**Как правильно читать объявления в Си**

**http://habrahabr.ru/post/116255/**

Даже совсем зеленые программисты на Си, не испытывают проблем с чтением таких объявлений:  
int foo[5]; // foo массив из 5 элементов типа int  
char \*foo; // foo указатель на char  
double foo(); // foo функция возвращающая значение типа double  
  
Но как только объявления становятся немного сложнее, проблематично точно сказать что это. Например:  
char \*(\*(\*\*foo[][8])())[];   
  
  
Оказывается, что правила чтения произвольно совокупных объявлений легко учатся даже начинающими программистами (хоть и невозможно использовать такую объявленную переменную).

### Основные и производные типы

В дополнении к имени переменной, объявление состоит из одного основного типа и может содержать еще и производный тип, и это ключ к пониманию различий между ними.  
  
Основные типы:  
**• char   
• signed char   
• unsigned char   
• short   
• unsigned short   
• int   
• unsigned int   
• long   
• unsigned long   
• float   
• double  
• long double   
• void   
• struct tag   
• union tag   
• enum tag   
• long long   
• unsigned long long**   
  
Объявление может содержать только **один основной тип**, и он всегда находится слева выражения. Основные типы дополняются производными типами, в Си их три:  
  
1) **\* — указатель на ...**  
Обозначается символом \*, и важно понимать что указатель **всегда** на что-нибудь указывает.  
  
2)**[] — массив из…**   
Массив может быть безразмерный — [], а может быть и размерный [10]. Правда размерный массив или нет, это неважно при чтении объявлений (обычно все же пишется размер массива). Должно быть понятно что массив всегда «массив из чего-нибудь».  
  
3) **() — функция возвращающая ...**  
Обычно обозначается парой круглых скобок (), но также возможно что внутри их будут модели параметров.Список параметров (если он есть) не играет существенной роли при чтении объявлений, и мы его обычно игнорируем. Заметим, что круглые скобки используемые для обозначения функций, отличаются от скобок служащих для группировки: группирующие скобки окружают переменные, тогда как скобки для обозначения функция находятся справа.Функция не имеет смысла если она ничего не возвращает(когда мы объявляем функцию с возвращаемым типом значения void, то это просто выглядит как будто функция возвращает значения типа void)  
  
Производные типы **всегда** что то модифицируют, будь то основной тип или производный, и что бы правильно читать объявления, всегда нужно вставлять предлог («на», «из», «возвращающая»). Используя при чтении «указатель» вместо «указатель на», вы точно прочитаете объявление неправильно.

### Приоритет операторов.

Почти каждый программист Си знаком с таблицами приоритетов операторов, в которых говорится что (например) умножение и деление имеют более высокий приоритет (выполняются раньше) чем сложение и вычитание, и группирующие скобки используются для изменения этого приоритета. Это кажется нормальным для «обычных» выражений, но те же правила применимы и к объявлениям — они просто «типовые», а не вычислительные.   
Операторы «массив из»**[]** и «функция возвращающая»**()** имеют более высокий приоритет чем «указатель на», что приводит к довольно простому правилу декодирования:  
**Всегда начинайте с имени переменной:**  
  
**foo — это .....**  
  
**И заканчивайте декодирование основным типом:**  
  
**… типа int**  
  
То, что будет в середине, обычно сложнее разобрать, но можно сформулировать правило:  
**двигайтесь вправо, если это возможно, и перемещайтесь влево если это необходимо**   
Начиная с имени переменной, соблюдая правила приоритета, двигайтесь вправо насколько это возможно вычеркивая лексемы, пока не дойдете до группирующих скобок. После же перемещайтесь налево в соответствии со скобками.

### Простой пример.

Давайте начнем с простого примера:  
-> **long \*\*foo[7];**  
  
Давайте попробуем разобраться, сосредоточившись на одной или двух частях, выделяя их **жирным шрифтом**, а то с чем мы уже определились будем ~~зачеркивать~~  
  
-> long \*\*foo [7];  
  
Начинаем с имени переменной и заканчиваем основным типом:  
foo — это… типа long  
  
Разбираем дальше:  
-> ~~long~~ \*\*~~foo~~[7];  
  
В данной момент имя переменной окружает лексема значащая «массив из 7 » и лексема значащая «указатель на», и в соответствии с правилом двигаемся вправо и дописываем к нашему описанию «массив из 7 »:  
foo — это массив из 7 … типа long  
  
-> ~~long~~ \*\*~~foo[7]~~;  
  
Вправо больше некуда двигаться, а ближайшая лексема это «указатель на». Добавим её:   
foo — это массив из 7 указателей на… значение типа long  
  
-> ~~long~~ \*~~\*foo[7]~~;  
  
Ближайшая лексема так же «указатель на», добавим и её:  
foo — это массив из 7 указателей на указатели на значение типа long  
  
Ну вот и все.

