1. У чому полягає концепція успадкування?

Успадкування (inheritance) застосовують для створення класу, похідного від певного базового класу. Похідні класи успадковують елементи базових класів, а також можуть додавати нові елементи.

Синтаксично успадкування визначається списком базових класів після імені класу і двокрапки. Перед іменем базового класу вказується тип успадкування (public або інше):

class X // Базовий клас

{

. . .

};

class Y : public X // Похідний клас

{

. . .

};

1. Чим відрізняється відкрите, захищене і закрите успадкування?

У С++ прийняті такі типи успадкування:

* *відкритий базовий клас*(**public**): відкриті елементи базового класу залишаються відкритими у похідному класі, захищені елементи базового класу залишаються захищеними у похідному класі, закриті елементи базового класу є закритими у похідному класі;
* *захищений базовий клас* (**protected**): відкриті та захищені елементи базового класу є захищеними у похідному класі, закриті елементи базового класу є закритими у похідному класі;
* *закритий базовий клас* (**private**): відкриті та захищені елементи базового класу є закритими у похідному класі, закриті елементи базового класу є закритими у похідному класі; цей спосіб є прийнятим за умовчанням.

1. Які переваги й недоліки множинного успадкування?

Мова С++ підтримує одиночне та множинне успадкування. Найбільш розповсюдженим є одиночне успадкування, коли клас походить від одного базового класу. У випадку використання множинного успадкування клас може походити від кількох базових класів. Наприклад,

class X // Базовий клас

{

. . .

};

class Z // Базовий клас

{

. . .

};

class Y : public X, public Z // Похідний клас

{

. . .

};

Множинне успадкування (multiple inheritance) застосовується в тому випадку, коли об'єкт являє собою поняття, що поєднує в собі кілька загальних понять, кожне з яких може бути представлено базовим класом.

До специфікації базового класу можна додати службове слово virtual. Окремий об'єкт віртуального базового класу V розділяється між всіма класами, які вказали V, визначаючи свої базові класи, наприклад:

class A : virtual public V

{

. . .

};

class B : virtual public V

{

. . .

};

class C : public A, public B

{

. . .

};

Тут об'єкт класу C матиме тільки один вкладений об'єкт класу V (ромбовидне успадкування

У визначенні класів можна використовувати **using**-оголошення. Оголошення **using** дозволяють створювати набір перевантажених функцій з базових і похідних класів; **using**-оголошення у визначенні класу має відношення до членів базового класу. **using**-оголошення не може бути використано для отримання доступу до закритої інформації базового класу.

**class** Base

{

**public**:

**void** setValue(**int**);

};

**class** Descendant : **private** Base

{

**public**:

**using** Base::setValue;

**void** setValue(string);

};

**int** main()

{

Descendant d;

d.setValue("a");

d.setValue(2); // Без using-оголошення була б помилка

**return** 0;

}

Множинне успадкування класів незалежно від способів його реалізації є небезпечним з точки зору можливого конфлікту імен, його використання може призвести до виникнення суперечливих і дубльованих даних. Використання множинного успадкування слід уникати в більшості випадків. Єдиний сенс його застосування - коли один з базових класів є абстрактним і зовсім не містить даних. Абстрактні класи будуть розглянуті нижче.

1. Для чого визначають віртуальний базовий клас?

Окремий об'єкт віртуального базового класу V розділяється між всіма класами, які вказали V, визначаючи свої базові класи, наприклад:

1. Для чого створюють ієрархії винятків?

**class** Error

{

// ...

};

**class** MathError : **public** Error

{

// ...

};

Оброблювач базового типу обробляє також винятки похідних типів:

...

**catch** (Error)

{

// обробляє виняток типу Error та MathError

}

1. У чому полягає концепція поліморфізму?

*Поліморфізм* (polymorphism) - це механізм визначення серед функцій з однаковими іменами функції для виклику, заснований на типі параметрів (*поліморфізм часу компіляції*) або об'єкта, для якого викликається метод (*поліморфізм часу виконання*). Поліморфізм часу виконання у C++ використовує ієрархії успадкування.

В [языках программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) и [теории типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D1%82%D0%B8%D0%BF%D0%BE%D0%B2) **полиморфизмом** называется способность [функции](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) обрабатывать данные разных [типов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)

**class** X

{

. . .

};

**class** Y **: public** X

{

. . .

};

**class** Z **: public** Y

{

. . .

};

X \*px = **new** Y();

Z z;

X &rx = z;

Для інших перетворень типів указівників та посилань на об'єкти однієї ієрархії вживають спеціальний оператор **dynamic\_cast**<тип>(вираз).

Тепер можна створити масив указівників на базовий клас та здійснити ініціалізацію елементів указівниками на похідні класи:

X \*a[3];

x[0] = **new** X();

x[1] = **new** Y();

x[2] = **new** Z();

1. Чим відрізняється поліморфізм часу компіляції від поліморфізму часу виконання?

**Compile-time Polymorphism**(полиморфизм времени компиляции) - полиморфизм, который разрешается на этапе компиляции. Пример такого полиморфизма - использование шаблонов.

**Runtime Polymorphism** - обычный, наиболее распространенный полиморфизм, который разрешается на этапе исполнения(запуска) программы.

