перехоплення та опрацювання винятків

Виняток потрібно перехопити та опрацювати.

Після того, як виняток був згенерований, управління із захищеного блоку `try` передається в блоки опрацювання винятків `catch`.

Безпосередньо після блоку `try{}` обов'язково слід записати один або кілька блоків `catch`, які називаються *обробниками винятків* або *блоками опрацювання винятків*.

Обробники винятків починаються з ключового слова `catch`, за яким в круглих дужках записується *специфікація обробника винятку*, потім – *блок команд опрацювання винятку*:

```
catch (специфікація_обробника_винятку)  // заголовок блоку опрацювання винятку
{
    ...                                 // блок команд опрацювання винятку
}
```

Специфікація обробника винятку може мати наступні три форми:

```
(тип ім'я)
(тип)
(...)
```

тип повинен бути одним із допустимих типів винятків – або вбудованим, або визначеним програмістом.

Перші дві форми із наведених специфікацій обробників винятків призначені для опрацювання конкретного типу винятку. Якщо ж замість специфікації обробника винятку записано три крапки, то такий обробник перехоплює всі винятки. Тому секцію-пастку з параметром – трьома крапками слід вказувати в списку catch-блоків останньою.

Коли за допомогою throw генерується виняток, виконуються наступні дії:

1) створюється копія параметра throw у вигляді статичного об'єкту, який існує до тих пір, поки виняток не буде опрацьований;
2) в пошуку відповідного обробника розкручується стек, при цьому викликаються деструктори локальних об'єктів, які виходять із області дії;
3) управління передається тому обробнику, який має параметр, сумісний з типом об'єкта-винятку.

Розкручування стеку – це процес знищення локальних об'єктів при виході із ланцюжка виклику функцій. При розкручуванні стеку всі обробники на кожному рівні переглядаються послідовно, від внутрішнього рівня до зовнішнього, поки не буде знайдено відповідний обробник.

Обробник буде знайдено, якщо тип об'єкта, вказаного після throw:

- такий же самий, що і вказаний в параметрі catch (параметр catch може бути записаний як параметр-значення T, параметр-константа const T, параметр-посилання T&, параметр-константне посилання const T&, де T – тип винятку);

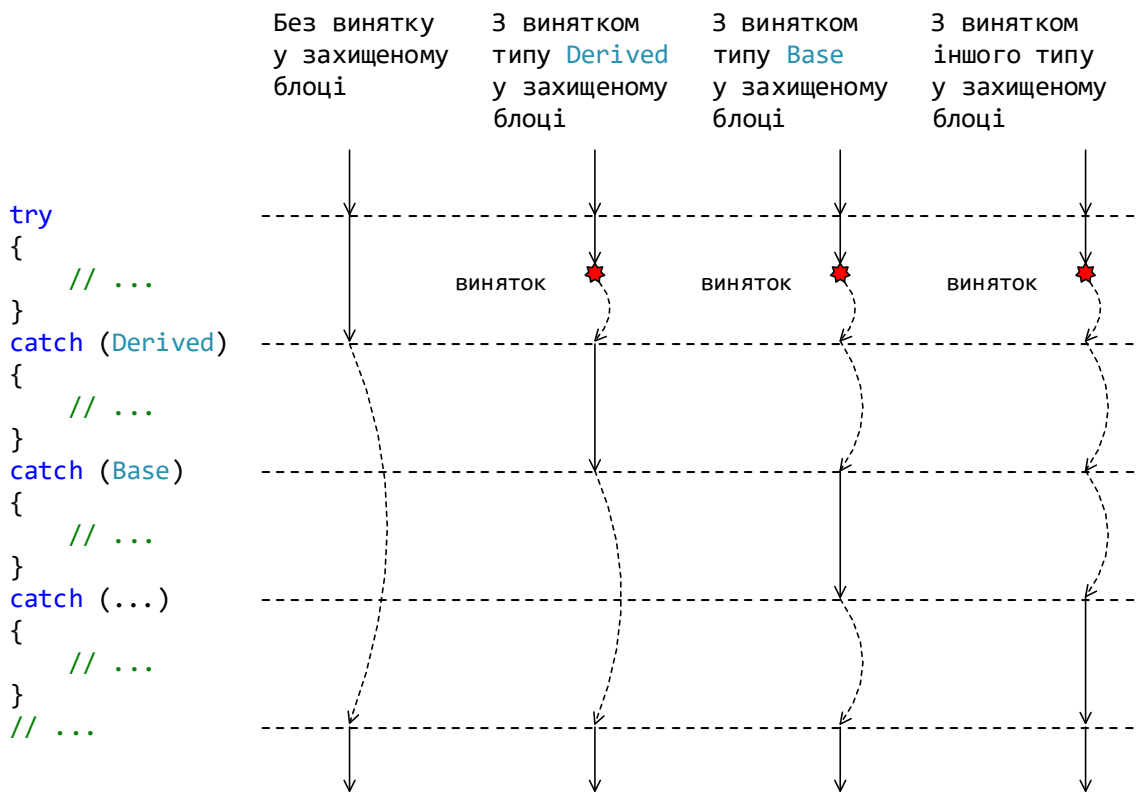
- є похідним від вказаного в параметрі `catch` (якщо успадковування – відкрите, тобто з ключем `public`);
- є вказівником, який можна перетворити згідно стандартних правил перетворення вказівників до типу вказівника в параметрі `catch`.

Із цього випливає, що *обробники похідних класів слід розміщувати до обробників базових*, оскільки в іншому випадку вони ніколи не отримають управління. Обробник вказівника типу `void*` автоматично перехоплює вказівник будь-якого іншого типу, тому його слід розміщувати після обробників вказівників конкретних типів.

Розглянемо приклад:

```
#include <iostream>

using namespace std;

class Base           // базовий клас винятків
{
};

class Derived        // похідний клас винятків
    : public Base
{
};

int main()
{
    try              // захищений блок
    {
        // ...
    }
    catch (Derived)  // перехоплюємо винятки похідного класу
    {
        // ...
    }
    catch (Base)     // перехоплюємо винятки базового класу
    {
        // ...
    }
    catch (...)      // перехоплюємо всі інші винятки
    {
        // ...
    }

    return 0;
}
```

Хід виконання блоків `try-catch` такої програми зображено на діаграмі:



Блоки try-catch можуть бути вкладеними, причому як у try-блок, так і у catch-блок:

```
try                         // зовнішній захищений блок
{
    ...                     // захищені команди зовнішнього блоку

    try                     // перший внутрішній захищений блок
    {
        ...                 // захищені команди 1-го внутрішнього блоку
    }
    catch (int)             // 1-й внутрішній блок опрацювання винятку
    {
        ...                 // команди внутрішнього блоку опрацювання винятку
    }

}
catch (int)                 // зовнішній блок опрацювання винятку
{
    ...                     // команди зовнішнього блоку опрацювання винятку

    try                     // другий внутрішній захищений блок
    {
        ...                 // захищені команди 2-го внутрішнього блоку
    }
    catch (int)             // 2-й внутрішній блок опрацювання винятку
    {
        ...                 // команди внутрішнього блоку опрацювання винятку
    }
}
```

Передавання інформації в блок опрацювання

Розглянемо наступний варіант специфікації обробника винятку:

(*тип ім'я*)

Він означає, що об'єкт-виняток передається у блок опрацювання, щоб там його деяким чином використати, наприклад – для виводу інформації в повідомленні про помилку. При цьому об'єкт-виняток може передаватися у секцію-пастку будь-яким способом: за значенням, за посиланням або за вказівником:

```
catch (exception e)             // за значенням
catch (exception& e)            // за посиланням
catch (const exception& e)      // за константним посиланням
catch (exception* e)            // за вказівником
```

Рекомендується передавати параметри за посиланням, оскільки в цьому випадку не створюється тимчасова копія об'єкту-винятку.

Якщо тип параметра секції-пастки `exception` – це вбудований тип, то ніяких перетворень за замовчуванням не відбувається. Це означає, що якщо заголовок секції-пастки має такий вигляд

```
catch (float e)                 // за значенням
```

то в цю секцію попадуть лише ті винятки, тип виразів яких – `float`. Команда

```
throw 3.14;
```

генерує виняток типу `double` (дійсні константи за замовчуванням мають саме цей тип), тому буде опрацьовуватися секцією-пасткою з заголовком

```
catch (double e)                // за значенням
```

В якості власних типів винятків можна реалізувати клас з полями, в якому конструктор має аргументи. Це дозволить при опрацюванні винятків отримувати та аналізувати необхідну інформацію.

Розглянемо, наприклад, клас-трикутник Triangle. Конструктор повинен перевірити сторони, якщо сторони задано не коректно, слід генерувати виняток.

Для того, щоб передати інформацію про помилку в секцію опрацювання винятку, створимо клас винятків Error:

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

class Error
{
    string message;

public:
    Error(const string& s)
        : message(s)
    {}

    string What() const { return message; }
};
```

```
class Triangle
{
    double a, b, c;

public:
    Triangle(double a, double b, double c)
    {
        if (a==0 || b==0 || c==0)
            throw Error("Zero-length side!");
        if (a > b+c || b > a+c || c > a+b)
            throw Error("It is not the triangle!");

        this->a = a;
        this->b = b;
        this->c = c;
    }

    double P() const { return a+b+c; }
};

int main()
{
    try
    {
        Triangle t(1, 4, 1);
        cout << t.P() << endl;
    }
    catch (Error e)
    {
        cout << e.What() << endl;
    }

    return 0;
}
```

Порядок та особливості опрацювання виняткiв

Коли в try-блоці виникає виняток, починається порівняння типу винятку і типів параметрів в секціях-пастках. Виконується той catch-блок, тип параметра якого збігається з типом винятку. Якщо catch-блоку такого типу немає, але є catch з трьома крапками, то виконується його блок. Якщо ж такого блоку в поточному переліку обробників немає, то здійснюється пошук іншого переліку обробників в місці виклику цієї функції. Такий пошук продовжується аж до функції main() включно. Якщо ж і там не буде знайдено потрібного catch-блоку, то буде викликана стандартна функція terminate(), яка, в свою чергу, викличе функцію abort().

Якщо catch-блок знайдено і виконано, то після виконання команд catch-блоку за відсутності явних команд переходу чи команди throw; виконуються команди, записані після усієї конструкції try-catch. Якщо під час виконання в try-блоці не виявлено виняткової ситуації, то всі catch-блоки пропускаються і програма продовжує виконання з першої команди після останнього catch-блоку.

Якщо управління перейшло до catch-блоку, то вихід із секції-пастки виконується одним із наступних способів:

1. Виконалися всі команди обробника (catch-блоку) – відбувається неявний перехід до першої команди, записаної після конструкції try-catch.
2. В обробнику виконується команда goto; – дозволено перехід до будь-якої команди поза конструкцією try-catch. Заборонено перехід всередину блоку try та в інший блок catch.
3. В обробнику виконується команда return; – відбувається нормальний вихід із функції.
4. В обробнику виконується команда throw; – така форма цієї команди допустима лише всередині секції-пастки. Спроба виконати цю команду поза catch-блоком приведе до негайного аварійного завершення програми.
5. В обробнику генерується інший виняток.

Команда throw; без виразу генерування винятку генерує повторний виняток такого ж типу. Тоді здійснюється пошук іншого обробника вгору по ієрархії вкладеності. Якщо потрібного catch-блоку не виявлено, програма аварійно завершується.

Вихід із обробника із-за генерування винятку може привести до виходу із функції. При цьому гарантується виклик деструкторів для знищення локальних об’єктів. Процес знищення локальних об’єктів при виході із ланцюжка виклику функцій із-за виникнення винятку називається «*розкручування*» (unwinding) стеку. Розкручування стеку виконується лише для локальних об’єктів – для динамічних об’єктів, створених операцією `new` деструктори автоматично не викликаються (вони викликаються при виконанні операції `delete`).

Класи та винятки

Винятки надають ідеальний механізм розв'язання проблем, пов'язаних з помилками при виконанні конструкторів та перевантаження операцій.

В цьому параграфі розглядаються особливості опрацювання винятків в конструкторах та деструкторах.

Винятки в списку ініціалізації конструктора

Поля класу можна ініціалізувати не лише в тілі, а і у списку ініціалізації конструктора, причому в якості виразу ініціалізації може виступати виклик функції або явний виклик конструктора. Тоді виникає питання: що робити, якщо при виконанні виразу ініціалізації виникає виняток? Блок try-catch не можна записати в тілі конструктора, бо тіло конструктора виконується після списку ініціалізації. Спеціально для вирішення цієї проблеми введено контрольований блок, який є тілом функції. Він називається контрольованим блоком рівня функції (*function-try-block*), і призначений для того, щоб ловити та опрацьовувати винятки, які виникають в списку ініціалізації конструктора:

```
int f(int);             // зовнішня функція, визначена в іншому місці

class C
{
    int i; double d;

public:
    C(int, double);     // конструктор ініціалізації
};

C::C(int k, double x)   // реалізація конструктора
try                     // контролюємо
: i( f(k) ), d(x)       // список ініціалізації
{
    // ...              // тіло конструктора
}
catch (...)             // обробник винятків
{
    // ...
}
```

Зверніть увагу: слово try тут записане перед списком ініціалізації конструктора – список ініціалізації тепер контролюється. Будь-який виняток, який згенерує функція f(), буде перехоплений та опрацьований в секції-пастці. При такій організації конструктора відсутні команди після блоку try-catch. Але вони і не потрібні, оскільки, якщо в конструкторі виник виняток, то процес створення об'єкту не завершився і нормального конструювання не відбулося. Вихід із цієї ситуації лише один – виконати ті дії щодо опрацювання винятку, які можливо, і перенаправити виняток далі за допомогою однієї з команд:

- throw без аргументу – повторно генеруємо перехоплений виняток;
- throw з аргументом – генеруємо новий виняток.

Можна оголосити контрольований блок-функцію для будь-якої функції. Наприклад, можна контролювати всі винятки, які виникають при виконанні програми за допомогою наступної конструкції:

