Объектно-ориентированное программирование с использованием языка C++

компьютерная академия ШАГ

Урок №1

Объектно-ориентированное программирование с использованием языка C++

Содержание

Основы объектно-ориентированного программирования

И, снова, здравствуйте!!! Мы очень надеемся, что вы успешно сдали экзамен по программированию на языке С и теперь можете переходить к следующему этапу обучения. А, именно, программированию на языке С++. Прежде всего, напомним некоторые исторические факты, касающиеся этих двух языков программирования. Эту информацию вы уже получали в первом уроке, но всё же:

В 1972 году 31-летний специалист по системному программированию из фирмы Bell Labs Деннис Ритчи разработал язык программирования Си. Первое описание языка было дано в книге Б. Кернигана и Д. Ритчи, которая была переведена на русский язык. Долгое время это описание являлось стандартом, однако ряд моментов допускали неоднозначное толкование, которое породило множество трактовок языка С. Для исправления этой ситуации при Американском национальном институте стандартов (ANSI) был образован комитет по стандартизации языка С. В 1983 году был утвержден стандарт языка С, получивший название ANSI C. В начале 80-х годов в той же Bell Laboratory Бьерном Страуструпом в результате дополнения и расширения языка С был создан новый по сути язык, получивший название "С с классами". В 1983 году это название было заменено на С++.

Вспомнили? Однако история создания языков — это не единственное различие между ними :)))). Дело всё в

том, что существует два основных подхода к программированию: **процедурный и объектно-ориентированный**. Рассмотрим их определения:

> **Процедурное программирование — подход, при котором функции и переменные, относящиеся к какому-то конкретному объекту свободно располагаются в коде и никак между собой не связаны. (Язык С).**
>
> **Объектно-ориентированное программирование — подход, при котором функции и переменные, относящиеся к какому-то конкретному объекту объединены в коде определенным образом и тесно связаны между собой. (Язык С++).**

Итак, с сегодняшнего дня мы будем заниматься объектно-ориентированным программированием. Будем называть его сокращенно ООП.

ООП — концепция, которая в свое время произвела настоящую революцию в программировании. ООП предполагает, что приложение строится из набора независимых по своему внутреннему устройству частей. ООП держится на трёх основных принципах, с которыми мы с вами сейчас познакомимся.

"Кит первый" — инкапсуляция (Encapsulation)

Принцип независимости данных в ООП называется **инкапсуляцией**. Таким образом, каждая часть может содержать собственные данные, недоступные другим частям системы. Очевидно, что абсолютно независимыми эти части быть не могут, поскольку им необходимо взаимодействовать между собой, использовать общие данные и обмениваться собственными данными.

Простой пример — живой организм, состоящий из множества живых клеток, каждая из которых имеет свое собственное поведение и свои собственное устройство, но взаимодействующая с другими клетками и обменивающаяся с ними веществами. В программировании такой живой организм — это приложение, а клетка — объект, вещества — данные, а пути взаимодействия — методы (функции) и события.

Попробуем сформулировать определение объекта:

Объект — это некоторая уникальная единица имеющая свои переменные и функции, эти переменные обрабатывающие".

Для создания объекта необходимо определить некий тип данных, который будет использоваться в программе и каждая переменная этого типа (экземпляр) ,будет представлять собой объект. В принципе, как объект можно представить тип данных — структура. Надеемся, что вы еще помните, что это такое. В структуре все данные, касающиеся одного объекта собраны воедино под одним именем, являющим собой пользовательский тип данных. В C++ таким типом данных будет **КЛАСС**.

Представьте себе черный ящик. С точки зрения программирования это объект. Но, что же тогда является классом, если ящик-объект? Класс в данном случае — это чертеж, по которому строится этот ящик. Теперь подумаем, что может относится к этому черному ящику. Совершенно очевидно, что у этого черное ящика есть некоторые свойства. Во-первых его форма, а во вторых то, что в нем лежит и механизм с помощью которого он закрывается и открывается и с помощью которого содержимое из него извлекается.