### Сложный пример

Чтобы проверить наши навыки, нам нужно попробовать прочитать очень сложное объявление, которое никогда не встретится в реальной жизни (на самом деле мы очень долго думали как можно применить это объявление). Но нужно показать что правила работают и для очень сложных деклараций.  
  
-> char \*(\*(\*\*foo [][8])())[];  
Все объявления стоит начинать читать с «имя переменной… основной тип»   
foo — это… типа char;  
  
->~~char~~ \*(\*(\*\*~~foo~~ [][8])())[];  
К имени примыкают «указатель на» и «массив из», идем вправо:  
foo — это массив из… типа char;  
  
~~char~~ \*(\*(\*\*~~foo[]~~[8])())[];  
Мы можем выбрать правую или левую примыкающую лексему, но правило гласит что, необходимо двигаться вправо насколько это возможно, пока к внутренней части группирующих скобок что нибудь примыкает, поэтому идем вправо.  
foo — это массив из массив из… типа char;  
  
->~~char~~ \*(\*(\*\* ~~foo[][8]~~)())[];  
Мы дошли до группирующих скобок, и дальше двигаться направо не представляется возможным, поэтому двигаемся влево пока не дойдем до парной группирующей скобки, чтоб вычеркнуть все остальные лексемы.  
foo — это массив из массив из указателей на… типа char;  
  
->~~char~~ \*(\*(\* ~~\*foo[][8]~~)())[];  
Снова двигаемся влево и приписываем «указатель на».   
foo — это массив из массив из указателей на указатели на… типа char;  
  
->~~char~~ \*(\* ~~(\*\* foo[][8])~~())[];  
После того как мы дописали «указатель на» в предыдущем шаге, мы дошли до парной группирующей скобки, так что продолжим присоединять и к «группирующим скобкам». Сейчас к ним примыкает«функция возвращает» справа и «указатель на» слева. Двигаемся вправо.  
foo — это массив из массив из указателей на указатели на функцию возвращающую… типа char;  
  
->~~char~~ \*(\* ~~(\*\* foo[][8])()~~)[];  
Мы снова уперлись в группирующие скобки, поэтому снова возвращаемся налево.  
foo — это массив из массив из указателей на указатели на функцию возвращающую указатели на… типа char;  
  
->~~char~~ \*~~(\*(\*\* foo[][8])())~~[];  
Обойдя группирующие скобки, видим что сейчас к вычеркнутым лексемам примыкает «массив из» справа и «указатель на» слева, «массив из» находится справа, добавим.  
foo — это массив из массив из указателей на указатели на функцию возвращающую указатели на массив из… типа char;  
  
->~~char~~ \*~~(\*(\*\* foo[][8])())[]~~;  
Ну и добавляем последнюю лексему.  
foo — это массив из массив из указателей на указатели на функцию возвращающую указатели на массив из указателей на тип char;  
  
Мы правда не знаем как это применить, но описание типа корректно.

### Абстрактные объявления

Стандарт Си позволяет использовать абстрактные объявления, когда тип должен быть объявлен, но не связан с именем переменной. Это используется при приведении типов, и как аргумент **sizeof** — иногда это выглядит ужасающе:   
  
**int (\*(\*)())()**;  
  
Естественно возникает вопрос с чего же начать, так вот ответ будет звучать так «надо найти место, где будет стоять имя переменной и рассматривать как обычное объявление». Такое место будет только одно, и найти его на самом деле очень просто. Используя правила синтаксиса, которые мы знаем:  
  
•справа от всех лексем «указатель на»   
•слева от всех лексем «массив из»   
•слева от всех лексем «функция возвращает»  
•внутри всех группирующих скобок  
  
А теперь посмотрим на пример. Мы видим что левый набор лексем «указатель на» устанавливает одну границу и правый набор лексем «функция возвращает» устанавливает другую границу.  
~~int (\*(\*~~ •)• ~~())()~~;  
  
Красные точки • показывают куда можно поместить имя переменной, но только одно место удовлетворяет условиям (внутри группирующих скобок). И что же у нас тогда с объявлением? А вот что:  
  
**int (\*(\*foo)())()**;  
которое наши правила описывают как:  
foo — это указатель на функцию возвращающую указатель на функцию возвращающую значение типа int

### Семантические ограничения / Примечания

Не все комбинации производных типов допускаются. Возможно создать объявления, прекрасно вписывающееся в синтаксические правила, но которые тем не менее будут ошибочны (будут правильны синтаксически, но ошибочны семантически, например)  
  
