

# Типы, определяемые пользователем: Структуры

Язык С предлагает встроенные численные типы, действия над которыми определяются системой и процессором. Однако во многих ситуациях возникает необходимость создать собственные типы, данные экземпляров которых не могут быть представлены только одним элементом примитивного, встроенного типа.

Например, для того чтобы представить одно рациональное число, необходимо сохранить два целых числа. При этом крайне важно, что пара целых чисел связана, то есть объект, в котором зашифровано рациональное число, должен неделимым образом ссылаться на два целых, иначе в программе легко запутаться.

Первый вариант – использование массива из двух элементов. Такой вариант может быть удобным, если экземпляры типа могут быть представлены набором элементов одного типа, как рациональное число – пара целых чисел. Тем не менее, во многих ситуациях возникает необходимость сохранять экземпляр данных в виде набора из нескольких элементов разного типа.

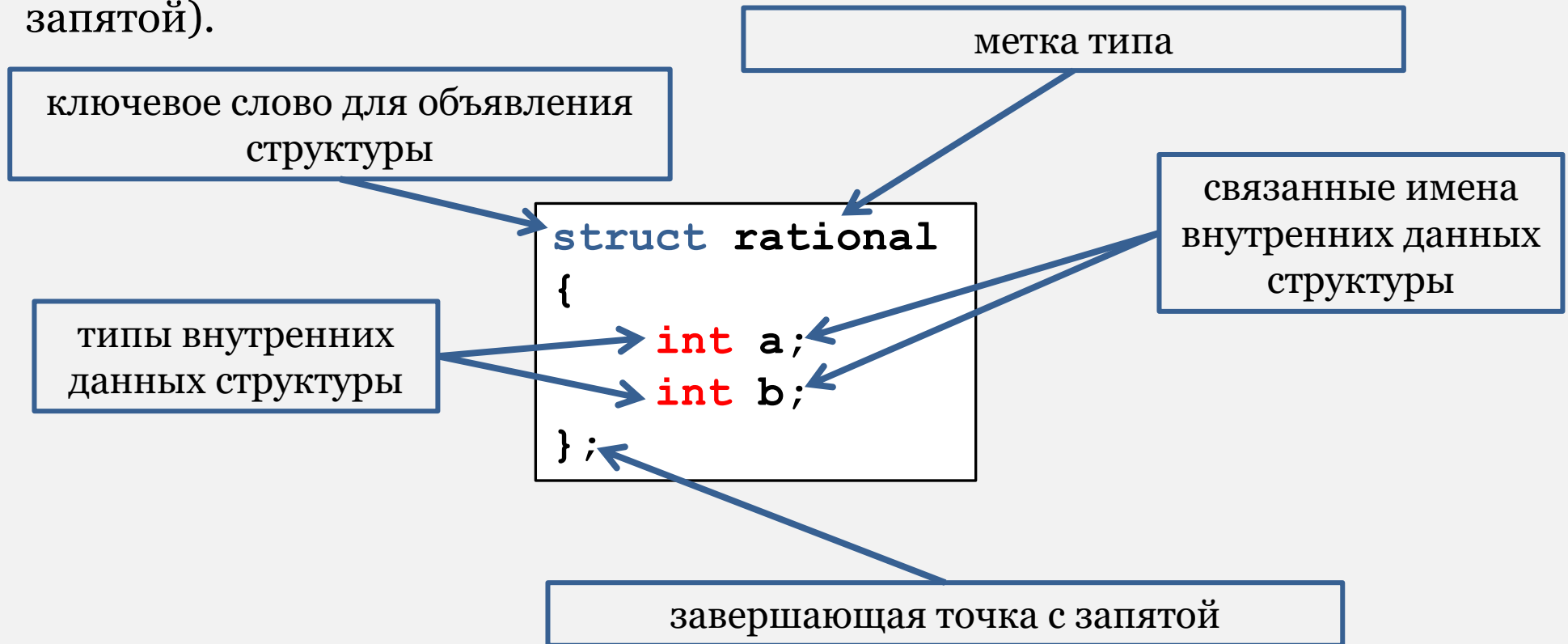
# Структуры

Элементы разных типов, связанные в единый объект данных, называются в программировании **кортежем** или **записью**. Для создания записей в языке **C** есть специальный инструмент – **структуры**. Структура – это определяемый пользователем композитный тип данных, который является набором из элементов данных других типов, возможно также определённых пользователем.



