В.Л. Тарасов

Лекции по программированию на C++

Лекция 15

Наследование

На основе ранее разработанных классов можно создавать новые классы. Это можно делать либо *включая* объекты одного класса в состав другого класса, либо делая новый класс *производным* от другого. Вначале рассмотрим включение объектов одного класса в состав другого.

15.1. Объекты как члены класса

Пусть есть класс:

```
class Member{
    int a;
public:
    Member(int i)    // Конструктор класса Member
    { a = i; }
};
```

Объявим новый класс `Container`, который будет включать в качестве своего члена объект класса `Member`:

```
class Container{     // Класс, содержащий объект другого класса
    Member aa;       // Объект aa класса Member – член класса Container
    double x;
public:
    Container(int i, double xx);      // Конструктор
};
```

При создании объекта класса `Container` должен создаваться и объект `aa` класса `Member`, для чего должен вызываться соответствующий конструктор. Это реализуется с помощью специальной синтаксической конструкции:

```
Container::Container(int i, double xx) : aa(i)     // Конструктор
{ x = xx; }                                        // класса Container
```

После заголовка конструктора ставится двоеточие, за которым идет *имя объекта* и в скобках аргументы конструктора объекта, то есть `aa(i)`.

Далее идет реализация остальной части конструктора класса, содержащего объект.

15.2. Конструкторы встроенных типов

В языке C++ встроенные типы (int, double,…) можно рассматривать как классы, имеющие конструкторы, поэтому конструктор для Member можно реализовать в виде:

```
Member(int i) : a(i) { }
```

Здесь запись a(i) означает вызов конструктора встроенного типа int для создания объекта (переменной) a. Конструктор типа int присвоит переменной a значение i.

Аналогично, конструктор класса Container можно записать в виде:

```
Container::Container(int i, double xx) : aa(i), x(xx) { }
```

Запись x(xx) означает вызов конструктора типа double для создания переменной (объекта) x, который присвоит члену x значение xx.

Конструкторы по умолчанию встроенных типов присваивают переменным *нулевое* значение.

Программа 15.1. Объекты как члены класса

В программе в классы Member и Container добавлены конструкторы по умолчанию и перегруженные операторы для вывода.

```
#include <iostream>
using namespace std;

class Member{
    int a;
public:
    Member(int i)        // Конструктор класса Member
    { a = i; }

    Member() : a()       // Конструктор по умолчанию класса Member
    { }                  // a присваивается 0

    friend ostream& operator<<(ostream& os, const Member& mem)
    {
        return os << mem.a;
    }
};

class Container{         // Класс, содержащий объект другого класса
    Member aa;           // Объект aa класса Member – член класса Container
    double x;
public:
```

```
    // Конструктор
    Container(int i, double xx) : aa(i), x(xx)    // aa(i) – это вызов
    { }                                           // конструктора для создания aa

    // Конструктор по умолчанию
    Container() : aa(), x()
    { }                                           // aa.a = 0, x = 0.0

    friend ostream& operator<<(ostream& os, const Container& con)
    {
        return os << con.aa << " " << con.x;
    }
};

int main()
{
    Member mem0;             // mem0 создается конструктором по умолчанию
    cout << "mem0 = " << mem0 << endl;
    Member mem1(1);
    cout << "mem1 = " << mem1 << endl;
    Container con0;          // con0 создается конструктором по умолчанию
    cout << "con0 = " << con0 << endl;
    Container con1(2, 3.14159);
    cout << "con1 = " << con1 << endl;
    return 0;
}
```

Программы выводит:

```
mem0 = 0
mem1 = 1
con0 = 0 0
con1 = 2 3.14159
```

Пример показывает, что действительно, конструкторы по умолчанию встроенных типов присваивают переменным нули.

Для вывода значений классов `Member` и `Container` перегружен оператор вывода (<<). В реализации оператора вывода для класса `Container` выражение `os << con.aa` приводит к выводу функции оператора `ostream& operator<<(ostream&os, const Member& mem)`, выводящей значения класса `Member`.

Программа 15.2. Личные данные

Напишем программу, для работы с массивами персональных данных. Эти сведения должны включать фамилию человека и его дату рождения.

