Атрибуты переменных в C

Хохлов С.А.

МГТУ им. Н.Э. Баумана

20 сентября 2025 г.

Атрибуты переменных в C

[Deitel2014] с. 151-157

- имя
- тип
- размер
- значение
- класс хранения
- продолжительность хранения
- область видимости
- связывание

Базовые типы

Базовые типы данных I

- char
  - 8 бит минимум (на практике почти всегда 8)
  - может быть signed или unsigned (зависит от реализации)
- short int / short
  - минимум 16 бит
  - обычно 2 байта на современных платформах
- int
  - минимум 16 бит
  - чаще всего 4 байта на 32-битных и 64-битных системах, но на старых МК может быть 2 байта

Базовые типы данных II

- long int / long
  - минимум 32 бита
  - на 32-битных системах — обычно 4 байта
  - на 64-битных UNIX — 8 байт (LLP64/LP64 различие: в LP64 (Linux, macOS, Unix) long расширяется до 64 бит, а в LLP64 (Windows 64-bit) остаётся 32-битным).
- long long int / long long
  - минимум 64 бита
  - почти всегда 8 байт

Базовые типы данных III

- Для любого целого базового типа (char, short, int, long, long long) есть вариант с unsigned.
  - разрядность совпадает с соответствующим знаковым типом.
  - все биты идут под значение, диапазон сдвигается от [0 ... 2^N-1] вместо [-2^(N-1) ... 2^(N-1)-1].
- Есть также вариант с signed – явное указание “знаковости”, полезно для char.

Базовые типы данных IV

- float
  - обычно 32 бита (формат IEEE-754 single precision)
- double
  - минимум как float, чаще 64 бита (IEEE-754 double precision)
- long double
  - не фиксировано стандартом, зависит от компилятора и платформы:
  - на x86 GCC – 80 бит (x87),
  - на ARM/Windows – часто просто 64 бита как double.

Базовые типы данных V

- Важно: стандарт C гарантирует только минимальные размеры и относительные соотношения:

sizeof(char) <= sizeof(short) <= sizeof(int) <=

<= sizeof(long) <= sizeof(long long)

Переносимые типы данных из <stdint.h> (C99) I

Нужны для того, чтобы уйти от неоднозначностей int/long на разных платформах.

- Фиксированной ширины (если поддерживаются платформой):
  - int8_t / uint8_t — ровно 8 бит
  - int16_t / uint16_t — ровно 16 бит
  - int32_t / uint32_t — ровно 32 бита
  - int64_t / uint64_t — ровно 64 бита
- Минимальной ширины:
  - int_least8_t / uint_least8_t — не меньше 8 бит (может быть шире, если так быстрее/удобнее для CPU)
  - int_least16_t / uint_least16_t — не меньше 16 бит
  - int_least32_t / uint_least32_t — не меньше 32 бит
  - int_least64_t / uint_least64_t — не меньше 64 бит

Переносимые типы данных из <stdint.h> (C99) II

- Наибольшей скорости (оптимальные):
  - int_fast8_t / uint_fast8_t — «самый быстрый» $\geq$ 8 бит
  - int_fast16_t / uint_fast16_t — $\geq$ 16 бит
  - int_fast32_t / uint_fast32_t — $\geq$ 32 бит
  - int_fast64_t / uint_fast64_t — $\geq$ 64 бит
- Особые:
  - intptr_t / uintptr_t — целое, способное хранить указатель (ширина равна размеру адресации)
  - intmax_t / uintmax_t — максимально широкие целые типы, доступные в реализации

Главное отличие от «базовых» типов: здесь чётко фиксирована ширина в битах или хотя бы гарантированный минимум, поэтому код переносим между архитектурами.

Логический тип

- В C99 был добавлен логический тип _Bool.
- Дополнительный заголовочный файл <stdbool.h> определяет для него псевдоним bool, а также макросы true (истина) и false (ложь).
- _Bool ведёт себя так же, как обычный встроенный тип, за одним исключением: любое ненулевое (не ложное) присваивание _Bool хранится как единица.
- Такое поведение защищает от переполнения и позволяет сравнивать любые не ложные значения:

```
_Bool a = 1, b = 2;

if (a == b) {
    /* this code will run */
}
```

Классы хранения

Спецификаторы и классы хранения I

Спецификаторы классов хранения

- auto
- register (запрещена операция взятия адреса)
- extern
- static

Классы хранения

- Автоматический
- Статический

Автоматический класс хранения I

- Автоматический класс хранения переменной объявляется с помощью ключевого слова auto (а также register).
- Автоматические переменные создаются при входе в блок, где они объявляются.
- Они продолжают существовать, пока данный блок выполняется, и уничтожаются при выходе программы из блока.

