Виртуальные функции и полиморфизм в C++

Полиморфизм (polymorphism) (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения **двух или более схожих**, но технически разных задач. **Целью** полиморфизма, применительно к ООП, **является использование одного имени** для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных.

В C++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций (**function overloading**).

В более общем смысле, концепцией полиморфизма является идея "один интерфейс, множество методов". Это означает, что можно создать общий интерфейс для группы близких по смыслу действий. Преимуществом полиморфизма является то, что он помогает понижать сложность программ, разрешая использование того же интерфейса для задания единого класса действий.

Полиморфизм может применяться также и к операторам. Фактически во всех языках программирования ограниченно применяется полиморфизм, например, в арифметических операторах. Так, в Си, символ + используется для складывания целых, длинных целых, символьных переменных и чисел с плавающей точкой. В этом случае компилятор автоматически определяет, какой тип арифметики требуется. В C++ вы можете применить эту концепцию и к другим, заданным вами, типам данных. Такой тип полиморфизма называется перегрузкой операторов (**operator overloading**).

Ключевым в понимании полиморфизма является то, что он позволяет вам манипулировать объектами различной степени сложности путём создания общего для них стандартного интерфейса для реализации похожих действий.

Полиморфизм обеспечивается за счет использования **производных классов и виртуальных функций**.

Виртуальная функция — это функция, объявленная с ключевым словом virtual в базовом классе и переопределенная в одном или в нескольких производных классах. Виртуальные функции являются особыми функциями, потому что при вызове объекта производного класса с помощью указателя или ссылки на него C++ определяет во время исполнения программы, какую функцию вызвать, основываясь на типе объекта. Для разных объектов вызываются разные версии одной и той же виртуальной функции.

Класс, содержащий одну или более виртуальных функций, называется полиморфным классом (**polymorphic class**).

Виртуальные функции в C++

```
class Base{
public:
  virtual void  function1 () {
          cout << “Base::function1()" << endl; }
            void function2 () {
          cout << “Base::function2()" << endl; }
};

class First : public Base{
public:
  virtual void  function1 () {
          cout << " First ::function1()" << endl; }
            void  function2 () {
          cout << " First ::function2()" << endl; }
};
```

Работа с виртуальными функциями в C++

```
First*  pointer = new First();
Base* pointer_copy  =  pointer;

pointer->function1 ();
pointer->function2 ();

pointer_copy->function1 ();
pointer_copy->function2 ();
```

Класс **Base** определяет две функции, одну **виртуальную**, другую — нет. Класс **First** переопределяет обе функции. Однако, одинаковое обращение к функциям даёт разные результаты. При выводе программа даст следующее:

First::function1()
First ::function2()
First ::function1()
Base::function2()

```
class Base {
public:
    virtual void who() { // определение виртуальной функции
        cout << "Base\n";
    }
};

class first_d: public Base {
public:
    void who() { // определение who() применительно к first_d
        cout << "First derivation\n";
    }
};

class second_d: public Base {
public:
    void who() { // определение who() применительно к second_d
        cout << "Second derivation\n";
    }
};
```

```
int main(void){
    Base base_obj;
    Base *p;
    first_d first_obj;
    second_d second_obj;
    p = &base_obj;
    p->who(); // доступ к who класса Base
    p = &first_obj;
    p->who(); // доступ к who класса first_d
    p = &second_obj;
}
```

Программа вернет следующий результат:

Base
First derivation
Second derivation

Наиболее распространенным способом вызова виртуальной функции служит использование параметра функции. Например, рассмотрим следующую модификацию предыдущей программы:

```
// использование в качестве параметра ссылки на базовый класс
void show_who (Base &r) {
    r.who();
}
int main(){
    Base base_obj;
    first_d first_obj;
    second_d second_obj;
    show_who (base_obj) ; // доступ к who класса Base
    show_who(first_obj); // доступ к who класса first_d
    show_who(second_obj); // доступ к who класса second_d
}
```

Программа выводит на экран те же самые данные, что и предыдущая версия. В данном примере функция show_who() имеет параметр типа ссылки на класс Base. В функции main() вызов виртуальной функции осуществляется с использованием объектов типа Base, first_d и second_d. Вызываемая версия функции who() в функции show_who() определяется типом объекта, на который ссылается параметр при вызове функции.

- Если функция была объявлена как виртуальная, то она и остается таковой вне зависимости от количества уровней в иерархии классов, через которые она прошла. Например, если класс second_d получен из класса first_d, а не из класса Base, то функция who() останется виртуальной и будет вызываться корректная ее версия, как показано в следующем примере:
- Если в производном классе виртуальная функция не переопределяется, то тогда используется ее версия из базового класса.

Абстрактные классы и чистые виртуальные функции

```
class Base{
        virtual void who() = 0;
};
```

Класс не реализует поведение какого-то конкретного действия, а представляет интерфейс для работы с функцией. В этом случае, метод **who(...)** этого класса нужно сделать чистым виртуальным, дописав **"= 0"** после его сигнатуры: