ЛАБОРАТОРНОЙ РАБОТЫ № 1

По дисциплине: Объектно-ориентированное программирование

Тема работы: **Описание класса. Создание объектов при отсутствии конструктора в классе. Обращение к общедоступным полям и методам класса.**

Цель работы: Дать понятие класса, объекта класса. Научить описывать классы, обращаться к полям и методам класса.

Количество часов: 2ч

Содержание работы (Задание, Задачи):

1. Разработать классы, определенные в вариантах.
2. Создать инициализированные и неинициализированные объекты в каждом из разработанных у пункте 1 классе. Продемонстрировать невозможность создания инициализированного объекта (при отсутствии конструктора) в любом из классов, на выбор.
3. Продемонстрировать различные способы инициализации общедоступных полей объекта, на примере одного из классов.

Варианты:

1. a) **Student:** Фамилия, Имя, Отчество, Дата рождения, Адрес, Телефон, Факультет, Курс.
   b) **Train:** Пункт назначения, Номер поезда, Время отправления, Число общих мест, Купейных, Плацкартных.

   c) «Комплексные числа».

   d) «Треугольник».

2. a) **Abiturient:** Фамилия, Имя, Отчество, Адрес, Оценки.
   b) **Product:** Наименование, Производитель, Цена, Срок хранения, Количество.

   c) «Правильная дробь».

   d) «Дата».

3. a) **Aeroflot:** Пункт назначения, Номер рейса, Тип самолета, Время вылета, Дни недели.
   b) **Patient:** Фамилия, Имя, Отчество, Адрес, Номер медицинской карты, Диагноз.

c) «Окружность».

d) «Вектор».

4. a) **Book:** Автор, Название, Издательство, Год, Количество, страниц.
   b) **Bus:** Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

   c) «Рациональная дробь».

   d) «Отрезок».

5. a) **Worker:** Фамилия и инициалы, Должность, Год поступления на работу, Зарплата.
   b) **Customer:** Фамилия, Имя, Отчество, Адрес, Телефон Номер, кредитной карточки, Номер банковского счета.
   c) «Точка в пространстве».
   d) «Время».

Методические указания по выполнению:

1. **Описание класса**

В C++ так же, как и в других языках программирования, класс – создаваемый программистом структурный тип данных, который используется для описания множества объектов предметной области, имеющих общие свойства и поведение.

Класс объявляется следующим образом:

```
class <Имя класса>
{
 private:      <Внутренние (недоступные) компоненты класса>
 protected:    <Защищенные компоненты класса>
 public:       <Общие (доступные) компоненты класса>
};
```

Описание предусматривает три секции. Компоненты класса, объявленные в секции *private*, называются *внутренними*. Они доступны только компонентным функциям того же класса и функциям, объявленным *дружественными* (они будут рассмотрены позже) описываемому классу.

Компоненты класса, объявленные в секции *protected*, называются *защищенными*. Они доступны компонентным функциям не только данного

класса, но и его потомков. При отсутствии наследования – интерпретируются как внутренние.

Компоненты класса, объявленные в секции *public*, называются *общими*. Они

доступны за пределами класса в любом месте программы. Именно в этой секции

осуществляется объявление полей и методов интерфейсной части класса.

Если при описании класса тип доступа к компонентам не указан, то по умолчанию

принимается тип *private*.

В качестве компонентов в описании класса фигурируют *поля*, применяемые для хранения параметров объектов, и *функции*, описывающие правила взаимодействия с этими полями. В соответствии со стандартной терминологией ООП функции – компоненты класса или *компонентные функции* можно называть *методами*.

Компонентные функции или методы могут быть описаны как внутри, так и вне

определения класса. В последнем случае определение класса должно содержать прототипы этих функций, а заголовок описываемой функции должен включать *квалификатор видимости*, который состоит из имени класса и знака «::». Таким образом, компилятору сообщается, что определяемой функции доступны внутренние поля класса:

<Тип результата> <Имя класса> :: <Имя функции>(<Список параметров>)

{

<Тело компонентной функции>

}

Пример 1.1. Описание класса.

А. Описание компонентных функций внутри класса.

```
#include <stdio.h>
class First
{
public:
      char c;
      int x,y;
      /* компонентные функции, определенные внутри класса */
      void print(void)
      {
            printf ("%c %d %d ",c,x,y);
      }
      void set(char ach,int ax,int ay)
      {
            c=ach;     x=ax;     y=ay;
      }
};
```

Б. Описание компонентных функций вне класса.

```
#include <stdio.h>
class First
{
        public:   char c;
                  int x,y;
  void print(void);
  void set(char ach,int ax,int ay);
};
     /* компонентные функции, описанные вне класса */
void First::print(void)
{   printf ("%c %d %d ",c,x,y);   }
void First::set (char ach,int ax,int ay)
{   c=ach;    x=ax;    y=ay;  }
```

Согласно стандарту C++, если тело компонентной функции размещено в описании класса, то эта функция по умолчанию считается *встраиваемой (inline)*. Коды таких функций компилятор помещает непосредственно в место

их вызова, что ускоряет работу программы, но увеличивает ее размер и накладывает некоторые ограничения на использование языковых средств. Так, встраиваемыми не могут быть функции, содержащие операторы цикла, операторы безусловного перехода, ассемблерные вставки, а также виртуальные компонентные функции и функции, реализующие рекурсивные алгоритмы.

Обычно при попытке встраивания таких функций компиляторы C++ выдают сообщение об ошибке или предупреждение. Однако в старших версиях Visual C++, начиная с Visual Studio 2005, компилятор самостоятельно принимает решение о реализации конкретного метода как встраиваемого, а соответственно и не выдает никаких сообщений, игнорируя описатель inline.

2. Создание объектов при отсутствии конструктора в классе.

В программе, использующей классы, по мере необходимости объявляют объекты этих классов.

Объекты – переменные программы, соответственно на них распространяются общие

правила длительности существования и области действия переменных, а именно:

• внешние, статические и внешние статические объекты создаются до вызова функции *main()* и уничтожаются по завершении программы;

• автоматические объекты создаются каждый раз при вызове функции, в которой они объявлены, и уничтожаются при выходе из нее;

• объекты, память под которые выделяется динамически, создаются оператором *new* и уничтожаются оператором *delete.*

При объявлении полей в описании класса не допускается их инициализация, поскольку в момент описания класса память для размещения его полей еще не выделена. Выделение памяти осуществляется не для класса, а для объектов этого класса, поэтому возможность инициализации полей появляется только во время или после объявления объекта конкретного класса.

Объявление объектов и способы инициализации их полей зависят от наличия или отсутствия в классе специального инициализирующего метода –

конструктора, а также от того, в какой секции класса описано инициализируемое поле. Конструктор, являясь методом класса, может инициализировать любое поле объекта при его создании.

Если в классе отсутствует конструктор, но описаны защищенные *protected* или скрытые *private* поля, то возможно создание только неинициализированных объектов. Для этого используется стандартная конструкция объявления переменных или указателей на них. Например:

```
First   a,   // объект класса First
       *b,   // указатель на объект класса First
     c[4];   // массив с из четырех объектов класса First
```

При объявлении указателя, как и для обычных переменных, память под объект не выделяется. Это необходимо сделать отдельно, используя операцию *new*, после работы с динамическим объектом память необходимо освободить:

```
b=new First; ... delete b;
```

Объект, созданный таким способом, называют *динамическим*.

Значения полей неинициализированных статических и динамических объектов или

массивов объектов задают в процессе дальнейшей работы с объектами: защищенных и скрытых – только в методах класса, а общедоступных – в методах класса или

непосредственным присваиванием в программе.

При отсутствии в классе конструктора и защищенных *protected* или скрытых *private* полей для объявления инициализированных объектов используют оператор инициализации, применяемый при создании инициализированных структур, например:

```
First   a = {'A',3,4},
      c[4] = {{'A',1,4},{'B',3,5},{'C',2,6},{'D',1,3}};
```

Инициализирующие значения при этом должны перечисляться в порядке следования полей в описании класса.