# Пример объявления структуры

Для объявления структуры используется ключевое слово **struct**. После ключевого слова указывается **синтаксическая метка типа**, которая будет присваиваться экземплярам структуры. За меткой типа следует блок кода, внутри которого перечисляются типы и связанные имена внутренних элементов. Объявление структуры завершается оператором **;** (точка с запятой).



# Объявление структуры

Следующие два определения структуры эквивалентны:

```
struct somestruct
{
    int a;
    int b;
    double d;
};
```

```
struct somestruct
{
    int a,b;
    double d;
};
```

Допускается немедленное определение экземпляра структуры сразу за объявлением структуры:

```
struct rational
{
    int a,b;
} q;
```

Допускается определение экземпляра структуры без типовой метки (экземпляра анонимной структуры):

```
struct {
    int a,b;
} p;
```

# Объявление структуры

Структура может быть объявлена внутри функции. В этом случае определение структуры распространяется только на тело функции.

```
int summ(int lha, int rha)
{
    struct rational
    {
        int a,b;
    } q;
    return lha + rha;
}
```

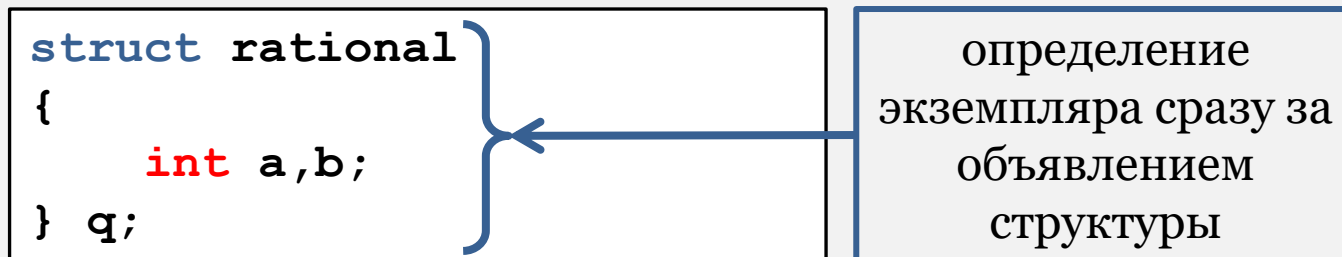
определение  
структуры  
**rational** внутри  
функции **summ**.

```
int main()
{
    struct rational
    {
        int a,b;
    } q;
    return 0;
}
```

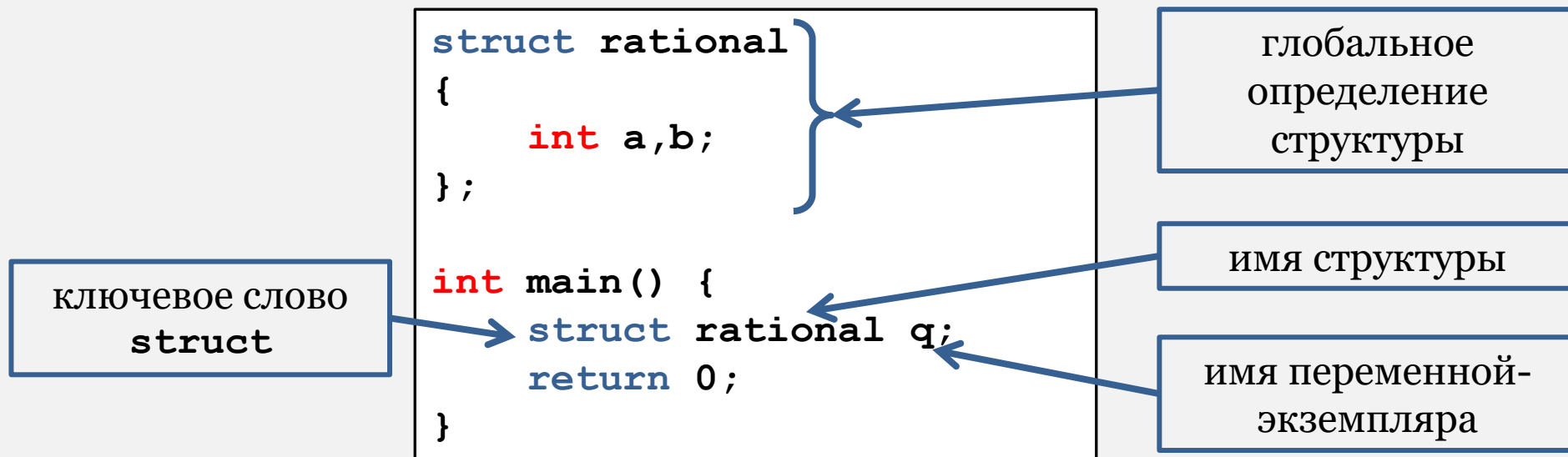
определение  
структуры  
**rational** внутри  
функции **main**.

