ЛАБОРАТОРНАЯ РАБОТА №6
НАСЛЕДОВАНИЕ

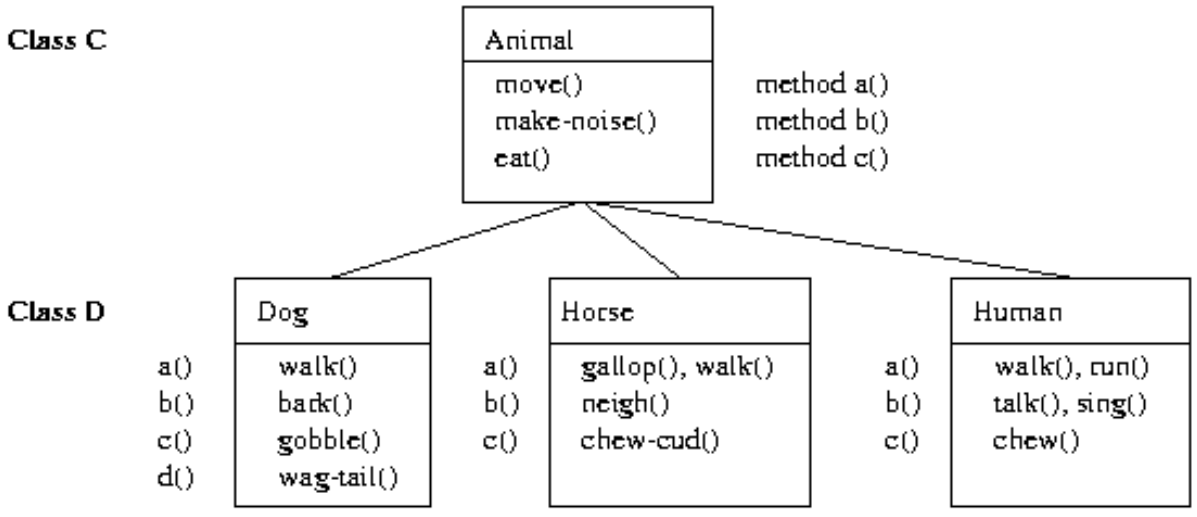
ЦЕЛЬ ЛАБОРАТОРНОЙ РАБОТЫ:

Целью данной лабораторной работы является изучение наследованная классов в языке C++.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ:

Наследование — это механизм создания нового класса на основе уже существующего. При этом к существующему классу могут быть добавлены новые элементы (данные и функции), либо существующие функции могут быть изменены. Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.



Иерархия классов позволяет определять новые классы на основе уже имеющихся. Имеющиеся классы обычно называют базовыми(иногда порождающими), а новые классы, формируемые на основе базовых, – производными (порожденными, классами-потомками или наследниками).

Производные классы «получают наследство» – данные и методы своих базовых классов, и могут пополняться собственными компонентами (данными и собственными методами). Наследуемые компоненты не перемещаются в производный класс, а остаются в базовых классах. Сообщение, обработку которого не могут выполнить методы производного класса, автоматически передается в базовый класс. Если для обработки сообщения нужны данные, отсутствующие в производном классе, то их пытаются отыскать автоматически в базовом классе.

При наследовании некоторые имена методов (функций-членов) и данных-членов базового класса могут быть по-новому определены в производном классе. В этом случае соответствующие компоненты базового класса становятся недоступными из производного класса. Для доступа из производного класса к компонентам базового класса, имена которых повторно определены в производном, используется операция разрешения контекста ::

Для порождения нового класса на основе существующего используется следующая общая форма

```
class Имя : МодификаторДоступа  ИмяБазовогоКласса
{ объявление_членов;};
```

При объявлении порождаемого класса МодификаторДоступа может принимать значения public, private, protected либо отсутствовать, по умолчанию используется значение private. В любом случае порожденный класс наследует все члены базового класса, но доступ имеет не ко всем. Ему доступны общие (public) члены базового класса и недоступны частные (private).

Для того, чтобы порожденный класс имел доступ к некоторым скрытым членам базового класса, в базовом классе их необходимо объявить со спецификацией доступа защищенные (protected). Члены

класса с доступом protected видимы в пределах класса и в любом классе, порожденном из этого класса.

1. Public - наследование

При общем наследовании порожденный класс имеет доступ к наследуемым членам базового класса с видимостью public и protected. Члены базового класса с видимостью private – недоступны.