В программировании свойства так и называются — свойства. А, способы извлечения содержимого, применимые к этому ящику — это методы. С помощью свойств мы работаем с внутренними данными, читаем и устанавливаем значения, а методы — это те действия, которые может выполнять сам объект, то есть в нашем примере, мы нажимаем кнопку на ящике, которая "вызывает метод" — открыть ящик и ящик открывается самостоятельно. Другими словами, метод, это функция, принадлежащая объекту и выполняемая самим объектом.

Итак, резюмируем, инкапсуляция — механизм, с помощью которого все свойства и методы, относящиеся к одному определенному объекту собраны вместе под общим именем (типом), который мы будем называть — классом.

"Кит второй" — наследование (Inheritance)

Для начала дадим определение данному термину:

Наследование — это процесс, с помощью которого, один объект может наследовать свойства и методы другого объекта в дополнение к своим свойствам и методам.

Давайте продолжим анализировать черный ящик. Предположим, на основе черного ящика мы собираемся создать красочную упаковку (коробку) для подарка. Для этого мы просто добавим к черному ящику те особенности, которые характерны для подарочной упаковки. Например, цвет. Такие характеристики, как форма и размер, а также возможность открываться и закрываться имеются и у ящика и у коробки. Поэтому, не имеет смысл описывать их повторно, достаточно просто унаследовать "коробку" от "ящика". В этом и заключается главное преимущество

использования механизма наследования. Мы сначала создаем некую простую конструкцию, а затем добавляя к ней новые свойства и методы, получаем новый усовершенствованный объект.

"Кит третий" — полиморфизм (Polymorphism)

Полиморфизм — способность объекта вести себя по-разному, в зависимости от ситуации и реагировать на определенное действие строго специфичным для себя образом.

Банальная ситуация — вы приходите в магазин, что бы купить колбасу. Выбираете продукт. Если продавец — ваш знакомый, он подскажет вам — стоит или нет брать данный товар. Если — абсолютно чужой человек, равнодушно отрежет кусок. Продавец ведет себя так или иначе, в зависимости от ситуации. Так и наш с вами объект — самостоятельный тип данных будет различным образом реагировать на внешние раздражители. Сейчас вам немного сложно понять, как это будет происходить. Однако с принципом полиморфизма, вы уже знакомы, вспомните — перегруженные функции. Функция в зависимости, от переданных в нее параметров вызывала ту, или иную свою версию.

Итак, с теоретической частью покончено — пора переходить к практической.

Знакомство с классами

Определение класса и создание его объекта

Класс — это производный структурированный тип, введенный программистом на основе уже существующих типов. Другими словами, механизм классов задает некоторую структурированную совокупность типизированных данных и позволяет определить набор операций над этими данными.

Общий синтаксис класса можно определить с помощью конструкции:

```
class имя_класса {список_компонентов};
```

1. **имя_класса** — произвольно выбираемый идентификатор
2. **список_компонентов** — определения и описания типизированных данных и принадлежащих классу функций. Компонентами класса могут быть данные, функции, классы, перечисления, битовые поля и имена типов. Вначале для простоты будем считать, что компоненты класса — это типизированные данные (базовые и производные) и функции.
3. Заключенный в фигурные скобки список компонентов называют телом класса.
4. Телу класса предшествует заголовок. В простейшем случае заголовок класса включает слово class и имя.

5. Определение класса всегда заканчивается точкой с запятой.

Итак, принадлежащие классу функции мы будем называть **методами класса** или **компонентными функциями**. Данные класса — **компонентными данными** или **элементами данных класса**.

Вернемся к определению: класс — это тип, введенный программистом. А, так как, каждый тип служит для определения объектов, определим синтаксис создания объекта класса.

```
имя_класса имя_объекта;
```

Определение объекта класса предусматривает выделение участка памяти и деление этого участка на фрагменты, соответствующие отдельным элементам объекта, каждый из которых отображает отдельный компонент данных класса.

