

Объектно-ориентированный подход к программированию

До сих пор мы работали в рамках структурного программирования, в котором поначалу любая программа представляла структуру, состоящую из трех базовых конструкций: *следование, ветвление и цикл*.

Однако, рост объемов и сложности программ уже в рамках структурного программирования все яснее показывали необходимость структурировать информацию, выделяя в ней главное и отбрасывая несущественное. Это называют *степенью повышения абстракции программы*. Возможность повышения абстракции программы в рамках структурного программирования тоже имела место. Например, повышение степени абстракции программы было реализовано с помощью пользовательских функций, которые можно *снова использовать*. Это позволяло отвлечься от деталей реализации функции и сосредоточить внимание только на ее интерфейсе. Если вы не применяете глобальные переменные, то интерфейс полностью определяется заголовком функции. Помещение фрагмента кода в функцию и передача всех необходимых ей данных в качестве параметров были не только повышением степени абстракции программы, но и нашим первым опытом знакомства с таким понятием как *инкапсуляция*, то есть с объединением и сокрытием деталей реализации.

Следующий шаг на пути повышения степени абстракции программ - описание собственных типов данных. Это позволяет структурировать и группировать информацию, представлять ее в более естественном для нас виде. Мы изучали с вами структуры. Структура – пользовательский тип данных. Можно представить, например, в виде одной структуры все сведения о студенте (ф.и.о., год рождения, номер группы, оценки и т.п.)

Если мы хотим работать с не стандартными, а собственными типами данных, то для работы с ними нам потребуются *специальные функции*. Естественно было бы сгруппировать описания собственных типов и описание этих специальных функций в одном месте программы и по возможности отделить их от остальных частей программы.

Для этого вводят *понятие класса*. Класс – это главное отличие объектно-ориентированных программ. Класс описывает тип, который объединяет (помещает в одну капсулу- класс, инкапсулирует) как данные, так и функции, с помощью которых эти данные обрабатываются. В ООП данные принято называть *свойствами*, а функции для их обработки – *методами*. Изменять и обрабатывать данные можно только с помощью методов данного класса.

Но если класс описывает тип, то мы можем создавать переменные этого типа, содержащие и методы, и свойства. Такие созданные переменные называются *объектами*. Каждый объект принадлежит определенному классу. И если мы сделали описание класса, то мы определили характеристики всех объектов, принадлежащих этому классу.

Итак, в чем главное отличие ООП?

В ООП каждый объект принадлежит определенному классу.

Класс – это основной инструмент, с помощью которого осуществляется *инкапсуляция (объединение данных и методов и их сокрытие)* в C++.

Сокрытие – это запрет на доступ к внутренней структуре данных напрямую извне, минуя специально создаваемые для этого методы.

Класс – это тип, определяемый пользователем.

Если описан класс, то определены характеристики всех объектов, принадлежащих этому классу. Но *класс определяет* не только *структуры данных*, но и *функции их обработки*.

Функции, инкапсулированные в классе и предназначенные для выполнения каких-то операций с данными любого из объектов этого класса, называют *методами класса*.

Константы и переменные класса(данные) называются *свойствами*. Иногда по аналогии со структурами *их называют полями*

Доступ к свойствам возможен только через методы


Инкапсуляция (encapsulation)– это ограничение доступа к данным и их объединение с методами, обрабатывающими эти данные. Когда данные и методы соединяются, то создается объект. Доступ к отдельным частям класса регулируется с помощью специальных ключевых слов:

- ☐ *public* (открытая часть),
- ☐ *private* (закрытая часть),
- ☐ *protected* (защищенная часть).

Объявление класса

```
class MyClass
{
public:
    // доступно всем

private:
    // доступно только данному классу

protected:
    //доступно классу и его наследникам
};
 Создание объекта
MyClass myObject;
```

Методы расположены в открытой части. Они формируют интерфейс класса и могут свободно вызываться через соответствующий объект класса.

Доступ к закрытой части класса возможен только из его собственных методов.