```
int main()
try         // контрольований блок - тіло функції
{
    // ...
    return 0;
}
catch (...)
{
    // опрацювання винятків
}
```

Якщо в контрольованому блоці не виникло винятків, то блок функції нормально завершиться. якщо ж виникнуть винятки, то (як і завжди) виконується секція-пастка і лише після цього програма завершується. Зрозуміло, що секцій-пасток може бути стільки, скільки потрібно.

Якщо така конструкція використовується в звичайній функції чи методі, то вийти із секції-пастки можна за допомогою команди return.

Зауваження: не всі компілятори підтримують контрольований блок рівня функції.

Винятки та деструктори

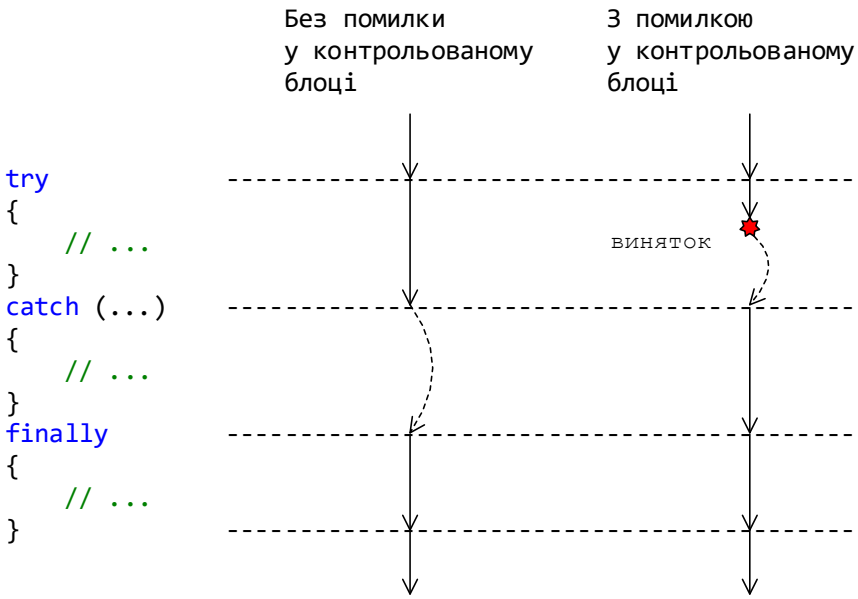
В зв'язку з тим, що деструктор викликається для знищення локальних об'єктів при генеруванні будь-якого винятку, сам деструктор генерувати винятки не повинен (хоча, взагалі кажучи, може). Деструктори повинні створюватися так, начебто вони мають специфікацію винятків throw() (див. наступний параграф), тобто жодний виняток не може вийти за межі деструктора. Якщо ж це відбулося, то програма завершується аварійно. Це означає, що винятки, які виникають в деструкторі, повинні бути перехоплені та опрацьовані в самому ж деструкторі.

Є ще один цікавий момент, пов'язаний із деструкторами та опрацюванням винятків:

Багато мов програмування (зокрема, java та C#, але не C++) крім *захищеного блоку* (блоку try) та *блоків опрацювання* (блоків catch) мають ще одну синтаксичну конструкцію, яка стосується роботи з винятками: *блок очистки* (блок finally).

В мові C++ окремої синтаксичної конструкції для «блоку очистки» (блоку finally) немає!

Блок очистки записується одразу після блоків опрацювання і завжди гарантовано виконується (не залежно від того, чи виникли винятки, чи ні).



Блок finally використовується для гарантованого звільнення ресурсів, які були виділені до виникнення винятку.

Оскільки в мові C++ блоку очистки немає, то його роль повинні виконувати деструктори локальних об'єктів: визначаємо клас, конструктор якого виділяє необхідні ресурси, а деструктор – звільняє. В захищеному блоці створюємо локальний об'єкт (при цьому конструктором будуть виділені ресурси).

При нормальному завершенні блоку try локальний об'єкт буде автоматично знищено. Деструктор, який буде автоматично викликаний, звільнить ресурси.

Якщо при виконанні блоку try виник виняток, управління передається в секцію catch, при цьому виконується розкручування стеку і, знову ж таки, автоматично викликається деструктор, який звільняє ресурси:

```
#include <iostream>

using namespace std;


class Resource
{
public:
    Resource()                                  // конструктор
    {                                           // виділяє ресурси
        cout << "create resources" << endl;
    }
    ~Resource()                                 // деструктор
    {                                           // звільняє ресурси
        cout << "release resources" << endl;
    }
};
```

```
int main()
{
    try
    {
        Resource r;                         // виділили ресурси
        throw 1;                            // згенерували виняток
    }
    catch (...)                             // опрацювання винятку
    {
        cout << "process exception" << endl;
    }

    return 0;
}
```

Специфікація винятків (список винятків функції)

Для кожної функції, метода, конструктора чи деструктора можна в заголовку вказати *специфікацію винятків*.

Якщо в заголовку не вказана специфікація винятків, вважається, що функція може породжувати і передавати назовні будь-який виняток. Це означає, що будь-який виняток може бути згенерований, переданий назовні та опрацьований поза функцією.

В заголовку можна явно вказати список винятків, які породжує і передає назовні функція, – тобто, винятків, які функція генерує і може передавати назовні для опрацювання. Це – специфікація винятків функції:

```
// прототипи функцій:
void f1() throw(int, double);  // генерує і передає назовні винятки int, double
void f2() throw(NullArgument); // генерує і передає назовні виняток NullArgument
void f3() throw(MyException);  // генерує і передає назовні виняток MyException
```

Якщо в заголовку вказано порожню специфікацію (порожній список винятків функції):

```
// прототипи функцій:
void f4() throw();             // не генерує і не передає назовні винятків
```

то це означає, що функція не породжує винятків (тобто, функція не може згенерувати ніяких винятків та передати їх для опрацювання назовні).

Наявність специфікації винятків насправді не накладає ніяких обмежень при реальному генеруванні винятків – функція фактично може згенерувати виняток будь-якого типу. Проте такі неспецифіковані винятки повинні бути опрацьовані в тілі функції (тіло функції для цього повинно містити блоки try-catch). Якщо ж функція генерує неспецифікований виняток, який залишився не опрацьованим в тілі функції, то запускається стандартна функція unexpected(), яка викликає функцію terminate(). Це приводить до аварійного завершення програми.

Не всі компілятори C++ підтримують специфікацію винятків в повному обсязі відповідно до стандарту. Зокрема, MS Visual C++ .NET реалізує специфікацію винятків в наступних формах:

- throw() – функція не повинна породжувати винятків;
- throw(...) – функція може породжувати будь-які винятки;
- throw(*типи*) – вказівка типів ігнорується, ця специфікація еквівалентна throw(...) – така функція може породжувати будь-які винятки.

Якщо специфікація винятків оголошена у заголовку, то вона повинна бути вказана і у всіх прототипах. Проте, специфікація винятків не входить до прототипу функції, тому функції, які відрізняються лише специфікаціями винятків, не вважаються різними при перевантаженні.

Компілятор Microsoft Visual Studio C++ ігнорує специфікацію винятків:

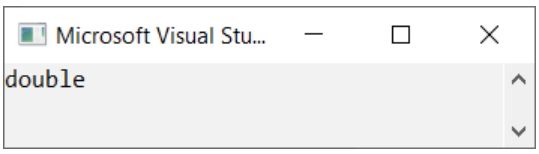
```
#include "pch.h"
#include <iostream>

using namespace std;

void f() throw(int)
{
    throw 1.0;
}

int main()
{
    try
    {
        f();
    }
    catch (int)
    {
        cout << "int" << endl;
    }
    catch (double)
    {
        cout << "double" << endl;
    }

    return 0;
}
```

– виводить повідомлення:

C++ exception specification ignored except to indicate a function is not __declspec(nothrow)

– вказівка типу ігнорується, ця специфікація еквівалентна `throw(...)` – функція може породжувати будь-які винятки.

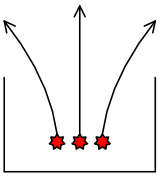
Результат виконання:



Специфікацію винятків функції можна уявити як певний фільтр, через який можуть пройти лише специфіковані винятки, – наче своєрідні „ворота”, що „випускають назовні” лише ті згенеровані функцією винятки, які вказані явно в специфікації:

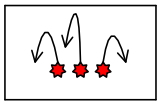
```
void f();
```

„воріт” взагалі немає



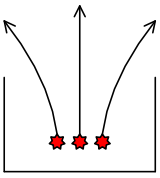
```
void f() throw();
```

„ворота” закриті



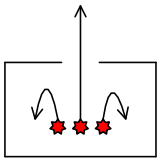
```
void f() throw(...);
```

„ворота” відкриті



```
void f() throw(int);
```

„ворота випускають” назовні лише винятки типу `int`



Стандартні винятки

В стандартній бібліотеці C++ реалізовано ряд стандартних винятків, які утворюють ієрархію класів. Для роботи із ними в програмі необхідно вказати команду

```
#include <stdexcept>
```

Ієрархія стандартних винятків

```
class exception { /* ... */ };
    class logic_error:          public exception { /* ... */ };
        class domain_error:         public logic_error { /* ... */ };
        class invalid_argument:     public logic_error { /* ... */ };
        class length_error:         public logic_error { /* ... */ };
        class out_of_range:         public logic_error { /* ... */ };
    class runtime_error:        public exception { /* ... */ };
        class range_error:          public runtime_error { /* ... */ };
        class overflow_error:       public runtime_error { /* ... */ };
        class underflow_error:      public runtime_error { /* ... */ };
    class bad_alloc:            public exception { /* ... */ };
    class bad_cast:             public exception { /* ... */ };
    class bad_typeid:           public exception { /* ... */ };
    class bad_exception:        public exception { /* ... */ };
    class ios_base::failure:    public exception { /* ... */ };
```

Ця ієрархія – основа для створення власних винятків та ієрархій винятків. Ми можемо визначати власні винятки як нащадки класу exception.

Клас exception визначено в стандартній бібліотеці таким чином:

```
class exception
{
public:
    exception() throw();
    exception(const exception&) throw();
    exception& operator = (const exception&) throw();
    virtual ~exception() throw();
    virtual const char* what() const throw();
};
```

Всі конструктори і методи мають специфікацію, яка забороняє породження винятків. Метод what() видає повідомлення про помилку. Успадковування від стандартних класів дозволяє використовувати метод what() для виведення повідомлення про причини помилки. Для успадковування від стандартних винятків слід включити файл

```
#include <exception>
```

Припускається, що винятки типу logic_error сигналізують про помилки в логіці програми, наприклад про невиконання деякої умови. Категорія runtime_error – це помилки, які виникають в результаті непередбачених обставин при виконанні програми, наприклад, переповнення при операціях з дробовими числами. Такі винятки програма повинна породжувати самостійно командою throw.

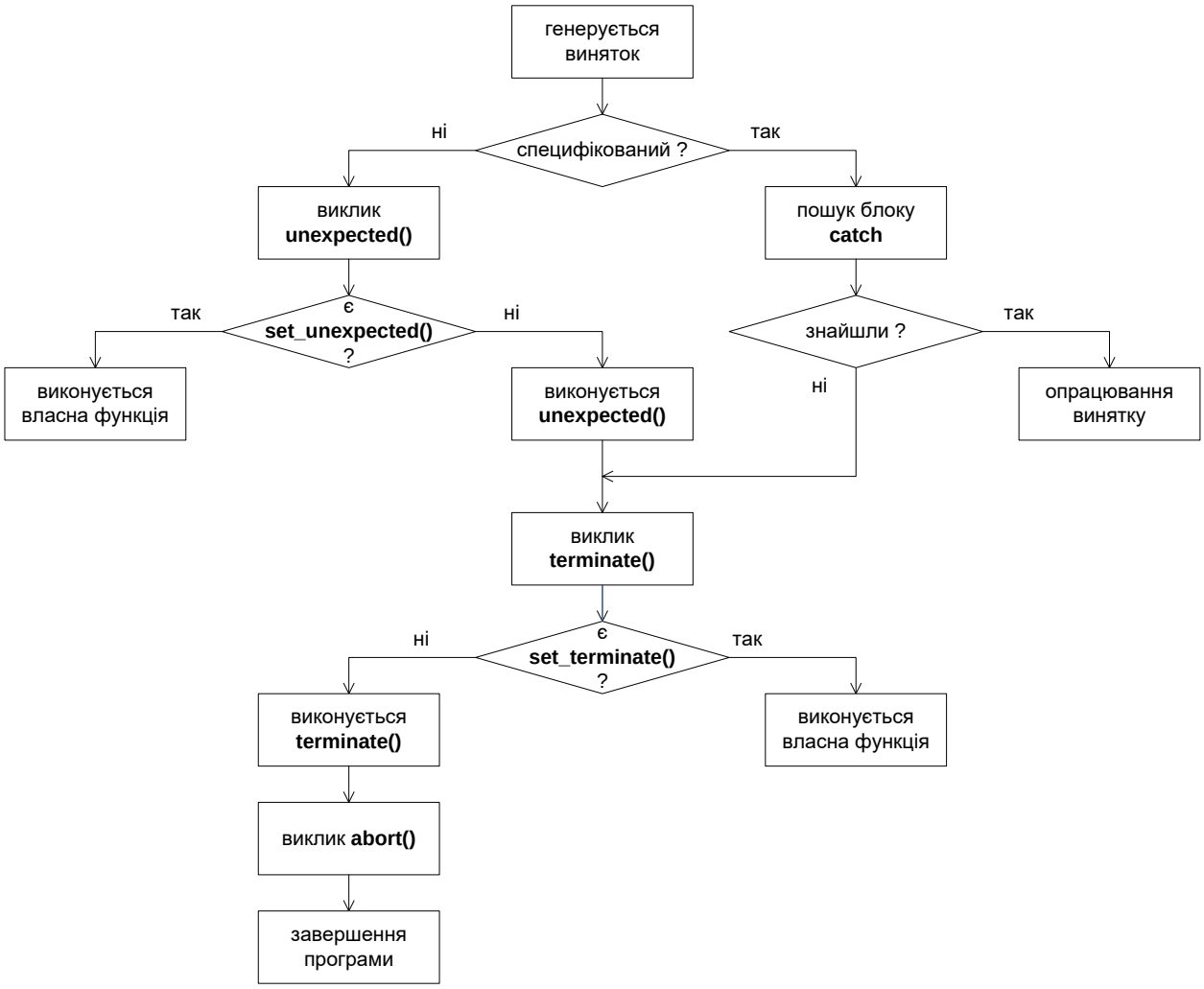
П’ять стандартних винятків породжують різні механізми C++. Ці винятки теж можна явно використовувати в команді throw, проте це робити не рекомендується. Виняток bad_alloc породжує операція new (або new[ ]), якщо запит на виділення пам’яті не може бути виконаний. Винятки bad_cast та bad_typeid породжуються механізмом RTTI (Run-Time Type Identification – динамічна ідентифікація типів). Системою вводу-виводу породжується виняток ios_base::failure. Виняток bad_exception породжується при порушенні специфікації винятків функції і відіграє важливу роль в підміні стандартних функцій завершення.

Підміна функцій стандартного завершення

Коли функція породжує специфікований виняток, – в штатному режимі управління передається відповідному обробнику.

В цьому параграфі розглядається нетривіальна ситуація, коли виняток, породжений функцією, не входить у список специфікації.

Схематично алгоритм дій при породженні винятків зображено на діаграмі:



Функція terminate() викликається або із функції unexpected(), якщо порушена специфікація винятку, або механізмом опрацювання винятків, якщо відсутня відповідна секція-пастка.

Функції стандартного завершення

Стандартна функція unexpected() за замовчуванням просто викликає стандартну функцію terminate(). За допомогою функції set_unexpected() можна встановити власну функцію, яка буде викликатися замість unexpected() і визначати дії програми при виникненні непередбаченої виняткової ситуації.

Функція terminate() за замовчуванням викликає стандартну функцію abort(), яка завершує виконання програми. За допомогою функції set_terminate() можна встановити власну функцію, яка буде викликатися замість terminate() і визначати спосіб завершення програми.

Функції set_unexpected() та set_terminate() описані у файлі <exception>.

Механізм підміни функцій стандартного завершення

Для підміни стандартних функцій завершення необхідно підключити заголовок