# Объявление экземпляра структуры

Ранее мы познакомились с одним способом объявления экземпляра структуры: сразу за определением самой структуры.



Экземпляр структуры можно объявить и в любом другом месте, где видимо само определение структуры. В этом случае необходимо написать ключевое слово **struct**, за ним имя метки структуры и затем имя переменной.



# Объявление экземпляра структуры

Если структура объявлена внутри функции, то и экземпляр структуры может быть объявлен только внутри функции.

```
int summ(int lha, int rha)
{
    struct rational
    {
        int a,b;
    };
    struct rational q;
    return lha + rha;
}

int main() {
    struct rational q;
    return 0;
}
```

структура **rational** определена в функции **summ** – мы можем объявить переменную, которая является экземпляром этой структуры, внутри этой функции

на этой строке при компиляции возникнет ошибка, так как структура **rational** недоступна в функции **main**

# Обращение к внутренним данным структуры

Переменные, которые являются составными частями структуры, называются **полями** структуры или её **свойствами**. Для того чтобы определить действие над структурой, необходимо определить действия над её полями. Получив доступ к полю структуры, мы можем совершать с ним все действия, которые мы совершаем над любой другой переменной.


Если экземпляр структуры, к полям которого мы хотим обратиться, является объектом (не указателем), то есть имеет типовую метку структуры, можно обратиться к свойствам структуры через оператор `.` (точка).

определение структуры для хранения рационального числа

объявление экземпляра структуры **rational**

обращение к полям структуры **rational** – присваиваем значения

```
int main() {  
    struct rational {  
        int numerator, denominator;  
    };  
  
    struct rational q;  
    q.numerator = 1;  
    q.denominator = 2;  
    return 0;  
}
```





# Обращение к внутренним данным структуры

Если для работы со структурой используется указатель на экземпляр структуры, то прежде чем использовать оператор `.` (точка), необходимо разыменовать указатель оператором `*` (звёздочка/asterisk). Однако оператор разыменования имеет более низкий приоритет, чем оператор обращения к полю структуры, соответственно операцию разыменования следует брать в скобки, что приводит к громоздкому синтаксису:

в функцию передаётся указатель на экземпляр структуры

```
int summ(struct rational *q) {  
    return (*q).numerator + (*q).denominator;  
}
```

разыменовываем указатель,  
операцию берём в круглые скобки

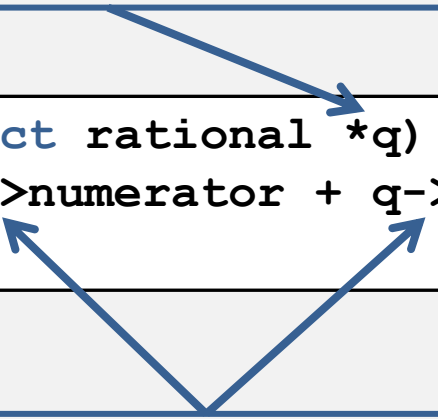
после разыменования обращаемся к  
полю структуры через оператор точки

# Обращение к внутренним данным структуры

Избежать сложных конструкций при обращении к свойствам структуры помогает специальный оператор стрелки `->`, который записывается в виде двух символов: знака минус и закрывающейся угловой скобки. Оператор стрелки позволяет обращаться к полю структуры без разыменования и имеет такой же приоритет, как и оператор точки.