Автоматический класс хранения II

- Автоматический класс хранения обычно получают локальные переменные функций.
- Ключевое слово auto позволяет явно объявить автоматический класс хранения.
- Локальные переменные получают автоматический класс по умолчанию, поэтому ключевое слово auto редко используется на практике.

Статический класс хранения I

- Ключевые слова extern и static используются в объявлениях имен переменных и функций со статическим классом хранения.
- Идентификаторы со статическим классом хранения существуют на протяжении всего времени выполнения программы до ее завершения.

Статический класс хранения II

- Память для статических переменных выделяется и инициализируется только один раз, перед тем как программа начнет выполняться. Имена функций также продолжают существовать вплоть до завершения программы.
- Однако тот факт, что имена переменных и функций продолжают существовать до самого завершения программы, еще не означает, что они доступны из любой точки программы.

Статический класс хранения. Внешние идентификаторы

- Имена глобальных переменных и функций по умолчанию получают класс хранения extern.
- Глобальные переменные создаются размещением их объявлений за пределами любых функций, и они сохраняют свои значения на протяжении всего времени выполнения программы.
- Глобальные переменные и функции доступны любым функциям, определения которых следуют за объявлениями этих переменных и функций.

Статический класс хранения. Локальные переменные static I

- доступны только внутри функций, где они определяются, но, в отличие от автоматических переменных, статические локальные переменные сохраняют свои значения после выхода из функции.
- Когда функция будет вызвана в следующий раз, она обнаружит в статической локальной переменной то же значение, которое было оставлено в ней в момент последнего выхода.
- Следующая инструкция объявляет локальную переменную count статической и инициализирует ее значением 1: `static int count = 1;`

Статический класс хранения. Локальные переменные static II

```
#include <stdio.h>
void test(void) {
    static int count = 1;
    printf("%i\n", count);
    count++;
}
int main() {
    test();
    test();
    return 0;
}
```

Статический класс хранения. Локальные переменные static III

- Все числовые переменные со статическим классом хранения по умолчанию инициализируются нулями, если явно не будут указаны иные значения.
- При применении к внешним идентификаторам ключевые слова extern и static получают специальное значение.

Продолжительность хранения

Продолжительность хранения

- период времени, в течение которого идентификатор существует в памяти
- некоторые идентификаторы существуют очень короткий период времени, некоторые многократно создаются и уничтожаются, а иные существуют на протяжении всего времени выполнения программы

Область видимости

Область видимости идентификаторов I

- область программы, где данный идентификатор доступен для обращения к нему
- некоторые идентификаторы доступны в любой точке программы, другие – только в ограниченной области

Область видимости файла I

- Идентификатор, объявленный за пределами какой-либо функции, получает область видимости файла.
- Такой идентификатор будет «известен» (то есть доступен) из любых функций, объявления которых находятся в том же файле, ниже строки с объявлением идентификатора.
- Глобальные переменные, определения функций и прототипы функций, находящиеся за пределами функций, получают область видимости файла.

Область видимости блока I

- Идентификаторы, объявленные внутри блока, получают область видимости блока.
- Область видимости блока заканчивается в конце блока в позиции закрывающей фигурной скобки (}).
- Локальные переменные, объявленные в начале функции, получают область видимости блока, как и параметры функции, которые считаются локальными переменными функции.
- Любой блок может содержать определения переменных.

Область видимости блока II

- Когда один блок вложен в другой и идентификатор, объявленный во вложенном блоке, совпадает с идентификатором, объявленным во внешнем блоке, идентификатор во внешнем блоке становится недоступен, пока вложенный блок не завершит выполнения. Это означает, что внутри вложенного блока доступен только его собственный локальный идентификатор.
- Локальные переменные, объявленные с ключевым словом static, также получают область видимости блока, даже при том, что они начинают свое существование с момента запуска программы. То есть класс хранения не влияет на область видимости идентификатора.

