Принцип полиморфизма – подразумевает возможность перегрузки имен операции и позволяет использовать одни и те же функции для решения различных задач. Различают полиморфные (поддерживают принцип полиморфизма в теории типов) и мономорфные (предполагают, что процедуры и операторы имеют уникальный тип) языки программирования. При наследовании часто возникают ситуации когда необходимо чтобы поведение базовой функции было модифицировано в зависимости от того объектом какого класса эта функция вызвана. Простое переопределение функции может привести к конфликту имен.

При определении полиморфных функций следует различать 2 понятия:

1. перегрузка функций предполагает использование одноименных методов в рамках иерархии отличающихся типами или количеством параметров
2. перекрытие – сигнатуры родительских и дочерних методов должны полностью совпадать. Особенность ярко иллюстрируется при вызове внутри переопределенного метода другого переопределенного метода, при этом возникает вопрос о выборе в качестве вызываемого метода родительского или дочернего компонента. Для того чтобы изменить данную ситуацию необходимо определять перекрытые методы как виртуальные при этом процесс вызова метода будет изменен. Каждый объект при своем создании запрашивает выделение памяти под свою структуру. Память выделяется под член данные объекта и указатели на 2 таблицы:
3. таблица статических методов (STM)
4. таблица виртуальных методов (VMT) - содержит адреса функций, расположенных в порядке иерархии создания. При вызове виртуальной функции осуществляется переход по адресу, указанному в таблице виртуальных методов со смещением присущим данному классу объектов.

Правила описания и использования виртуальных функций:

1. Виртуальная функция может являться только методом класса и не может являться внешней функцией программы.
2. Любую перегружаемую операцию можно сделать виртуальной.
3. Виртуальные функции, как и сама виртуальность, наследуются.
4. Виртуальная функция может быть const.
5. Если в базовом классе впервые объявлена виртуальная функция, то она должна быть, либо чистой (virtual int f(void)=0;), либо у нее должно быть задано определение, т.е. тело. Если в базовом классе объявлена чистая виртуальная функция, то в порожденных классах нужно дать ей определение или определить снова как чистую.
6. Если в базовом классе определена виртуальная функция, то метод производного класса с таким же именем автоматически, то будет являться виртуальным.
7. Конструкторы не могут быть виртуальными.
8. Методы, объявленные как static, не могут быть виртуальными.
9. Деструкторы чаще всего должны быть виртуальными.
10. Если некоторая функция вызывается с использованием в виде полного имени, то виртуальный механизм игнорируется.

Выделяют 2 типа полиморфизма (отличаются моментом связывания объекта с вызываемым методом):

- статический (времени компиляции) – привязка метода к объекту осуществляется на этапе компиляции программы, следовательно статический полиморфизм реализуется через механизм перегрузки или шаблона.

- динамический (времени выполнения) – привязка метода к объекту осуществляется на этапе выполнения программы, что связано с изъятием адреса используемого метода из таблицы метода данного объекта.

Реализуется через механизм виртуальных функций. Класс, включающий виртуальную функцию, называется полиморфным. Если в классе присутствует чистая виртуальная функция, то такой класс называется абстрактным. Нельзя создать объект абстрактного класса. Абстрактные классы используются только для дальнейшего наследования. Необходимость введения виртуальных функций определяется следующими обстоятельствами:

1. вся тяжесть по определению типа объекта при написании в статическом стиле ложится на программиста, а не на компилятор.
2. программа, написанная в статическом стиле, привязана к деталям иерархии приложения, при изменении иерархии необходимо менять текст приложения.
3. при изменении программы сторонним разработчиком в случае статического программирования требуются знания подробности иерархии наследования.

Вызов виртуальных функций может являться не виртуальным в следующих случаях:

1. если осуществляется не через указатель или ссылку при этом вызов становится статическим.
2. если вызов осуществляется через указатель или ссылку, но с уточнением имени класса.
3. если он осуществляется внутри конструктора или деструктора базового класса.

Пример :

#include <iostream>

using namespace std;

// Класс, содержащий функцию вычисления длины окружности

class CalcLength

{

public:

// Виртуальная функция

virtual double Calc(double radius)

{

return 2 \* 3.1415 \* radius;

}

};

// Класс, содержащий функцию вычисления площади окружности

class CalcArea : public CalcLength

{

public:

// Виртуальная функция

double Calc(double radius) override

{

return 3.1415 \* radius \* radius;

}

};

// Класс, содержащий функцию вычисления объема шара

class CalcVolume : public CalcArea

{

public:

// Виртуальная функция

double Calc(double radius) override

{

return 4.0 / 3 \* 3.1415 \* radius \* radius \* radius;

}

};

// Некоторая функция, получающая указатель на базовый класс ClassLength и параметр радиуса,

// в данной функции демонстрируется полиморфизм

void ShowResult(CalcLength\* p, double radius)

{

// Вызов метода Calc() по указателю p.

// Для данной функции неизвестно, метод какого класса будет вызван.

// Нужный метод будет сформирован во время выполнения - это есть полиморфизм

double res = p->Calc(radius); // <=== общий интерфейс для разных реализаций

cout << "Result = " << res << endl;

}

void main()

{

// 1. Объявить указатель на базовый класс - это важно

CalcLength\* p = nullptr;

// 2. Создать экземпляры 3-х классов

CalcLength obj1;

CalcArea obj2;

CalcVolume obj3;

// 3. Ввести номер функции

int num;

cout << "Enter number of function (1-3): ";

cin >> num;

if ((num < 1) || (num > 3))

return;

// 4. Ввести радиус

double radius;

cout << "radius = ";

cin >> radius;

// 5. Установить указатель p в зависимости от введеного num

if (num == 1) p = &obj1;

if (num == 2) p = &obj2;

if (num == 3) p = &obj3;

// 6. Вызвать метод ShowResult()

// Нужный объект подставляется в зависимости от ситуации

ShowResult(p, radius);

// 7. Вызвать метод ShowResult() непосредственно подставляя экземпляр класса

if (num == 1) ShowResult(&obj1, radius);

if (num == 2) ShowResult(&obj2, radius);

if (num == 3) ShowResult(&obj3, radius);

}