Пример 2.1. Создание инициализированных и неинициализированных объектов при отсутствии конструктора.

```
#include <conio.h>
class Num
{
public:
	int n;
};
void main(int argc, char* argv[])
{
	Num N = {56}; // инициализированный объект
	Num NN;       // неинициализированный объект
	_getch();
}
```

3. Обращение к общедоступным полям и методам объекта из программы.

Обращение к общедоступным полям и методам объекта из программы может осуществляться с помощью полных имен, каждое из которых имеет вид

<Имя объекта> . <Имя класса> :: <Имя поля или функции> ;

Например:

a.First::set('A',3,4); // статический объект

b->First::set('B',3,4); // динамический объект

c[i].First::set('C',3,4); // массив объектов

Однако обычно доступ к компонентам объекта обеспечивается с помощью

укороченного имени, в котором квалификатор доступа опущен, тогда принадлежность к классу определяется по типу объекта:

<Имя объекта>.<Имя поля или функции>

<Имя указателя на объект> -> <Имя поля или функции>

<Имя объекта>[<Индекс>].<Имя поля или функции>

Например:

a.x	*b->x*	*c[i].x*
a.set('A',3,4)	*b->set('B',3,4)*	*c[i].set('C',3,4)*

Пример 3.1. Различные способы инициализации общедоступных полей объекта.

```
#include <locale.h>
#include <string.h>
#include <stdio.h>
#include <conio.h>



class sstro
{
public:
     char str1[80];
     int x,y;
     void set_str(char *vs) // инициализация полей
     {
          strcpy(str1,vs);    x=0;   y=0;
     }
     void print(void)        // вывод содержимого полей
     {
          printf("x=%5d    y=%5d   str: ",x,y);
          puts(str1);
     }
};
```

```
void main()
{
	setlocale(0,"russian");
	sstro aa = {"Строка",200,400}; // инициализированный объект
	sstro bb,cc;    // неинициализированные объекты
	bb.x=200;    // инициализация посредством прямого обращения
	bb.y=150;
	strcpy(bb.str1,"Строка");
	cc.set_str("Строка"); //  вызов инициализирующего метода
	aa.print();  bb.print(); cc.print();
	_getch();
}
```

Результат работы программы:

```
x=  200   y=  400   str: Строка
x=  200   y=  150   str: Строка
x=    0   y=    0   str: Строка
```

Пособия и инструменты:

Visual Studio 2008

Вопросы для защиты лабораторной работы:

1. Что такое класс в C++? Как выполнить описание класса?
2. Какие существуют способы ограничения доступа к компонентам класса? Как и где они используются? Чем отличается описание компонентных функций внутри и вне определения класса?
3. Посредством каких операций осуществляется доступ к элементам класса совместно с объектом класса, совместно с указателем на объект класса?
4. Какой тип доступа имеют элементы класса, доступные только для элементов-функций класса и друзей класса?
5. Доступом по умолчанию для элемента класса является …

6. Какая функция используется для присваивания значений закрытым элементам данных класса?
7. Как называется множество открытых элементов-функций класса?
8. Найдите ошибку в следующем фрагменте. Объясните, как ее необходимо исправить?

```
class Time {
    public:
        // прототипы функций
    private:
        int hour = 0;
        int minute = 0;
        int second = 0;
};
```

Литература:

1. Скляров В.А. Язык C++ и объектно-ориентированное программирование. – М.: Высшая школа., 1997.
2. Иванова Г.С., Ничушкина Т.Н., Пугачев Е.К. Объектно-ориентированное программирование: Учебник для ВУЗов. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2007.
3. Березин Б.И., Березин С.Б. Начальный курс С и C++. – М.: «Диалог-МИФИ», 1997.
4. Вайнер Р., Пинсон Л. C++ изнутри: Пер. с англ. – Киев: «ДиаСофт», 1993.
5. Дейтел Х., Дейтел П. Как программировать на C++: пер. с англ./ – М.: ЗАО «Издательство БИНОМ», 2005 г. – 1024 с.: ил.