**Желтое – то, в чем не уверен.**

1. **Які синтаксичні конструкції надає C++ для створення користувацьких типів?**

У більшості випадків для створення користувацьких типів застосовують класи. Але іноді також можна використовувати інші мовні конструкції – переліки (перечисления), структури й об'єднання.

1. **Для чого використовують безіменні переліки?**

Безіменний перелік фактично є списком констант. Наприклад,

**enum** { red, green, blue };

Такий перелік еквівалентний визначенню констант:

**const int** red = 0;

**const int** green = 1;

**const int** blue = 2;

Іменований перелік визначає новий тип даних. Цей новий тип може бути використаний для визначення змінних:

**enum** Colors { red, green, blue };

Colors c = blue;

1. **Як змінній типу переліку присвоїти ціле значення?**

**Ніяк**

1. **Що спільного і чим відрізняються структури й масиви?**

Структури дозволяють об'єднати кілька елементів даних різних типів

На відміну від масивів, операція присвоєння здійснює поелементне копіювання структури.

City someCity;

someCity = Kharkiv; // Поелементне копіювання

1. **Як надіслати структури до функцій?**

На відміну від масивів, об'єктів типу структури передаються у функції за значенням. Це означає, що функція працює з копіями об'єктів. Якщо необхідно змінити значення елементів структури всередині тіла функції, слід використовувати аргументи типу посилання. Навіть якщо не потрібно змінювати дані всередині функції, можна використовувати посилання на константні типи, що підвищить продуктивність програми через копіювання адреси замість величезного блоку даних:

**struct** LargeData

{

// Деяка велика структура

};

**void** someFunc(**const** LargeData& data)

{

**...**

};

1. **Для чого використовують операцію ->?**

Для доступу до елементів структури використовують операцію ".". Спеціальний оператор -> використовують для того, щоб отримати доступ до елементів через указівник на структуру:

cout << someCity.name;

City \*pc = **new** City;

pc->population = 10000000;

1. **У чому полягають особливості об'єднань і для чого їх використовують?**

Об'єднання (**union**) - це структура, у якій всі елементи розташовані за однією адресою. В кожен момент у об'єднанні може зберігатися значення тільки одного з елементів. У наведеному нижче прикладі об'єкт об'єднання може зберігати дійсне або ціле значення, але не одночасно обидва. Спроба присвоїти значення другому елементу призводить до псування значення першого елемента, оскільки насправді пам'ять виділена тільки для одного зі значень.

**union** FloatAndInt

{

**float** f;

**int** i;

};

FloatAndInt fi;

fi.i = 2;

fi.f = 100;

cout << fi.i << endl; // 1120403456

fi.i = 3;

cout << fi.f << endl; // 4.2039e-45

Об'єднання може бути використане для створення масивів, елементи яких мають різні типи. Крім того, можна використати об'єднання для кодування інформації, а також для аналізу внутрішнього представлення даних. У наведеному вище прикладі число 1120403456, якщо його перевести у двійкову систему, показує внутрішнє представлення числа 100.0 з плаваючою крапкою.

1. **Що таке клас і з чого він складається?**

*Клас* (class) - це структурований тип даних, набір елементів даних різних типів і функцій для роботи з цими даними.

1. **Що таке інкапсуляція?**

Одним з важливих принципів об'єктно-орієнтованого підходу є *інкапсуляція* (приховування даних). Зміст інкапсуляції полягає у приховуванні від зовнішнього користувача деталей реалізації об'єкта.

1. **Які рівні доступу до елементів класу підтримує C++?**

Для реалізації інкапсуляції мова C++ надає рівні доступу до елементів класу: **public**, **protected** та **private**. Елементи класу, декларовані як **private** (закриті) можуть використовуватись лише у функціях-елементах цього класу а також у його друзях (класах та функціях). Елементи, декларовані як **protected**(захищені), додатково можуть бути використані у похідних класах. Елементи, декларовані як **public** (публічні) можуть бути використані у будь-якій частині програми. Відповідні ключові слова з двокрапкою використовують для групування елементів класу.

 Елементи класу, декларовані як **private** (закриті) можуть використовуватись лише у функціях-елементах цього класу а також у його друзях (класах та функціях). Елементи, декларовані як **protected**(захищені), додатково можуть бути використані у похідних класах. Елементи, декларовані як **public** (публічні) можуть бути використані у будь-якій частині програми. Відповідні ключові слова з двокрапкою використовують для групування елементів класу.

