Задание № 10

Конструирование простейшего класса

Цель работы: познакомиться с основными понятиями объектно-ориентированного программирования (класс, объект, свойство, метод, конструктор, деструктор), выполнить конструирование класса, предназначенного для хранения заданной структуры данных, изучить способы создания объектов.

Задание.

В работе требуется сконструировать класс с заданным набором свойств. Набор свойств следует взять из лабораторной работы № 1 в соответствии со своим вариантом. В класс также должно быть добавлено достаточное количество методов для просмотра и редактирования значений любого из свойств.

Требования к конструированию класса: доступ к свойствам — закрытый, к методам — открытый. В классе следует предусмотреть конструктор по умолчанию, конструктор с параметрами.

Действия, выполняемые программой:

1. создание объекта с помощью конструктора по умолчанию,
2. создание объекта с помощью конструктора с параметрами,
3. создание массива объектов,
4. редактирование и просмотр свойств каждого объекта (можно однократное).
5. обработка массива объектов в соответствии с заданием лабораторной работы 1.

Ввод исходных данных осуществляется с клавиатуры, вывод на экран.

Справочный материал

Класс — это пользовательский тип данных, объединяющий данные и алгоритмы для обработки этих данных. Класс моделирует группу каких-либо реальных объектов (студенты, машины), процессов (путешествия), явлений (погода).

Данные класса представлены в виде переменных и называются **свойствами**.

Алгоритмы представлены в виде функций и называются **методами**.

В классе существует разграничение доступа к его членам. Внутреннюю (закрытую) часть класса, доступную только этому классу, составляет раздел `private`, защищенная часть класса доступна классу и его наследникам — раздел `protected`, доступны для любых объектов без ограничения члены класса из раздела `public` (открытая часть класса).

Пример объявления класса:

```
class Book
{
public:        // открытая часть класса
          Book(); //  Конструктор
          ~Book(); // Деструктор
          void setAuthor(std::string);//Метод, устанавливающий
                                        // новое значение свойству
          std::string getAuthor(); // Метод, позволяющий
                                   //прочитать значение свойства
          . . .
private:  // закрытая часть класса
          string author;
          string title;
```

```
        int year; // год издания
} ;
```

Данные для их защиты от воздействий извне помещаются в раздел private. Для доступа к таким данным используются методы, которые берут на себя контроль за корректностью использования данных сторонними объектами. Объединение в классе данных и методов с целью защиты данных называется **инкапсуляция**.

Для хранения текстовых данных в примере использовался стандартный тип string, объявленный в стандартной библиотеке языка C++. Он удобнее и надежнее для использования, чем массив символов в языке C, так как не надо заботиться о распределении памяти, кроме того, для типа string определена операция конкатенации строк +.

В класс можно включить несколько функций с одинаковыми именами, но различающихся списками параметров. Такая возможность основана на свойстве языка C++, называемом перегрузкой функций, а функции при этом называются перегруженными. Каждая перегруженная функция решает свою задачу и имеет свой алгоритм. Выбор функции, которая должна быть вызвана, определяется на этапе компиляции программы в точном соответствии со списком параметров. В данной работе перегруженными являются конструкторы. Использование перегруженных функций в классе представляет собой вариант **полиморфизма**, а именно, статический полиморфизм.

Объект создается по шаблону, который дает класс, при этом используется специальный метод — **конструктор**. Имя конструктора совпадает с именем класса, обычно он помещается в разделе public, для него не указывается тип возвращаемого значения.

Конструктор по умолчанию создает объект с не инициализированными свойствами:

```
Book::Book()
{

}
```

или всегда с одним и тем же набором значений:

```
Book::Book()
{
     author = "Noname";
     title = "Noname";
     year = 0;
}
```

Конструктор с параметрами создает объект с заранее определенным набором свойств:

```
Book::Book(std::string auth, std:: string ttl, int y)
{
     author = auth;
     title = ttl;
     year = y;
}
```

Деструктор выполняет разрушение объекта, он не имеет параметров, находится в разделе public, не имеет типа возвращаемого значения, а имя отличается от имени конструктора одним символом - знаком ~ (тильда) в начале.

```
Book::~Book()
{
// Деструктор по умолчанию
}
```

Не требуется вставлять в программу явный вызов деструктора — он вызывается

автоматически. Также деструктор можно не включать в класс — он будет добавлен автоматически компилятором. Деструктор должен быть обязательно включен в класс, если какие-либо данные объекта хранятся в динамической памяти.

