Практическое занятие
Тема: Проектирование классов и создание объектов

Цель: Получение практических навыков разработки классов, создания объектов и использования дружественных функций для обработки данных.

Общие сведения о классах и объектах

Ключевым понятием ООП является объект. Под **объектом** понимается переменная особого типа. Этот особый тип в Си++ создан на базе типа данных **структура** (struct) и называется **класс** (class). *Класс* отличается от *структуры* в языке С++ тем, что в него помимо собственно данных различных типов входят также функции, обрабатывающие эти данные. Класс – это тип, группирующий данные и функции в единое целое с точки зрения их последующей обработки. Объявленная переменная типа class есть объект. Такое объединение данных и обрабатывающих их функций называется *инкапсуляцией*. Она защищает данные от внешнего вмешательства или неправильного использования. В то же время объект может оснащаться средствами программной связи с другими объектами.

Данные и функции, входящие в класс, называются *представителями* или *членами* этого класса. Данные класса ещё называют полями, а функции – методами класса. Члены класса разделяются на закрытые (**private**) и открытые (**public**). К закрытым членам обращение возможно только внутри класса и из других частей программы они недоступны. Обычно таким образом защищаются данные от внешнего вмешательства. К открытым членам класса может производиться обращение извне.

Синтаксис объявления *класса* похож на синтаксис объявления *структуры*. Пример объявления.

```
#include <iostream>
using namespace std;
class myclass              // класс под именем myclass
{
  private:
    int a;                 // закрытая переменная
  public:                  // ключевое слово (для открытых членов класса)
     /* функции – открытые члены класса  для доступа к закрытой переменной */
     void set_a(int n){a=n;} -
      int get_a() {return a;}
};
```

Объявление класса myclass не задает ни одного объекта типа myclass. Оно определяет только **тип** объекта (шаблон). Чтобы создать объект, необхо-

димо указать имя класса - как спецификатор типа данных, после которого записать имя объекта. Например, в следующей строке заданы два объекта типа myclass .

```
myclass obj1, obj2;
```

Создание объекта связано с выделением для него памяти. При объявлении же класса память не выделяется.

После того как объекты созданы, можно обращаться к открытым членам класса. Как и при работе со структурами, обращение к открытым членам класса производится через точку **“ . ”**. В следующем фрагменте показано обращение к функциям- членам класса myclass.

```
obj1.set_a(10);      // обращение к функции set_a( ) первого объекта
obj2.set_a(22);      // обращение к функции set_a( ) второго объекта
cout<<obj1.get_a( )<<’\t’;      /*обращение к функции get_a( )
                                  первого объекта */
cout<<obj2.get_a( )<<’\n’;      /*обращение к функции get_a( )
                                  второго объекта */
```

Функции set_a( ) первого объекта в качестве аргумента передаётся численное значение 10, которое присваивается переменной **a** этого объекта. Аналогично переменной **a** второго объекта присваивается значение 22.

Каждый объект имеет свою переменную **a**, и использует принадлежащие классу функции set_a( ) и get_a( ). Поэтому объекты еще называют экземплярами классов. В третьей строке фрагмента на экран выводится значение, возвращаемое функцией get_a( ) первого объекта, а в четвертой – второго объекта. После выполнения указанных операций на экране будет: 10 22.

Конструкторы и деструкторы

Среди членов класса могут быть две особенные функции. Их особенность заключается в том, что обращение к ним производится по умолчанию. Первая функция называется *конструктор*. Она вызывается автоматически при создании объекта. Её назначение состоит в том, чтобы присвоить начальные значения (инициализировать) переменным объекта. Вторая функция называется *деструктор*. Она служит для ликвидации последствий использования объекта (например, для освобождения памяти). Поэтому она автоматически вызывается при удалении объекта. Конструктор имеет то же имя, что и класс, частью которого он является. Имя деструктора также совпадает с именем класса, но перед ним ставится символ ~ (тильда). Конструктор и деструктор не возвращают никаких значений.

```
#include<iostream>
using namespace std;
class abc          // объявление класса abc
{
```

```
    int a;          // закрытая переменная
public:
  abc(){ cout<<"I am constructor \n"; a=10;} // конструктор
  ~abc(){ cout<<"I am destructor \n";}        // деструктор
   void show(){ cout<<a<<"\n";}         // функция-член класса
};

int main()
{
  abc obj;        // создание объекта
  obj.show();
return 0;
}
```

В этом примере конструктор не содержал параметров. Но обычно параметры присутствуют, т.к. основное назначение конструкторов – производить инициализацию переменных объекта. В следующем примере используется конструктор с параметрами.

```
#include<iostream>
using namespace std;
class black                 // объявление класса black
{
     int a,b;               // закрытые переменные
  public:
     black(int x, int y)    // конструктор с параметрами
       {a=x; b=y;}
     void show()
       {cout<<a<<'\t'<<b;}
};
int main()
{
     // создание объекта с инициализацией переменных
  black ob(20,100);
  ob.show();
return 0;
}
```

Массивы объектов

Объекты – это переменные, и они имеют те же возможности и признаки, что и переменные любых других типов. Поэтому объекты могут объединяться в массивы. Синтаксис объявления массива объектов совершенно аналогичен тому, который используется для объявления массива переменных любого другого типа. То же касается и доступа к элементам массива.

