Сибирь I

# Структуры

# Проверка остаточных знаний

# Что такое контейнеры?

# Что такое итераторы?

# Задача

Создать свою базу данных

# Пример

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Категория | Наименование | Дата | Сумма | Менеджер | Заказчик | |
| 2 | Простуда и грипп | Гастрацид | 02.01.2011 | 49 299,00 | Иванов | 36,6 | |
| 3 | Naturino | Леденцы | 13.01.2011 | 132 305,00 | Дубинин | Старый Лекарь | |
| 4 | Витамины | Асковит | 25.01.2011 | 19 584,00 | Михайлов | Старый Лекарь | |
| 5 | Naturino | 7 витаминов | 09.02.2011 | 41 983,50 | Михайлов | Старый Лекарь | |
| 6 | Простуда и грипп | Анти-Гриппин грейпфрут | 23.03.2011 | 34 873,00 | Дубинин | Ригла | |
| 7 | Простуда и грипп | Анти-Гриппин ромашковый | 05.04.2011 | 79 941,00 | Иванов | Городская аптека | |
| 8 | Витамины | Натуретто витамины + железо | 06.04.2011 | 40 368,00 | Дубинин | Нео-Фарм | |
| 9 | Простуда и грипп | Анти-Гриппин для детей | 07.04.2011 | 297 669,00 | Иванов | Нео-Фарм | |
| 10 | Зеленый Доктор | Фиторелакс | 10.04.2011 | 72 136,50 | Иванов | Ригла | |
| 11 | Простуда и грипп | Анти-Ангин Спрей | 11.04.2011 | 56 470,00 | Петров | Нео-Фарм | |
| 12 | Зеленый Доктор | Эвкалипт-М | 24.04.2011 | 5 890,88 | Дубинин | Городская аптека | |
| 13 | Витамины | Глюкозамин | 09.05.2011 | 65 467,00 | Михайлов | Апельсин | |
| 14 | Простуда и грипп | Анти-Гриппин для детей | 09.05.2011 | 21 186,00 | Дубинин | Апельсин | |
| 15 | Витамины | Глюкозамин | 19.05.2011 | 36 608,50 | Волина | 36,6 | |
| 16 | Простуда и грипп | Анти-Ангин Формула | 29.05.2011 | 11 906,00 | Иванов | Имплозия | |
| 17 | Витамины | Асковит | 30.05.2011 | 17 906,50 | Петров | Нео-Фарм | |

Продажи

# Можем ли мы это сделать уже сейчас?


КАК БЫ ДА, НО НЕТ

# Вроде того

```
std::map<std::string, std::string> category;
std::map<std::string, std::string> product_name;
std::map<std::string, unsigned int> sum;
...

// Новый заказ от Neofarm
customer = "Neofarm";
category[customer] = "Vitamines";
product_name[customer] = "Ascorbinka";
sum[customer] = 42000;
```

# Смущает ли что-нибудь?

Новый запрос заказчика перетирает старый

**Неудобно, нужно что-то ещё**

# Структуры

*Это производные типы данных, включающие в себя множество элементов разных типов*

```
struct order {
  // поля структуры
  char category[64]; // плохо
  std::string product_name; // гораздо лучше
  std::string date;
  unsigned int cost;
  std::string customer;
};

int function() {
  order example = {
    "Vitamines",
    "Askorbinki",
    "17.10.2022",
    42000,
    "Neofarm"
  };
  ...
}
```

# Другие способы инициализации

```
int function() {
  order example;
  example.category = "Vitamines";
  example.product_name = "Askorbinki";
  example.date = "17.10.2022";
  example.cost = 42000;
  example.customer = "Neofarm";

  order example_2 {"str", "str", "str", 1000, "str"};  // C++11

  order zero_order {};  // устанавливает все поля в 0
  zero_order = example;
  ...
}
```

# В контейнерах тоже красиво

```
int function() {
  // будет использоваться только внутри функции
  struct pair {
    int first,
    int second
  };

  int x1 = 1;
  int x2 = 2;
  int y1 = 3;
  int y2 = 4;

  std::vector<pair> pairs ({{x1, x2}, {y1, y2}});
  std::cout << pairs[0].first << ' ' << pairs[0].second;
  ...
}
```

# Немного о структурах в памяти

```
// сколько структура занимает места в памяти?

struct Foo {
char c;     // 1 bytes
int iiii;   // 4 bytes
};
```

# В данном случае поле char дополняется 3-мя байтами

```
struct Foo {
char c;     // 1 bytes
int iiii;   // 4 bytes
};          // sizeof(Foo) == 8
```

```
struct Foo {  // для выравнивания размер кратен 4 байтам
int iiii;
char c;
};            // sizeof(Foo) == 8

struct Bar {  // для выравнивания размер кратен 1 байту
char c8[8];
};            // sizeof(Bar) == 8

struct Test1 {
char c;
Foo foo;
};

struct Test2 {
char c;
Bar bar;
};
```

```
struct Foo {  // для выравнивания размер кратен 4 байтам
int iiii;
char c;
};            // sizeof(Foo) == 8

struct Bar {  // для выравнивания размер кратен 1 байту
char c8[8];
};            // sizeof(Bar) == 8

struct Test1 {  // для выравнивания размер кратен 4 байтам
char c;
Foo foo;
};            // sizeof(Test1) == 12

struct Test2 {  // для выравнивания размер кратен 1 байту
char c;
Bar bar;
};            // sizeof(Test2) == 9
```