```
#include <exception>
```

Щоб підмінити стандартну функцію terminate(), потрібно визначити власну функцію з прототипом

```
void F();                       // без параметрів і без результату
```

та вказати її ім'я в команді виклику функції set_terminate() :

```
set_terminate(F);               // встановили нашу функцію
```

При цьому можна зберегти адресу попереднього обробника:

```
void (*old_terminate)() = set_terminate(F);
```

де old_terminate – вказівник на функцію, який отримує значення попереднього обробника неперехоплених винятків.

Після цього замість terminate() при опрацюванні не перехоплених винятків буде викликатися наша функція F().

Наступна програма демонструє приклад підміни функції terminate().

Якщо інтегроване середовище налаштоване так, щоб перехоплювати всі неопрацьовані винятки, то виконувати цю програму потрібно запуском .exe-файлу.

```
#include <iostream>
#include <exception>

using namespace std;


void F()                                    // наша функція підміни – буде замість
{                                           // функції terminate()
    cout << "uncaught exception!" << endl;
}

int main()
{
    set_terminate(F);                        // встановили нашу функцію
    try
    {
        throw 1;                             // згенерували виняток
    }
```

```
    catch (double)                          // опрацювання винятку
    {
        cout << "normal process exception" << endl;
    }

    return 0;
}
```

Функція-«термінатор» не повинна повертати управління командою return, не повинна генерувати виняток командою throw – вона може лише завершити програму функцією exit() чи abort(). Зате перед завершенням можна виконати дії, які дозволять розібратися в причині виклику «термінатора», наприклад, записати інформацію про помилку у файл.

Аналогічно реалізується підміна стандартної функції unexpected():

```
set_unexpected(F);                          // встановили нашу функцію
```

Функція опрацювання не перехопленого винятку, як і термінальна функція, не повинна повертати управління командою return і завершує програму функцією exit() чи abort(). Окрім цього вона може згенерувати виняток, вказаний в специфікації винятків. Відбудеться підміна не перехопленого винятку «легальним» і далі опрацювання винятків відбуватиметься стандартним чином.

Функція може згенерувати інший виняток, – не специфікований, або просто «відправити» незаявлений виняток «далі» командою throw. В цьому випадку, якщо в специфікації винятків не вказано винятку bad_exception, викликається функція terminate(). А якщо специфікація винятків містить bad_exception, згенерований виняток підмінюється винятком bad_exception, і починається пошук його обробника. Оскільки передбачити тип несподіваного винятку неможливо, пропонується підмінювати *всі* не перехоплені винятки одним типом: або стандартним винятком bad_exception, або нашим власним типом. Перший варіант реалізується так:

```
#include <exception>

void _unexpected_to_bad()
{
    throw;                                  // генеруємо bad_exception
}

int main()
{
    set_unexpected(_unexpected_to_bad);     // встановили нашу функцію
    // ...
    return 0;
}
```

А другий – так:

```
#include <exception>

class MyUnexpectedException {};                 // наш клас невідомих винятків

void _unexpected_to_my()
{
    throw MyUnexpectedException();              // генеруємо наш виняток
}

int main()
{
    set_unexpected(_unexpected_to_my);          // встановили нашу функцію
    // ...
    return 0;
}
```

У всі специфікації винятків функцій потрібно добавити відповідний тип – і можна забути про несподівані винятки!

Зауваження 1.

Для того, щоб спрацювала підміна стандартних функцій завершення, слід або відключити опрацювання винятків IDE Microsoft Visual Studio, або запускати на виконання .exe-файл (тобто, запускати програму не з середовища Microsoft Visual Studio).

Зауваження 2.

Оскільки специфікація винятків функції з вказаними типами винятків throw(*типи*) ігнорується Microsoft C++, то неможливо продемонструвати виняток bad_exception, роботу функції unexpected() та її підміну за допомогою функції set_unexpected().

Assert – примусове генерування винятку

Використання команд розгалуження для виявлення помилкових команд, а також виведення повідомлень про помилки та завершення виконання програми – настільки поширене рішення проблем, що в C ++ є окрема конструкція для цього – assert.

Assert – це спеціальна конструкція, що дозволяє перевіряти значення довільних даних в довільному місці програми. Ця конструкція може автоматично сигналізувати при виявленні некоректних даних, що призводить до аварійного завершення програми із зазначенням місця виявлення некоректних даних.

Навіщо примусово вивалювати програму у виняткову ситуацію? – це ж по крайній мірі дивно..! Справа в тому, що програма з некоректними даними може ще довго виконуватися, але її поведінка стає непередбачуваною. Оскільки assert завершує програму відразу ж після виявлення некоректних даних, він дозволяє швидко локалізувати і виправити баги в програмі, які привели до некоректних даних – це його основне призначення.

Assert-и доступні в багатьох мовах програмування, включаючи C++, java та C#.

Макрос assert

Макрос assert (або «команда перевірки твердження») в мові C ++ – це макрос препроцесора, який опрацьовує умовний вираз під час виконання. Якщо умовний вираз істинний, то макрос assert нічого не робить. Якщо ж вираз хибний, то виводиться повідомлення про помилку, і програма завершується. Це повідомлення про помилку містить хибний умовний вираз, а також ім'я файлу з кодом і номером рядка з assert. Таким чином, можна легко знайти та ідентифікувати проблему, що дуже допомагає при відлагодженні програм.

Сам assert реалізований у заголовному файлі cassert і часто використовується як для перевірки коректності переданих параметрів функції, так і для перевірки значення, що повертається функцією:

```
1  #include <iostream>
2  #include <cassert>  // для assert()
3
4  using namespace std;
5
6  int getArrayValue(int* array, int size, int index)
7  {
8      // Значення index-у має бути між 0 та size-1
9      assert(index >= 0 && index < size);
10
11     return array[index];
12 }
13
```

```
14  int main()
15  {
16      const int size = 10;
17      int a[size] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
18
19      cout << getArrayValue(a, size, 1)  << endl;
20      cout << getArrayValue(a, size, 10) << endl;
21
22      return 0;
23  }
```

Програма виводить наступне:

Рекомендується використовувати assert. Іноді умови assert бувають не зовсім описовими, наприклад:

```
assert(found);
```

Якщо цей assert спрацює, то отримаємо наприклад, – таке:

```
Assertion failed: found, file C:\\VCProjects\\Program.cpp, line 42
```

Про що це повідомлення? Очевидно, що щось не було знайдено, але що саме? Вам потрібно буде самому пройтися по коду, щоб це визначити.

На щастя, є хитрість, яку можна використовувати для виправлення цієї ситуації. Просто додайте повідомлення в якості рядка C-style разом з логічним оператором &&:

```
assert(found && "Animal could not be found in database");
```

Рядок C-style завжди приймає значення `true`. Тому, якщо `found` прийме значення `false`, то `false && true == false`. Якщо ж `found` прийме значення `true`, то `true && true == true`. Таким чином, рядок C-style взагалі не впливає на опрацювання умови макросу assert.

Однак, якщо assert спрацює, то рядок C-style буде добавлений до повідомлення assert – наприклад, так:

```
Assertion failed: found && "Animal could not be found in database", file
C:\\VCProjects\\Program.cpp, line 42
```

Це дасть додаткове пояснення причини помилки.

Директива NDEBUG

Макрос assert() витрачає ресурси на перевірку умови. Крім того, макроси assert (в ідеалі) ніколи не мають зустрічатися у версії коде Release (бо ваш код до цього моменту вже має бути ретельно протестований). Тому багато розробників вважають за краще використовувати assert лише в конфігурації Debug. У мові C ++ є можливість відключити

всі assert-и в Release-версії коду – для цього слід використовувати директиву `#define NDEBUG`:

```
#define NDEBUG // Всі assert-и будуть проігноровані до кінця цього файлу
```

Деякі IDE встановлюють NDEBUG за умовчанням, як частина параметрів проекту в конфігурації Release.

Зверніть увагу: функція exit() і assert (якщо він спрацьовує) негайно припиняють виконання програми, без можливості виконати подальшу будь-яку очистку (наприклад, закрити файл або базу даних). Через це їх слід використовувати акуратно.

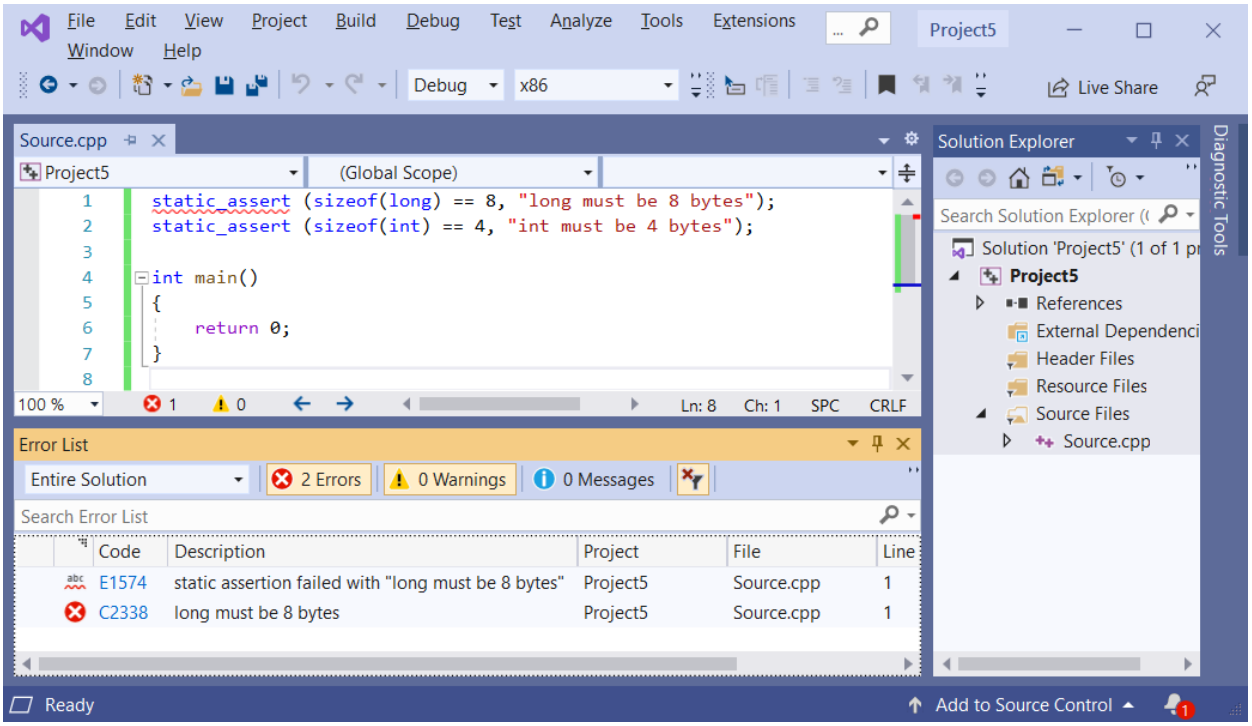
Макрос static_assert

В мові C++ 11 додали ще один тип assert-у – static_assert. На відміну від assert, який спрацьовує під час виконання програми, static_assert спрацьовує під час компіляції, викликаючи помилку компілятора, якщо умова не є істинною. Якщо умова – хибна, то виводиться діагностичне повідомлення.

Ось приклад використання static_assert для перевірки розмірів певних типів даних:

```
static_assert (sizeof(long) == 8, "long must be 8 bytes");
static_assert (sizeof(int) == 4, "int must be 4 bytes");

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

Ось результат спроби компіляції:



Оскільки static_assert опрацьовується компілятором, то умова static_assert теж має опрацьовуватися під час компіляції. Оскільки макроси static_assert не опрацьовуюься під час виконання програми, то вони можуть бути розміщені в будь-якому місці коду (навіть в глобальному просторі).

В мові C++ 11 діагностичне повідомлення має бути обов'язково надано в якості другого параметра. В C++ 17 надання діагностичного повідомлення є необов'язковим.

Клас Assert

Клас Assert простору імен Microsoft.VisualStudio.TestTools.UnitTesting служить для автоматичного тестування функціональних можливостей проекту. Метод модульного тесту використовує код деякого методу з основного проекту, але повідомляє про коректність поведінки коду тільки в тому випадку, якщо включені команди Assert.