Пример с массивом объектов.

```
#include<iostream>
using namespace std;
class one              //  имя класса
{
    int a;             // закрытая переменная
  public:
    void set_a(int n){a=n;}    // функция доступа к переменной
     int get_a( ) {return a;}  // функция доступа к переменной
};

int main(
{
   one obj[8];               // объявление массива из 8-и объектов
   int i;
   for(i=0; i<8; i++)        // инициализация переменных объектов
        obj[i].set_a(i);
   for(i=0;i<8;i++)          // вывод значений переменных объектов
        cout<<obj[i].get_a()<<'  ';
 return 0;
}
```

В данном примере объявлен класс **one**, в котором имеется закрытая переменная и две функции доступа к ней. В главной функции создан массив из 8-и объектов, в котором циклической операцией производится инициализация закрытой переменной каждого элемента массива с помощью функции **set_a().** Во второй циклической операции осуществляется вывод на экран значения переменной каждого объекта с использованием функции **get_a().**

Дружественные функции

Дружественные функции введены для прямого доступа к private-членам объектов класса. Дружественная функция определяется в программе как обычная функция. Чтобы показать, к какому классу она дружественна, её объявляют в этом классе с ключевым словом *friend*. Особенность дружественной функции состоит в следующем. Дружественная функция не является членом класса. К закрытым членам класса она не может обращаться непосредственно. Она это может сделать только через объекты этого класса. Поэтому в качестве параметра у дружественной функции должен указываться объект со спецификатором дружественного класса.

```
#include<iostream>
using namespace std;
class data
{
```

```
    int a,b;
public:
    data(int x,int y)    // конструктор с параметрами x и y
        { a=x; b=y;}
    friend int diff(data ob); // дружественная функция
};

int diff(data ob)     // описание дружественной функции
    { return ob.a-ob.b;}
int main()
{
  int d;
  data obj(8,5);
  d=diff(obj);   // вызов дружественной функции
   cout<<"Difference = "<<d;
return 0;
}
```

Практикум

Вариант А:

Задание 1. Спроектировать класс **point,** обозначающий координаты точки на плоскости: x,y. Разработать:

а) конструктор с параметрами - для инициализации объектов класса **point**;

б) функции доступа к закрытым членам класса;

в) деструктор, обозначающий момент уничтожения объекта.

Создать два объекта и продемонстрировать работу методов класса.

Задание 2. В проект добавить дружественную функцию, вычисляющую площадь прямоугольника, заданного точками его вершин. Создать массив из 4-х объектов, обозначающий точки вершин прямоугольника. По желанию пользователя выводить на экран:

- координаты заданных точек;
- расстояние между любыми точками;
- площадь прямоугольника, заданного точками его вершин.

Задание 3. Спроектировать класс в соответствии с *индивидуальным заданием* (см. приложение). Создать массив из 6-ти объектов спроектированного класса. Рассчитать *Вычисляемый показатель*. Интерфейс программы оформить в виде меню.

Вариант Б:

Задания 1 и 3 из Варианта А.

ПРИЛОЖЕНИЕ

Варианты заданий

№ п\п	*Класс*	*Вычисляемый показатель*
1.	Компьютер	Экземпляр с наибольшей оперативной памятью
2.	Локальная сеть	Минимальная стоимость монтажа
3.	Транспортное средство	Максимальный пробег на полном бензобаке
4.	Программный продукт	Последняя версия
5.	Документ	Количество документов, выданных в прошлом году
6.	Диета	Максимальное дневное количество белков
7.	Периферийное устройство компьютера	Минимальная цена устройства
8.	Строительный товар	Сумма покупки
9.	Представитель университета	Количество студентов, обучающихся у конкретного преподавателя
10.	Предприятие малого бизнеса	Название предприятия с максимальным числом сотрудников
11.	СУБД	Количество СУБД заданного производителя
12.	Страховой полис	Количество полисов на заданную фамилию
13.	Наушники	Тип с максимальным частотным диапазоном
14.	Недвижимость	Максимальная жилая площадь
15.	Часы	Самый дорогой экземпляр
16.	Телефон	Самая новая модель
17.	Бытовая техника	Количество товаров заданной фирмы
18.	Магнитная карта для проезда на транспорте	Количество карт без поездок
19.	Насекомое	Максимальный размер крыльев
20.	Канцелярские товары	Количество товаров заданной фирмы
21.	Товар	Сумма покупки
22.	Учащийся	Учащийся с максимальным IQ (коэффициентом интеллекта)
23.	Путешествие (тур)	Самый дешевый тур на 7 и более дней
24.	Мебель	Количество предметов из дерева
25.	Программное обеспечение	Продукт с максимальным объемом памяти
26.	Представление	Минимальное число зрителей в зале
27.	Запоминающее устройство	Экземпляр с минимальным размером
28.	Принтер	Экземпляр с наибольшей производительностью
29.	Книга	Количество книг одного автора
30.	Телефон	Самая новая модель
31.	Документ	Количество документов на одну фамилию
32.	Товар	Минимальная по весу покупка
33.	Строительные материалы	Сумма покупки