Доступ к защищенной части возможен из его собственных методов и из методов классов – потомков. Все это повышает надежность программ.

Конструкторы и деструкторы

Среди методов класса есть специальные методы – *конструкторы и деструкторы*.

Конструктор – это метод, который обеспечивает выделение памяти при создании объекта и инициализирует переменные объекта данного класса.

Особенности конструктора:

- ☐ *Конструктор носит то же имя, что и сам класс*, в котором он определен.
- ☐ *Конструктор* ничего не возвращает в точку вызова, поэтому при объявлении он *не должен иметь описатель типа возвращаемого значения*. Это значит, что конструктор не может быть явно вызван из пределов программы.

Виды конструкторов

Мы познакомимся только с двумя видами конструктора:

- ☐ Конструктор по умолчанию
- ☐ Конструктор с параметрами

Если все параметры конструктора имеют значение по умолчанию или если конструктор вовсе не имеет параметров, то он называется *конструктором по умолчанию*.

В других случаях конструктор может иметь любое количество параметров.

Деструктор – это метод, который используется для освобождения памяти, выделенной при создании объекта конструктором.

Особенности деструктора :

- ☐ Деструктор имеет такое же имя как и класс, но предваряется символом ~ (тильда).
- ☐ Объект, созданный как локальная переменная в некотором блоке, *уничтожается неявно*, когда при выполнении программы будет достигнут конец блока. Деструктор особенно важен, если память выделяется динамически из области так называемой кучи (heap). Такая память *освобождается явно* с помощью операции *delete*.

Пример программы с использованием классов

Объявление класса

```
class Person
{
    public:
        Person();           // конструктор по умолчанию
        std::string getName();
        void setName(std::string val);
        int getYear();       // методы
        void setYear(int val);
    private:
        std::string name; // свойства
        int year;
};
```

Реализация функций - членов класса

```
Person::Person()
{
    //Конструктор по умолчанию
}
или
Person::Person()
{
    name = "Noname";
    year = 1992;
}

void Person::setName(std::string nval)
{
    name = nval;
}
void Person::setYear(int val)
{
    year = val;
}

std::string Person::getName()
{
    return name;
}
int Person::getYear()
{
    return year;
}
```

Программа с использованием объекта

```
int main()
{
    // Вызов конструктора по умолчанию
    Person p;
    p.setName("Anita");
    p.setYear(1978);
    cout<<p.getName()<<" "<< p.getYear()<<endl;
    return 0;
}
```

Пример-задача. На плоскости имеются 2 точки: A(x1,y1) и B(x2,y2). Координаты точек вводятся с клавиатуры. Определить расстояние между двумя точками.

```
#include <iostream>

#include <math.h>

using namespace std;

class point
{
public:
    point();
    ~point();
    void Setpointx(int);
    int Getpointx();
    void Setpointy(int);
    int Getpointy();
private:
    int x;
    int y;
};

double calc(class point A, class point B);

int main()
{
    double S;
```

```

    point A;

    point B;

    int x1,x2,y1,y2;

    cout<<"vvedite x1=";

    cin>>x1;

    A.Setpointx(x1);

    cout<<"vvedite x2=";

    cin>>x2;

    B.Setpointx(x2);

    cout<<"vvedite y1=";

    cin>>y1;

    A.Setpointy(y1);

    cout<<"vvedite y2=";

    cin>>y2;

    B.Setpointy(y2);

    S=calc(A,B);

    cout<<"S="<<S;

    return 0;
}

point::point()
{
}

point::~~point()
{
}

void point::Setpointx(int a)
{
    x=a;
}

int point::Getpointx()
{
    return x;
}

```

```
}  
  
void point::Setpointy(int a)  
{  
    y=a;  
}  
  
int point::Getpointy()  
{  
    return y;  
}  
  
double calc(class point A, class point B)  
{  
    return sqrt(pow(B.Getpointx()-A.Getpointx(),2)+pow(B.Getpointy()-A.Getpointy(),2));  
}
```