Пособие предназначено для студентов, изучающих язык C++. Является дополнением к существующему методическому пособию[1]. Может быть также полезно преподавателям, ведущим практические занятия по языку C++.

Виртуальные функции.

Виртуальной становится функция, объявленная в базовом классе с ключевым словом **virtual**.
Класс, содержащий виртуальную функцию, называется полиморфным.

Если в производном классе объявлена функция, профиль которой полностью совпадает с профилем виртуальной функции базового класса, говорят, что функция производного класса замещает функцию базового класса. Использование ключевого слова **virtual** в производном классе не обязательно, но рекомендуется для большей ясности программы.

Механизм виртуальных функций используется для того, чтобы иметь возможность для каждого из производных классов иметь собственный вариант некоторой функции базового класса. При обращении к виртуальной функции через указатель или ссылку на базовый класс определение того, какая именно функция должна быть вызвана, происходит динамически. Если указатель или ссылка настроены на объект производного класса, то происходит вызов функции из данного класса, т.е. вызов по типу объекта, в отличие от ситуации с невиртуальными функциями, когда вызов происходит по типу указателя.

Пример.
```
class B { . . .      public:   . . .   virtual void fun();  };
class D1 : public B  { . . .      public:   . . .   virtual void fun();  }; //  замещение функции из B
class D2 : public B { . . .      public:   . . .   void fun();  };// слова virtual нет, но замещение тоже есть

void work (B* b){ . . . b->fun();. . .} // функция может работать с объектами как базового,
                                                  так и любого из производных классов.
B b;   D1 d1;    D2 d2;
B* pb=&b; //  указатель базового класса  указывает на объект базового класса B
work(pb); //будет вызвана функция  B::fun
pb=&d1;    //  настроили указатель базового класса на объект производного класса D1
work(pb); //будет вызвана функция  D1::fun
pb=&d2;    //  настроили указатель базового класса на объект производного класса D2
work(pb); //будет вызвана функция  D2::fun
```

Чтобы правильно решить, какая функция будет вызываться, нужно вспомнить, что виртуальность – это динамическое связывание. При статическом связывании, т.е. для невиртуальных функций (точнее, тех функций, которые вызываются напрямую, без использования таблицы виртуальных функций) при компиляции сразу пишется конкретный вызов : **call _fun** (имя функции=адрес функции).

При динамическом связывании в коде пишется косвенный вызов через указатель на таблицу виртуальных функций:

call this->vptr[№ функции], соответственно вызываться будет функция в зависимости от типа объекта ***this**. Поскольку в коде был указатель на объект базового класса, вызваны таким образом могут быть только те функции, которые объявлены как виртуальные длябазового класса и указатели на них есть в таблице виртуальных функций данного класса.

Ниже приводятся таблицы, в которых рассмотрены основные случаи, вызывающие обычно сложности при изучении виртуальных функций.

Виртуальные функции. Примеры. Нормальная виртуальность.

	class B{ public:	class D: public B{	D d; B*pb=&d;// указатель на базовый класс указывает на объект произв. класса

1	virtual void f1();	<virtual> void f1();	pb->f1() ; // D::f1
			полное совпадение профиля функции

2	virtual void f2()=0;	<virtual> void f2();	pb->f2() ; // D::f2
			замещение чисто виртуальной функции

3	virtual B* f3();	virtual D* f3();	pb->f3() ; // D::f3
			допустимое различие типов возвращаемого значения: указатель на базовый – указатель на производный

4	virtual B& f4();	virtual D& f4();	pb->f4() ; // D::f4
			допустимое различие типов возвращаемого значения: ссылка на базовый – ссылка на производный

5	virtual ~B();	virtual ~D();	Деструкторы виртуальны, хотя их имена, разумеется, всегда различны. D* pd=new D; pb=pd; ….delete pb;// удаляет объект полностью, а не только «срез» базового класса.

Виртуальные функции. Примеры. Виртуальность и ее отсутствие.

	class B{ public:	class D: public B{	D d; B*pb=&d;// указатель на базовый класс указывает на объект произв. класса
1	virtual **void** f1();	virtual **int** f1();	Error! Различаются типы возвращаемых значений, совпадает только сигнатура функции, а для виртуальности необходимо полное совпадение профиля. Совпадение сигнатур не позволяет различать эти функции при перегрузке. Ошибка времени компиляции.