Спецификация доступа	внутри класса	в порожденном классе	вне класса
private	+	-	-
protected	+	+	-
public	+	+	+

Общее наследование означает, что порожденный класс – это подтип базового класса. Таким образом, порожденный класс представляет собой модификацию базового класса, которая наследует общие и защищенные члены базового класса.
Пример:

```
1) class student {
2) protected:
3)   char fac[20];
4)   char spec[30];
5)   char name[15];
6) public:
7)   student(char *f, char *s, char *n);
8)   void print();};
9) class grad_student : public student {
10) protected:
11)   int year;
12)   char work[30];
13) public:
14)   grad_student(char *f, char *s, char *n, char *w,
   int y);
15)   void print(); };
```

Порожденный класс наследует все данные класса student, имеет доступ к protected и public-членам базового класса. В новом классе добавлено два член-данных, и порожденный класс переопределяет функцию print().

```
1) student :: student(char *f, char *s, char *n) {
2)   strcpy(fac, f);
3)   strcpy(spec, s);
4)   strcpy(name, n);
5) }
6) grad_student :: grad_student(char *f, char *s, char
   *n, char *w, int y) :
7)     student(f,s,n) {
8)   year = y;
9)   strcpy(work, w);
10) }
```

Конструктор для базового класса вызывается в списке инициализации.

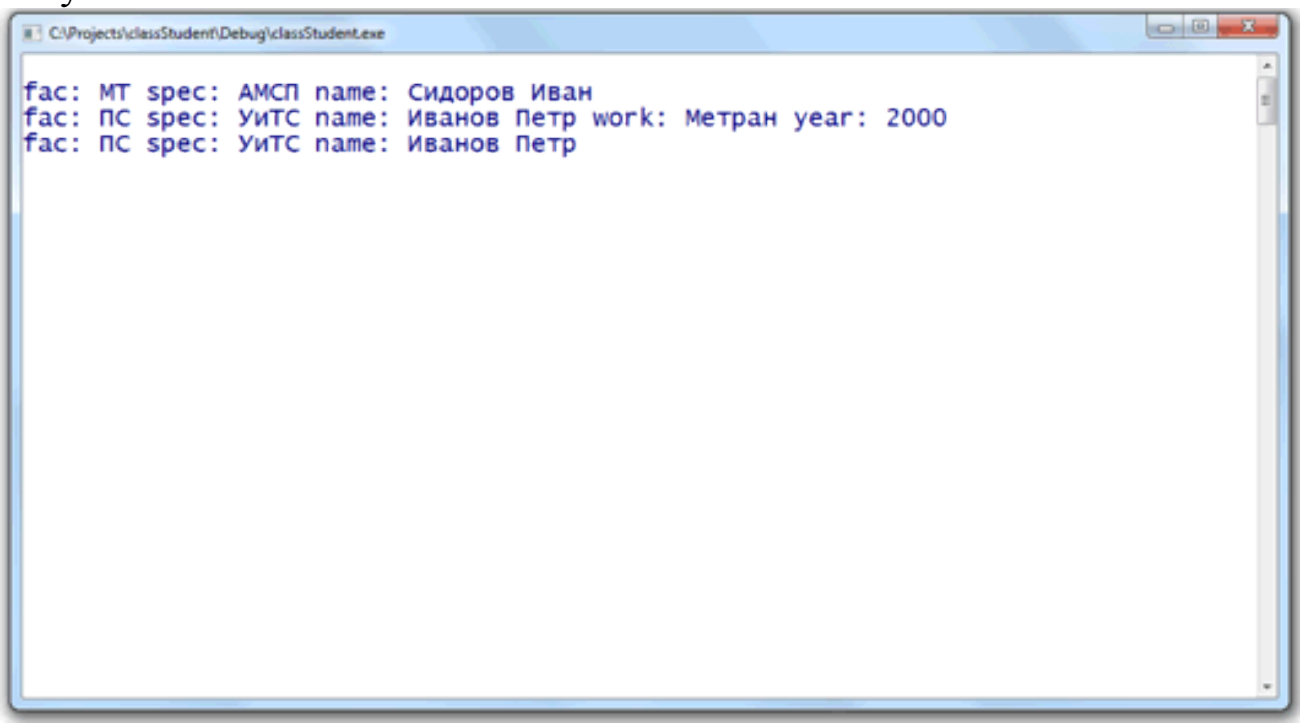
Перегрузка функции print().

```
1) int main() {
2)   system("chcp 1251");
3)   system("cls");
4)   student s("МТ", "АМСП", "Сидоров Иван");
5)   grad_student stud("ПС", "УиТС", "Иванов Петр",
   "Метран", 2000);
6) student *p = &s;
7)   p->print();
```

```
1) void student :: print() {
2)   cout << endl << "fac: " << fac << " spec: " << spec
3) << " name: " << name;
4) }
5) void grad_student :: print() {
6)   student :: print();
7)   cout << " work: " << work << " year: " << year;
8) }
```

```
8)   grad_student *gs = &stud;
9)   student *m;
10)    gs->print();
11)    m = gs;
12)    m->print();
13)    cin.get();
14)    return 0;
15)  }
```

Результат выполнения:



Указатель на порожденный класс может быть неявно передан в указатель на базовый класс. При этом переменная-указатель m на базовый класс может указывать на объекты как базового, так и порожденного класса.