В методі тесту можна викликати будь-які методи класу Microsoft.VisualStudio.TestTools.UnitTesting.Assert, наприклад Assert.AreEqual(). Клас Assert містить багато методів, і багато з цих методів мають кілька перевантажень.

Метод Assert.AreEqual() перевіряє, – чи однакові зазначені об'єкти, і генерує виняток типу AssertFailedException, якщо два об'єкти – різні. Різні числові типи вважаються неоднаковими, навіть якщо їх значення збігаються: 42L не дорівнює 42.

Лабораторний практикум

Оформлення звіту про виконання лабораторних робіт

Вимоги до оформлення звіту про виконання лабораторних робіт №№ 5.1–5.3

Звіт про виконання лабораторних робіт №№ 5.1–5.3 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 №№ 5.1–5.3

ЗВІТ

про виконання лабораторної роботи № < *номер* >

« *назва теми лабораторної роботи* »

з дисципліни

«Об’єктно-орієнтоване програмування»

студента(ки) групи КН-26

< *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 № 5.1. Класи з опрацюванням виняткових ситуацій

Мета роботи

Освоїти опрацювання виняткових ситуацій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення винятку.
2) Поняття та призначення контрольованого (захищеного) блоку.
3) Поняття та призначення блоку опрацювання винятку.
4) Генерування (створення) винятку.
5) Механізм опрацювання винятків.
6) Повторне генерування винятку в блоці опрацювання.
7) Винятки в конструкторах та деструкторі.
8) Специфікація винятків в заголовку функції.
9) Стандартні винятки та їх опрацювання.
10) Підміна функцій стандартного завершення.

Варіанти завдань

Реалізувати опрацювання винятків в трьох варіантах:

- використовувати відповідні стандартні винятки;
- використовувати винятки-нащадки від стандартних винятків;
- визначити власні винятки;

Передачу об’єкта-винятку виконати різними способами:

- за значенням,
- за посиланням,
- за вказівником.

Конструктори і методи, що генерують винятки, повинні бути оголошені із специфікацією винятків.

Завдання А

Виконати свій варіант Лабораторної роботи № 1.1 «Класи з двома полями» з конструкторами та опрацюванням винятків (Лабораторна робота № 2.1). Продемонструвати опрацювання стандартного винятку bad_exception.

Лабораторна робота № 2.1:

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 повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.1:

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

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( ); метод повинен 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 повинна 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 повинен 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 повинен 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 повинен перевіряти нерівність знаменника нулю.

Варіант 4.

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

Варіант 5.

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

Варіант 6.

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

Варіант 7.

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

Варіант 8.

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

Варіант 9.

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

Варіант 10.

Реалізувати клас Line. Лінійне рівняння $y = Ax + B$. Поле first – дійсне число,

коефіцієнт A ($A \neq 0$); поле second – дійсне число, коефіцієнт B. Реалізувати метод function( ) – обчислення для заданого x значення функції y.

Варіант 11.

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

Варіант 24.

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

Варіант 25.

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

Варіант 26.

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

Варіант 27.

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

Варіант 28.

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

Варіант 29.

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

Варіант 30.

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

Варіант 31.

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

Завдання В

Виконати свій варіант Лабораторної роботи № 1.3 «Об’єкти – параметри методів (дії над кількома об’єктами)» з конструкторами, перевантаженням операцій (Лабораторна робота № 2.3) та опрацюванням винятків.

Лабораторна робота № 2.3:

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 повинні 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 повинні бути продемонстровані різні способи створення об’єктів і масивів об’єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.3:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково повинні бути реалізовані наступні методи:

- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів – різними конструкторами. Програма повинна 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 – поточнний 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

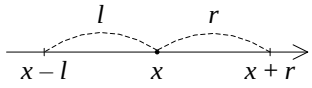
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

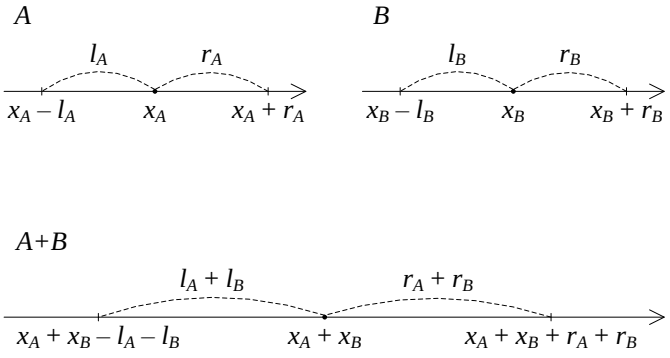
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 8.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,

- порівняння довжин векторів.

Варіант 14.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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().

* Повинна 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,

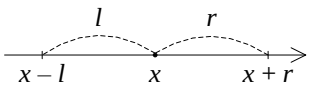
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

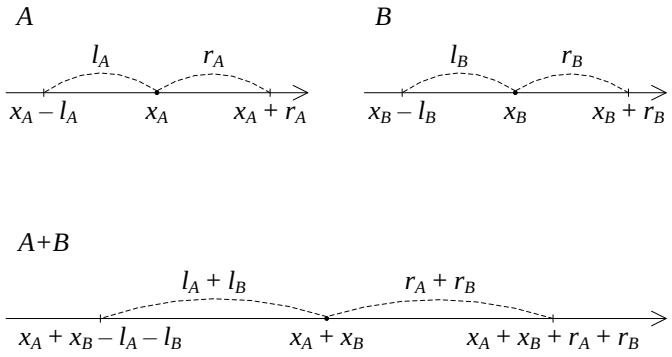
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 19.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,

- скалярний добуток векторів.

Варіант 28.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 – поточнний 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

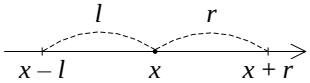
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

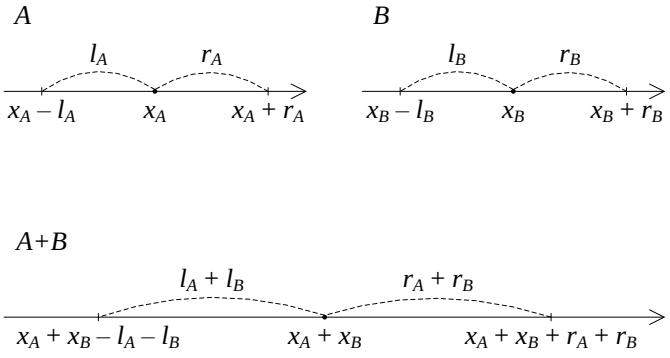
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 33.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,

- порівняння довжин векторів.

Варіант 39.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Завдання С

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання А*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання А):

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 2.3 «Конструктори та перевантаження операцій для класів» з конструкторами, але без перевантаження операцій, – в якості базових класів. Реалізувати для базових класів методи вводу-виводу як дружні функції. Лабораторна робота № 2.3 виконується на основі Лабораторної роботи № 1.3 «Об'єкти – параметри методів (дії над кількома об'єктами)».

Реалізувати класи-нащадки, які розширюють контент базових класів перевантаженими операціями: базові класи містять конструктори та операції вводу-виводу, а похідні – всі інші операції. В нащадках методи базових класів можна використовувати для реалізації перевантажених операцій.

Класи-нащадки реалізувати в двох варіантах: як відкриті і як закриті класи-нащадки.

При відкритому успадковуванні не визначати функції вводу-виводу для класів-нащадків.

Лабораторна робота № 2.3:

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 повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.3:

У всіх завданнях, крім вказаних в завданні операцій, обов'язково повинні бути реалізовані наступні методи:

- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів – різними конструкторами. Програма повинна 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 повинні

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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

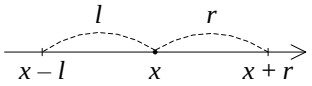
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

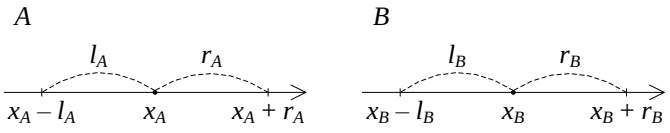
r – відстань від правої границі до центру:

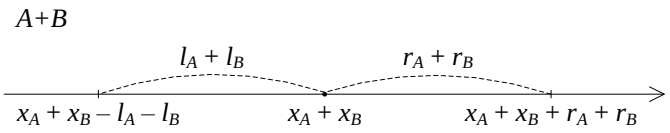


Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 8.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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() conj(x, y) = (x, $–y$).

Варіант 13.

Створити клас Vector3D, що задається трійкою координат. Поля

- x
- y
- z

Обов'язково повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 14.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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().

* Повинна 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,

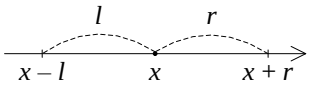
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

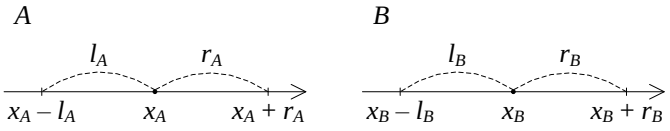
r – відстань від правої границі до центру:

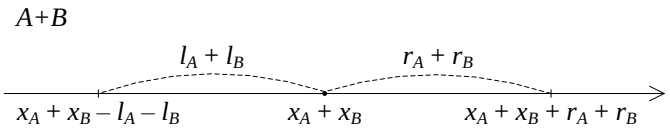


Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 19.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 28.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

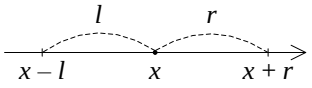
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

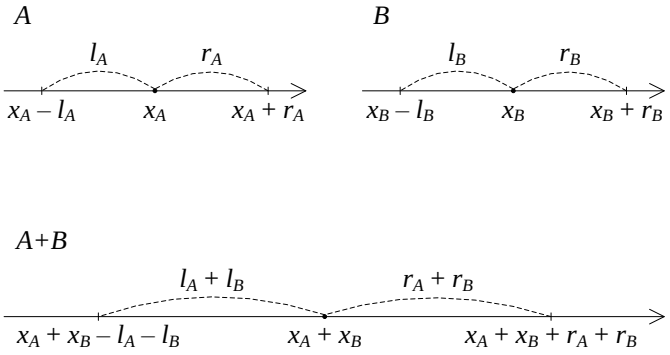
r – відстань від правої границі до центру:



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 повинна бути

представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 33.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 39.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні бути реалізовані:

- перетворення у полярні координати,

- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Завдання D

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання B*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання B):

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 1.3 «Об'єкти – параметри методів (дії над кількома об'єктами)» як класи-нащадки; використовувати відкрите успадковування.

В якості базового класу реалізувати відповідний клас: пару чисел (Pair) або трійку чисел (Triad) з конструкторами та повним набором операцій порівняння.

Клас Pair (пара чисел): пара p1 більше пари p2, якщо

(p1.first > p2.first)

або

(p1.first = p2.first) і (p1.second > p2.second).

Клас Triad (трійка чисел): 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).

Реалізувати для базових класів методи вводу-виводу як дружні функції.

Не визначати функції вводу-виводу для класів-нащадків.

Лабораторна робота № 1.3:

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

- методи доступу (константні методи зчитування та методи запису) значення кожного поля;
- метод ініціалізації Init( ); метод повинен 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 повинна 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.

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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

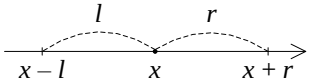
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

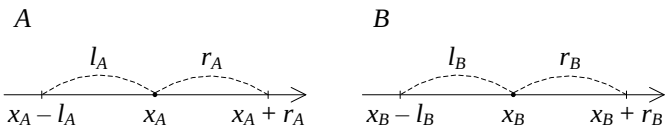
r – відстань від правої границі до центру:

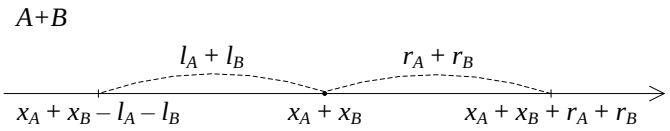


Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 8.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 14.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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().

* Повинна 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,

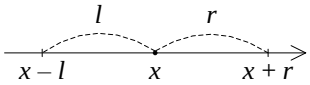
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

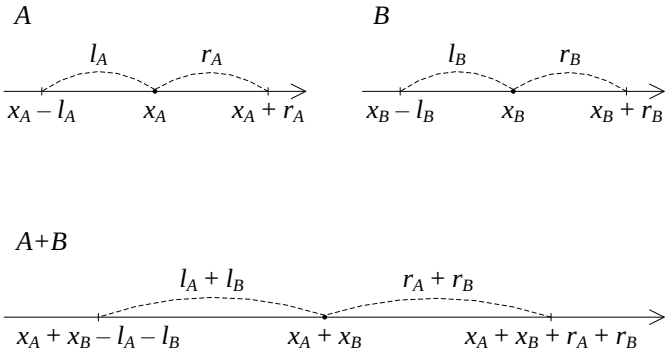
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 19.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 28.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

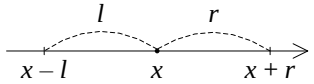
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

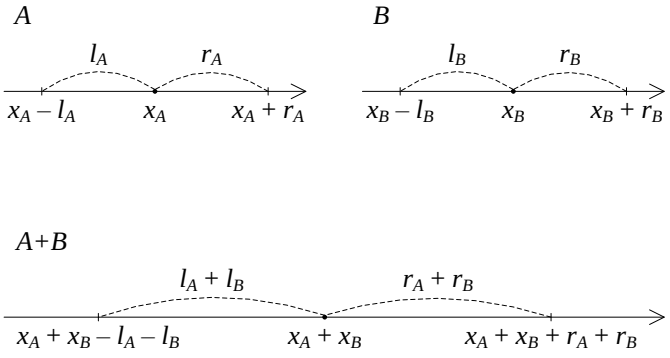
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 33.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,

- порівняння довжин векторів.

Варіант 39.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Завдання E

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання C*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання C):

Реалізувати класи із завдання свого варіанту Лабораторної роботи № 1.3 «Об'єкти – параметри методів (дії над кількома об'єктами)» як класи-нащадки (відкрите успадковування) з перевантаженням операцій та конструкторами (оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.3 «Конструктори та перевантаження операцій для класів»).

В якості базового класу реалізувати клас Object з лічильником кількості створених об'єктів.