2	virtual **void** f2();	virtual **void** f2() **const**;	pb->f2(); //B::f2 функции различны по сигнатуре => нет виртуальности
			const B* pbc =&d;
			pbc->f2(); //Error! В классе B нет функции, которую можно вызвать для const-объекта.
			const D dc; pbc=&dc;
			pbc->f2(); //Error! (то же)

3	virtual **void** f3()const;	virtual **void** f3();	pb->f3(); //B::f3const функции различны по сигнатуре => нет виртуальности
			const B* pbc =&d;
			pbc->f3(); // B::f3const
			const D dc; pbc=&dc;
			pbc->f3(); //B::f3const
			(B* pb=&dc ; //Error! Типы указателей различны)

	class B{ public:	class D: public B{	D d; B*pb=&d;// указатель на базовый класс указывает на объект произв. класса
4	void f4();	virtual void f4();	pb->f4();// B::f4 нет виртуальности
5	virtual void f5();	Нет такой функции	pb->f5();// B::f5 нет виртуальности
6	Нет такой функции	virtual void f6();	pb->f6();// Error! нет такой функции в базовом классе, нет виртуальности

7	virtual void f7();	virtual void f7(**int**);	pb->f7(); // B::f7 - нет виртуальности
			pb->f7(1);// Error! в B нет f7(int)

8	virtual void f8(B*);	virtual void f8(D*);	pb->f8(&d) ; // B::f8 - нет виртуальности
			недопустимо никакое различие типов параметров, в том числе, указатель на базовый – указатель на производный

9	virtual void f9(B&);	virtual void f9(D&);	pb->f9(d) ; // B::f9 - нет виртуальности
			недопустимо никакое различие типов параметров, в том числе, ссылка на базовый – ссылка на производный

10	virtual
	void f10(double);
		virtual void f10(int);	pb->f10(1); // B::f10(double)
			pb->f10(1.5);// B::f10(double)
			нет виртуальности,
			в базовом классе ищется наилучшим образом подходящая функция

11	private: virtual void f11();	public: virtual void f11();	pb->f11();// Error! f11() – private в B
12	public: virtual void f12();	private: (protected:) virtual void f12();	pb->f12(); // D::f12 есть виртуальность, получили доступ к закрытой (защищенной) функции. Опасно!

Виртуальные функции. Виртуальность и значения параметров по умолчанию.

```
class A { int ax;
          public:     A (int n = 1) {ax = n; };
                      virtual int f1 ( int a = 5, int b = 5 ){ cout<< " A::f ";return 0;}
                      virtual int f2( int x = 10 ){ cout<< " A::f2 x= "<<x<<endl ; return 0; }
                      virtual int f3( int x){ cout<< " A::f3 x= "<<x<<endl ; return 0; }

         };
class C:  public A { int cx;
                     public:    C ( int n = 3 ){ cx = n; };
                       int f1 (int a, int b = 0){cout<< " C::f a="<<a<<" b="<<b<<endl;  return 0;} ;
                       int f2 (int x){cout<< " C::f2 x= "<<x<<endl ; return 0; }
                       int f3( int x = 10 ){ cout<< " C::f3 x= "<<x<<endl ; return 0; }
                      };
int main()
{
       A * p; C c;    //
       p=&c;          //    Здесь функции виртуальные и вызываются как виртуальные – по типу объекта
       p->f1();       //C::f1 a= 5 b=5     значения параметров по умолчанию, заданные в функции базового класса, передаются
                      //                   в функцию производного класса
       p->f1(1);      //C::f1 a= 1 b=5
       p->f1(2,2);    //C::f1 a= 2 b=2
       p->f2();       //C::f2 x= 10
       p->f2(3);      //C::f2 x= 3
       p->f3();       //ERROR! Функция  A::f3 не может быть вызвана без параметров
       p->f3(1);      // C::f3 x=1
}
```

Список литературы.

1. Волкова И. А., Иванов А. В., Карпов Л. Е. Основы объектно-ориентированного программирования. Язык программирования C++. Учебное пособие для студентов 2 курса. – М.: Издательский отдел факультета ВМК МГУ, 2011 – 112 с.
2. Standard for the C++ Programming Language ISO/IEC 14882, 1998.
3. Страуструп Б. Язык программирования C++. Специальное изд./Пер. с англ. - М.: "Бином", 2005.
4. Мейерс С. Эффективное использование C++. 35 новых рекомендаций по улучшению ваших программ и проектов: Пер. с англ. – М.:ДМК Пресс; Спб.: Питер, 2006.
5. Элджер Дж. C++: библиотека программиста – Спб.: Питер, 2001.