Область видимости прототипа функции

- Единственными идентификаторами, получающими область видимости прототипа функции , являются идентификаторы в списке параметров внутри прототипа.
- Прототипы функций не требуют наличия имен в списке параметров – обязательными являются только типы. Если имена присутствуют в списке параметров в прототипе функции, они игнорируются компилятором.
- Идентификаторы, указанные в прототипе функции, можно использовать в любом месте в программе, не рискуя породить конфликта имен.

Область видимости функции

метки в switch и goto

Связывание

Связывание

- Под связыванием идентификатора понимается его доступность в одном или во всех исходных файлах программы при соответствующем объявлении
- При применении к внешним идентификаторам ключевые слова extern и static получают специальное значение.
- Существует возможность компилировать программы, состоящие из нескольких файлов с исходными текстами.

Прототипы функций I

- Подобно тому, как объявления extern позволяют распространить область действия глобальных переменных на несколько файлов, входящих в состав программы, объявления прототипов функций дают возможность использовать эти функции за пределами файлов, где они определены (указывать спецификатор extern в прототипах функций не требуется).
- Просто включите прототип функции в каждый файл, где она вызывается, и скомпилируйте файлы вместе.

Прототипы функций II

- Прототип функции указывает компилятору, что соответствующая функция определена либо ниже в этом же файле, либо в другом файле.
- И снова компилятор не будет пытаться разрешить ссылки на такие функции, оставив решение этой задачи компоновщику. Если компоновщик не встретит подходящее определение функции, он выведет сообщение об ошибке и прервет процедуру создания выполняемого файла.

Глобальные переменные I

- Глобальные переменные доступны из любых функций, объявленных в том же файле, ниже объявления переменной. Глобальные переменные также доступны функциям, объявленным в других файлах, но при этом их необходимо явно объявлять в каждом файле, где они используются. Например, чтобы получить возможность обращаться к глобальной переменной flag в другом файле, ее необходимо объявить в этом файле, как показано ниже: `extern int flag;`

Глобальные переменные II

- Компилятор сообщит компоновщику о неразрешенной ссылке на внешнюю переменную flag. Если компоновщик найдет подходящее определение глобальной переменной, он подставит ее адрес в машинную инструкцию, обращающуюся к этой переменной.
- Если же компоновщик не найдет определение переменной flag, он выведет сообщение об ошибке и прервет процедуру создания выполняемого файла.
- Любой идентификатор, объявляемый в области видимости файла, автоматически получает класс хранения extern.

Ограничение видимости глобальных переменных и прототипов функций I

- Когда к глобальной переменной или функции применяется спецификатор класса хранения static, это предотвращает возможность использования таких переменных и функций за пределами данного файла. Это называется внутренним связыванием (internal linkage).
- Глобальные переменные и функции, определяемые без спецификатора static, поддерживают внешнее связывание (external linkage) – они доступны в других файлах, при наличии в этих файлах соответствующих объявлений.

Ограничение видимости глобальных переменных и прототипов функций II

- Определение глобальной переменной

  ```
  static const double PI = 3.14159;
  ```

  создаст неизменяемую переменную PI типа double, инициализированную значением 3.14159 и доступную только функциям в файле, где объявлена эта переменная.

Ограничение видимости глобальных переменных и прототипов функций III

- Спецификатор static часто применяется при создании вспомогательных функций, используемых лишь в пределах данного файла.
- Если функция не нужна за пределами текущего файла, в соответствии с принципом наименьших привилегий ее следует объявить с классом static.
- Если функция определяется до ее использования в файле, спецификатор static следует применить к определению функции, в противном случае – к прототипу.

Классы хранения и связывание

Ключевое слово	Класс хранения	Связывание
`auto`	автоматический (в стеке, живёт внутри блока)	не влияет
`static` (локальная переменная)	статический (живёт всё время работы программы)	не влияет
`static` (глобальная переменная или функция)	статический	внутреннее (видно только в этом файле)
без ключевого слова (глобальная переменная/функция)	статический	внешнее (видно из других файлов)
`extern`	не выделяет память, только объявление	внешнее (символ определяется в другом месте)