Для работы с датами напишем класс `Date`, включив в него конструктор копирования и оператор сравнения дат. Объявление класса разместим в файле `ClassDate.h`.

```
// Файл ClassDate.h

#ifndef ClassDateH
#define ClassDateH

#include <iostream>
#include <fstream>
#include <ctime>
#include <string>
#include <vector>
#include <iomanip>
using namespace std;

class Date{                                 // Класс для работы с датами
    int d, m, y;                            // День, месяц, год
public:

    Date(int = 0, int = 0, int = 0);        // Конструктор

    Date(const Date& D)                     // Конструктор копирования
    {
        d = D.d; m = D.m; y = D.y;
    }

    Date& operator=(const Date& D)          // Оператор присваивания
    {
        d = D.d; m = D.m; y = D.y;
        return *this;
    }

    bool operator<=(const Date& D) const  // Сравнение двух дат
    {
        return (y < D.y) || (y == D.y && m < D.m) ||
               (y == D.y && m == D.m && d <= D.d);
    }

    friend ostream& operator<<(ostream& os, const Date& dt)// Вывод даты
    {
        return cout << setw(2) << setfill('0') << dt.d << '.'
                    << setw(2) << setfill('0') << dt.m << '.' << dt.y;
    }
};
#endif
```

Слово const в заголовке функции operator<=() const означает, что она не изменяет объект класса для которого вызывается.

Реализацию класса Date разместим в файле ClassDate.CPP.

```
// Файл ClassDate.cpp

#include "ClassDate.h"

// Конструктор класса Date

Date::Date(int dd, int mm, int yy)    // Значения аргументов по умолчанию
{                                     // второй раз не указываются
```

```
    time_t seconds;                  // Переменная для секунд
    time(&seconds);                  // Число секунд от 01.01.1970
    struct  tm*  ptm_time;           // Указатель на структуру tm
    ptm_time = localtime(&seconds);  // Системная дата

// Если аргументы конструктора не заданы, берем системную дату
    d = dd ? dd : ptm_time->tm_mday;          // Если день не задан,
                                              // берется текущий день
    m = mm ? mm : ptm_time->tm_mon + 1;       // Если месяц не задан,
                                              // берется текущий месяц
    y = yy ? yy : ptm_time->tm_year + 1900;   // Если год не задан,
                                              // берется текущий год
}
```

Для работы со сведениями об одном человеке напишем класс Pers.

```
// Файл Pers.h

#ifndef PersH
#define PersH

#include "ClassDate.h"               // Подключение класса Date

class Pers                           // Класс со сведениями о человеке
{
    string name;                     // Строка с фамилией и инициалами
    Date bd;                         // Дата рождения
public:
    // Конструктор
    Pers(const string& nm, const Date& D) : bd(D), name(nm)
    { }

    Pers() : bd(), name()            // Конструктор по умолчанию
    { }

    Pers(const Pers& ps)             // Конструктор копирования
    {
        name = ps.name;              // Копирование имени
        bd = ps.bd;                  // Копирование даты
    }

    Pers& operator=(const Pers &);   // Оператор присваивания

    // Оператор вывода сведений о человеке
    friend ostream& operator<<(ostream& os, const Pers& ps)
    {
        return os << ps.name << "     \t" << ps.bd;
    }

    // Оператор ввода сведений о человеке
    friend istream& operator>>(istream& is, Pers& ps);

    // CmpDate: возвращает true, если дата объекта ps1
    // более ранняя или та же, что и у объекта ps2
    friend bool CmpDate(const Pers& ps1, const Pers& ps2)
    {                                  // Используется
        return ps1.bd <= ps2.bd;       // функция-оператор сравнения дат
```

```
    }

    // CmpName: сравнение фамилий
    friend bool CmpName(const Pers& ps1, const Pers& ps2)
    {
        return ps1.name <= ps2.name;
    }
};

#endif
```

В конструкторе по умолчанию класса Pers:

```
Pers :: Pers() : bd(), name() { }
```

инструкция bd() вызывает конструктор по умолчанию класса Date, который заносит в bd текущую дату. Инструкция name() вызывает конструктор по умолчанию класса string, который присваивает name пустую строку.

В операторе вывода объекта класса Pers в инструкции

```
return os << ps.name << "      \t" << ps.bd;
```

после имени ps.name выводится строка из нескольких пробелов и символа табуляции. Число пробелов подобрано так, чтобы последующий вывод даты ps.bd начинался с одной и той же позиции.