1. **Що таке функції доступу?**

**class** Country

{

**private:**

**char** name[40];

**double** area;

**int** population;

**public:**

**char**\* getName();

**double** getArea();

**int** getPopulation();

**void** setName(**char\*** value);

**void** setArea(**double** value);

**void** setPopulation(**int** value);

**double** density();

};

Функції-елементи getName(), getArea(), getPopulation(), setName(), setArea() та setPopulation() надають доступ до закритих елементів класу. Такі функції мають назву функцій доступу. Їх ще називають геттерами і сеттерми (getters / setters).

1. **Чи можна в C++ поза класом реалізовувати методи, оголошені всередині класу?**

Функції-елементи, які були оголошені всередині тіла класу, можуть бути реалізовані поза класом. Для того, щоб визначити приналежність функції до класу, використовується операція дозволу області видимості (::). Наведемо реалізацію функції density():

**double** Country::density()

{

**return** population / area;

}

1. **Що таке конструктор і як він викликається?**

У класів є спеціальні функції-елементи, які називаються *конструкторами*. Конструктори автоматично викликаються під час створення об'єкту. Ім'я конструктору повинно збігатись з ім'ям класу. Можна описати будь-яку кількість конструкторів класу. Вони повинні відрізнятися кількістю або типами параметрів.

**class** X

{

**int** j;

**public**:

X(); // Конструктор без параметрів

X(**int** i); // Конструктор з параметрами

. . .

};

1. **Скільки конструкторів без параметрів може бути створено в одному класі?**

**1**

1. **Як створити клас, у якому немає жодного конструктора?**

 Якщо конструктор без параметрів не визначений явно, компілятор створює такий конструктор автоматично. Такий конструктор здійснює ініціалізацію елементів даних усталеними значеннями (нулями). Якщо у класі визначений хоча б один явний конструктор, компілятор не створює автоматично усталеного конструктора.

1. **Що таке конструктор копіювання і коли його слід визначати?**

Можна здійснювати ініціалізацію об'єкта іншим об'єктом того ж типу. В цьому випадку викликається спеціальний конструктор - так званий *конструктор копіювання*. Такий конструктор, якщо його не перекрили, створюється автоматично. Для явного створення конструктору копіювання класу X слід додати такий опис:

**class** X

{

. . .

**public**:

. . .

X(**const** X&);

. . .

};

Явне визначення конструктора копіювання необхідне у випадках, коли поелементне копіювання даних одного об'єкта в інший не є достатнім або коректним.

1. **Що таке деструктор і як він викликається?**

Деструктор - це спеціальна функція-елемент, яка має ім'я ~Ім'яКласу та автоматично викликається перед тим, як об'єкт має бути знищений.Деструктор не може мати параметрів:

**class** X

{

. . .

**public**:

. . .

~X() { /\* Тіло деструктора \*/ }

. . .

};

1. **Скільки деструкторів може бути визначено в класі?**

**1**

1. **Як створити константну функцію-елемент?**

Є спеціальний різновид функцій-елементів - так звані константні функції-елементи. Такі функції не можуть змінити даних об'єкту, для якого вони викликані. Для того, щоб описати таку функцію, після списку аргументів слід розмістити модифікатор **const**:

**class** Country

{

**...**

**public**:

**char**\* getName() **const** { **return** name; }

...

};

Константні функції-елементи можна викликати для константних об'єктів:

**const** Country c = "France";

cout << c.getName();

c.setName("Belgium"); // Помилка!

1. **Що таке вказівник this і для чого його використовують?**

Функції-елементи мають доступ до інших елементів класу через так званий вказівник **this**. Функції-елементи отримують цей вказівник як неявний аргумент. Цей вказівник вказує на об'єкт, для якого викликана функція-елемент. Найчастіше вказівник **this**застосовують для того, щоб уникнути конфліктів імен:

**class** X

{

**int** value;

**public**:

X(**int** value) { **this**->value = value; }

};

Іноді необхідно повернути посилання або вказівник на об'єкт:

Country&Country::getCountry()

{

. . .

**return** \***this**;

}

Якщо функція-елемент має модифікатор **const**, вона одержує вказівник **this**, який вказує на константний об'єкт.

1. **Які елементи входять в область видимості класу?**

Клас визначає свою область видимості. Усі елементи класу входять до його області видимості. Іноді до елементів класової області видимості можна звертатись за допомогою оператору ::, який застосовується до імені класу. Наприклад, можна використовувати цей оператор замість указівника **this**:

**class** X

{

**int** i;

**public**:

**void** f() { **int** i; i = X::i; }

};

1. **Чи можна створювати класи всередині інших класів?**

Клас може бути описаний у глобальній області видимості, а також у класовій та локальній областях. Класи, які визначені всередині інших класів, мають назву внутрішніх. Об'єкт внутрішнього класу не створюється автоматично у зовнішньому класі. Такий об'єкт необхідно створювати окремо:

**class** X

{

**public**:

**class** Y

{

**public:**

**int** t;

};

**private**:

**class** Z

{

**public:**

**int** w;

};

Z z; // Об'єкт внутрішнього класу

};

X::Y y; // OK

X::Z z; // Помилка! Немає доступу до Z

1. **Що таке статичний елемент класу?**

Іноді нам необхідно створити змінні, які логічно мають відношення до певного класу, але їх не доцільно робити елементами даних об'єктів, бо вони повинні бути спільними для усіх об'єктів. Статичні елементи даних пропонують спосіб розміщення глобальних змінних в області видимості класу. Звертатись до таких елементів можна як через імена об'єктів, так і імена класів:

**class** X

{

. . .