Пример полиморфизма времени компиляции :

**#include <iostream>**

**template** **<class** **T>**

**struct** my\_template {

T foo;

my\_template() **:** foo(T()) {}

};

**template** **<>**

**struct** my\_template**<int>** {

**enum** { foo **=** 42 };

};

**int** **main**() {

my\_template**<int>** x;

my\_template**<long>** y;

my\_template**<**std**::**string**>** z;

std**::**cout **<<** x.foo **<<** std**::**endl;

std**::**cout **<<** y.foo **<<** std**::**endl;

std**::**cout **<<** z.foo **<<** std**::**endl;

**return** 0;

}

Пример полиморфизма времени исполнения :

**void** foo(**float**);

**void** foo(**int**);

1. Що таке віртуальна функція?

Віртуальна функція (virtual function) - це функція-елемент, адреса якої може визначатись динамічно під час виконання програми. *Віртуальні функції* - це методи з фіксованим інтерфейсом виклику та різними реалізаціями у різних похідних класах. У визначені класу для ідентифікації віртуальних функцій вживається слово **virtual**:

Віртуальні функції не можуть бути описані з модифікатором **static**. Віртуальні функції можуть бути **inline**, але цей специфікатор має ефект лише для виклику функції для об'єкту (не для вказівника або посилання).

1. Які класи вважаються поліморфними?

Об'єкт класу, у якому є віртуальні функції, має назву *"поліморфний об'єкт"*, а відповідний клас - поліморфним класом (polymorphic class).

1. Чому в поліморфних класах доцільно визначати віртуальні деструктори?

Деструктори також можуть бути віртуальними. Після оголошення деструктору віртуальним деструктори усіх похідних класів є віртуальними автоматично. Віртуальні деструктори дозволяють точно визначити обсяг пам'яті, яку необхідно звільнити, коли знищується об'єкт. Це особливо важливо для об'єктів, які створені у динамічній пам'яті та знищуються за допомогою операції **delete**.

 Только так обеспечивается корректное разрушение объекта производного класса через указатель на соответствующий базовый класс.

Уничтожение объекта производного класса через указатель на базовый класс с невиртуальным деструктором дает неопределенный результат. На практике это выражается в том, что будет разрушена только часть объекта, соответствующая базовому классу

1. Що таке таблиця віртуальних методів?

 Кожний поліморфний об'єкт містить спеціальне поле, у яке конструктор записує адресу так званої *таблиці віртуальних методів* (Virtual Method Table, VMT). Така таблиця створюється для усіх класів з віртуальними функціями. Таблиця віртуальних методів містить адреси віртуальних функцій класу. Ці адреси використовуються для виклику віртуальних функцій.

Для реализации механизма виртуальных функций каждый объект класса хранит указатель на таблицу виртуальных функций vptr, что увеличивает его общий размер. Обычно, при объявлении виртуального деструктора такой класс уже имеет виртуальные функции, и увеличения размера соответствующего объекта не происходит.

1. Що таке суто віртуальна функція?

Іноді у базовому класі не доцільно здійснювати реалізацію певної віртуальної функції, оскільки клас створено для представлення базової абстракції. В такому випадку функцію можна описати як суто віртуальну (pure virtual function). Для цього використовують спеціальний синтаксис:

virtual void f() = 0;

1. Що таке абстрактний клас?

*Абстрактний клас* (abstract class) - це клас, який містить оголошення хоча б однієї суто віртуальної функції:

**class** AbstractBaseClass

{

. . .

**public:**

**virtual void** f() = 0;

**virtual int** secondFunc(**int** x);

};

1. Для чого використовують шаблонні функції?

Шаблонні функції (template functions) призначені для запису узагальнених функцій, що можуть працювати з даними різних типів.

1. Чи можна шаблонну функцію окремо реалізувати для певного типу?

Функція-елемент шаблонного класу вважається неявною шаблонною функцією, а параметри шаблона типу для її класу - параметрами її шаблона. Для деяких типів стандартні функції-елементи не підходять. У таких випадках можна явно задавати реалізацію функції, розрахованої на конкретний тип. Перед реалізацією таких функцій потрібно спеціальне оголошення **template**<> без параметрів.

1. Навіщо параметр шаблону в шаблонній функції вказувати явно?

если процесс вывода дает два различных типа для одного и того же параметра шаблона, то компилятор сообщает об ошибке – неудачном выводе аргументов.  
В таких ситуациях приходится подавлять механизм вывода и задавать аргументы явно, указывая их с помощью заключенного в угловые скобки списка разделенных запятыми значений, который следует после имени конкретизируемого шаблона функции

1. Як створити шаблон класу?

оголошення і визначення шаблону класу починається зі службового слова **template**. За ним іде список формальних параметрів шаблона типу. Цей список не може бути порожнім. Наприклад:

**template** <**class** T> **class** X

{

T t;

**public**:

**void** set(T t1) { t = t1; }

};

1. Якими можуть бути параметри шаблону?

Параметрами шаблонів можуть бути параметри-типи, параметри звичайних типів і параметри-шаблони. У шаблона може бути кілька параметрів.

1. Для чого використовують цілі параметри шаблону?

Цілі аргументи використовуються найчастіше для завдання розмірів і границь масивів. Цілий аргумент шаблона повинен бути константою. Можливе завдання усталених параметрів шаблона:

template <class T, int size = 64> class Y

{

. . .

};

Фактичне значення цілого параметру повинно бути константним виразом:

const int N = 128;

int i = 256;

Y<int, 2\*N> b1;// OK

Y<float, i> b2;// Помилка: i не константа

1. Що таке інстанціювання шаблону?

*Інстанціювання шаблону* (template instantiation) - це створення певних типів з шаблону. Такі класи мають назву *екземплярів шаблону* (template instances):

X<**int**> xi;

X<**double**> xd;

Instance – пример, экземляр

1. Як здійснюється перетворення об'єктів інстанційованих типів

---