в функцию передаётся указатель на экземпляр структуры

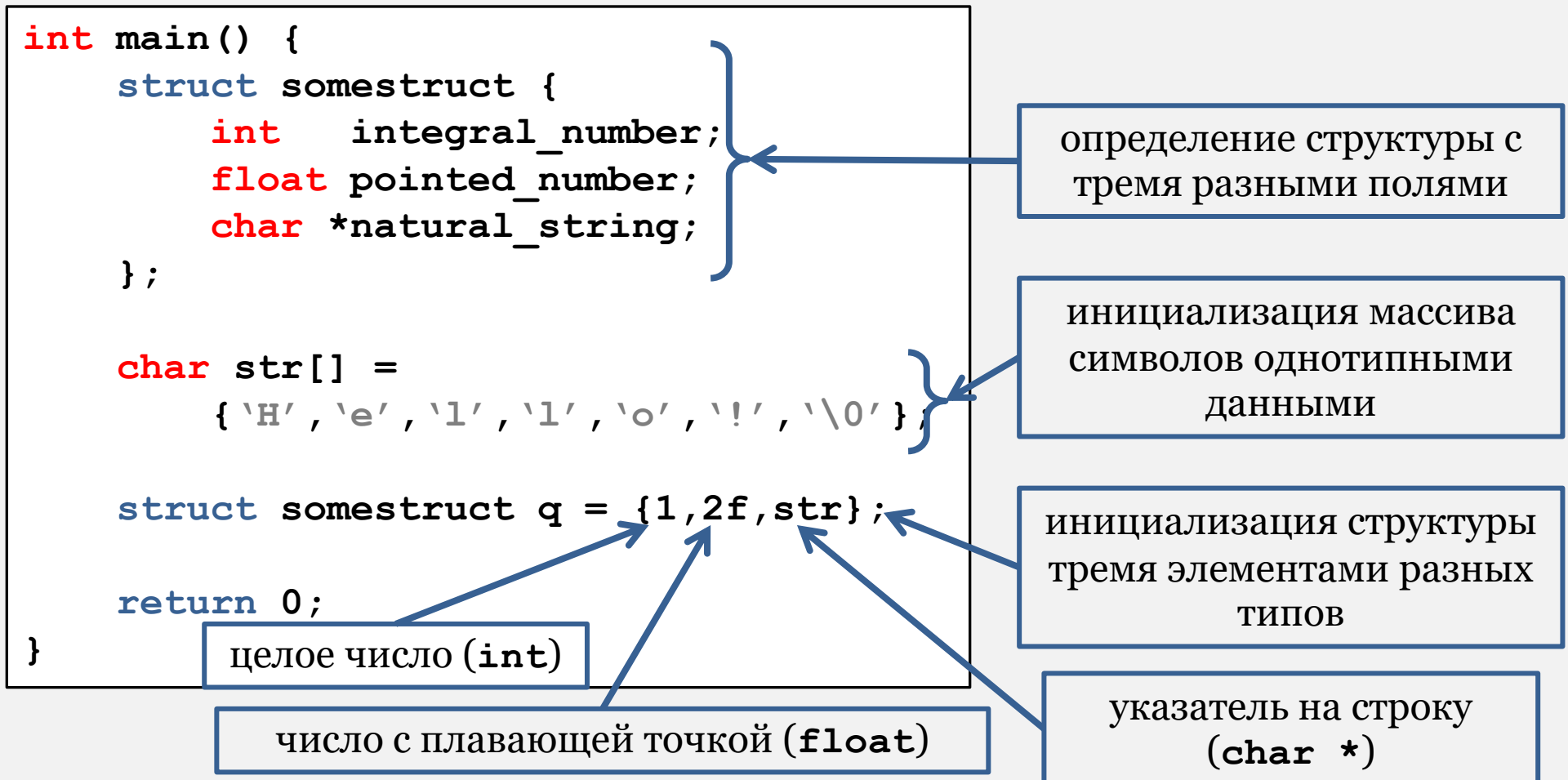
```
int summ(struct rational *q) {  
    return q->numerator + q->denominator;  
}
```



обращаемся к свойству структуры, на которую указывает переменная, без явного разыменования

# Инициализация структур

Как и массив, структура может быть инициализирована кортежем значений, тип которых соответствует типам полей структуры. Порядок данных в списке инициализации должен соответствовать порядку объявления переменных в определении структуры.



# Сравнение структур

В отличие от фундаментальных, встроенных типов между структурами нет правил сравнения и операции `==`, `!=`, `>` и `<` не определены. Для типов, определённых пользователем необходимо реализовывать собственные функции сравнения.

```
struct rational
{
    int numerator, denominator;
};

int eq_rational (struct rational *lha, struct rational *rha)
{
    return ( lha->numerator*rha->denominator ==
             rha->numerator*lha->denominator );
}

int gt_rational (struct rational *lha, struct rational *rha)
{
    return ( lha->numerator*rha->denominator >
             rha->numerator*lha->denominator );
}
```

# Примеры использования структур

```
struct rational {
    int numerator, denominator;
};

struct rational summ(struct rational lha, struct rational rha)
{
    struct rational q;
    q.numerator    = lha.numerator*rha.denominator
                    + rha.numerator*lha.denominator;
    q.denominator = lha.denominator*rha.denominator;

    return q;
}

int main()
{
    struct rational q = {1,2}, p = {1,2};
    struct rational s = summ(p,q);

    printf("1/2 + 1/2 = %d/%d\n",s.numerator,s.denominator);
}
```

$1/2 + 1/2 = 4/4$