Лабораторна робота № 2.3:

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 повинні 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 повинні бути продемонстровані різні способи

створення об’єктів і масивів об’єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.3:

У всіх завданнях, крім вказаних в завданні операцій, обов’язково повинні бути реалізовані наступні методи:

- метод ініціалізації Init( );
- метод введення з клавіатури Read( );
- метод виведення на екран Display( );
- метод перетворення до літерного рядку toString( ).

Всі завдання повинні 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 повинні бути продемонстровані різні способи створення об’єктів і масивів об’єктів – різними конструкторами. Програма повинна 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 3.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

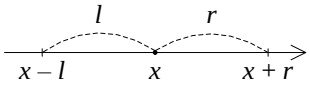
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

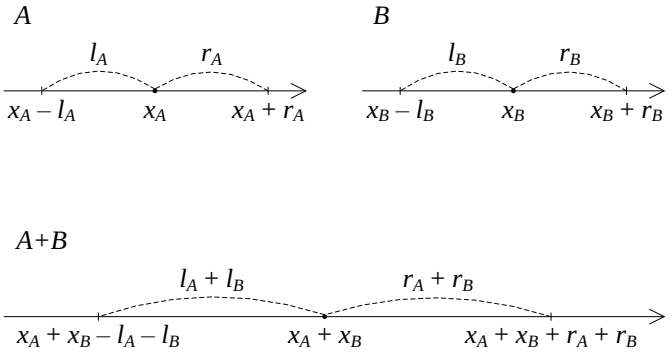
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 8.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 14.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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().

* Повинна 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,

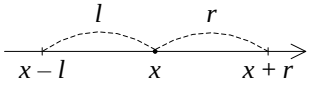
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

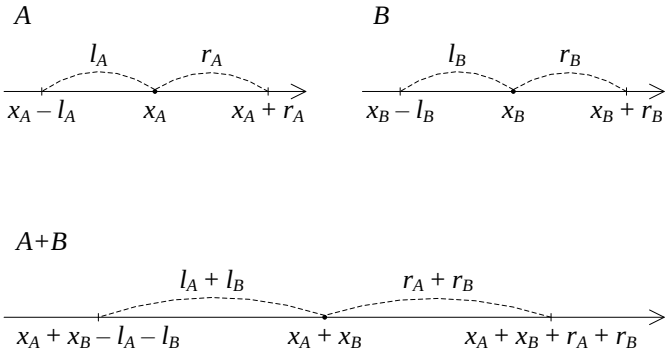
r – відстань від правої границі до центру:



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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 19.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен

бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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 повинні 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 повинні бути реалізовані:

- додавання векторів,
- віднімання векторів,
- скалярний добуток векторів.

Варіант 28.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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(), $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2, b_1 \cdot b_2)$.

* Повинна 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,

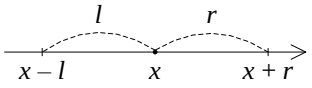
$x - l$ – координата лівої границі,

$x + r$ – координата правої границі.

Відповідно:

l – відстань від лівої границі до центру,

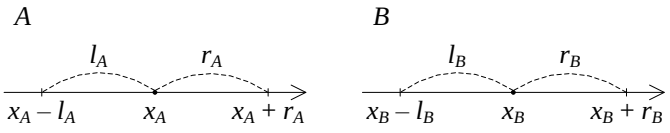
r – відстань від правої границі до центру:

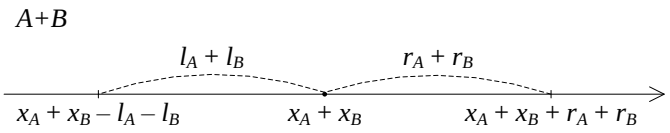


Клас FuzzyNumber містить три поля: $\{x, l, r\}$.

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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 33.

Створити клас Fraction для роботи з дробовими числами. Число повинне 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 повинен бути представлений двома полями типу long. Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу long – молодша частина.

Повинні 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 повинні 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 повинні 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() conj(x, y) = (x, $–y$).

Варіант 38.

Створити клас Vector3D, що задається трійкою координат. Поля

- x
- y
- z

Обов'язково повинні бути реалізовані:

- множення на скаляр,
- порівняння векторів,
- обчислення довжини вектора,
- порівняння довжин векторів.

Варіант 39.

Створити клас Money для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу byte – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні бути реалізовані:

- перетворення у полярні координати,
- визначення відстані до початку координат,
- порівняння на рівність та нерівність.

Завдання F

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання D*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання D):

Реалізувати основні класи із завдання свого варіанту Лабораторної роботи № 1.5 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.5 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.5 «Конструктори та перевантаження операцій для класів з композицією».

Реалізувати два види успадковування – відкрите та закрите.

Лабораторна робота № 2.5:

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 повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.5:

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», повинні бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 2.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 3.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле

«міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 4.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 5.

Створити клас Man (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 6.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 7.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 8.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 9.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 10.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 11.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 12.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 13.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

Варіант 14.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи

множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь довільного степеня, і метод для обчислення числа π в заданій степені.

Варіант 15.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на n секунд і хвилин.

Варіант 16.

Створити клас Pair (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) - (c, d) = (a - c, b - d).$$

Створити клас-контейнер Rational, що містить поле «пара цілих чисел»; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) - (c, d) = (ad - bc, bd).$$

Варіант 17.

Створити клас Pair (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Complex, що містить поле «пара чисел» – пара чисел описує дійсну і мниму частини. Визначити методи множення

$$(a, b) \times (c, d) = (ac - bd, ad + bc)$$

і віднімання

$$(a, b) - (c, d) = (a - c, b - d).$$

Варіант 18.*

Створити клас Pair (пара цілих чисел); визначити методи зміни полів і операцію

додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Long, що містить поле «пара цілих чисел» – пара чисел описує старшу і молодшу частини числа. Перевизначити операцію додавання і визначити методи множення і віднімання.

Варіант 19.

Створити клас Triad (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити клас-контейнер Vector3D, що містить поле «трійка чисел». Вектор задається трійкою координат, які описуються полем – трійкою чисел. Повинні 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 заданій степені.

Варіант 40.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

Завдання G

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання E*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання E):

Реалізувати завдання свого варіанту Лабораторної роботи № 1.5 «Композиція класів та об’єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.5 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.5 «Конструктори та перевантаження операцій для класів з композицією».

При цьому допоміжний клас слід реалізувати як відкритий клас-нащадок від базового класу Object. Клас Object реалізує лічильник кількості створених об’єктів.

Лабораторна робота № 2.5:

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 повинні 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 повинні бути продемонстровані різні способи створення об’єктів і масивів об’єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.5:

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», повинні бути визначені як незалежні.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Створити клас Car (машина), що характеризується торговою маркою (літерний рядок), кількістю циліндрів, потужністю. Визначити методи пере-присвоєння та зміни потужності.

Створити клас-контейнер Lorry (вантажівка), що містить поле «машина» і характеризується також вантажопідйомністю кузова. Визначити функції пере-присвоєння марки та зміни вантажопідйомності.

Варіант 2.

Створити клас Pair (пара чисел); визначити методи зміни полів і порівняння пар:

пара p1 більше пари p2, якщо

(p1.first > p2.first) або (p1.first = p2.first) і (p1.second > p2.second).

Визначити клас-контейнер Fraction, що містить поле «пара чисел» (число з цілою та дробовою частинами). Визначити повний набір методів порівняння.

Варіант 3.

Створити клас Liquid (рідина), що має поля «назва» і «густина». Визначити методи перепризначення і зміни густини.

Створити клас-контейнер Alcohol (спирт), що містить поле «рідина» і поле «міцність». Визначити методи пере-присвоєння та зміни міцності.

Варіант 4.

Створити клас Pair (пара чисел); визначити методи зміни полів та обчислення добутку чисел.

Визначити клас-контейнер Rectangle (прямокутник), що містить поле «пара чисел» – пара чисел описує сторони. Визначити методи обчислення периметру та площі прямокутника.

Варіант 5.

Створити клас Man (людина) з полями: ім'я, вік, стать і вага. Визначити методи пере-присвоєння імені, зміни віку і зміни ваги.

Створити клас-контейнер Student, що має поля «людина» та «рік навчання». Визначити методи пере-присвоєння та збільшення року навчання.

Варіант 6.

Створити клас Triad (трійка чисел); визначити методи зміни полів і обчислення суми чисел.

Визначити клас-контейнер Triangle (трикутник), що містить поле «трійка чисел» – трійка чисел описує сторони. Визначити методи обчислення кутів і площі трикутника.

Варіант 7.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер Equilateral (рівносторонній трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 8.

Створити клас Triangle (трикутник) з полями-сторонами. Визначити методи зміни сторін, обчислення кутів, обчислення периметру.

Створити клас-контейнер RightAngled (прямокутний трикутник), що має поля «трикутник» та «площа». Визначити метод обчислення площі.

Варіант 9.

Створити клас Pair (пара чисел); визначити методи зміни полів і обчислення добутку

чисел.

Визначити клас-контейнер RightAngled (прямокутний трикутник), що містить поле «пара чисел», яке описує катети. Визначити методи обчислення гіпотенузи і площі трикутника.

Варіант 10.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Визначити повний набір методів порівняння дат.

Варіант 11.

Створити клас Triad (трійка чисел); визначити метод порівняння тріад:
тріада t1 більше тріади t2, якщо

(t1.first > t2.first) або (t1.first = t2.first) і (t1.second > t2.second)

або (t1.first = t2.first) і (t1.second = t2.second) і (t1.third > t2. third).

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину і секунду. Визначити повний набір методів порівняння моментів часу.

Варіант 12.

Реалізувати клас-оболонку Number для числового типу float. Реалізувати методи додавання і ділення.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод піднесення до довільного степеня, та метод для обчислення логарифму числа.

Варіант 13.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Date, що містить поле «трійка чисел» – трійка чисел описує рік, місяць і день. Перевизначити методи збільшення полів на 1 і визначити метод збільшення дати на *n* днів.

Варіант 14.

Реалізувати клас-оболонку Number для числового типу double. Реалізувати методи множення і віднімання.

Створити клас-контейнер Real, що містить поле типу Number, в класі Real реалізувати метод, що обчислює корінь довільного степеню, і метод для обчислення числа π в заданій степені.

Варіант 15.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на n секунд і хвилин.

Варіант 16.

Створити клас Pair (пара цілих чисел) з операціями перевірки на рівність і множення полів. Реалізувати операцію віднімання пар за формулою

$$(a, b) – (c, d) = (a – c, b – d).$$

Створити клас-контейнер Rational, що містить поле «пара цілих чисел»; визначити нові операції:

додавання

$$(a, b) + (c, d) = (ad + bc, bd)$$

і ділення

$$(a, b) / (c, d) = (ad, bc);$$

перевизначити операцію віднімання

$$(a, b) – (c, d) = (ad – bc, bd).$$

Варіант 17.

Створити клас Pair (пара чисел); визначити метод множення полів та операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Complex, що містить поле «пара чисел» – пара чисел описує дійсну і мниму частини. Визначити методи множення

$$(a, b) \times (c, d) = (ac – bd, ad + bc)$$

і віднімання

$$(a, b) – (c, d) = (a – c, b – d).$$

Варіант 18.*

Створити клас Pair (пара цілих чисел); визначити методи зміни полів і операцію додавання пар

$$(a, b) + (c, d) = (a + c, b + d).$$

Визначити клас-контейнер Long, що містить поле «пара цілих чисел» – пара чисел описує старшу і молодшу частини числа. Перевизначити операцію додавання і визначити методи множення і віднімання.

Варіант 19.

Створити клас Triad (трійка чисел) з операціями: додавання числа, множення на число, перевірки на рівність.

Створити клас-контейнер Vector3D, що містить поле «трійка чисел». Вектор задається трійкою координат, які описуються полем – трійкою чисел. Повинні 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 степеню, і метод для обчислення числа π в заданій степені.

Варіант 40.

Створити клас Triad (трійка чисел); визначити методи збільшення полів на 1.

Визначити клас-контейнер Time, що містить поле «трійка чисел» – трійка чисел описує годину, хвилину, секунду. Перевизначити методи збільшення полів на 1 і визначити методи збільшення часу на *n* секунд і хвилин.

Завдання H

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання F*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання F):

Реалізувати основні класи із завдання свого варіанту Лабораторної роботи № 1.7 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.7 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.7 «Конструктори та перевантаження операцій для класів з композицією – складніші завдання».

Реалізувати два види успадковування – відкрите та закрите.

Лабораторна робота № 2.7:

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 повинні 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 повинні бути продемонстровані різні способи створення об'єктів і масивів об'єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.7:

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є основним, всі решту – допоміжні. Допоміжні класи повинні бути визначені як незалежні. Об'єкти допоміжних класів повинні використовуватися як поля основного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинна бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна 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 повинне 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 повинні 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 повинен виконувати перевірку на коректність суми, що знімається: вона не повинна бути менше мінімального значення і не повинна перевищувати максимальне значення.

- метод toString( ) повинен 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 5*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число повинне 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 повинне бути представлене символами-цифрами, які утворюють літерний рядок.

Повинні 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 повинне 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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.

– повинні 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 повинні 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 повинні 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( ) повинен повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран повинна 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 повинне 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 повинні здійснювати перевірку на перетин меж екрану. Функція виводу на екран повинна 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 повинен бути представлений двома полями типу unsigned long. Повинні 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) повинен 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 повинні 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().

Повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,

- типу unsigned long – молодша частина.

Повинні бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 17.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинен 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 повинні 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( ) повинен 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 повинен 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 повинні 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 повинен 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 повинен 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 повинен 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( ) повинен видавати суму в гривнях.

Варіант 21.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинна бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна 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 повинне 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 повинні 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 повинен виконувати перевірку на коректність суми, що знімається: вона не повинна бути менше мінімального значення і не повинна перевищувати максимальне значення.

- метод toString( ) повинен 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 25*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число повинне 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 повинне бути представлене символами-цифрами, які утворюють літерний рядок.

Повинні 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 повинне 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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.

– повинні 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 повинні 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 повинні 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( ) повинен повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран повинна 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 повинне 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 повинні здійснювати перевірку на перетин меж екрану. Функція виводу на екран повинна 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 повинен бути представлений двома полями типу unsigned long. Повинні 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) повинен 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 повинні 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().

Повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене

двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 37.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинен 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 повинні 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( ) повинен 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 повинен 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 повинні 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 повинен 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 повинен 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 повинен 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( ) повинен видавати суму в гривнях.

Завдання І

Виконати свій варіант Лабораторної роботи № 3.3 (*Завдання G*) «Успадковування замість композиції» з конструктором ініціалізації (ініціалізація літерним рядком*) та опрацюванням винятків.

Лабораторна робота № 3.3 (Завдання G):

Реалізувати завдання свого варіанту Лабораторної роботи № 1.7 «Композиція класів та об'єктів» з конструкторами та перевантаженням операцій як класи-нащадки від класів із завдання Лабораторної роботи № 1.7 (агрегований клас слід використовувати як базовий, а клас-контейнер стане похідним класом).

Оскільки слід реалізувати конструктори та операції, то зручніше за основу брати завдання свого варіанту Лабораторної роботи № 2.7 «Конструктори та перевантаження операцій для класів з композицією – складніші завдання».

При цьому допоміжний клас слід реалізувати як відкритий клас-нащадок від базового класу Object. Клас Object реалізує лічильник кількості створених об'єктів.

Лабораторна робота № 2.7:

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 повинні 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 повинні бути продемонстровані різні способи створення об’єктів і масивів об’єктів. Програма повинна демонструвати використання всіх функцій і методів. Вона повинна 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.

Лабораторна робота № 1.7:

У всіх завданнях потрібно реалізувати по два-три класи. Один клас є основним, всі решту – допоміжні. Допоміжні класи повинні бути визначені як незалежні. Об’єкти допоміжних класів повинні використовуватися як поля основного класу.

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинна бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна 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 повинне 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 повинні 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 повинен виконувати перевірку на коректність суми, що знімається: вона не повинна бути менше мінімального значення і не повинна перевищувати максимальне значення.

- метод toString( ) повинен 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 5*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число повинне 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 повинне бути представлене символами-цифрами, які утворюють літерний рядок.

Повинні 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 повинне 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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.

– повинні 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 повинні 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 повинні 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( ) повинен повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран повинна 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 повинне 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 повинні здійснювати перевірку на перетин меж екрану. Функція виводу на екран повинна 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 повинен бути представлений двома полями типу unsigned long. Повинні 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) повинен 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 повинні 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().

Повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,

- типу unsigned long – молодша частина.

Повинні бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 17.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинен 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 повинні 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( ) повинен 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 повинен 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 повинні 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 повинен 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 повинен 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 повинен 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( ) повинен видавати суму в гривнях.

Варіант 21.

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинна бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна 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 повинне 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 повинні 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 повинен виконувати перевірку на коректність суми, що знімається: вона не повинна бути менше мінімального значення і не повинна перевищувати максимальне значення.

- метод toString( ) повинен 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 повинна бути представлена полями-номіналами, значеннями яких повинна 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 повинна бути відокремлена від цілої частини комою.

Варіант 25*.

Реалізувати клас Fraction:

Клас Fraction – для роботи з дробовими числами. Число повинне 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 повинне бути представлене символами-цифрами, які утворюють літерний рядок.

Повинні 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 повинне 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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.

– повинні 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 повинні 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 повинні 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( ) повинен повертати у вигляді літерного рядка вартість товару.

Для представлення ціни використовувати клас Money:

Клас Money – для роботи з грошовими сумами. Число повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинні 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 повинні 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 повинне бути представлене двома полями:

- типу long для гривень і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене вкладеним об’єктом класу Fraction.

Дробова частина (копійки) при виводі на екран повинна 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 повинне 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 повинні здійснювати перевірку на перетин меж екрану. Функція виводу на екран повинна 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 повинен бути представлений двома полями типу unsigned long. Повинні 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) повинен 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 повинні 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().

Повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене

двома полями:

- типу LongLong для гривень (див. далі) і
- типу unsigned char – для копійок.

Дробова частина (копійки) при виводі на екран повинна 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні 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 повинне бути представлене двома полями:

- типу long – старша частина,
- типу unsigned long – молодша частина.

Повинні бути реалізовані:

- всі арифметичні операції, присутні в C++ (без присвоєння), та
- операції порівняння.

Варіант 37.*

Реалізувати клас Account:

Клас Account, – банківський рахунок. У класі повинні 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 повинен 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 повинні 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( ) повинен 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 повинен 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 повинні 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 повинен 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 повинен 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 повинен 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( ) повинен видавати суму в гривнях.

Лабораторна робота № 5.2. Функції, що генерують виняткові ситуації

Мета роботи

Освоїти опрацювання виняткових ситуацій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення винятку.
2) Поняття та призначення контрольованого (захищеного) блоку.
3) Поняття та призначення блоку опрацювання винятку.
4) Генерування (створення) винятку.
5) Механізм опрацювання винятків.
6) Повторне генерування винятку в блоці опрацювання.
7) Винятки в конструкторах та деструкторі.
8) Специфікація винятків в заголовку функції.
9) Стандартні винятки та їх опрацювання.
10) Підміна функцій стандартного завершення.

Приклад розв'язку завдання

В кожному варіанті завдання слід реалізувати функції, які виконують перевірку параметрів, що передаються; і генерують виняткові ситуації у випадку помилкових значень параметрів. Всі функції реалізуються в 6 варіантах:

1) без специфікації виняткових ситуацій;
2) із специфікацією throw( );
3) з конкретною специфікацією підходящої стандартної виняткової ситуації;
4) специфікація з власною винятковою ситуацією, яка реалізована як порожній клас;
5) специфікація з власною винятковою ситуацією, яка реалізована як незалежний клас з полями – параметрами функції;
6) специфікація з власною винятковою ситуацією, яка реалізована як нащадок від стандартної виняткової ситуації з полями.

Перехоплення і опрацювання виняткових ситуацій повинна виконувати головна функція програми.

Реалізувати підміну стандартних функцій terminate( ) та unexpected( ). Підмінена функція користувача повинна виводити повідомлення про відсутність обробника виняткової

ситуації і завершувати роботу.

Умова завдання – наступна:

Варіант 0.

Функція, яка з’ясовує, чи є рік високосний. Високосний рік визначається таким чином: якщо номер року не кратний 100, то високосним вважається той, який ділиться на 4 без залишку; якщо номер року кратний 100, то високосним вважається той, номер якого ділиться на 400 без залишку.

Виняткова ситуація генерується, коли номер року – за модулем більший 5000.

Розв’язок

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

using namespace std;


class Empty
{};

class Error
{
    string message;

public:
    Error(string message)
        : message(message)
    {}

    string What() { return message; }
};

class E: public exception
{
    string message;

public:
    E(string message)
        : message(message)
    {}

    string What() { return message; }
};


bool IsVysokosny1(int year)
{
    if (abs(year) > 5000)
        throw 1;

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}
```

```
bool IsVysokosny2(int year) throw()
{
    if (abs(year) > 5000)
        throw 1.0;

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

bool IsVysokosny3(int year) throw(char)
{
    if (abs(year) > 5000)
        throw '1';

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

bool IsVysokosny4(int year) throw(Empty)
{
    if (abs(year) > 5000)
        throw Empty();

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

bool IsVysokosny5(int year) throw(Error)
{
    if (year > 5000)
        throw Error("too large year");
    if (year < -5000)
        throw Error("too small year");

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

bool IsVysokosny6(int year) throw(int)
{
    if (year > 5000)
        throw E("too large year");
    if (year < -5000)
        throw E("too small year");

    return (year % 400 == 0) || (year % 100 != 0 && year % 4 == 0);
}

void FU()
{
    cout << "unexpected error! - bad_exception" << endl;
    throw;
}

void FT()
{
    cout << "unknown error! - terminate" << endl;
    abort();
}


int main()
{
    set_unexpected(FU);
    //set_terminate(FT);
```

```
    int y;
    cout << " year = ? "; cin >> y;

    try
    {
        // cout << IsVysokosny1(y) << endl;
        // cout << IsVysokosny2(y) << endl;
        // cout << IsVysokosny3(y) << endl;
        // cout << IsVysokosny4(y) << endl;
        // cout << IsVysokosny5(y) << endl;
        cout << IsVysokosny6(y) << endl;
    }
    catch (int)
    {
        cout << " catch (int) <= IsVysokosny1()" << endl;
    }
    catch (double)
    {
        cout << " catch (double) <= IsVysokosny2()" << endl;
    }
    catch (char)
    {
        cout << " catch (char) <= IsVysokosny3()" << endl;
    }
    catch (Empty)
    {
        cout << " catch (Empty) <= IsVysokosny4()" << endl;
    }
    catch (Error e)
    {
        cout << " catch (Error) <= IsVysokosny5() :" << endl;
        cout << e.What() << endl;
    }
    catch (bad_exception)
    {
        cout << " catch (bad_exception)" << endl;
    }
    //catch (exception)
    //{
    //    cout << " catch (exception) <= IsVysokosny6() :" << endl;
    //}
    //catch (E e)
    //{
    //    cout << " catch (E) <= IsVysokosny6() :" << endl;
    //    cout << e.What() << endl;
    //}

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

Варіанти завдань

В кожному варіанті завдання слід реалізувати функції, які виконують перевірку параметрів, що передаються; і генерують виняткові ситуації у випадку помилкових значень параметрів. Всі функції реалізуються в 6 варіантах:

1) без специфікації виняткових ситуацій;
2) із специфікацією throw( );

3) з конкретною специфікацією підходящої стандартної виняткової ситуації;
4) специфікація з власною винятковою ситуацією, яка реалізована як порожній клас;
5) специфікація з власною винятковою ситуацією, яка реалізована як незалежний клас з полями – параметрами функції;
6) специфікація з власною винятковою ситуацією, яка реалізована як нащадок від стандартної виняткової ситуації з полями.

Перехоплення і опрацювання виняткових ситуацій повинна виконувати головна функція програми.

Реалізувати підміну стандартних функцій terminate( ) та unexpected( ). Підмінена функція користувача повинна виводити повідомлення про відсутність обробника виняткової ситуації і завершувати роботу.

Варіанти завдань наступні:

Варіант 1.

Функція, яка обчислює площу трикутника за трьома сторонами

$S = \sqrt{p(p-a)(p-b)(p-c)}$

де $p = (a + b + c) / 2$.

Виняткова ситуація генерується, коли введено неправильні значення довжин сторін.

Варіант 2.

Функція, яка обчислює корінь лінійного рівняння $ax + b = 0$.

Виняткова ситуація генерується, коли $a = 0$.

Варіант 3.

Функція, яка обчислює периметр трикутника.

Виняткова ситуація генерується, коли введено неправильні значення довжин сторін.

Варіант 4.

Функція, яка переводить години і хвилини в секунди.

Виняткова ситуація генерується, коли введено неправильний час (наприклад, 25:78 – години:хвилини).

Варіант 5.

Функція, яка обчислює корінь квадратного рівняння $ax^2 + bx + c = 0$.

Виняткова ситуація генерується, коли $b^2 – 4 \cdot a \cdot c < 0$.

Варіант 6.

Функція, яка обчислює суму геометричної прогресії $S_n = (a_0 – a_n r) / (1 – r)$.

Виняткова ситуація генерується, коли $r = 1$.

Варіант 7.

Функція, яка виконує ділення нечітких чисел A і B.

Нечіткі числа представляються трійками чисел $(x – l, x, x + r)$. Поля:

- x
- l
- r

Для нечітких чисел $A = (x_A – l_A, x_A, x_A + r_A)$ та $B = (x_B – l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

  $A + B = (x_A + x_B – l_A – l_B, x_A + x_B, x_A + x_B + r_A + r_B)$;
- віднімання

  $A – B = (x_A – x_B – l_A – l_B, x_A – x_B, x_A – x_B + r_A + r_B)$;
- множення

  $A \times B = (x_A \times x_B – x_B \times l_A – x_A \times l_B – l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$;
- зворотне число

  $1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A – l_A)), x_A > 0$;
- ділення

  $A / B = ((x_A – l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B – l_B)), x_B > 0$;

Виняткова ситуація генерується, коли $x_B = 0$ або $x_B = l_B$ або $x_B = – r_B$.

Варіант 8.

Функція, яка обчислює нечітке число, зворотне заданому.

Нечіткі числа представляються трійками чисел $(x – l, x, x + r)$. Поля:

- x
- l
- r

Для нечітких чисел $A = (x_A – l_A, x_A, x_A + r_A)$ та $B = (x_B – l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

$A + B = (x_A + x_B - l_A - l_B,\ x_A + x_B,\ x_A + x_B + r_A + r_B)$;

- віднімання

  $A - B = (x_A - x_B - l_A - l_B,\ x_A - x_B,\ x_A - x_B + r_A + r_B)$;

- множення

  $A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B,\ x_A \times x_B,\ x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$;

- зворотне число

  $1 / A = (1/(x_A + r_A),\ 1 / x_A,\ 1/(x_A - l_A)),\ x_A > 0$;

- ділення

  $A / B = ((x_A - l_A)/(x_B + r_B),\ x_A / x_B,\ (x_A + r_A)/(x_B - l_B)),\ x_B > 0$;

Виняткова ситуація генерується, коли $x_A = 0$ або $x_A = l_A$ або $x_A = -r_A$.

Варіант 9.

Функція, яка виконує ділення комплексних чисел A (a_1, b_1) і B (a_2, b_2).

Комплексне число представляються парою дійсних чисел (a, b), де поля

- a – дійсна частина,
- b – мнима частина.

Для комплексних чисел арифметичні операції виконуються за наступними формулами:

- додавання add() $(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$;
- віднімання sub() $(a_1, b_1) - (a_2, b_2) = (a_1 - a_2, b_1 - b_2)$;
- множення mul() $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2 - b_1 \cdot b_2, a_1 \cdot b_2 + a_2 \cdot b_1)$;
- ділення div() $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot a_2 + b_1 \cdot b_2, a_2 \cdot b_1 - a_1 \cdot b_2) / (a_2^2 + b_2^2)$;
- порівняння equ() $(a_1, b_1) = (a_2, b_2)$, якщо $(a_1 = a_2)$ і $(b_1 = b_2)$;
- комплексно спряжене число conj() $\text{conj}(a, b) = (a, -b)$.

