**ООП**

**Введение в ООП**

Объектно-ориентированное программирование – это подход при котором программа строится из объектов. Объекты -это некие сущности, обладающие определенными характеристиками, состоянием и поведением.

Объекты могут иметь границы, например: стул, телефон, машина;

или не иметь четких границ, например: вода, воздух;

но важно то, что объекты существуют в пространстве и времени

Объекты принято классифицировать, т.е. объединять в классы.

Класс - это множество объектов с одинаковым набором характеристик и состояний.

Класс - это синтаксическая конструкция, которая позволяет классифицировать(описывать) объекты,

Класс в общем и целом описывает объект, но класс не является объектом, он лишь в общих чертах описывать какими могут быть объекты/

Можно сказать, что класс создает четкий набор характеристик объектов и их поведение.

Кроме классов объекты также можно описывать с помощью структур.

Класс и структура – это фактически одно и тоже с одним единственным отличием, которое мы сегодня рассмотрим.

Классы и структуры состоят из членов (Members) или полей. Члены класса делятся на переменные и методы.

**Class /struct**

|  |
| --- |
| **Data** |
| **Methods** |

Data -> переменные члены класса. Это обычная переменная объявленная внутри класс или структуры, которые характеризуют объект и отображают его состояние.

Methods-> это функция внутри класса. Методы определяют поведение объектов и взаимодействие их с другими объектами того же класса или других классов

|  |
| --- |
| Human |
| Name;  Surname;  Bith date;  Nationality;  Weight;  Height;  IQ; |
| Go();  Talk();  Breath();  Eat();  Drink();  Sleep();  Love(); |

Создавая структуру или класс, мы создаем новый тип данных. Следовательно, объекты классов и структур являются переменными нашего типа. Из всего выше сказанного

Класс – это тип данных

Структура – это тип данных.

Объекты являются переменными нашего типа, также они являются экземплярами нашего класса или структуры.

struct Point {

double x;

double y;

};

void main() {

Point A; // объявление переменной ‘A’, типа Point

// объявление экземпляра ‘A’, структуры Point

Point\* pa = &A; // создаем указатель pa на объект A

}

Доступ к членам объектов

Для доступа к членам объекта можно использовать “.” либо “->”

“.” Оператор прямого доступа (point operator). Используется для доступа к полям объекта по имени объекта.

“->” Оператор косвенного доступа (Arrow operator). Открывает доступ к полям объекта по адресу объекта.

Дело в том, что часто возникает ситуация, когда имя объекта узнать не возможно, но мы всегда можем узнать адрес в памяти.

**Основные концепции ООП**

Объектно-ориентированное программирование базируется на трех основных концепциях:

**Инкапсуляция**

**Наследование**

**Полиморфизм**

Инкапсуляция – это способность классов скрывать определенную свою часть от внешнего мира. Это сокрытие определенной области класса от внешнего мира. Под внешним миром имеется ввиду все что находится за пределами класса. Все что скрыто внутри класса инкапсулировано в нем. Инкапсуляция реализуется модификаторами доступа и Get/Set методами.

В языке C++ имеется три **модификатор**а доступа

**Private:**

**Public:**

**Protected:**

**Private** – это закрытые поля, private члены доступны внутри класса

**Public** – это открытые поля, доступны как внутри класса так и за его пределами.

**Protected** – могут быть использованы только при наследовании, доступны внутри класса и внутри дочерних классов

В private-секции обычно находятся переменные, более того переменные члены класса обязательно должны быть закрытыми, к ним нельзя давать прямой доступ извне. Это защищает объекты класса от случайной перезаписи.

В public-секции обычно расположены методы, хотя некоторые из них могут располагаться и в private-секции.

**Единственным отличием между классом и структурой является то что по умолчанию у структуры все поля открытые (public), а у класса закрытые (private).**

Если переменные скрыты в классе, то и толка в них нет. Поэтому для доступа к переменным используют Get/Set методы

**Get-методы** нужны для того чтобы получить значения закрытой переменной, прочитать значение из памяти. Фактически get-методы открывают доступ, к переменной класса, на чтение. Get-методы всегда должны быть **константными**. Константным называется метод, который не изменяет объект. Кроме того, для константного объекта могут быть вызваны только константные методы.

int get\_x() const {

return x;

}

**Set-методы** нужны для того чтобы задавать значения переменным внутри класса, т.е. set-методы открывают доступ к переменным класса на запись. Кроме того, set-методы обеспечивают фильтрацию данных, т.е. предотвращают запись не правильных значений в переменные члены классов. Например, возраст человека минус 30 лет, масса человека 10 тон.

Исходя из всего выше сказанного, инкапсуляция предотвращает случайную перезапись переменных класса, а также запись не правильных значений в переменные члены класса.

У каждой переменной в классе должен быть свой Get/Set метод. Следовательно, Get/Set будет столько сколько и переменных.

Внутри объекта невозможно узнать имя этого объекта, но всегда можно узнать адрес объекта в памяти, для этого служит ключевое слово **this.**

This – это указатель на объект, для которого вызывается метод. Методы всегда вызываются для какого-то объекта. В нашем случае

void set\_x(double x) {

this->x = x;

}

This позволяет отличить принимаемый параметр x от члена класса x.

