# Наследование

**Определение**: Класс, который наследуется, называется базовым классом. Наследуемый класс называется производным классом.

Базовый класс определяет все те качества, которые будут общими для всех производных от него классов. Производный класс наследует эти общие черты и добавляет свойства, характерные только для него.

**Пример**: // *Определение базового и производного классов*

//Определение базового класса

class B {

int i;

public:

void set\_i(int n);

int get\_i();

};

//Определение производного класса

class D: public B {

int j;

public:

void set\_j(int n);

int mul();

};

Основная форма наследования базового класса:

class имя\_производного\_класса : спецификатор\_доступа имя\_базового\_класса {

// тело класса

}

**Спецификатор\_доступа** – одно из трех ключевых слов: public (открытый), private (закрытый), protected (защищенный).

Если выбран спецификатор доступа public, то все открытые члены базового класса будут также открытыми членами производного класса. Все закрытые члены базового класса останутся закрытыми, и к ним не будет доступа из производного класса.

Если выбран спецификатор доступа private, то все открытые члены базового класса в производном классе становятся закрытыми. Однако они доступны для функции – членов производного класса.

Если выбран спецификатор доступа protected, то закрытые члены базового класса становятся недоступными для членов производного класса.

Если спецификатор доступа не указан явно, то, по умолчанию, производный класс наследуется как *закрытый*.

**Режим доступа при наследовании класса:**

К элементам в К элементам в

базовом классе: производном классе:







**Пример**: // *Наследование с уровнем доступа* ***public***

#include <iostream>

using namespace std;

class B {

int x;

public:

int y;

void set\_xy(int n, int m) { x = n; y = m; }

void mul\_B\_xy() { cout << "mul\_B\_xy=" << x \* y << endl; }

};

// Класс наследуется как открытый

class D : public B {

int z;

public:

void set\_z(int n) { z = n; }

// void mul\_D\_xy(){cout<<"mul\_D\_xy="<<x\*y<<endl;} // ошибка

void mul\_yz() { cout << "mul\_yz=" << y \* z << endl; }

};

int main()

{

B ob;

D ob1;

ob.set\_xy(5, 7);

ob1.set\_xy(10, 15); // инициализация членов базового класса в D

ob1.set\_z(20); // инициализация члена производного класса D

ob.mul\_B\_xy();

ob1.mul\_B\_xy();

// ob1.mul\_D\_xy(); // ошибка

ob1.mul\_yz();

system("pause");

return 0;

}

**Комментарий**: В программе иллюстрируется уровень доступа ***public*** к членам объекта производного класса ***D*** (для открытых и закрытых членов базового класса). Следует обратить внимание:

1. На работу функций mul\_B\_xy() и mul\_D\_xy(). Эти функции идентичны, но mul\_B\_xy() определена как *public* в базовом классе и у нее есть доступ к закрытой переменной **x**. Эта функция будет работать в объектах и базового и производного классов. Функция mul\_D\_xy() определена в производном классе и ей будет недоступна закрытая переменная **x** базового класса.
2. Функция mul\_D\_xy() определена в производном классе как *public*, поэтому у нее есть доступ к открытой переменной базового класса **y** и закрытой переменной производного класса **z**.

**Пример**: /\* *Наследование с уровнем доступа* ***private*** \*/

#include <iostream>

using namespace std;

class B {

int x;

public:

void set\_x(int n) { x = n; }

void show\_x() { cout << "x=" << x << endl; }

};

// Класс наследуется как закрытый

class D : private B {

int y;

public:

void set\_y(int n) { y = n; }

void show\_y() { cout << "y=" << y << endl; }

};

int main()

{

B ob;

D ob1;

ob.set\_x(5);

ob.show\_x();

// ob1.set\_x(10); // ошибка (функция теперь является private)

ob1.set\_y(20);

// ob1.show\_x(); // ошибка (функция теперь является private)

ob1.show\_y();

system("pause");

return 0;

}

**Пример**: /\* *Модифицированный вариант предыдущей программы* \*/

#include <iostream>

using namespace std;

class B {

int x;

public:

void set\_x(int n) { x = n; }

int get\_x() { return x; }

void show\_x() { cout << "x=" << x << endl; }

};

// Класс наследуется как закрытый

class D : private B {

int y;

public:

// void set\_y(int n) { y = n; }

void set\_xy(int n, int m) { set\_x(n); y = m; }

void show\_xy() { show\_x(); cout << "y=" << y << endl; }

void mul() { cout << "mul=" << get\_x()\*y << endl; }

};

int main()

{

B ob;

D ob1;

ob.set\_x(5);

ob.show\_x();

ob1.set\_xy(15, 20);

ob1.show\_xy();

ob1.mul();

system("pause");

return 0;

}

Наследование и защищенные члены

Спецификатор ***protected*** повышает гибкость использования механизма

наследования. Если член класса объявлен защищенным (*protected*), то вне класса он недоступен. Этим он ничем не отличается от закрытого члена (*private*). Отличие появляется, когда мы используем механизм наследования.