Теперь приведем файл Pers.cpp, содержащий оставшейся части класса Pers.

```
// Файл Pers.cpp

#include "Pers.h"

Pers& Pers::operator=(const Pers& ps) // Оператор присваивания
{
    if(this != &ps){              // Если присваивание не самому себе,
        name = ps.name;               // копирование имени,
        bd = ps.bd;                   // копирование даты
    }
    return *this;                     // Возвращение ссылки на объект
}

// Ввод данных об одном человеке
istream& operator>>(istream& is, Pers& ps)
{
    is >> ps.bd;                      // Ввод даты рождения
    getline(is, ps.name);             // Ввод фамилии и имени (инициалов)

// Поиск позиции первого непробельного символа в строке
    int indexNotSpaceTab = ps.name.find_first_not_of(" \t");

// Удаление начальных пробелов
    ps.name = ps.name.substr(indexNotSpaceTab, ps.name.length());
    return is;
```

```
}
```

Для работы со сведениями о группе лиц разработаем класс Persons.

```
// Файл Persons.h

#ifndef PersonsH
#define PersonsH

#include "Pers.h"                          // Подключение класса Pers

class Persons                              // Класс для работы с группой лиц
{
    vector<Pers> G;                        // Вектор сведений о людях
public:
    void Add(const Pers& pd)               // Добавление человека в группу
    {                                      // нового человека
        G.push_back(pd);
    }

// Сортировка по критерию, задаваемому функцией сравнения Compare()
    void SortPersons ( bool (*Compare)(const Pers&, const Pers&) );

// Вывод сведений о группе лиц
    friend ostream& operator<<(ostream&, const Persons&);

// Ввод сведений о группе лиц
    friend istream& operator>>(istream&, Persons&);
private:
    void Swap(int i, int j); // Поменять местами элементы i и j массива G
};

#endif
```

Теперь приведем файл Persons.cpp, содержащий оставшуюся часть реализации класса Persons.

```
// Файл Persons.cpp

#include “Persons.h”

void Persons :: Swap(int i, int j)    // Переставить элементы i и j массива G
{
    Pers tmp = G[i];
    G[i] = G[j];
    G[j] = tmp;
}

// Реализация метода сортировки
// Compare – указатель на функцию сравнения сведений о двух людях
void Persons::SortPersons(bool (*Compare)(const Pers&, const Pers&))
{
    for(int i = G.size() - 1; i > 0; i--)
        for(int j = 0; j < i; j++)
            if(!Compare(G[j], G[j + 1]))      // Если не тот порядок,
                Swap(j, j + 1);               // переставить элементы
}
```

```
// Реализация функций-друзей класса Persons

// Вывод списка группы
ostream& operator<<(ostream& os, const Persons& group)
{
    for(int i = 0; i < group.G.size(); i++)   // Вывод данных
        os << group.G[i];                     // о каждом человеке
    return os;
}

// Чтение данных о группе из файла
istream& operator>>(istream& is, Persons& group)
{
    Pers tmp;                             // Сведения о человеке
    while(!is.eof()){
        is >> tmp;                        // Чтение сведений о человеке
        if(is.eof())                      // Если достигнут конец файла,
            break;                    // выход из цикла
        group.Add(tmp);                   // Добавление сведений в группу
    }
    return is;
}
```

Аргументом функции сортировки

```
Void Persons::SortPersons(bool (*Compare)(const Pers&, const Pers&))
```

является указатель Compare на функцию, которая сравнивает личные сведения двух людей.

В главной программе создадим один объект класса Persons (одну группу). Заполним объект сведениями, прочитанными из файла PersData.txt, распечатаем эти сведения, отсортируем сначала по возрастанию даты рождения, затем - в алфавитном порядке. Все это – в файле MainPers.cpp.

```
// Файл MainPers.cpp

#include "Persons.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    Persons Writers;                      // Writers – объект класса Persons
    ifstream inf("PersData.txt");         // Входной файловый поток
    if(!inf){                             // Если файл не удалось открыть,
        cerr << "Не удалось открыть файл PersData.txt ";
        exit(1);                          // завершаем программу
    }
    inf >> Writers;                       // Чтение из файла
    cout << "Состав группы: \n";
    cout << Writers << endl;              // Вывод состава группы

    // Сортировка по возрастанию даты рождения
```

```
    Writers.SortPersons(CmpDate);
    cout << "Состав группы по возрастанию даты: \n";
    cout << Writers << endl;

    // Сортировка в алфавитном порядке
    Writers.SortPersons(CmpName);
    cout << "Состав группы по алфавиту: \n";
    cout << Writers << endl;
    system("pause");
    return 0;
}
```

Инструкция

```
ifstream inf("PersData.txt");
```

создает входной файловый поток с вызовом конструктора класса `ifstream`, которому передается строка с именем открываемого файла.