В описании конструктора и любого другого метода класса (в реализации) используется оператор разрешения области видимости ::. Этот оператор позволяет включить идентификаторы в заданное пространство имен namespace. Для функций-членов класса пространством имен будет класс. Если не использовать оператор ::, получится глобальная функция, не связанная с классом.

Примеры создания объектов:

```
Book b1; // Создание объекта с помощью конструктора
// по умолчанию
Book b2("Pushkin","Evgeny Onegin",2003); // Создание
// объекта с помощью конструктора с параметрами
Book b3[3]; // Создание массива объектов с помощью
// конструктора по умолчанию
```

Для доступа к свойствам и методам в функциях-членах данного класса используется только имя свойства или метода

```
Book::Book(std::string auth, std::string ttl, int y)
{
     author = auth;
     . . .
}
```

Также возможно обратиться к члену того же класса с помощью указателя на текущий объект `this`

```
Book::Book(std::string auth, std:: ttl, int y)
{
     this->author = auth;
     . . .
}
```

Для доступа сторонних объектов к свойствам и методам объекта какого-либо класса используется имя объекта и операция точка (.)

```
b1.setAuthor("Gogol");
```

Сторонние объекты также могут обращаться к свойствам и методам какого-либо объекта с помощью указателя на объект и операции стрелка (->):

```
Book* ptr = &b1;
ptr->getAuthor();
```

Пример простейшей реализации методов класса `Book`:

```
void Book::setAuthor(std::string auth)
{
     author = auth;
}

std::string Book::getAuthor()
{
     return author;
}
```

Пример выполнения задания (без обработки массива)

```
#include <iostream>
```

```
using namespace std;

// Объявление класса
class Person
{
 public:
        Person();
        Person(std::string st, int y);
        ~Person();
        std::string getName();
        void setName(std::string val);
        int getYear();
        void setYear(int val);
 private:
        std::string name;
        int year;
};

// Программа, использующая класс
int main()
{
    Person p1; // Вызов конструктора по умолчанию
    Person p2("Kate",1977); // Вызов конструктора с параметрами
    p1.setName("Anita");
    p1.setYear(1978);

    cout << p1.getName() << "  " << p1.getYear() << endl;
    cout << p2.getName() << "  " << p2.getYear() << endl;
    Person p3[3];
    std::string sbuf;
    int ybuf;
    for(int i=0;i<3;i++)
    {
        cout <<"enter name->";
        cin >> sbuf;
        p3[i].setName(sbuf);
        cout <<"enter year->";
        cin >> ybuf;
        p3[i].setYear(ybuf);
    }
    for(int i=0;i<3;i++)
       cout << p3[i].getName() << "  " << p3[i].getYear() << endl;
    return 0;
}

// Реализация функций-членов класса
Person::Person()
{
    //Конструктор по умолчанию
     name = "Noname" ;
     year = 0;
```

```
}

Person::Person(std::string st, int y)
{
    //Конструктор с параметрами
    name = st;
    year = y;
}

Person::~Person()
{
    //Деструктор по умолчанию, можно не включать в класс,
     // явно в программе не вызывается
}

std::string Person::getName()
{
    return name; // Возвращает значение свойства name
}

void Person::setName(std::string val)
{
    name = val; // Перезаписывает значение свойства name
}

int Person::getYear()
{
    return year; // Возвращает значение свойства year
}

void Person::setYear(int val)
{
    year = val; //  Перезаписывает значение свойства year
}
```

Вопросы к защите

1. Чем класс отличается от структуры?
2. Как запретить посторонним доступ к свойствам класса?
3. Что такое конструктор?
4. Какие типы конструкторов вы знаете?
5. Покажите на примере вызов конструктора.