Закрытый член базового класса не доступен другим элементам программы. Защищенный член класса по отношению к своему классу является закрытым и может наследоваться производным классом.

Полная основная форма объявления класса:

class *имя\_класса* {

// *закрытые члены*

protected:

// *защищенные члены*

public:

// *открытые члены*

};

**Пример**: // *Защищенные члены класса при наследовании с уровнем доступа* ***public***

#include <iostream>

using namespace std;

class B {

protected:

int x, y;

public:

void set\_xy(int n, int m) { x = n; y = m; }

};

class D : public B {

int z;

public:

void set\_z(int n) { z = n; }

// эта функция имеет доступ к переменным x, y класса base

void mul() {

cout << "mul=" << x \* y \* z << endl;

}

};

int main()

{

D ob;

ob.set\_xy(2, 3);

ob.set\_z(5);

ob.mul();

system("pause");

return 0;

}

**Конструкторы, деструкторы и наследование**

Базовый класс и производный класс могут содержать несколько конструкторов и деструкторов.

Если у базового и производного классов имеются конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы наоборот. То есть, при создании объекта производного класса конструктор базового класса выполняется раньше конструктора производного класса, а при уничтожении объекта производного класса деструктор производного класса раньше деструктора базового класса.

**Пример**: // *Конструкторы без параметров (очередность выполнения)*

#include <iostream>

using namespace std;

class B {

public:

B() { cout << " Constructor B" << endl; }

~B() { cout << " Destructor B" << endl; }

};

class D1 : protected B {

public:

D1() { cout << " Constructor D1" << endl; }

~D1() { cout << " Destructor D1" << endl; }

};

class D2 : protected D1 {

public:

D2() { cout << " Constructor D2" << endl; }

~D2() { cout << " Destructor D2" << endl; }

};

int main()

{

D2 ob;

system("pause");

return 0;

}

Результаты выполнения программы:

Constructor B

Constructor D1

Constructor D2

Destructor D2

Destructor D1

Destructor B

**Передача параметров конструктору при наследовании**

Когда инициализация проводится только в производном классе, используется стандартная синтаксическая форма конструктора с параметрами.

Если инициализация происходит и в базовом и в производном классах, то:

* необходимые аргументы базового и производного классов передаются конструктору производного класса;
* соответствующие аргументы передаются в базовый класс.

*Синтаксис передачи аргументов из производного класса в базовый класс*:

*конструктор\_произв\_класса(список\_арг) : имя\_базового\_класса(список\_арг) {*

*// тело конструктора производного класса*

*}*

**Пример**: // *Передача аргументов в конструктор производного класса*

#include <iostream>

using namespace std;

class B {

protected:

int i;

public:

B(int x) { i = x; cout << " Constuctor B" << endl; }

~B() { cout << " Destructor B" << endl; }

};

class D : public B {

int j;

public:

D(int x, int y) : B(y) {

j = x; cout << " Constuctor D" << endl;

cout << "i=" << i << " j=" << j << endl;

}

~D() { cout << " Destructor D" << endl; }

};

int main()

{

D ob(5, 10);

system("pause");

return 0;

}

Результаты выполнения программы:

Constructor B

Constructor D

i=10 j=5

Destructor D

Destructor B

Комментарий: В конструкторе производного класса используется одна переменная **x,** а переменная **y** передается конструктору базового класса. Аргумент, предназначенный для производного класса, идет в конструкторе первым.

Рассмотрим случай, когда в базовом классе есть конструктор с параметрами, а в производном нет.

**Пример**:

#include <iostream>

using namespace std;

class B {

protected:

int i;

public:

B(int x) { i = x; cout << " Constuctor B" << endl; }

~B() { cout << " Destructor B" << endl; }

};

class D : public B {

public:

D(int x) : B(x) {

cout << " Constuctor D" << endl;

cout << "i=" << i << endl;

}

~D() { cout << " Destructor D" << endl; }

};

int main()

{

D ob(5);

system("pause");

return 0;

}

Результаты выполнения программы:

Constructor B

Constructor D

i=5

Destructor D

Destructor B

Комментарий: В производном классе в конструкторе определяется аргумент, который будет передан конструктору базового класса.

**Множественное наследование**

Существует два способа наследования производным классом более одного базового класса:

* Производный класс может использоваться в качестве базового для другого производного класса, создавая многоуровневую иерархию классов. Тогда говорят, что исходный базовый класс является косвенным (indirect) базовым классом для второго производного класса (этот случай рассмотрен выше).
* Производный класс может прямо наследовать более одного базового класса.