Виняткова ситуація генерується, коли дійсна та мнима частина другого числа дорівнюють нулю.

Варіант 10.

Функція, яка обчислює цілу частину неправильного дробу, представленого чисельником і знаменником – цілими числами.

Виняткова ситуація генерується, коли знаменник дорівнює нулю.

Варіант 11.

Функція, яка переводить комплексне число $z = x + i\,y$ із алгебраїчної форми в

тригонометричну $z = radius(\cos(angle) + i \sin(angle))$.

Комплексне число представляються структурою-парою дійсних чисел (x, y), де поля

- x – дійсна частина,
- y – мнима частина.

Перетворене число теж представляється структурою-парою (*radius*, *angle*):

$radius = \sqrt{x^2 + y^2}$; $angle = \arcsin\frac{x}{y}$

Виняткова ситуація генерується, коли $y = 0$.

Варіант 12*.

Функція, яка обчислює різницю між двома датами в днях. Дати представлені структурою з трьома полями: рік, місяць, день.

Виняткова ситуація генерується, коли введено неправильну дату (наприклад, 30.02.2015 або 13.13.2013).

Варіант 13*.

Функція, яка обчислює тривалість телефонної розмови в хвилинах, приймаючи фіксуючи час початку і закінчення. Час представлений структурою з трьома полями: година, хвилина, секунда. Неповна хвилина вважається за повну.

Виняткова ситуація генерується, коли введено неправильний час (наприклад, 25:78:65 – години:хвилини:секунди) або час закінчення розмови передує часу початку.

Варіант 14***.

Функція, яка обчислює день тижня по даті поточного року. Дати представлені структурою з трьома полями: рік, місяць, день. День тижня першого січня поточного року – повинен відповідати календарному.

Виняткова ситуація генерується, коли введена неправильна дата (наприклад, 30.02 – тридцятого лютого або 13.13 – тринадцятий день тринадцятого місяця).

Варіант 15.

Функція, яка обчислює кути прямокутного трикутника. Параметри – катети a та b. Синус кута A, що протилежний катету a, обчислюється за формулою $\sin(A) = a / c$, де c – гіпотенуза трикутника.

Виняткова ситуація генерується, коли довжина хоч однієї сторони дорівнює нулю, або

коли трикутник – не прямокутний.

Варіант 16.

Функція, яка перевіряє, чи рядок, що передається, є палиндромом (читається так само: з початку до кінця та з кінця до початку).

Виняткова ситуація генерується, коли рядок – порожній.

Варіант 17.

Функція, яка визначає, чи прямі $A_1x + B_1y + C_1 = 0$ та $A_2x + B_2y + C_2 = 0$ – перетинаються. Прямі перетинаються, якщо вираз $d = A_1B_2 – A_2B_1$ не рівний нулю. Прямі задаються структурою з трьома полями.

Виняткова ситуація генерується, коли числа A_i та B_i (i = 1, 2) – дорівнюють нулю (тобто, коли структура $\{A_i, B_i, C_i\}$ (i = 1, 2) не описує рівняння прямої).

Варіант 18.

Функція, яка обчислює відстань між двома точками $P_1(x_1, y_1)$ і $P_2(x_2, y_2)$ за формулою

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Виняткова ситуація генерується, коли P_1 і P_2 – одна і та ж точка.

Варіант 19.

Функція, яка обчислює відстань від точки $P(x_1, y_1)$ до прямої $Ax + By + C = 0$ за формулою:

$$D = \frac{|Ax_1 + By_1 + C|}{\sqrt{A^2 + B^2}}$$

Виняткова ситуація генерується, коли $A = B = 0$ (тобто, не задано прямої).

Варіант 20*.

Функція, яка спрощує неправильний дріб, представлений чисельником і знаменником – цілими числами (4/6 → 2/3).

Виняткова ситуація генерується, коли знаменник дорівнює нулю.

Варіант 21.

Функція, яка обчислює площу трикутника за трьома сторонами

$$S = \sqrt{p(p-a)(p-b)(p-c)}$$

де $p = (a + b + c) / 2$.

Виняткова ситуація генерується, коли введено неправильні значення довжин сторін.

Варіант 22.

Функція, яка обчислює корінь лінійного рівняння $ax + b = 0$.

Виняткова ситуація генерується, коли $a = 0$.

Варіант 23.

Функція, яка обчислює периметр трикутника.

Виняткова ситуація генерується, коли введено неправильні значення довжин сторін.

Варіант 24.

Функція, яка переводить години і хвилини в секунди.

Виняткова ситуація генерується, коли введено неправильний час (наприклад, 25:78 – години:хвилини).

Варіант 25.

Функція, яка обчислює корінь квадратного рівняння $ax^2 + bx + c = 0$.

Виняткова ситуація генерується, коли $b^2 – 4 \cdot a \cdot c < 0$.

Варіант 26.

Функція, яка обчислює суму геометричної прогресії $S_n = (a_0 – a_n r) / (1 – r)$.

Виняткова ситуація генерується, коли $r = 1$.

Варіант 27.

Функція, яка виконує ділення нечітких чисел A і B.

Нечіткі числа представляються трійками чисел $(x – l, x, x + r)$. Поля:

- x
- l
- r

Для нечітких чисел $A = (x_A – l_A, x_A, x_A + r_A)$ та $B = (x_B – l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

  $A + B = (x_A + x_B – l_A – l_B, x_A + x_B, x_A + x_B + r_A + r_B)$;
- віднімання

  $A – B = (x_A – x_B – l_A – l_B, x_A – x_B, x_A – x_B + r_A + r_B)$;

- множення

  $A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$;

- зворотне число

  $1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0$;

- ділення

  $A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0$;

Виняткова ситуація генерується, коли $x_B = 0$ або $x_B = l_B$ або $x_B = - r_B$.

Варіант 28.

Функція, яка обчислює нечітке число, зворотне заданому.

Нечіткі числа представляються трійками чисел $(x - l, x, x + r)$. Поля:

- x
- l
- r

Для нечітких чисел $A = (x_A - l_A, x_A, x_A + r_A)$ та $B = (x_B - l_B, x_B, x_B + r_B)$ арифметичні операції виконуються за наступними формулами:

- додавання

  $A + B = (x_A + x_B - l_A - l_B, x_A + x_B, x_A + x_B + r_A + r_B)$;

- віднімання

  $A - B = (x_A - x_B - l_A - l_B, x_A - x_B, x_A - x_B + r_A + r_B)$;

- множення

  $A \times B = (x_A \times x_B - x_B \times l_A - x_A \times l_B - l_A \times l_B, x_A \times x_B, x_A \times x_B + x_B \times r_A + x_A \times r_B + r_A \times r_B)$;

- зворотне число

  $1 / A = (1/(x_A + r_A), 1 / x_A, 1/(x_A - l_A)), x_A > 0$;

- ділення

  $A / B = ((x_A - l_A)/(x_B + r_B), x_A / x_B, (x_A + r_A)/(x_B - l_B)), x_B > 0$;

Виняткова ситуація генерується, коли $x_A = 0$ або $x_A = l_A$ або $x_A = - r_A$.

Варіант 29.

Функція, яка виконує ділення комплексних чисел A (a_1, b_1) і B (a_2, b_2).

Комплексне число представляються парою дійсних чисел (a, b), де поля

- a – дійсна частина,
- b – мнима частина.

Для комплексних чисел арифметичні операції виконуються за наступними формулами:

- додавання add() $(a_1, b_1) + (a_2, b_2) = (a_1 + a_2, b_1 + b_2)$;
- віднімання sub() $(a_1, b_1) – (a_2, b_2) = (a_1 – a_2, b_1 – b_2)$;
- множення mul() $(a_1, b_1) \times (a_2, b_2) = (a_1 \cdot a_2 – b_1 \cdot b_2, a_1 \cdot b_2 + a_2 \cdot b_1)$;
- ділення div() $(a_1, b_1) / (a_2, b_2) = (a_1 \cdot a_2 + b_1 \cdot b_2, a_2 \cdot b_1 – a_1 \cdot b_2) / (a_2^2 + b_2^2)$;
- порівняння equ() $(a_1, b_1) = (a_2, b_2)$, якщо $(a_1 = a_2)$ і $(b_1 = b_2)$;
- комплексно спряжене число conj() $\text{conj}(a, b) = (a, –b)$.

Виняткова ситуація генерується, коли дійсна та мнима частина другого числа дорівнюють нулю.

Варіант 30.

Функція, яка обчислює цілу частину неправильного дробу, представленого чисельником і знаменником – цілими числами.

Виняткова ситуація генерується, коли знаменник дорівнює нулю.

Варіант 31.

Функція, яка переводить комплексне число $z = x + i\,y$ із алгебраїчної форми в тригонометричну $z = radius(\cos(angle) + i \sin(angle))$.

Комплексне число представляються структурою-парою дійсних чисел (x, y), де поля

- x – дійсна частина,
- y – мнима частина.

Перетворене число теж представляється структурою-парою $(radius, angle)$:

$radius = \sqrt{x^2 + y^2}$; $angle = \arcsin\frac{x}{y}$

Виняткова ситуація генерується, коли $y = 0$.

Варіант 32*.

Функція, яка обчислює різницю між двома датами в днях. Дати представлені структурою з трьома полями: рік, місяць, день.

Виняткова ситуація генерується, коли введено неправильну дату (наприклад, 30.02.2015 або 13.13.2013).

Варіант 33*.

Функція, яка обчислює тривалість телефонної розмови в хвилинах, приймаючи фіксуючи час початку і закінчення. Час представлений структурою з трьома полями: година, хвилина, секунда. Неповна хвилина вважається за повну.

Виняткова ситуація генерується, коли введено неправильний час (наприклад, 25:78:65 – години:хвилини:секунди) або час закінчення розмови передує часу початку.

Варіант 34***.

Функція, яка обчислює день тижня по даті поточного року. Дати представлені структурою з трьома полями: рік, місяць, день. День тижня першого січня поточного року – повинен відповідати календарному.

Виняткова ситуація генерується, коли введена неправильна дата (наприклад, 30.02 – тридцятого лютого або 13.13 – тринадцятий день тринадцятого місяця).

Варіант 35.

Функція, яка обчислює кути прямокутного трикутника. Параметри – катети a та b. Синус кута A, що протилежний катету a, обчислюється за формулою $\sin(A) = a / c$, де c – гіпотенуза трикутника.

Виняткова ситуація генерується, коли довжина хоч однієї сторони дорівнює нулю, або коли трикутник – не прямокутний.

Варіант 36.

Функція, яка перевіряє, чи рядок, що передається, є палиндромом (читається так само: з початку до кінця та з кінця до початку).

Виняткова ситуація генерується, коли рядок – порожній.

Варіант 37.

Функція, яка визначає, чи прямі $A_1x + B_1y + C_1 = 0$ та $A_2x + B_2y + C_2 = 0$ – перетинаються. Прямі перетинаються, якщо вираз $d = A_1B_2 – A_2B_1$ не рівний нулю. Прямі задаються структурою з трьома полями.

Виняткова ситуація генерується, коли числа A_i та B_i ($i = 1, 2$) – дорівнюють нулю (тобто, коли структура $\{A_i, B_i, C_i\}$ ($i = 1, 2$) не описує рівняння прямої).

Варіант 38.

Функція, яка обчислює відстань між двома точками $P_1(x_1, y_1)$ і $P_2(x_2, y_2)$ за формулою

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Виняткова ситуація генерується, коли P_1 і P_2 – одна і та ж точка.

Варіант 39.

Функція, яка обчислює відстань від точки $P(x_1, y_1)$ до прямої $Ax + By + C = 0$ за формулою:

$$D = \frac{|Ax_1 + By_1 + C|}{\sqrt{A^2 + B^2}}$$

Виняткова ситуація генерується, коли $A = B = 0$ (тобто, не задано прямої).

Варіант 40*.

Функція, яка спрощує неправильний дріб, представлений чисельником і знаменником – цілими числами (4/6 → 2/3).

Виняткова ситуація генерується, коли знаменник дорівнює нулю.

Лабораторна робота № 5.3. Опрацювання виняткових ситуацій

Мета роботи

Освоїти опрацювання виняткових ситуацій.

Питання, які необхідно вивчити та пояснити на захисті

1) Поняття та призначення винятку.
2) Поняття та призначення контрольованого (захищеного) блоку.
3) Поняття та призначення блоку опрацювання винятку.
4) Генерування (створення) винятку.
5) Механізм опрацювання винятків.
6) Повторне генерування винятку в блоці опрацювання.
7) Винятки в конструкторах та деструкторі.
8) Специфікація винятків в заголовку функції.
9) Стандартні винятки та їх опрацювання.
10) Підміна функцій стандартного завершення.

Приклад

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення об’єкта базового класу A в об’єкт похідного класу B.

```
// exception.cpp : Defines the entry point for the console application.
//

#include <iostream>

using namespace std;

class A
{
    protected:
        int x;
    public:
        A(int x = 0)
        {
            this->x = x;
        }
};

class B
    : public A
{
        int y;
```

```
    public:
        B(int x = 0, int y = 0)
            : A(x)
        {
            this->y = y;
        }
        B(A a)
        {
            throw a;
        }
};

int main()
{
    A a(1);
    B b(2, 3);

    try
    {
        b = B(a);
        cout << "Continue" << endl;
    }
    catch(B)
    {
        cout << "catch(B)" << endl;
        exit(1);
    }
    catch(A)
    {
        cout << "catch(A)" << endl;
        exit(2);
    }

    cout << "End" << endl;
    return 0;
}
```

Варіанти завдань

Визначення класів та реалізації методів слід розмістити в окремих модулях.

Варіанти завдань наступні:

Варіант 1.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника на клас B до вказівника на абстрактний клас.

Варіант 2.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника базового класу A у вказівник похідного класу B.

Варіант 3.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі присвоєння об’єкту базового класу A об’єкта похідного класу B.

Варіант 4.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при отриманні через список параметрів одним із методів класу B копії об’єкта класу A.

Варіант 5.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі самоприсвоєння об’єктів одного класу.

