Полиморфизм в C++ — это одна из ключевых концепций объектно-ориентированного программирования (ООП), которая позволяет объектам разных классов обрабатываться единообразно через общий интерфейс или базовый класс. Слово "полиморфизм" буквально означает "много форм" (от греч. "poly" — много, "morph" — форма), и в контексте C++ это означает, что одна и та же операция может выполняться по-разному в зависимости от типа объекта.

В C++ полиморфизм реализуется в основном через **наследование** и **виртуальные функции**. Давайте разберём это подробнее.

**Типы полиморфизма в C++**

1. **Полиморфизм времени компиляции (статический полиморфизм):**
   * Реализуется через перегрузку функций, операторов или шаблоны.
   * Компилятор заранее определяет, какая именно функция или оператор будет вызвана, основываясь на типах аргументов.

Пример перегрузки функций:

cpp

СвернутьПереносКопировать

#include <iostream>

class Example {

public:

void print(int x) { std::cout << "Integer: " << x << std::endl; }

void print(double x) { std::cout << "Double: " << x << std::endl; }

};

int main() {

Example obj;

obj.print(5); *// Вызовет print(int)*

obj.print(5.5); *// Вызовет print(double)*

return 0;

}

Пример с шаблонами:

cpp

СвернутьПереносКопировать

#include <iostream>

template <typename T>

void print(T value) { std::cout << value << std::endl; }

int main() {

print(5); *// T = int*

print(5.5); *// T = double*

print("Hi"); *// T = const char\**

return 0;

}

1. **Полиморфизм времени выполнения (динамический полиморфизм):**
   * Реализуется через наследование и виртуальные функции.
   * Позволяет вызывать методы производных классов через указатель или ссылку на базовый класс. Какой именно метод будет вызван, определяется во время выполнения программы.

Пример:

cpp

СвернутьПереносКопировать

#include <iostream>

class Animal {

public:

virtual void speak() { std::cout << "Some sound\n"; } *// Виртуальная функция*

virtual ~Animal() = default; *// Виртуальный деструктор*

};

class Dog : public Animal {

public:

void speak() override { std::cout << "Woof!\n"; } *// Переопределение*

};

class Cat : public Animal {

public:

void speak() override { std::cout << "Meow!\n"; }

};

int main() {

Animal\* ptr = nullptr;

ptr = new Dog();

ptr->speak(); *// Вывод: "Woof!"*

delete ptr;

ptr = new Cat();

ptr->speak(); *// Вывод: "Meow!"*

delete ptr;

return 0;

}

**Ключевые элементы динамического полиморфизма**

1. **Виртуальные функции:**
   * Ключевое слово virtual в базовом классе указывает, что функция может быть переопределена в производных классах.
   * Вызов нужной версии функции определяется в runtime через таблицу виртуальных функций (vtable).
2. **Ключевое слово override:**
   * Используется в производном классе, чтобы явно указать, что функция переопределяет виртуальную функцию базового класса.
3. **Виртуальный деструктор:**
   * Если базовый класс имеет виртуальный деструктор, это гарантирует правильное уничтожение объектов производных классов через указатель на базовый класс.
4. **Чисто виртуальные функции и абстрактные классы:**
   * Если функция объявлена как virtual void func() = 0;, она становится чисто виртуальной, а класс — абстрактным. Такие классы нельзя инстанцировать напрямую, только через наследование.

Пример:

cpp

СвернутьПереносКопировать

class Abstract {

public:

virtual void doSomething() = 0; *// Чисто виртуальная функция*

};

class Concrete : public Abstract {

public:

void doSomething() override { std::cout << "Doing something\n"; }

};

**Зачем нужен полиморфизм?**

* **Гибкость:** Позволяет писать код, который работает с общим интерфейсом, не заботясь о конкретных типах объектов.
* **Расширяемость:** Легко добавлять новые классы, не меняя существующий код.
* **Абстракция:** Скрывает детали реализации, предоставляя единый способ взаимодействия с объектами.

**Отличие статического и динамического полиморфизма**

* **Статический:** Решение о вызове принимается на этапе компиляции (быстрее, но менее гибко).
* **Динамический:** Решение о вызове принимается во время выполнения (гибче, но с накладными расходами на vtable).

Если есть конкретный пример или вопрос по полиморфизму, дайте знать — разберём его вместе!

|  |
| --- |
| Для конченной залупы с ебаной кодировки  #include <windows.h>  SetConsoleOutputCP(CP\_UTF8); |

|  |
| --- |
| **Автоматическая компиляция с задачами (tasks.json)**:   * Нажмите Ctrl+Shift+P, введите "Tasks: Configure Task" и выберите "C/C++: g++ build active file". * Это создаст файл .vscode/tasks.json в папке проекта: |

|  |
| --- |
| Скачивание компилятора с дебагером  pacman -S mingw-w64-x86\_64-gcc mingw-w64-x86\_64-gdb  pacman -S mingw-w64-x86\_64-gdb |