# Выравнивание

*Это свойство адреса в памяти, согласно которому адрес переменной в памяти должен быть кратен её размеру*

Данные в памяти могут идентифицироваться по адресу в памяти

# Примерчик

```
struct x_ {
    char a;      // 1 byte
    int b;       // 4 bytes
    short c;     // 2 bytes
    char d;      // 1 byte
} bar[3];

struct actual_x_ {
    char a;             // 1 byte
    char _pad0[3];      // выравнивание 'b' по 4-байтной границе
    int b;              // 4 bytes
    short c;            // 2 bytes
    char d;             // 1 byte
    char _pad1[1];      // выравнивние sizeof(x_) для кратности 4
} actual_bar[3];
```


C++ STRUCT SIZES
ARE NOT MAKING SENSE
CIVIL WAR
CAPTAIN AMERICA

# Внимание – нелинейность

Примешь синюю таблетку — и сказке конец. Ты отправишься делать лабу и поверишь, что это был сон. Примешь красную таблетку — войдешь в страну ООП.

# В C++ структуры почти не отличаются от классов

# Классы

*Это производные типы, включающие в себя множество элементов различных типов данных и методы их обработки*

*Типы, объявляемые ключевыми словами `class` и `struct` являются **классами**?*

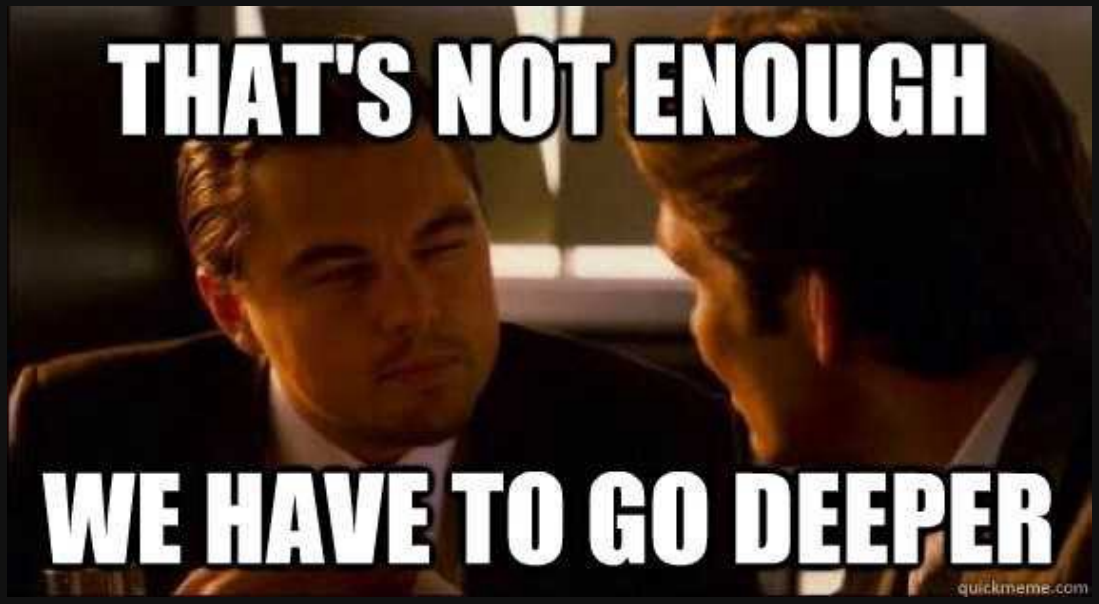
# Найдите 10 отличий

```
struct order {
  std::string category;
  std::string product_name;
  unsigned int cost;
  std::string customer;
};

class order_analog {
 public:
  std::string category;
  std::string product_name;
  unsigned int cost;
  std::string customer;
};
```

## У классов и структур также есть

1. Спецификаторы доступа
   - public
   - private
   - protected
2. Методы


THAT'S NOT ENOUGH
WE HAVE TO GO DEEPER
quickmeme.com

*У `struct` все поля по умолчанию* ***публичные***, *у `class` –* ***приватные***

# Спецификаторы доступа

```
class order {
 public:  // доступны вне класса
  std::string category;
  std::string product_name;
  unsigned int cost;

 private:  // доступны только внутри класса
  std::string customer;
};

int function() {
  order example = {...};
  std::cout << example.cost << std::endl;  // можно
  std::cout << example.customer << std::endl;  // нельзя
}
```

# Методы

```
class order {
  ...
 public:  // Доступно извне
  void print_customer() {  // описание метода
    std::cout << "Order " << order_id << "customer: " << customer
  }

 private:  // Доступно только внутри класса
  std::string customer;
};

int function() {
  order example = {...};

  // Можно
  example.print_customer();
}
```

# Еще пара отступлений

# Битовые поля в структурах

Типы должны быть перечислимыми или целочисленными

```
struct register {
  int var_1 : 4;        // значение занимает 4 бита
  int var_2 : 4;
  int : 4;              // пропуск 4 бит
  bool flag : 1;        // занимает 1 бит
  bool good : 1
};
```

Нужно в низкоуровневом программировании

# Объединения (Union)

*В одной области могут храниться данные, которые могут читаться по-разному – в зависимости от требуемого в конкретный момент типа данных*

```
union one4all {
  int int_val;
  long long_val;
  double double_val;
};

int function() {
  one4all example;
  example.int_val = 42;
  std::cout << example.int_val << std::endl;
  example.double_val = 3.1415926;
  std::cout << example.double_val << std::endl;
}
```

# Нужны ли они нам?


Как бы да, но нет