Варіант 6.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі створення одним із методів класу B об’єкта класу A.

Варіант 7.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення посилання на клас B у посилання на абстрактний клас.

Варіант 8.

Клас B є похідним від класу A. У класі B оголошено метод, який повертає об’єкт цього ж класу. Визначити обробник виняткової ситуації при спробі повернення методом класу B об’єкта класу A.

Варіант 9.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення об’єкта класу B в об’єкт класу A.

Варіант 10.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі ініціалізації об’єкта класу B об’єктом класу A.

Варіант 11.

Визначити обробник виняткової ситуації, якщо кількість створених об’єктів класу A

перевищує задане значення.

Варіант 12.

Визначити обробник виняткової ситуації, якщо об’єкт класу A створюється в області динамічної пам’яті.

Варіант 13.

Визначити обробник виняткової ситуації, якщо індекс об’єкта-масиву виходить за межі допустимого діапазону.

Варіант 14.

Побудувати клас-годинник, який має поля для збереження годин (0..23), хвилин (0..59) та секунд (0..59). Визначити обробник виняткової ситуації, якщо створюється об’єкт-годинник з недопустимими значеннями годин, хвилин або секунд.

Варіант 15.

Побудувати клас, який контролює виведення рядків у межах вікна (x_1, y_1, x_2, y_2). Визначити обробник виняткової ситуації при виведенні рядків за межами вікна.

Варіант 16.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника на клас B до вказівника на абстрактний клас.

Варіант 17.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника базового класу A у вказівник похідного класу B.

Варіант 18.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі присвоєння об’єкту базового класу A об’єкта похідного класу B.

Варіант 19.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при отриманні через список параметрів одним із методів класу B копії об’єкта класу A.

Варіант 20.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі самоприсвоєння об’єктів одного класу.

Варіант 21.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі створення одним із методів класу B об’єкта класу A.

Варіант 22.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення посилання на клас B у посилання на абстрактний клас.

Варіант 23.

Клас B є похідним від класу A. У класі B оголошено метод, який повертає об’єкт цього ж класу. Визначити обробник виняткової ситуації при спробі повернення методом класу B об’єкта класу A.

Варіант 24.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення об’єкта класу B в об’єкт класу A.

Варіант 25.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі ініціалізації об’єкта класу B об’єктом класу A.

Варіант 26.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника на клас B до вказівника на абстрактний клас.

Варіант 27.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення вказівника базового класу A у вказівник похідного класу B.

Варіант 28.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі присвоєння об’єкту базового класу A об’єкта похідного класу B.

Варіант 29.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при отриманні через список параметрів одним із методів класу B копії об’єкта класу A.

Варіант 30.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі самоприсвоєння об’єктів одного класу.

Варіант 31.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі створення одним із методів класу B об’єкта класу A.

Варіант 32.

Неабстрактний клас B є похідним від абстрактного класу A. Визначити обробник виняткової ситуації при спробі перетворення посилання на клас B у посилання на абстрактний клас.

Варіант 33.

Клас B є похідним від класу A. У класі B оголошено метод, який повертає об’єкт цього ж класу. Визначити обробник виняткової ситуації при спробі повернення методом класу B об’єкта класу A.

Варіант 34.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі перетворення об’єкта класу B в об’єкт класу A.

Варіант 35.

Клас B є похідним від класу A. Визначити обробник виняткової ситуації при спробі ініціалізації об’єкта класу B об’єктом класу A.

Варіант 36.

Визначити обробник виняткової ситуації, якщо кількість створених об’єктів класу A

перевищує задане значення.

Варіант 37.

Визначити обробник виняткової ситуації, якщо об’єкт класу A створюється в області динамічної пам’яті.

Варіант 38.

Визначити обробник виняткової ситуації, якщо індекс об’єкта-масиву виходить за межі допустимого діапазону.

Варіант 39.

Побудувати клас-годинник, який має поля для збереження годин (0..23), хвилин (0..59) та секунд (0..59). Визначити обробник виняткової ситуації, якщо створюється об’єкт-годинник з недопустимими значеннями годин, хвилин або секунд.

Варіант 40.

Побудувати клас, який контролює виведення рядків у межах вікна (x_1, y_1, x_2, y_2). Визначити обробник виняткової ситуації при виведенні рядків за межами вікна.

Питання та завдання для контролю знань

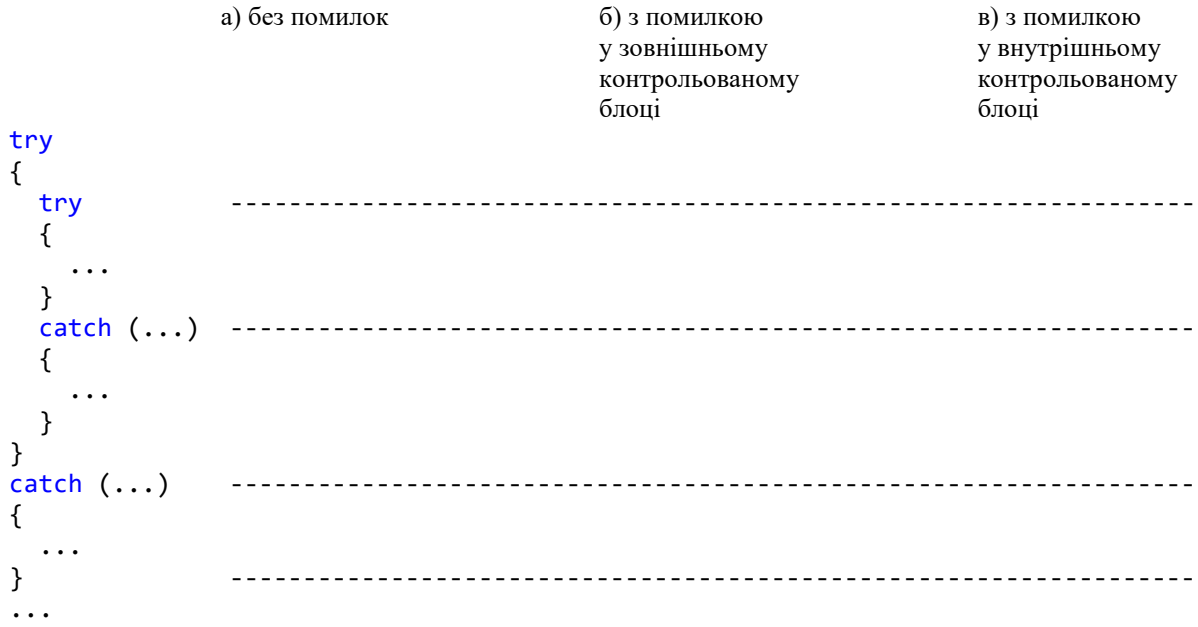
1. Виняток в мові C++ – це:
   а) переривання
   б) подія
   в) помилка в програмі
   г) ситуація
   д) об’єкт

2. Які винятки може генерувати функція:
   А)
```
int f(int x) throw(int, string) {
   ...
}
```
   Б)
```
int f(int x) throw(int) {
   ...
}
```
   В)
```
int f(int x) throw() {
   ...
}
```
   Г)
```
int f(int x) {
   ...
}
```

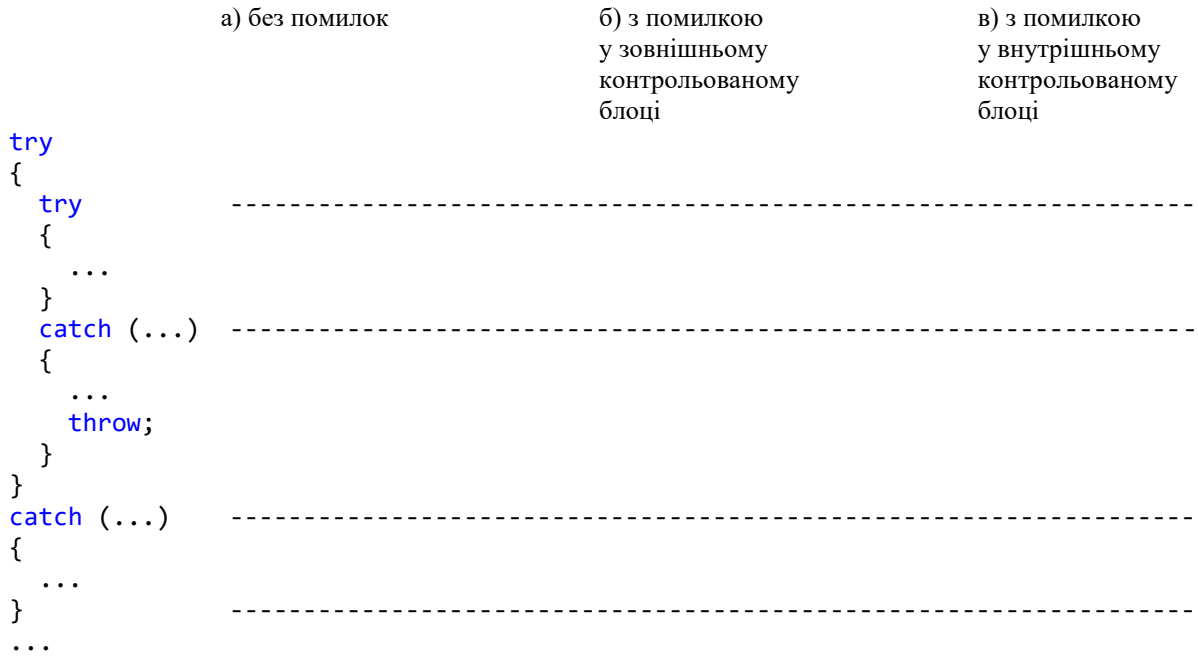
3. Які винятки, згенеровані наступною функцією, можуть бути опрацьовані ззовні (в блоці, що викликає цю функцію):
   А)
```
int f(int x) throw(int, string) {
   ...
}
```
   Б)
```
int f(int x) throw(int) {
   ...
}
```
   В)
```
int f(int x) throw() {
   ...
}
```
   Г)
```
int f(int x) {
   ...
}
```

4. Чим throw відрізняється від return ?
5. Припустимо, що є ієрархія класів винятків, породжена деяким базовим класом винятку. В якому порядку слід розмістити блоки catch?
6. Якщо в блоці try не генеруються ніякі винятки, куди передається управління після того, як блок try завершить роботу?
7. Що відбудеться, якщо виняток буде згенерований поза блоком try?
8. Що відбудеться, якщо жодний обробник не відповідає типу згенерованого винятку?
9. Що відбудеться, якщо кілька обробників відповідають типу згенерованого винятку?
10. Чи повинна генерація винятку приводити до завершення виконання програми?
11. Що відбувається, коли обробник catch генерує виняток?
12. Що робить команда throw; ?
13. Що відбувається, коли функція генерує виняток того типу, який не допускається специфікацією винятків цієї функції?
14. Що відбувається з автоматичними об’єктами, які були створені в блоці try, коли цей блок генерує виняток?
15. Якого типу може бути виняток?
16. Скільки параметрів можна писати в заголовку секції catch – обробнику виняткової ситуації?
17. Якими способами можна передати виняток в блок опрацювання виняткової ситуації?
18. Як можна подолати обмеження на передавання єдиного параметру у блок опрацювання виняткової ситуації?
19. Напишіть конструкцію, яка дозволяє перехопити будь-який виняток.
20. Яким способом виняток «передається далі»?
21. Що таке «розкручування стеку»?
22. Що таке «специфікація винятків»?
23. Що відбувається, якщо функція порушує специфікацію винятків?
24. Чи може конструктор генерувати винятки?
25. Чи може деструктор генерувати винятки?
26. Чи повинен деструктор генерувати винятки?
27. Чи відрізняється функція з «порожньою» специфікацією throw() від функції, в якої взагалі відсутня специфікація винятків?

28. Нарисувати діаграму виконання фрагменту програми:
а) коли виняткова ситуація не виникає,
б) коли виникла виняткова ситуація у зовнішньому контрольованому блоці,
в) коли виникла виняткова ситуація у внутрішньому контрольованому блоці.



29. Нарисувати діаграму виконання фрагменту програми:
а) коли виняткова ситуація не виникає,
б) коли виникла виняткова ситуація у зовнішньому контрольованому блоці,
в) коли виникла виняткова ситуація у внутрішньому контрольованому блоці.



30. Нарисувати діаграму виконання фрагменту програми:
    а) коли виняткова ситуація не виникає,
    б) коли виникла виняткова ситуація у зовнішньому контрольованому блоці,
    в) коли виникла виняткова ситуація у внутрішньому контрольованому блоці.

а) без помилок	б) з помилкою у зовнішньому контрольованому блоці	в) з помилкою у внутрішньому контрольованому блоці

```
try
{
  ...
}
catch (...)    ----------------------------------------------------------------
{
  ...
  try          ----------------------------------------------------------------
  {
    ...
  }
  catch (...)  ----------------------------------------------------------------
  {
    ...
  }
}              ----------------------------------------------------------------
...
```

31. Нарисувати діаграму виконання фрагменту програми:
    а) коли виняткова ситуація не виникає,
    б) коли виникла виняткова ситуація у зовнішньому контрольованому блоці,
    в) коли виникла виняткова ситуація у внутрішньому контрольованому блоці.

а) без помилок	б) з помилкою у зовнішньому контрольованому блоці	в) з помилкою у внутрішньому контрольованому блоці

```
try
{
  ...
}
catch (...)    ----------------------------------------------------------------
{
  ...
  try          ----------------------------------------------------------------
  {
    ...
  }
  catch (...)  ----------------------------------------------------------------
  {
    ...
    throw;
  }
}              ----------------------------------------------------------------
...
```

32. Для наступного фрагменту:

```
try
{
    try
    {
        try
        {
            A();
            B();
        }
        catch (...)
        {
            C();
            throw;
            D();
        }
        E();
    }
    catch (...)
    {
        F();
    }
}
catch (...)
{
    G();
}
H();
```

Які із функцій, наведених у вказаному коді, будуть виконані, якщо функція `A()` викликає виняткову ситуацію

1. B, C, D, E, F, G
2. C, D, E, F, G
3. D, E, F, G
4. C, F, G
5. C, G
6. інший варіант (тоді слід вказати послідовність виконання функцій)

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.