





## Раздел «Язык Си» . OOP-Instrumental :

- [Понятие объекта и класса](#)
  - [Простые объекты](#)
    -  [Задачи](#)
      -  [Задача 1.](#)
      -  [Задача 2.](#)
      -  [Задача 3.](#)

## Понятие объекта и класса

### Простые объекты

**Объектно-ориентированный** подход предполагает, что в работа программы определяется взаимодействием различных объектов в ответ на обращение к каждому из них.

При процедурном программировании каждая задача в соответствии с постановкой и условиями разбивалась на отдельные алгоритмы, которые могли быть использованы последовательно один или много раз. При этом в процессе работы каждого алгоритма изменяются значения тех или иных переменных, которые служат для обмена данными между алгоритмами.

Можно представить, что все переменные, которые используются в алгоритмах – это простейшие объекты, к которым обращается <<высший разум>> через действия алгоритма.

В C++ сущность объектов определяется через описание класса объектов. Класс позволяет породить хотя бы один объект. Если удалось получить хотя бы один объект, можно получить сколько угодно других (пока позволяют ресурсы).

При этом все объекты имеют свой собственный набор атрибутов и методов. Метод – это функция, которая принадлежит непосредственно объекту и имеет доступ ко всем атрибутам этого объекта.

**Объект** обладает:

1. идентичностью,
2. набором атрибутов, значения которых определяет **состояние** объекта,
3. и поведением, которое определяется методами, присущими объекту.

Совокупность объектов, обладающих одинаковым набором атрибутов и поведением определяет общий для них **класс или понятие**.

В C++ сущность объектов определяется через описание класса подобных объектов. Класс позволяет породить хотя бы один реальный объект. Если удалось породить хотя бы один объект, то можно получить сколько угодно других объектов (если позволяют ресурсы)

Рассмотрим простейшую задачу.

#### Задача.

Время задается часами и минутами, которые отображаются на 24-часовом циферблате. Первоначально хотелось бы иметь такой инструмент, который:

1. для новых часов устанавливал бы их в 00:00
2. отображал бы время, указанное на 12-часовом циферблате в показаниях этих часов.

Понятно, что нам нужен объект, у которого есть атрибут <<час>> и атрибут <<минуты>>. Такой объект вполне можно описать используя структуры языка C и набор функций .

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

#### Поиск

#### Раздел «Язык Си»

[Главная](#)  
[Зачем учить C?](#)

[Определения](#)

[Инструменты:](#)

[Поиск](#)

[Изменения](#)

[Index](#)

[Статистика](#)

#### Разделы

[Информация](#)

[Алгоритмы](#)

[Язык Си](#)

[Язык Ruby](#)

[Язык](#)

[Ассемблера](#)

[El Judge](#)

[Парадигмы](#)

[Образование](#)

[Cemu](#)

[Objective C](#)

[Logon>>](#)

```

// Структура для описания показаний часов
typedef struct{
    int h;
    int min;
}CTime;

// установка часов в 0:0
void setTimeZero(CTime*);

void setTime(CTime*, int, int);
// Установить время по строке:
// H:MIN
void setTimeString(CTime*, char * );

// Установить время от системных часов
void setCurrentTime(CTime* a){
    struct tm tmp; // системная структура для даты и времени
    /*

struct tm {
//          int tm_sec;    /* Seconds (0-60) */
//          int tm_min;    /* Minutes (0-59) */
//          int tm_hour;    /* Hours (0-23) */
//          int tm_mday;    /* Day of the month (1-31) */
//          int tm_mon;     /* Month (0-11) */
//          int tm_year;    /* Year - 1900 */
//          int tm_wday;    /* Day of the week (0-6, Sunday = 0) */
//          int tm_yday;    /* Day in the year (0-365, 1 Jan = 0) */
//          int tm_isdst;   /* Daylight saving time */
//          };

// time_t тип для записи времени в секндах с 1970 г.
time_t tloc; // для записи результата выполнения time()

// получить секунды на текущий момент
time_t ctm = time(&tloc);

// преобразовать секунды к структуре tm
tmp = *localtime(&ctm);

// заполнить поля объекта типа
a->h = tmp.tm_hour;
a->min = tmp.tm_min;
};

// Преобразование к 12-часовому циферблату
//CTime convertTo12(int h24, int min24);

// Печать показаний часов
void printTime(CTime a){
    printf("%02d:%02d\n", a.h, a.min);
};

int main(){
    CTime t1,t2, t3;
    t1.h=-12;
    t1.min=450;
    // t1 - в 0:0
    // setTimeZero(&t1);
    printTime(t1);
    //
    // t2 = convertTo12(23,55);
    // printTime(t2);

    // получить текущее время
    setCurrentTime(&t3);
    printTime(t3);

```

```
return 0;
}
```

При объявлении переменных `t1` и `t2` выделяется память для всех атрибутов (`h, min`). Инициализация атрибутов не происходит, то есть значения их – случайные числа. При этом существует опасность установки значений для `h` и `min` не соответствующих показаниям наших часов. Работа программы при этом становится непредсказуемой.

Конечно, правильно используя функции, можно избежать неприятностей, но хотелось бы повысить "надежность" нашего инструмента – часов.

В языке C++ используется несколько иной подход. Для описания объектов используются как структуры, так и классы.

