Тема 1. Введение в C++

Наследование

Рассмотрим здесь еще одно понятие, пропущенное в предыдущих лекциях.

Наследование — это способ конструирования новых объектов (классов), которые перенимают (наследуют) функциональность уже имеющихся объектов, т.е. могут выполнять те же действия, что и имеющийся класс (родитель), но, дополняя их, или, может быть, модифицируя под новые конкретные условия. Обычно применяются термины базовый класс и производный класс или родительский и дочерний (порожденный).

С наследования связаны два основных понятия: преемственность доступа к объектам родительского класса и виртуальные функции.

Новый тип видимости protected (в подолнение к public, private) — снаружи не виден, но можно использовать внутри класса.

```
class  A
{
   private:                 // не наследуются
       int ia;
   protected:               // наследуются protected, protected, private
       int ja;
   public:                  // наследуются public, protected, private
       int ka;
};
.....
A aa;
aa.ka  - видно
aa.ia , aa.ja  - не видно
.....

class AB : public A
{                       также включает в себя
    public:             int ja  как protected
        int mab;        int ka  как public
};
......
AB  ab;
ab.mab  - видно
ab.ka   - видно
ab.ja   - не видно
......

class AC : protected A
{                       также включает в себя
    public:             int ja  как protected
       int mac;         int ka  как protected
};
```

```
......
AC  ac;
ac.mac  - видно
ab.ka   - не видно
ab.ja   - не видно
......

class AD : private A
{                       также включает в себя
     public:            int ja  как private
         int mad;       int ka  как private
};
......
AD  ac;
ad.mad  - видно
ad.ka   - не видно
ad.ja   - не видно
......
```

Обычно используется public наследование. Наследование можно продолжать, т.е. порождать все новые и новые производные классы по цепочке.

Порожденный класс может использовать все унаследованные методы (функции) базового класса в соответвствии с полученными правами доступа.

Порожденный класс может определить свою функцию с той же сигнатурой, что и в базовом классе (т.е. переопределить функцию базового класса). При работе с этими классами будет вызываться своя функция данного класса.

Виртуальная функция — объявление virtual. Пока что это обычный член класса.

Порожденный класс может переопределить виртуальную функцию базового класса (с сохранением типа возвращаемого значения).

Различие между просто переопределенными функциями и виртуальными переопределенными функциями возникает при доступе к функциям через указатель или ссылку на класс.

Вообще-то в C++ указатели автоматически не преобразуются в другие типы. Но, указатель на базовый класс может принять значение указателя на порожденный класс.

Самое главное правило наследования !!!

Если функция вызывается по указателю, то

конкретная переопределенная функция соответствует типу указателя (т.е. то, как он был объявлен)

конкретная виртуальная функция соответствует типу конкретного объекта за этим указателем (т.е. на что указывают)

Примеры.

```
class A
{  public:
     void f(int) { printf("A:f()\n"); }
     virtual void g(double) { printf("A:g()\n"); }
};
```

```
class AB : public A
{  public:
     void f(int) { printf("AB:f()\n"); }
     void g(double) { printf("AB:g()\n"); }
};

int main()
{
   A aa;
   AB ab;

   aa.f(0);  // обрабатывается при компиляции
   aa.g(0);  // обрабатывается при компиляции
   ab.f(0);  // обрабатывается при компиляции
   ab.g(0);  // обрабатывается при компиляции

   A &ra = ab;
   ra.f(0);  // обрабатывается при компиляции
   ra.g(0);  // обрабатывается при выполнении

   A *pa = &ab;
   pa->f(0);  // обрабатывается при компиляции
   pa->g(0);  // обрабатывается при выполнении

   return 0;
}
```

Чисто виртуальные функции.

```
virtual int myfun() = 0;
```

Необходимо определить в последующем порожденном классе.

Виртуальный класс — содержит чисто виртуальные функции.

Преобразование типов указателей в C++

```
C-style   (тип)объект или тип(объект) - не проверяется компилятором !!!

static_cast<тип>(объект)  - проверяется на этапе компиляции
                            (например, const указатель)
   int i = 200; (a если 300 ?)
   char c = i;   - предупреждение !
   char d = static_cast<char>(i);  - понимаем, что делаем !

dynamic_cast<тип>(объект) - приведение к другому типу
                  с проверкой возможности (иначе вернет NULL)
    обычно используется для приведения указателя с базового типа
```

```
    на производный (когда он реально указывает на производный
    и нужна специфика именно производного класса)

const_cast<тип>(объект) - снимает спецификацию const (надо понимать зачем !!!)

reinterpret_cast<тип>(объект) - почти произвольное преобразование
                                (кроме dynamic и const)
```