Текстовый файл исходных данных можно создать любым редактором, в частности редактором среды разработки, с помощью которого создаются исходные тексты программ. Подготовим в файле `PersData.txt` следующие исходные данные:

```
 9  9 1828   Толстой Л.Н.
 6  6 1799   Пушкин А.С.
15 10 1814   Лермонтов М.Ю.
13  2 1769   Крылов И.А.
28 10 1818   Тургенев И.С.
19 11 1711   Ломоносов М.В.
28  3 1868   Горький А.М.
11 11 1821   Достоевский Ф.М.
24  5 1905   Шолохов М.А.
```

Здесь сначала указаны день, месяц и год рождения человека, далее - фамилия и инициалы. На такую структуру файла исходных данных настроена функция `ReadFromFile()`.

Программа выдает следующее:

```
Состав группы:
Толстой Л.Н.            09.09.1828
Пушкин А.С.             06.06.1799
Лермонтов М.Ю.          15.10.1814
Крылов И.А.             13.02.1769
Тургенев И.С.           28.10.1818
Ломоносов М.В.          19.11.1711
Горький А.М.            28.03.1868
Достоевский Ф.М.        11.11.1821
Шолохов М.А.            24.05.1905

Состав группы по возрастанию даты:
Ломоносов М.В.          19.11.1711
Крылов И.А.             13.02.1769
Пушкин А.С.             06.06.1799
```

```
Лермонтов М.Ю.          15.10.1814
Тургенев И.С.           28.10.1818
Достоевский Ф.М.        11.11.1821
Толстой Л.Н.            09.09.1828
Горький А.М.            28.03.1868
Шолохов М.А.            24.05.1905

Состав группы по алфавиту:
Горький А.М.            28.03.1868
Достоевский Ф.М.        11.11.1821
Крылов И.А.             13.02.1769
Лермонтов М.Ю.          15.10.1814
Ломоносов М.В.          19.11.1711
Пушкин А.С.             06.06.1799
Толстой Л.Н.            09.09.1828
Тургенев И.С.           28.10.1818
Шолохов М.А.            24.05.1905
```

15.3. Наследование классов

Классы могут находиться в отношении *наследования*. Исходные классы называются *базовыми* или *классами – предками*. Новые классы, образуемые на основе базовых классов, называются *производными* или *классами – потомками*. Производные классы наследуют все компоненты предков, то есть все члены базового класса становятся членами производного класса.

Любой производный класс может стать в свою очередь базовым для другого класса, в результате чего может быть создана *иерархия* классов.

Определение производного класса `S` имеет вид:

```
class S : X, Y, Z
{
 … // Собственные компоненты класса S
};
```

Здесь `X, Y, Z` – базовые классы для класса `S`.

Использование сразу нескольких базовых классов при создании одного производного класса называется *множественным наследованием*.

Хотя при наследовании все члены базового класса становятся членами производного класса, закрытые члены базового класса *не становятся доступными* для функций производного класса. В следующей программе приведен простейший случай наследования.

Программа 15.3. Наследование

Пусть объявлен класс `Base`:

```
// Файл Base.h

#include <iostream>
using namespace std;
class Base{
   int a;                                        // Закрытый член
public:
   void set_a ( int aa ) { a = aa; }             // Установка значения
   void Print()
   {
        cout << "Base::" << a;
   }
   int get_a( )
   {
      return a;
   }
};
```

На основе класса Base объявим производный класс Derived:

```
// Файл Derived.h

#include "Base.h"

class Derived : public Base{ // public говорит, что открытые члены Base
   double x;             // остаются открытыми, становясь членами Derived
public:
   void set_x(double xx)
   {
      x = xx;
   }

   void Print( )
   {
      cout << "Derive::" << x;
   }

   double get_x( )
   {
      return x;
   }

   bool IntMoreDbl( )         // Сравнение, больше ли a чем x
   {                          // Функции производного класса имеют доступ
      return get_a() > x;     // только к открытым членам базового класса
   }
};
```

В базовом и производном классах имеется функции с одним и тем же именем Print(). Конфликта имен не возникает, так как зоной действия имен членов класса является их класс. В состав класса Derived входит не только собственная переменная x типа double, но и переменная целого типа a, переходящая к нему из базового класса Base, но в функции производного класса IntMoreDbl() нельзя записать:

```
return a > x;
```