**public:**

**static int** i;

};

. . .

X::i = 10; // Через ім'я класу

X x;

x.i = 11; // Через ім'я об'єкту

Як видно з наведеного прикладу, статичні елементи не потребують створення об'єкту.

Статичні елементи даних не можуть бути описані з модифікатором **mutable**.

1. **Які обмеження накладаються на реалізацію статичних функцій?**

Існують також статичні функції-елементи. Вони не отримують указівника **this** і тому не мають доступу до нестатичних елементів об'єкту, для якого вони викликані.

1. **Чому статичні елементи даних слід визначати поза межами класу?**

Поскольку статические данные-члены являются дискретными переменными, общими для всех объектов класса, их необходимо определять и инициализировать за пределами объявления класса. (Дополнительные сведения о классах и членах классов см. в разделе

1. **Що таке друзі класу? Які синтаксичні конструкції можуть бути друзями класу?**

Функції або класи, оголошені як друзі класу, мають доступ до його закритих та захищених елементів. Друзі не є частиною області видимості класу та не отримують указівника **this**.

**class** SomeClass

{

**friend void** f(SomeClass &sc);

**private**:

**int** i;

};

**void** f(SomeClass &sc)

{

cout << sc.i; // Доступ до закритого елементу даних

}

**void** main()

{

SomeClass s1;

f(s1);

}

Друзьями могут быть другие классы, функции-эдементы  
Якщо X - друг класу Y, Y - друг класу Z, X не є автоматично другом Z. Дружба не успадковується.

1. **Чи входять друзі класу в область видимості класу?**

**No**

1. **Для чого призначений механізм винятків?**

Дуже часто функція програми, у якій виникає певна помилка, не має можливості виправити цю помилку, бо невідомим є контекст виклику цієї функції. Помилку необхідно передати до тієї частини програми, у якій її можна обробити.

1. **Як створити об'єкт-виняток?**
2. **Яких типів можуть бути об'єкти-винятки?**

Для винятків можна застосовувати фундаментальні типи даних, але кращий підхід базується на створенні власних типів, оскільки розпізнавання винятків базується на типах. Найчастіше це нові класи, які в окремому випадку можуть бути порожніми:

**class** Not\_Found {};

**class** Bad\_Data {};

1. **Чи можна використовувати основний результат функції, якщо відбулася генерація винятку?**

**нет**

1. **Як перехопити й обробити виняток?**

**throw (тип или класс) try catch**

1. **Як створити блок обробки всіх винятків?**

Спеціальна конструкція **catch**(...) застосовується для створення оброблювача усіх типів винятків. Наприклад:

1. **Для чого використовують перевантаження операцій?**

**Для управления функциями вызываемыми операторами**

1. **Які операції не можна перевантажувати?**

Автор класу може визначити (перевантажити) всі операції, визначені в С++, крім чотирьох: "::", ".\*", "." та "?:"

1. **Коли операцію слід перевантажувати тільки визначивши функцію-елемент?**

Якщо операторна функція є елементом класу, кількість її параметрів повинна бути на 1 менше кількості операндів операції, що перевантажується, тому що в цьому випадку першим операндом є сам об'єкт. Деякі операції - присвоювання "=", індексація "[]", виклик функції "()" і вибір елемента "->" повинні бути перевантажені тільки як функції-елементи.

1. **Коли операцію слід перевантажувати тільки визначивши глобальну функцію?**

**---**

1. **Як перевантажити операцію перетворення типів?**

Існує особливий вид операторних функцій - операції перетворення типів, що дозволяють перетворювати об'єкт у заданий тип за умовчанням. Операція перетворення повинна бути реалізована у вигляді функції-елемента й у загальному випадку вона має такий вигляд:

X::**operator** T(); // T - ім'я типу

Для операції перетворення не можна задавати тип значення, що повертається, чи вказувати список формальних параметрів. Наприклад, завдяки операції перетворення припустимі такі дії:

**class** X

{

**int** i;

**public**:

**operator** **int**() { **return** i; }

. . .

};

. . .

X x;

**int** k = x; // Використовується елемент даних i

**int** n = x + k