

Структура как пользовательский тип и совокупность данных

Определение шаблона структуры и структурной переменной

Структура (структурный тип, шаблон структуры) – это тип, определяемый пользователем с использованием ключевого слова `struct`. Если массив – это совокупность однородных элементов (элементов одного типа), то структура – это *объединение в единое целое множества поименованных элементов (полей данных), в общем случае, разных типов, имеющее для простоты одно имя.*

Определение структуры (структурного типа, шаблона структуры) задает внутреннюю организацию структурных переменных после их определения и никак не связано с резервированием памяти компилятором, **не создает никаких переменных**. Структурный тип только задает правила формирования структурной переменной, используемые компилятором *при последующем определении структурной переменной* для выделения ей места в памяти и организации доступа к ее полям. Структурный тип имеет уникальное имя в пределах области определения. Имя структурного типа используется при определении структурных переменных аналогично встроенным типам.

Объявление структуры

Структуры языка C++ представляют поименованную совокупность компонентов, называемых **полями**, или **элементами структуры**. Элементом структуры может быть:

- переменная любого допустимого типа;
- битовое поле;
- функция.
- Объявление структуры имеет следующее формальное описание:
- `struct [имя_структуры] { тип_элемента_структуры имя_элемента1; тип_элемента_структуры имя_элемента2; ... тип_элемента_структуры имя_элементаN; } [список_объявляемых_переменных];`

Определение структуры состоит из двух шагов:

- объявление структуры (задание нового типа данных определенным пользователем), структура состоит из полей;

```
struct student
{
char fio[30]; // определено поле fio
char group[8]; // определено поле group
int year;
int informatika, math, fizika, history;
}
```

- определение переменных типа структура;

```
student Vasya, ES[50];
```

```
struct date{
```

```
int day;
```

```
int month;
```

```
int year;
```

```
char mon_name[10];
```

```
};
```

▲ имя (тег) структуры

элементы структуры (поля, члены-данные)

Возможно неполное объявление структуры, имеющее следующее формальное описание:

```
struct имя_структуры;
```

При отсутствии имени объявляемой структуры создается анонимная структура. При создании анонимной структуры обычно указывается список объявляемых переменных.

Список объявляемых переменных типа данной структуры может содержать:

- имена переменных;
- имена массивов;
- указатели.

Например: `struct sA {char a[2]; int i;} struA, struB[10], *struC;`

Синтаксис определения структуры (структурного типа, шаблона структуры):

```
struct pattern_name { type1 field_name1;  
                      type2 field_name2;  
                      .....
```

```
                      typeN field_nameN;  
};
```

или, в случае полей одного типа:

```
struct pattern_name { type1 field_name1, field_name2;  
                      .....
```

```
                      typeK field_nameN;  
};
```

где:

- pattern_name – имя шаблона, удовлетворяющее правилам задания идентификаторов языка;
- type1, type2, ... typeN – любые типы;
- field_name1, field_name2, ..., field_nameN – имена полей, удовлетворяющие правилам задания идентификаторов языка

Примеры определений структур:

```
//1
struct PointRecType //структура «точка»
    { int x, y;
};

//2
struct ColorPointRecType //структура «цветная точка»
    { int x, y;
      unsigned color;
};

//3
struct Book //структура «книга»
    {
      char name [20];
      chat title[44];
      int year;
      float price;
};

//4
struct Auto //структура «автомобиль»
    {
      char fio[40]; //ФИО владельца
      char adres[60]; //адрес владельца
      int nomer; //номер авто
      char marka[20]; //марка авто
    }
```

```
//5
struct Student //структура «студент»
{
    unsigned short группа;
    char fio[20];
    unsigned short ocenki[KOL_OC]; //массив оценок
};
```

```
//6
struct Rectangle //структура «прямоугольник»
{
    int left; //пара координат верхней левой точки
    int top;
    int right; //пара координат нижней правой точки
    int bottom;
};
```

```
//7
struct Box //структура «ящик»
{
    double length; //длина
    double width; //ширина
    double heigth; //высота
};
```


Некоторые функции из библиотеки обработки символов `ctype.h`

<i>Прототип функции</i>	<i>Описание функции</i>
<code>int isdigit(int c)</code>	Возвращает значение true , если <code>c</code> является цифрой, и false в других случаях
<code>int isalpha(int c)</code>	Возвращает значение true , если <code>c</code> является буквой, и false в других случаях
<code>int isalnum(int c)</code>	Возвращает значение true , если <code>c</code> является цифрой или буквой, и false в других случаях
<code>int islower(int c)</code>	Возвращает значение true , если <code>c</code> является буквой нижнего регистра, и false в других случаях
<code>int isupper(int c)</code>	Возвращает значение true , если <code>c</code> является буквой верхнего регистра, и false в других случаях
<code>int tolower(int c)</code>	Если <code>c</code> является буквой верхнего регистра, то результат – буква нижнего регистра, в других случаях возвращается аргумент без изменений
<code>int toupper(int c)</code>	Если <code>c</code> является буквой нижнего регистра, то результат – буква верхнего регистра, в других случаях возвращается аргумент без изменений

Некоторые функции преобразования строк из библиотеки `stdlib.h`

<i>Прототип функции</i>	<i>Описание функции</i>
<code>double atof(const char *s)</code>	Преобразует строку <code>s</code> в тип <code>double</code>
<code>int atoi(const char *s)</code>	Преобразует строку <code>s</code> в тип <code>int</code>
<code>long atol(const char *s)</code>	Преобразует строку <code>s</code> в тип <code>long int</code>

Некоторые функции из библиотеки string.h

<i>Прототип функции</i>	<i>Описание функции</i>
<code>size_t strlen(const char *s)</code>	Вычисляет длину строки <code>s</code> в байтах
<code>char *strcat(char *s1, const char *s2)</code>	Присоединяет строку <code>s1</code> в конец строки <code>s2</code>
<code>char *strcpy(char *s1, const char *s2)</code>	Копирует строку <code>s1</code> в место памяти, на которое указывает <code>s2</code>
<code>char *strncat(char *s1, const char *s2, size_t maxlen)</code>	Присоединяет строку <code>maxlen</code> символов строки <code>s2</code> в конец строки <code>s1</code>
<code>char *strncpy(char *s1, const char *s2, size_t maxlen)</code>	Копирует <code>maxlen</code> символов строки <code>s2</code> в место памяти, на которое указывает <code>s1</code>
<code>char * strstr(char *s1, char *s2)</code>	отыскивает позицию первого вхождения строки <code>s2</code> в строку <code>s1</code>
<code>int strcmp(const char *s1, const char *s2)</code>	сравнивает две строки в лексикографическом порядке с учетом различия прописных и

<i>Прототип функции</i>	<i>Описание функции</i>
	строчных букв, возвращает отрицательное число, если s1 располагается в упорядоченном по алфавиту порядке раньше, чем s2, и положительное в противном случае, функция возвращает 0, если строки совпадают.
<code>int stricmp(const char *s1, const char *s2)</code>	сравнивает две строки в лексикографическом порядке не различая прописные и строчные буквы, возвращает отрицательное число, если s1 располагается в упорядоченном по алфавиту порядке раньше, чем s2, и положительное в противном случае, функция возвращает 0, если строки совпадают.
<code>char *strtok(char *s1, char *s2)</code>	Последовательный вызов функции разбивает строку s1 на лексемы, разделенные символами, содержащимися в строке s2. При первом вызове функция получает в качестве аргумента строку s1, в последующих вызовах в качестве аргумента передается NULL. При каждом вызове возвращается указатель на текущую лексему строки s1, когда лексем не осталось, возвращается NULL.