**• Невозможно создать массив функций**  
Но зато можно использовать массив указателей на функцию  
  
**•Функция не может возвращать функцию**  
Но может возвращать указатель на функцию  
  
**•Функция не может вернуть массив**  
Опять таки функция может вернуть указатель на массив  
  
**•В массивах только левая лексема [] может быть пустой**  
Си поддерживает многомерные массивы (например foo[1][2][3][4]), представляющие собой очень простую структуру данных. Однако, когда массив имеет больше чем одно измерение, то только первые скобки могут быть пустыми. **char foo[]** и **char foo[][5]** имеют право на существование, а вот char **foo[5][]** уже запрещено  
**•Тип «void» ограниченный**  
Тип «void» -это псевдо-тип, и переменные такого типа могут быть только «указатель на» и «функция возвращающая». Запрещено (точнее невозможно) использовать «массив из void» и просто переменные типа «void».  
void \*foo; //разрешено  
void foo(); //разрешено  
void foo; //запрещено  
void foo[]; //запрещено

### Добавление типа соглашения вызова

При разработке на платформе windows, часто добавляется, к описанию функции соглашение вызова.Это указывает компьютеру какой метод использовать для вызова функции в запросе, и метод должен быть таким же, какой и ожидает функция. Вот как это выглядит:  
  
extern int \_\_cdecl main(int argc, char \*\*argv);  
  
extern BOOL \_\_stdcall DrvQueryDriverInfo(DWORD dwMode, PVOID pBuffer,  
DWORD cbBuf, PDWORD pcbNeeded);  
  
Такое добавление очень часто встречается в разработке под win32, оно достаточно простое для понимания. Больше информации в статье [Использование соглашения вызова win32](http://www.unixwiz.net/techtips/win32-callconv.html) .  
  
Где это становится каким-то более сложным, так это когда соглашение вызова должно быть включено в «указатель» (включая typedef), потому, что лексема не выглядит так, чтобы соответствовать нормальной схеме.Это часто используется когда речь идет о работе с **LoadLibrary()** и **GetProcAddress()** API для обращения к вызову функции из недавно загруженной библиотеки.  
Это можно часто встретить с typedef:   
typedef BOOL (\_\_stdcall \*PFNDRVQUERYDRIVERINFO)(  
DWORD dwMode,  
PVOID pBuffer,  
DWORD cbBuf,  
PDWORD pcbNeeded  
);  
  
...  
  
/\* get the function address from the DLL \*/  
pfnDrvQueryDriverInfo = (PFNDRVRQUERYDRIVERINFO)  
GetProcAddress(hDll, "DrvQueryDriverInfo")  
  
Согласование вызова это атрибут функции, а не указателя, поэтому при чтении это нужно ставить перед указателем, но все равно внутри группирующих скобок:  
  
**BOOL (\_\_stdcall \*foo)(...);**  
  
Читается:  
foo — это указатель на \_\_stdcall функцию возвращающую BOOL.   
  
p.s. О неточностях пишите, пожалуйста, в личку.

* [программирование](http://habrahabr.ru/search/?q=%5B%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%5D&target_type=posts)
* , [объявления](http://habrahabr.ru/search/?q=%5B%D0%BE%D0%B1%D1%8A%D1%8F%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F%5D&target_type=posts)
* , [Си](http://habrahabr.ru/search/?q=%5B%D0%A1%D0%B8%5D&target_type=posts)

+91

9370

240

[Stephen J. Friedl](http://www.unixwiz.net/techtips/reading-cdecl.html)

[IamKarlson](http://habrahabr.ru/users/IamKarlson/) 10,8

Похожие публикации

[Программирование по-японски, или что думает Microsoft об идентификаторах C++?](http://habrahabr.ru/post/238627/) сегодня в 13:19

[Программирование на клавиатуре… фортепиано](http://habrahabr.ru/post/238037/) 25 сентября в 16:14

[Kefir.js — новая библиотека для функционального реактивного программирования (FRP) в JavaScript](http://habrahabr.ru/post/237495/) 21 сентября в 03:11

[Изучаем C++ через программирование игр](http://habrahabr.ru/post/237375/) 19 сентября в 12:19

[Кружок по программированию. Итоги первого года](http://habrahabr.ru/post/235825/) 8 сентября в 10:54

[Первая Всесоюзная олимпиада школьников по программированию (информатике) 1988 года](http://habrahabr.ru/post/235089/) 2 сентября в 18:09

[BBC планирует обучать детей программированию с помощью проекта Bitesize и ТВ-шоу](http://habrahabr.ru/post/235299/) 2 сентября в 11:09

[Каким оружием могли бы быть разные языки программирования](http://habrahabr.ru/post/235281/) 2 сентября в 09:29

[Я не знаю Си](http://habrahabr.ru/post/136283/) 15 января 2012 в 23:28

[Актуален ли будет блог о программировании на Си под \*nix системы?](http://habrahabr.ru/post/55561/) 26 марта 2009 в 11:33