так как a – это *закрытая* переменная класса Base, *недоступная* функциям производного класса.

```
// Файл BaseDer.cpp

#include "derived.h"
int main()
{
    Base b;                      // Переменная (объект) базового класса
    b.set_a(1);                  // Установка значения члена a объекта b
    cout << "b = "; b.Print();   // Вызов функции Base::Print()
    Derived d;                   // Переменная производного класса
    d.set_a(2);                  // Установка целой части вызовом функции
                                 // базового класса
    d.set_x(3.0);                // Установка вещественной части
                                 // функцией производного класса
    cout <<  "\nd = ";
    d.Print();                   // Вызов функции Derived::Print()
    cout << ", ";
    d.Base::Print();             // Для вызова функции базового класса здесь
                                 // требуется явное указание класса функции
    cout << endl;
    if(d.IntMoreDbl())
        cout << "d::Base > d::Derived";
    else
        cout << "d::Base < d::Derived";
    cin.get();
    return 0;
}
```

Программа выводит:

```
b = Base::1
d = Derive::3, Base::2
d::Base < d::Derived
```

В Visual Studio есть возможность построения схемы классов, дающей графическую иллюстрацию взаимоотношений между классами. Для этого надо выделить проект и выполнить команду контекстного меню **Перейти к схеме классов** (рис.15.1).

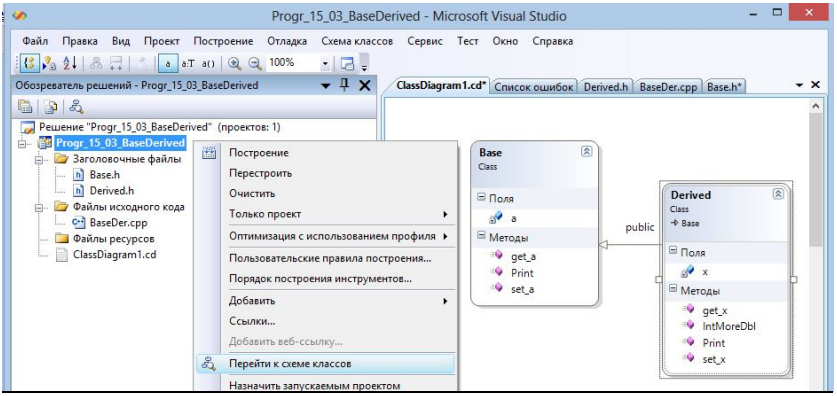

Рис. 15.1. Схема классов

Принято отношение наследования изображать стрелкой, направленной от производного класса к базовому.

15.4. Управление доступом

Члены класса могут быть закрытыми (`private`), открытыми (`public`) и *защищенными* (`protected`). Защищенные члены (`protected`) являются закрытыми, но доступными в классах, *производных* от данного класса.

Структуры и классы

По определению, структура есть класс, все члены которого по умолчанию являются открытыми, то есть объявление структуры:

```
struct S{
...
};
```

эквивалентно объявлению класса, все члены которого открыты:

```
class S{
public:
...
};
```

После ключевого слова `public` перечисляются открытые члены.

С помощью ключевого слова `private` можно закрыть часть членов структуры:

```
struct S{
private:
```

```
// Закрытые члены структуры
…
public:
// Открытые члены структуры
…
};
```

Разделов public и private может быть несколько, они могут располагаться в любой последовательности.

Управление доступом при наследовании

Закрытые члены класса, объявленные в его разделе private, доступны только функциям-членам класса и друзьям класса.

Закрытые члены класса можно сделать доступными в производных классах, объявив их *защищёнными*, то есть, поместив их в раздел protected. Изменим объявление класса Base из программы 15.3:

```
class Base{
protected:       // Закрытые члены, доступные в производных классах
   int a;
public:
…
};
```

Теперь в функции IntMoreDbl() класса Derived можно непосредственно обратиться к защищенному члену a базового класса:

```
bool IntMoreDbl() { return a > x; }
```

Можно изменять статус доступа членов базового класса через производный класс, указывая спецификатор доступа public, protected или private перед именем базового класса. Поясним это примером. Пусть имеется класс B, который по-разному наследуется другими классами.

```
class B{
   private:     float x_pri;     // Закрытый член   (private)
   protected:   int   i_pro;     // Защищенный член (protected)
   public:      char  c_pub;     // Открытый член   (public)
};

class PUB :  public B{       // x_pri наследуются как private
...                          // i_pro наследуются как protected
};                           // c_pub наследуются как public
```

Методы и друзья класса PUB могут обращаться к i_pro и c_pub. Внешние функции могут обращаться только к c_pub. Например,

```
void f1(PUB b)                  // Внешняя функция
{
   char c1 = b.c_pub;           // Можно, B::c_pub доступна через класс PUB
   int k = b.i_pro;             // Ошибка, B::i_pro недоступна через класс PUB
}
```

Напишем класс, который наследует класс B со спецификатором доступа protected:

```
class PROT : protected B{       // x_pri наследуются как private
...                             // i_pro наследуются как protected
};                              // c_pub наследуются как protected
```

Методы и друзья класса PROT могут обращаться к i_pro и c_pub. Для внешних функций переменные x_pri, i_pro и c_pub закрыты.