Рассмотрим описание объекта через **class**.

В C++ возможно:

1. определить для каждого объекта не только атрибуты, но и методы – его собственные функции, которые будут работать только с атрибутами данного объекта.
2. ограничить доступ к отдельным атрибутам или функциям, для того, чтобы обезопасить их значения от неразумного или вредоносного изменения.
3. описать специальные функции <<конструктор>> и <<деструктор>> для управления созданием и удалением каждого объекта.
4. и др.

Рассмотрим простейший вариант описания часов. В данном примере описание объекта(интерфейс) и реализация методов-функций приводятся в одном файле для простоты.

```
//=====Мантра=====
// Подключение заголовочного файла для ввода/вывода
#include <iostream>
// подключение стандартной библиотеки
#include <cstdlib>
// _____заклинание_____
// смысл заклинание будет ясен позже
using namespace std;
//=====конец мантры=====

// Описание интерфейса класса Time
class CTime{

    // по-умолчанию, атрибуты считаются недоступными
    // пользователю. К ним можно обращаться через
    // функции, которые будут объявлены доступными
    int h; // часы, от 0 до 11
    int min; // минуты, от 0 до 59

    // объявление открытой области:
    // все атрибуты и функции из этой области
    // доступны пользователю объекта
    public:

    // Объявление конструктора
    CTime();
    // Конструктор всегда имеет имя класса.
    // По умолчанию конструктор просто выделяет необходимую
    // память для атрибутов и методов объекта
    // но конструктор - тоже функция, и можно
    // добавить в него свой код

    // объявление методов:
    void setTime(int, int); // установка времени
    void setTimeString(char *); // установка строкой H:MIN
    // Установить время от системных часов
    void setCurrentTime();
    void print(); // печать значений часов
    CTime after(CTime a); // получить показание часов через a
```

```

/* *****
Все методы-функции класса имеют доступ к закрытой
области класса. И в каждом объекте методы работают
только со "своими" атрибутами. Именно поэтому
в методах нет надобности передавать в качестве
параметра объект как это было в C
***** */
};

//_____ Реализация функций _____
/*
Каждый атрибут или метод имеет короткое имя в своем классе
или полное имя.
Для класса CTime, полное имя метода или атрибута получается
приставкой имени класса перед именем атрибута или метода:
CTime::CTime() - конструктор, CTime::print() и т.д.
*/

// Конструктор.
CTime::CTime(){
// конструктор вызывается при создании объекта.
// Пусть атрибуты сразу будут равны 0
h = 0;
min = 0;
};

// печать значений атрибутов
void CTime::print(){
cout<<h<<':'<<min<<endl;
};

// Установка времени. Будем считать, что минуты не могут
// быть меньше 0 и больше 59, и часы должны быть не меньше 0
void CTime::setTime(int h24, int min24){
// сразу проверяем допустимость значений
// если не правильно - прерываемся
if(0 > min24 || 59 < min24 || 0 > h24 ){
cout<<"Неверное время\n";
exit(1);
}
min = min24;
// часы >0 конвертируем.
h = h24 % 12;
};

// Сумма времен
// Результат является объект типа CTime
CTime CTime::after(CTime a){
// Чтобы вернуть объект, его нужно иметь
// Создадим временный объект
CTime tmp;
// так как объекты a и tmp - это объекты класса CTime
// все функции этого класса имеют доступ ко всем атрибутам этого класса
// Значит эта функция CTime::add() может обращаться к закрытым атрибутам
// объекта a и tmp
int mins = min + a.min;
tmp.min = mins % 60;
tmp.h = (h + a.h + mins / 60) % 24;

return tmp;
};

// Как использовать объект часы:
int main(){
// Объявление объекта
CTime t1, t2; // h,min сразу 0
CTime res; // для суммы
// Доступ к методам как к атрибутам в структуре
// Для конкретного объекта t1 устанавливаются
// значения ЕГО атрибутов

```

```
t1.setTime(10,20);  
// печать атрибутов t1  
t1.print();  
  
t2.print();  
res = t1.after(t2);  
  
}
```

Итого наш "суперразум" создал часы (даже двое), установил время на часах и распечатал его.

### Задачи

#### Задача 1.

Реализовать и отладить все функции интерфейса **CTime**. Использовать C-код, написанный ранее для C-функций.

#### Задача 2.

Для класса **CTime** добавить следующие функции:

```
CTime before(CTime a); // возвращает объект время - показание часов на *a* раньше  
// возвращает 1, если текущее время на циферблате больше аргумента, 0 - если совпадают  
// и -1, если меньше.  
int isLater(CTime);
```

Проверить и отладить функции

#### Задача 3.

Сначала у Шляпсика и Бяксика были нормальные одинаковые часы. Когда они показывали время  $t_0$ , Шляпсик решил переводить их на  $t_1$  минут вперед каждые 30 минут, а Бяксик стал переводить из на  $t_2$  минут назад каждый час

Написать программу, которая выясняет через сколько суток, часы Шляпсика и Бяксика покажут одинаковое время и какое это будет время. Решить задачу, используя класс **CTime**.

-- [TatyanaOvsyannikova2011](#) - 22 Sep 2015

(с) Материалы раздела "Язык Си" публикуются под лицензией [GNU Free Documentation License](#).