**Наследование (Inheritance)** – это способность классов наследовать свойства других классов.

**Полиморфизм** (многоформенность) – способность объектов вести себя по разному, в зависимости от обстоятельств или же в зависимости от того кем они являются.

**Обязательные методы в классе**

В любом классе обязательно должны быть такие методы, как:

**Конструктор (constructor)**

**Деструктор(destructor)**

**Оператор присвоить(=)**

Конструктор – это метод, который создает объект.

Деструктор – это метод, который удаляет объект по истечении его времени жизни.

Оператор присвоить – очищает значение объекта и присваивает ему значение другого объекта.

Все эти методы вызываются не явно, а некоторые из них могут даже создаваться не явно.

Рассмотрим их более подробно.

***Конструктор:***

Конструкторы бывают c:

Параметрами;

без параметров;

по умолчанию;

копирования;

переноса;

Любой конструктор называется так же, как и класс и никогда не возвращает никаких значений, при этом даже не пишется ключевое слово void. Конструктор может принимать параметры, а может их и не принимать. Благодаря тому, что конструктор может принимать параметры его можно перегрузить, а это означает что в классе может быть сколько угодно конструкторов. Их число не ограничивается компилятором.

Деструктор же никогда не принимает никаких параметров, следовательно, перегрузить его нельзя и именно поэтому деструктор в классе может быть только один. Разные конструкторы по разному создают объекты, но удаляются все объекты одинаково, независимо как они созданы. Деструктор называется так же как и класс с ~название класса.

Пример:

Point(){

Cout << “Constructor:\t” << this << endl;

}

~Point(){

Cout << “Destructor:\t” << this << endl;

}

**Типы конструкторов:**

**Конструктор по умолчанию** – это конструктор, который может быть вызван без параметров. Это может быть конструктор без параметров, потому что он не принимает никаких параметров или же конструктор, каждый параметр которого имеет значение по умолчанию, и тот и другой можно вызвать без параметров. Какой из них лучше, зависит от ситуации. Если в классе нет ни одного конструктора, то компилятор сам добавит в него конструктор по умолчанию. Такой default-конструктор называют неявным (implicit) конструктор по умолчанию выделяет память под объект и инициализирует поля объекта значениями по умолчанию. Не явный конструктор по умолчанию выделяет память под объект и при этом все его поля будут заполнены мусором.

**Конструктор с параметром** –

В классе может быть сколько угодно конструкторов, но особое место среди них занимают copy-конструктор и single argument конструктор (конструктор с одним параметром)

copy-конструктор - копирует объект

single argument конструктор - может быть использован для преобразования типов

Point(double x = 0, double y = 0) {

this->x = x;

this->y = y;

cout << "Too parametr Конструктор :\t" << this << endl;

}

**Конструктор копирование**

Для копирования объекта могут быть использованы

Конструктор копирования

Оператор присваивания (=)

И тот, и другой копируют объект.

Конструктор копирования выполняет копирование при создании нового объекта, когда новый объект нужно сделать точной копией другого существующего объекта.

Оператор присвоить вызывается в том случае, когда существующий объект нужно сделать точной копией другого объекта.

Если мы в существующий объект копируем значение другого объекта, то его текущее значение уже не нужно и оно удаляется из памяти и заменяется другим значением другого объекта.

Конструктор копирования и оператор присваивания выполняют побитовое копирование объекта.

Copy-методы особенно актуальны, если в классе используется динамическая память.

Конструктор копирования всегда принимает константную ссылку на объект.

Class Point

Point(const Point& other) {

this->x = other.x;

this->y = other.y;

cout << "Too parametr Конструктор :\t" << this << endl;

}

Конструктор присвоения

void operator = (const Point& other) {

this->x = other.x;

this->y = other.y;

cout << "Too parametr Конструктор :\t" << this << endl;

}

**Перегрузка операторов**

Перегруженные операторы – это обычные функции имя которых состоит из ключевого слова **operator** и знака оператора. Операторы можно перегружать как в классе, так и за классом. Если оператор перегружен в классе, то он является методом, а это означает, что он может быть вызван только для какого-то объекта и этот объект не явно становится операндом и к нему можно обращаться через **this.** Если оператор перегружен за классом, то он является обычной функцией и в любом случае будет принимать свои операнды как параметры.

Если **унарный** оператор перегружается в классе, то он не принимает параметров, его единственным операндом будет объект, для которого он вызывается (**this**). Если **унарный** оператор перегружается за классом, то он в обязательном порядке должен принимать один параметр – свой операнд.

Если **бинарный** оператор перегружается в классе, то он в обязательном порядке принимает один из параметров – операнд справа, операндом слева будет объект(**this**), для которого он вызывается. Если бинарный оператор перегружается за классом, то он в обязательном порядке принимает два параметра:

1. Операнд слева
2. Операнд справа

**Правила перегрузки оператора**

1. Перегрузить можно только существующие операторы;

+

\*

++

1. Не все существующие операторы можно перегрузить.

Не перегружаются:

?: тернарные

. – оператор прямого доступа

.\* –

:: – оператор разрешения видимости

# –

## –

1. Перегруженные операторы сохраняют приоритет.
2. Изменить поведение операторов со встроенными типами данных невозможно.