//Теперь, собственно, реализация кортежа. Для этого мы воспользуемся рекурсивным
//наследованием структур. Так выглядит объявление структуры:

```
// Предварительное объявление.
template<typename... Ttypes> struct Tuple;

// Базовый случай: пустой кортеж - пустая структура.
template<>
struct Tuple<> {};

// Кортеж реализуется путем рекурсивного наследования.
// На каждом уровне наследования от списка типов отделяется один элемент.
template<typename Tfirst, typename... Trest>
struct Tuple<Tfirst, Trest...> : public Tuple<Trest...> {
    Tfirst value;
};
```

//Также нам понадобится вспомогательная структура, позволяющая получить тип элемента по его
//индексу и тип оставшейся части кортежа:

```
// Предварительное объявление.
template<size_t index, typename Ttuple> struct Element;

// Дошли до элемента с указанным индексом, сохраняем типы.
template<typename Tfirst, typename... Trest>
struct Element<0, Tuple<Tfirst, Trest...>> {
    using Type_t = Tfirst;
    using TupleType_t = Tuple<Tfirst, Trest...>;
};

// Двигаемся по списку типов с помощью рекурсивного наследования.
// На каждом уровне наследования от списка типов отделяется один элемент.
template<size_t index, typename Tfirst, typename... Trest>
struct Element<index, Tuple<Tfirst, Trest...>>
    : public Element<index - 1, Tuple<Trest...>> {};
```

//Теперь мы можем написать функцию доступа к элементу по его индексу:

```
template<size_t index, typename... Ttypes>
typename Element<index, Tuple<Ttypes...>>::Type_t&
Get(Tuple<Ttypes...>& tuple) {
    using TupleType_t = typename Element<index, Tuple<Ttypes...>>::TupleType_t;
    return static_cast<TupleType_t&>(tuple).value;
}
```

//Конечно, сам кортеж и доступ — это хорошо, но мы же хотим его создавать! Для этого сперва
//реализуем заполнение кортежа.

```
// Базоый случай - аргументов нет.
template<size_t index, typename Ttuple>
void FillTuple(Ttuple& tuple) {}

// Кладем первый элемент в кортеж, вызываем рекурсивно для остальных.
template<size_t index, typename Ttuple, typename Tfirst, typename... Trest>
void FillTuple(Ttuple& tuple, const Tfirst& first, const Trest&... rest) {
    Get<index>(tuple) = first;
    FillTuple<index + 1>(tuple, rest...);
}
```

//Теперь легко реализовать и функцию создания кортежа. Почему не конструктор? В нем не
//работает автоматическое выведение типов, а мы же не хотим писать огромный список
//параметров.

```
template<typename... Ttypes>
Tuple<Ttypes...> MakeTuple(const Ttypes&... args) {
    Tuple<Ttypes...> result;
    FillTuple<0>(result, args...);
    return result;
}
```

//Ну и напоследок демонстрация того, что у нас получилось. Распаковку и групповое
//присваивание пока не демонстрирую, и так достаточно кода

```
int main() {
    // Тут будет ошибка, нельзя присваивать C-строки.
    // auto tuple = MakeTuple("foo", 123, 2.5);

    // Так все работает.
    auto tuple = MakeTuple(std::string("foo"), 123, 2.5);

    // Достаем элементы из кортежа.
    auto a = Get<0>(tuple);
    auto b = Get<1>(tuple);
    auto c = Get<2>(tuple);

    std::cout << a << " " << b << " " << c << std::endl; // foo 123 2.5
    return 0;
}
```