Способы доступа к компонентам класса

Существует несколько уровней доступа к компонентам класса. Рассмотрим два основных.

public — члены класса открыты для доступа извне.

private — члены класса закрыты для доступа извне.

Давайте расшифруем.

По умолчанию все переменные и функции, принадлежащие классу, определены как закрытые (*private*). Это означает, что они могут использоваться только внутри функций-членов самого класса. Для других частей программы, таких как функция main(), доступ к закрытым членам запрещен. Это, ***кстати, единственное отличие класса от структуры — в структуре все члены по умолчанию — public.***

С использованием спецификатора доступа public можно создать открытый член класса, доступный для использования всеми функциями программы (как внутри класса, так и за его пределами).

Синтаксис для доступа к данным конкретного объекта заданного класса (как и в случае структур), таков:

```
имя_объекта.имя_члена класса;
```

Пришло время примера...

```
#include <iostream>
using namespace std;
class Test{
    //так как спецификатор доступа не указан
    //данная переменная будет по умолчанию закрыта
    //для доступа вне класса (private);
    int one;
    //спецификатор доступа public
    //все члены, идущие после него
    //будут открыты для доступа извне

public:
    //инициализировать переменные в классе
    //при создании запрещено, поэтому мы определяем
    //метод, реализующий данное действие
    void Initial(int o,int t){
        one=o;
        two=t;
    }
    //метод показывающий переменные класса
    //на экран
    void Show(){
        cout<<"\n\n"<<one<<"\t"<<two<<"\n\n";
    }
    int two;
};
```

```
void main(){
     //создается объект с типом Test
     Test obj;
     //вызывается функция, инициализирующая его свойства
     obj.Initial(2,5);

     //показ на экран
     obj.Show(); // 2 5
     //прямая запись в открытую переменную two
     //с переменной one такая запись невозможна, так
     //как доступ к ней закрыт
     obj.two=45;

     //снова показ на экран
     obj.Show(); //2 45
}
```

Вышеописанный пример вполне интуитивно прост, однако, выделим один из главных моментов — **переменные класса нет необходимости передавать в методы класса в качестве параметров, так как они видны в них автоматически.**

Конструкторы и деструкторы

Начальная инициализация объектов класса. Конструкторы

Иногда во время создания объекта его элементам необходимо присвоить начальные значения. Как вы уже знаете, сделать это непосредственно в определении класса нельзя. Решить данную проблему можно, написав функцию, которая будет присваивать начальные значения переменным класса, и вызывать эту функцию каждый раз сразу после объявления объекта, что мы благополучно сделали в примере предыдущего раздела урока.

Однако, в языке C++ есть механизм, позволяющий решить эту проблему иным путём. Это — конструкторы.

Конструктор (*construct — создавать*) — это специальная функция-член класса,объявленная с таким же именем, как и класс. Для конструктора не определяется тип возвращаемого значения. ДАЖЕ void!!! Рассмотрим пример создания конструктора:

```
#include <iostream>
using namespace std;
class Test{
     //так как спецификатор доступа не указан
     //данная переменная будет по умолчанию закрыта
     //для доступа вне класса (private)
```

```
      int one;
      //спецификатор доступа public
      //все члены, идущие после него
      //будут открыты для доступа извне

public:
      Test(){
            one=0;
            two=0;
      }

      //инициализировать переменные в классе
      //при создании запрещено, поэтому мы определяем
      //метод, реализующий данное действие
      void Initial(int o,int t){
            one=o;
            two=t;
      }

      //метод показывающий переменные класса на экран
      void Show(){
            cout<<"\n\n"<<one<<"\t"<<two<<"\n\n";
      }
      int two;
};

void main(){

      //создается объект с типом Test
      Test obj; //(здесь сработает конструктор)

      //показ на экран
      obj.Show(); // 0 0

      //вызывается функция, инициализирующая
      //его свойства
      obj.Initial(2,5);
      //показ на экран
      obj.Show(); //2 5

     //прямая запись в открытую переменную two
```

```
        //с переменной one такая запись невозможна, так
        //как доступ к ней закрыт
        obj.two=45;

        //снова показ на экран
        obj.Show(); //2 45
}
```

1. Конструктор автоматически вызывается при создании объекта, т.е не нужно специально его вызывать.
2. Основное назначение конструкторов — инициализация объектов.
3. Конструктор должен быть всегда public!!!