Теперь рассмотрим наследование со спецификатором доступа private:

```
class PRIV : private B{         // x_pri наследуются как private
...                             // i_pro наследуются как private
…                               // c_pub наследуются как private
   friend void f2(PRV d);       // Друг класса
};

void f2(PRIV d)
{
   char c2 = d.c_pub;           // Ошибка, B::c_pub недоступна через класс PRIV
}
```

Если перед базовым классом не указан спецификатор доступа, считается, что действует спецификатор private, то есть объявления

```
class PRIV : B{…};
```

и

```
class PRIV : private B{…};
```

эквивалентны.

Таким образом, можно сформулировать следующие правила доступа:

к закрытым (private) членам класса имеют доступ только методы класса и функции-друзья класса;

к защищенным (protected) членам имеют доступ методы и друзья данного класса и методы и друзья классов, производных от него;

к открытым (public) членам имеет доступ любая функция.

15.5. Наследование и конструкторы

При создании объекта производного класса должны быть созданы все его объекты–предки. Если в базовых классах есть конструкторы, то они должны быть вызваны. При написании конструкторов производного класса следует предусматривать вызов конструкторов базовых классов. В следующей программе демонстрируется схема вызова конструктора базового класса.

Программа 15.4. Конструкторы производного класса

Изменим классы из программы 15.3. Включим в класс Base конструктор:

```
// Файл Base_con.h

#include <iostream>
using namespace std;
class Base{
    int a;                                  // Закрытый член
public:

    Base( int aa ) { a = aa; }              // Конструктор

    void Print()
    { cout << "Base::" << a; }
    int get_a( )
    { return a; }
};
```

В производный класс Derived также включим конструктор:

```
// Файл Derived_con.h

#include "Base_con.h"

class Derived : public Base{ // public говорит, что открытые члены Base
    double x;            // остаются открытыми, становясь членами Derived
public:

    Derived ( int aa, double xx ) : Base(aa)      // Конструктор
    { x = xx; }

// Другой вариант конструктора класса Derived:
// Derived (int aa, double xx) : Base(aa), x(xx) { }

    void Print( )
    { cout << "Derive::" << x; }

    double get_x( )
    { return x; }

    bool IntMoreDbl( )       // Сравнение, больше ли a чем x
```

```
    {                          // Функции производного класса имеют доступ
        return get_a() > x;  // только к открытым членам базового класса
    }
};
```

Таким образом, конструктор производного класса Derived имеет вид:

```
Derived (int aa, double xx) : Base(aa){ x = xx; }
```

то есть, в заголовке конструктора ставится двоеточие, после которого вызывается конструктор базового класса: Base(aa).

Теперь выполним тестирование классов.

```
// Файл BaseDer_con.cpp

#include "Derived_con.h"

int main()
{
    Base b(1);       // Переменная (объект) базового класса
                     // Конструктор устанавливает значение члена a объекта b
    cout << "b = "; b.Print();   // Вызов функции Base::Print()

    Derived d(2, 3.0);           // Объект производного класса
                                 // Конструктор устанавливает значения целой
                                 // и вещественной части
    cout <<  "\nd = ";
    d.Print();                   // Вызов функции Derived::Print()
    cout << ", ";
    d.Base::Print();             // Для вызова функции базового класса здесь
                                 // требуется явное указание класса функции
    cout << endl;
    if(d.IntMoreDbl())
             cout << "d::Base > d::Derived";
    else
             cout << "d::Base < d::Derived";
    cin.get();
    return 0;
}
```

Вывод программы:

```
b = Base::1
d = Derive::3, Base::2
d::Base < d::Derived
```

Приведем более содержательный пример использования наследования.

Программа 15.5. Производный класс личных данных

Изменим программу 15.2 так, чтобы дата входила в класс Pers не как объект класса Date, а за счет объявления класса Pers производным от класса Date.

В класс Date не вносится изменений, поэтому включаем в состав проекта файлы ClassDate.h и ClassDate.cpp из программы 15.2.