Указатель на порожденный класс может указывать только на объекты порожденного класса.

Неявные преобразования между порожденным и базовым классами называются предопределенными стандартными преобразованиями:

• объект порожденного класса неявно преобразуется к объекту базового класса.

• ссылка на порожденный класс неявно преобразуется к ссылке на базовый класс.

• указатель на порожденный класс неявно преобразуется к указателю на базовый класс.

2. Private - наследование

Порожденный класс может быть базовым для следующего порождения. При порождении private наследуемые члены базового класса, объявленные как protected и public, становятся членами порожденного класса с видимостью private. При этом члены базового класса с видимостью public и protected становятся недоступными для дальнейших порождений. Цель такого порождения — скрыть классы или элементы классов от использования в дальнейших порождениях. При порождении private не выполняются предопределенные стандартные преобразования:

```
class grad_student : private student
{...};
int main() {...
  grad_student *gs = &stud;
  student *m;
  gs->print();
  m = gs;        // ошибка
  m->print();
  cin.get();
  return 0;
}
```

Однако порождение private позволяет отдельным элементам базового класса с видимостью public и protected сохранить свою видимость в порожденном классе. Для этого необходимо:

• в части protected порожденного класса указать те наследуемые члены базового класса с видимостью protected, уточненные именем базового класса, для которых необходимо оставить видимость protected и в порожденном классе;

• в части public порожденного класса указать те наследуемые члены базового класса с видимостью public, уточненные именем базового класса, для которых необходимо оставить видимость public и в порожденном классе.

```
class X {
private:
  int n;
protected:
  int m;
  char s;
public:
  void func(int);
};
class Y : private X {
private:
  ...
protected:
  ...
  X :: s;
public:
  ...
  X :: func();
};
```

3. Protected - наследование

Возможен и третий вариант наследования – с использованием модификатора доступа protected.

Доступ к элементам базового класса из производного класса, в зависимости от модификатора наследования:

Модификатор наследования → / Модификатор доступа ↓	public	protected	private
public	public	protected	private
protected	protected	protected	private
private	нет доступа	нет доступа	нет доступа

4. Конструкторы и деструкторы при наследовании

Как базовый, так и производный классы могут иметь конструкторы и деструкторы.

Если и у базового и у производного классов есть конструкторы и деструкторы, то конструкторы выполняются в порядке наследования, а деструкторы – в обратном порядке. То есть если A – базовый класс, B – производный из A, а C – производный из B (A-B-C), то при создании объекта класса C вызов конструкторов будет иметь следующий порядок:

- конструктор класса A
- конструктор класса B
- конструктор класса C.

Вызов деструкторов при удалении этого объекта произойдет в обратном порядке:

- деструктор класса C
- деструктор класса B
- деструктор класса A.

Поскольку базовый класс "не знает" о существовании производного класса, любая инициализация выполняется в нем независимо от производного класса, и, возможно, становится основой для инициализации, выполняемой в производном классе. Поскольку базовый класс лежит в основе производного, вызов деструктора базового класса раньше деструктора производного класса привел бы к преждевременному разрушению производного класса.

Конструкторы могут иметь параметры. При реализации наследования допускается передача параметров для конструкторов производного и базового класса. Если параметрами обладает только конструктор производного класса, то аргументы передаются обычным способом. При необходимости передать аргумент из производного класса конструктору родительского класса используется расширенная запись конструктора производного класса:

```
КонструкторПроизводногоКласса(СписокФормальныхАргументов
)
: КонструкторБазовогоКласса(СписокФактическихАргументов)
{ // тело конструктора производного класса }
```

Для базового и производного классов допустимо использование одних и тех же аргументов. Возможно, списки аргументов конструкторов производного и базового классов будут различны.

Конструктор производного класса не должен использовать все аргументы, часть предназначены для передачи в базовый класс:

```
1) student :: student(char *f, char *s, char *n) {
2)   strcpy(fac, f);
3)   strcpy(spec, s);
4)   strcpy(name, n);
5) }
6) grad_student :: grad_student(char *f, char *s, char
   *n, char *w, int y) :
7)      student(f,s,n) {
8)   year = y;
9)   strcpy(work, w);
10)  }
```

В расширенной форме объявления конструктора производного класса описывается вызов конструктора базового класса.

ЗАДАНИЕ

1) Создать класс «Староста», производный от класса «Студент». Новый класс должен содержать несколько дополнительных методов и полей.

2) Создать класс Alive и расширить его до Bird, Fish, Animal

3) Создать класс Animal, и расширить его до Dog, Cat.