С помощью параметров конструктору можно передать любые данные, необходимые для инициализации объектов класса.

Пример. *Класс, описывающий точку.*

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

//описание класса Point
class Point {
        int x, y;
        //координаты точки, по умолчанию имеют
        //уровень доступа private

public:
        //конструктор присваивает переменным класса x и y
        //начальные значения соответственно x0 и y0
        Point(int x0, int y0)
        {
                x = x0;
                y = y0;
        }
```

```
      //функция вывода координат точки на экран
      void ShowPoint()
      {
            cout << "\nx = " << x;
            cout << "\ny = " << y;
       }
};

void main()
{
      Point A(1,3); //создаем точку A
      //(объект типа Point) с координатами
      //x = 1, y = 3 (вызывается конструктор Point(1, 3)
      A.ShowPoint(); //выводим координаты точки A
                     //на экран
}
```

Примечание: При создании объекта значения параметров передаются конструктору с использованием синтаксиса, подобного обычному вызову функции.

Еще кое-что о конструкторах...

1. Конструктор без параметров называют конструктором по умолчанию. Такой конструктор обычно присваивает переменным-членам класса наиболее часто используемые значения.

```
Point()
{
      x = 0;
      y = 0;
}
```

2. Для каждого класса может существовать только один конструктор по умолчанию.

3. Если для класса не определено никакого конструктора, компилятор создает конструктор по умолчанию. Такой конструктор не задает никаких начальных значений, он просто существует :))).

Деструкторы

Деструктор выполняет функцию, противоположную функции конструктора. **Деструктор** (*destruct — разрушать*) — это специальная функция класса, которая автоматически вызывается при уничтожении объекта — например, когда объект выходит из области видимости.

Деструктор может выполнять любые задачи, в момент удаления объекта. Например, если в конструкторе была динамически выделена память, то деструктор должен освободить эту память перед удалением объкта класса.

Основные особенности при работе с деструктором

1. Деструктор не принимает никаких параметров и не возвращает никаких значений.
2. Класс может иметь только один деструктор.

И в заключении рассмотрим пример последовательности, в которой вызываются конструкторы и деструкторы.

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

//описание класса CreateAndDestroy
class CreateAndDestroy
{
public:
     CreateAndDestroy(int value) //конструктор
     {
          data = value;
```

```
          cout << " Object " << data << " constructor";
     }

     ~CreateAndDestroy() //деструктор
     {
          cout << " Object " << data << "
             destructor" << endl;
     }
private:
     int data;
};

void main ()
{
     CreateAndDestroy one(1);
     CreateAndDestroy two(2);
}

PROGRAM OUTPUT

Object 1 constructor
Object 2 constructor
Object 2 destructor
Object 1 destructor
```

Вывод к примеру — деструкторы вызываются в последовательности, обратной вызову конструкторов.

Домашнее задание

1. Цифровой счетчик, это переменная с ограниченным диапазоном. Значение которой сбрасывается, когда ее целочисленное значение достигает определенного максимума (например, k принимает значения в диапазоне от 0..100). В качестве примера такого счетчика можно привести цифровые часы, счетчик километража. Опишите класс такого счетчика. Обеспечьте возможность установления максимального и минимального значений, увеличения счетчика на 1, возвращения текущего значения.

2. Написать класс, описывающий группу студентов. Студент также реализуется с помощью соответствующего класса.