Объявление класса Pers, производного от класса Date разместим в файле Der_Pers.h:

```
// Файл Der_Pers.h

#ifndef Der_PersH
#define Der_PersH

// Подключение класса Date
#include "..\\Progr_15_02_PersonalData\\ClassDate.h"

class Pers : public Date              // Класс персональных данных Pers,
{                                     // производный от класса Date
    string name;                      // Строка с фамилией и инициалами
public:

// Конструктор с параметрами
    Pers(const string& nm, const Date& D) : Date(D), name(nm)
    { }

// Конструктор по умолчанию
    Pers :: Pers() : Date(), name()   // Текущая дата как дата рождения
    { }                               // Пустая строка как имя

// Конструктор копирования
    Pers::Pers(const Pers& ps) : Date(ps), name(ps.name)
    { }
    Pers& operator = (const Pers &);  // Оператор присваивания

// Оператор вывода сведений о человеке
    friend ostream& operator<<(ostream& os, const Pers& ps)
    { return os << ps.name << "     \t" << Date(ps); }

// Оператор ввода сведений о человеке
    friend istream& operator>>(istream& is, Pers& ps);

// CmpDate: возвращает true, если дата объекта ps1
// более ранняя или та же, что и у объекта ps2
    friend bool CmpDate(const Pers& ps1, const Pers& ps2)
    {                                 // Используется
        return ps1 <= ps2;            // функция-оператор сравнения дат
    }

// CmpName: сравнение фамилий
    friend bool CmpName(const Pers& ps1, const Pers& ps2)
    { return ps1.name <= ps2.name; }
};
#endif
```

Объявление класса Pers производным от класса Date делает все поля и методы класса Date полями и методами класса Pers. Ключевое слово public перед Date означает, что все открытые члены класса Date остаются открытыми, становясь членами класса Pers, закрытые члены остаются закрытыми, защищенные – защищенными.

В реализации конструктора с параметрами выражение Date(D) означает вызов конструктора копирования класса Date, который установит значение даты, входящей в класс Pers, равным дате D. Выражение name(ps.name) означает вызов конструктора копирования библиотечного класса string для создания строки name как копии строки ps.name.

В конструкторе по умолчанию выражение Date() означает вызов конструктора по умолчанию класса Date, который установит текущую дату, а выражение name() установит в качестве имени пустую строку.

В конструкторе копирования выражение Date(ps) означает вызов конструктора копирования класса Date, которому передается не дата, а объект ps класса Pers. Хотя в классе Date нет конструктора с аргументом типа Pers, ошибки не возникает, так как в производном классе есть все, что есть в базовом. Выражение Date(ps) создает дату, являющуюся членом класса Pers. Выражение name(ps.name) означает создание строки name как копии строки ps.name.

В функции CmpDate(const Pers& ps1, const Pers& ps2) сравнение дат, входящих в для ps1 и ps2 производится оператором <= :

```
return ps1 <= ps2;
```

В классе Pers нет оператора <=, но он есть в базовом классе Date, и вызывается в этой инструкции. В классе Date оператор <= объявлен с аргументами типа const Date&, а вызывается здесь в CmpDate() с аргументами типа const Pers&, то есть вместо базового класса Date используется производный класс Pers. Это допустимо, так как в производном классе есть все, что есть в базовом.

Непосредственно в объявлении класса Pers помещены реализации коротких функций. Остальные функции реализованы в файле Der_Pers.cpp.

```
// Файл Der_Pers.h

#include "Der_Pers.h"

// Реализация функции-члена класса Pers
Pers& Pers::operator=(const Pers& ps) // Оператор присваивания
{
    if (this != &ps){                 // Если присваивание не самому себе,
        name = ps.name;               // копирование имени,
        Date::operator=(ps);          // копирование даты
```

```
    }
    return *this;                    // Возвращение ссылки на объект
}

// Ввод данных об одном человеке
istream& operator>>(istream& is, Pers& ps)
{
    is >> (Date&)ps;                 // Ввод даты рождения
    getline(is, ps.name);            // Ввод фамилии и имени (инициалов)

// Поиск позиции первого непробельного символа в строке с фамилией
    int indexNotSpaceTab = ps.name.find_first_not_of(" \t");

// Удаление начальных пробелов
    ps.name = ps.name.substr(indexNotSpaceTab, ps.name.length());
    return is;
}
```

В операторе присваивания `operator=()` инструкцией

```
Date::operator=(ps);          // копирование даты
```

вызывается оператор присваивания класса `Date`, который скопирует дату, входящую в состав объекта `ps`.