**Пример**: // *Передача параметров в иерархически связанных классах (производный класс 2 -> производный класс 1 -> базовый класс)*

#include <iostream>

using namespace std;

class B {

int i;

public:

B(int x) {

i = x; cout << " Constuctor B" << endl;

cout << " i(B)=" << i << endl;

}

~B() { cout << " Destructor B" << endl; }

int get\_i() { return i; }

};

class D1 : public B {

int j;

public:

D1(int x, int y) :B(y) {

j = x; cout << " Constuctor D1" << endl;

cout << " j(D1)=" << j << endl;

}

~D1() { cout << " Destructor D1" << endl; }

int get\_j() { return j; }

};

class D2 : public D1 {

int k;

public:

D2(int x, int y, int z) :D1(y, z) { k = x; cout << " Constuctor D2" << endl; }

// void show(){cout<<"i="<<i<<" j="<<j<<" k="<<k<<endl;} // ошибка

void show() { cout << " i=" << get\_i() << " j=" << get\_j() << " k=" << k << endl; }

~D2() { cout << " Destructor D2" << endl; }

};

int main()

{

D2 ob(2, 4, 6);

ob.show();

system("pause");

return 0;

}

Результаты выполнения программы:

Constructor B

i(B)=6

Constructor D1

j(D1)=4

Constructor D2

i=6 j=4 k=2

Destructor D2

Destructor D1

Destructor B

Комментарий: В первой функции show() ошибка возникает из-за того, что к переменным i и j в производном классе D2 нет доступа. Но можно его получить, если использовать соответствующие открытые функции в базовом (B) и первом производном классе (D1).

Если производный класс напрямую наследует несколько базовых классов, то используется расширенное объявление:

class имя\_произв\_класса: СП\_доступа имя базового класса1,

СП\_доступа имя базового класса2,

………………………………………..

СП\_доступа имя базового классаN

{

// тело класса

}

Когда класс наследует несколько базовых классов, конструкторам которых необходимы аргументы, производный класс передает эти аргументы, используя расширенную форму объявления конструктора производного класса:

констр\_произв\_класса(список\_арг) : имя\_базового\_класса1(сп\_арг),

имя\_базового\_класса2(сп\_арг),

…………………………

имя\_базового\_классаN(сп\_арг)

{

// тело конструктора производного класса

}

**Пример**:

#include <iostream>

using namespace std;

class B1 {

protected:

int i;

public:

B1(int x) { i = x; cout << " Constuctor B1" << endl; }

~B1() { cout << " Destructor B1" << endl; }

};

class B2 {

protected:

int j;

public:

B2(int y) { j = y; cout << " Constuctor B2" << endl; }

~B2() { cout << " Destructor B2" << endl; }

};

class D : public B1, public B2 {

int k;

public:

D(int x, int y, int z) : B1(y), B2(z) { k = x; cout << " Constuctor D" << endl; }

~D() { cout << " Destructor D" << endl; }

void show() { cout << "i=" << i << " j=" << j << " k=" << k << endl; }

};

int main()

{

D ob(5, 10, 15);

ob.show();

system("pause");

return 0;

}

Результаты выполнения программы:

Constructor B1

Constructor B2

Constructor D

i=10 j=15 k=5

Destructor D

Destructor B2

Destructor B1

**Замечание**: В C++ есть возможность восстановить уровень доступа (public) к переменной в производном классе, определенной в базовом классе как public, когда он наследуется от базового класса как закрытый. Это достигается путем использования объявления уровня доступа в производном классе. (Использование этого способа считается нежелательным, так как он нарушает принцип инкапсуляции).

Синтаксис объявления уровня доступа:

базовый\_класс::член;

**Пример**:

#include <iostream>

using namespace std;

class B {

int i;

public:

int j, k;

void set\_i(int x) { i = x; }

int get\_i() { return i; }

};

class D : private B {

public:

B::j;

B::set\_i;

B::get\_i;

// B::i; // ошибка

};

int main()

{

D ob;

ob.j = 10;

//ob.k=15; // ошибка

ob.set\_i(5);

cout << ob.get\_i() << " " << ob.j << endl;

system("pause");

return 0;

}

**Виртуальные базовые классы**

Рассмотрим следующий вариант иерархии наследования классов:

Произв. 3

Произв. 2

Произв.1

Базовый

Базовый

В этой ситуации в классе "**Произв. 3**" имеется две копии класса "**Базовый**", поэтому существует неоднозначность.

Чтобы преодолеть эту неоднозначность, то есть включить тольку одну копию класса "**Базовый**", используют механизм включения одной копии.

Класс, поддерживающий этот механизм, называется виртуальным базовым классом (*virtual base class*).

Для того чтобы класс был виртуальным базовым классом, необходимо перед спецификатором доступа базового класса поставить ключевое слово ***virtual.***

**Синтаксис определения виртуального базового класса**:

class D1 : virtual public B { };

class D2 : virtual public B { };

Далее нигде не надо ставить virtual (копия base будет одна).

**Пример**:

#include <iostream>

using namespace std;

class B {

public:

int i;

};

class D1 : virtual public B {

public:

int j;

};

class D2 : virtual public B {

public:

int k;

};

class D3 : public D1, public D2 {

public:

int mul;

};

int main()

{

D3 ob;

ob.i = 5;

ob.j = 10;

ob.k = 15;

ob.mul = ob.i\*ob.j\*ob.k;

cout << "mul=" << ob.mul << endl;

system("pause");

return 0;

}