Прямое присваивание вида:

```
d = ps.d; m = ps.m; y = ps.y;
```

недопустимо, так как закрытые члены `d`, `m`, `y` класса `Date` *недоступны* для функций производного класса `Pers`, хотя и *являются* членами этого класса «по наследству». Присваивание дат можно произвести с использования `this` инструкцией:

```
this->Date::operator=(ps);
```

В операторе ввода `operator>>(istream& is, Pers& ps)` в инструкции

```
is >> (Date&)ps;
```

ссылка на объект `ps` типа `Pers` приводится к типу ссылки на `Date`, затем используется перегруженный оператор ввода (`>>`) для класса `Date`. Преобразование ссылки на объект производного класа к ссылке на объект базового класса допустимо, так как в производном классе есть все, что есть в базовом.

Файл `Der_Persons.h` содержит объявление класса `Persons`. Его содержимое совпадает с файлом `Persons.h`.

```
// Файл Der_Persons.h

#ifndef Der_PersonsH
#define Der_PersonsH

// Подключение класса Date
#include "Der_Pers.h"            // Подключение класса Date
```

```
class Persons                    // Класс для моделирования группы лиц
{
    vector<Pers> G;                      // Вектор сведений о людях
public:
    void Add(const Pers& pd)             // Добавление в группу
    {                                    // нового человека
        G.push_back(pd);
    }

// Сортировка массива по критерию, задаваемому функцией сравнения Compare()
    void SortPersons(bool (*Compare)(const Pers& ps1, const Pers& ps2));

// Вывод сведений о группе лиц
    friend ostream& operator<<(ostream& os, const Persons& group);

// Ввод сведений о группе лиц
    friend istream& operator>>(istream& is, Persons& group);

private:

    void Swap(int i, int j); // Поменять местами элементы i и j массива G
};
#endif
```

Файл Der_Persons.cpp содержит определения функций-членов и друзей класса Persons. Он совпадает с файлом Persons.cpp.

```
// Файл Der_Persons.cpp

#include "Der_Persons.h"

// Реализация функций-членов класса Persons
void Persons :: Swap(int i, int j)   // Переставить элементы i и j массива G
{
    Pers tmp = G[i];
    G[i] = G[j];
    G[j] = tmp;
}

// Сортировка массива методом пузырька по критерию,
// задаваемому функцией сравнения Compare()
void Persons::SortPersons(bool (*Compare)(const Pers&, const Pers&))
{
        for(int i = G.size() - 1; i > 0; i--)
        for(int j = 0; j < i; j++)
            if(!Compare(G[j], G[j + 1]))     // Если не тот порядок,
                Swap(j, j + 1);              // переставить элементы
}

// Реализация функций-друзей класса Persons

// Вывод списка группы
ostream& operator<<(ostream& os, const Persons& group)
{
    for(int i = 0; i < group.G.size(); i++)   // Вывод данных
        os << group.G[i] << endl;             // о каждом человеке
```

```
    return os;
}

// Чтение данных о группе из файла
istream& operator>>(istream& is, Persons& group)
{
    Pers tmp;
    while(!is.eof()){
        is >> tmp;
        if(is.eof())                        // Если достигнут конец файла,
                break;                      // выход из цикла
        group.Add(tmp);
    }
    return is;
}
```

Для испытания производного класса `Pers` используем функцию `main()` из программы 15.2.

```
// Файл TesDerivedPers.cpp

#include "Der_Persons.h"

int main()
{
    setlocale(LC_ALL, "Russian");
    Persons Writers;                    // Writers – объект класса Persons
    ifstream inf("PersData.txt");       // Входной файловый поток
    if(!inf){                           // Если файл не удалось открыть,
        cerr << "Не удалось открыть файл PersData.txt ";
        exit(1);                        // завершаем программу
    }

    inf >> Writers;                     // Ввод данных о группе из файла

    cout << "Состав группы: \n";
    cout << Writers << endl;            // Вывод состава группы

    // Сортировка по возрастанию даты рождения
    Writers.SortPersons(CmpDate);
    cout << "Состав группы по возрастанию даты: \n";
    cout << Writers << endl;

    // Сортировка в алфавитном порядке
    Writers.SortPersons(CmpName);
    cout << "Состав группы по алфавиту: \n";
    cout << Writers << endl;

    system("pause");
    return 0;
}
```

Вывод программы совпадает с выводом программы 15.2.