Спецификаторы класса памяти

Стандарт C поддерживает четыре спецификатора класса памяти:

```
extern
static
register
auto
```

Эти спецификаторы сообщают компилятору, как он должен разместить соответствующие переменные в памяти. Общая форма объявления переменных при этом такова:

спецификатор_класса_памяти тип имя переменой;

Спецификатор класса памяти в объявлении всегда должен стоять первым.

На заметку	Стандарты C89 и C99 из соображений удобства синтаксиса утверждают, что `typedef` – это спецификатор класса памяти. Однако `typedef` не является собственно спецификатором. Подробнее `typedef` рассматривается в книге далее.

Спецификатор extern

Прежде чем приступить к рассмотрению спецификатора `extern`, необходимо коротко остановиться на компоновке программы. В языке C при редактировании связей к переменной может применяться одно из трех связываний: внутреннее, внешнее или же не относящееся ни к одному из этих типов. (В последнем случае редактирование связей к ней не применяется.) В общем случае к именам функций и глобальных переменных применяется внешнее связывание. Это означает, что после компоновки они будут доступны во всех файлах, составляющих программу. К объектам, объявленным со спецификатором `static` и видимым на уровне файла, применяется внутреннее связывание, после компоновки они будут доступны только внутри файла, в котором они объявлены. К локальным переменным связывание не применяется и поэтому они доступны только внутри своих блоков.

Спецификатор `extern` указывает на то, что к объекту применяется внешнее связывание, именно поэтому они будут доступны во всей программе. Далее нам понадобятся чрезвычайно важные понятия объявления и описания. *Объявление* (*декларация*) объявляет имя и тип объекта. *Описание*[1] выделяет для объекта участок памяти, где он будет находиться. Один и тот же объект может быть объявлен неоднократно в разных местах, но описан он может быть только один раз.

В большинстве случаев объявление переменной является в то же время и ее описанием. Однако, если перед именем переменной стоит спецификатор `extern`, то объявление переменной может и не быть ее описанием. Таким образом, если нужно сослаться на переменную, определенную в другой части программы, необходимо объявить ее как внешнюю (`extern`).

Приведем пример использования спецификатора `extern`. Обратите внимание, что глобальные переменные `first` и `last` объявлены *после* `main()`.

```
#include <stdio.h>

int main(void)
{
  extern int first, last; /* используются глобальные переменные */

  printf("%d %d", first, last);

  return 0;
}
```

```
/* описание глобальных переменных first и last */
int first = 10, last = 20;
```

Программа напечатает `10 20`, потому что глобальные переменные `first` и `last` инициализированы этими значениями. Объявление `extern` сообщает компилятору, что переменные `first` и `last` определены в другом месте, поэтому программа компилируется без ошибки, несмотря даже на то, что `first` и `last` используются до своего описания.

Обратите внимание, в этом примере объявление переменных со спецификатором `extern` необходимо только потому, что они не были объявлены до `main()`. Если бы их объявление встретилось перед `main()`, то в объявлении со спецификатором `extern` не было бы необходимости.

При компиляции выполняются следующие правила. Если компилятор находит переменную, не объявленную внутри блока, он ищет ее объявление во внешних блоках. Если не находит ее и там, то ищет среди объявлений глобальных переменных. В предыдущем примере, если бы не было объявления `extern`, компилятор не нашел бы `first` и `last` среди глобальных переменных, потому что они объявлены после `main()`. Здесь спецификатор `extern` сообщает компилятору, что эти переменные будут объявлены в файле позже.

Как сказано выше, спецификатор `extern` позволяет объявить переменную, не описывая ее. Но если в объявлении со спецификатором `extern` инициализировать переменную, то это объявление становится также и описанием. При этом программист обязательно должен учитывать, что объект может иметь много объявлений, но лишь одно описание.

Спецификатор `extern` играет большую роль в программах, состоящих из многих файлов. В языке С программа может быть записана в нескольких файлах, которые компилируются раздельно, а затем компонуются в одно целое. В этом случае необходимо как-то сообщить всем файлам о глобальных переменных программы. Самый лучший (и наиболее переносимый) способ сделать это – определить (описать) все глобальные переменные в одном файле и объявить их со спецификатором `extern` в остальных файлах, как показано на рис. 2.1.

Файл 1

```
int x, y;
char ch;

int main(void)
{
  /* ... */
}

void func1(void)
{
  x = 123;
}
```

Файл 2

```
extern int x, y;
extern char ch;

void func22(void)
{
  x = y / 10;
}

void func23(void)
{
  y = 10;
}
```

Рис. 2.1. Использование глобальных переменных в раздельно компилируемых модулях

Во втором файле спецификатор `extern` сообщает компилятору, что эти переменные определены в других файлах. Таким образом компилятор узнает имена и типы переменных, размещенных в другом месте, и может отдельно компилировать второй файл, ничего не зная о первом. При компоновке этих двух модулей все ссылки на глобальные переменные будут разрешены.

На практике программисты обычно включают объявления `extern` в заголовочные файлы, которые просто подключаются к каждому файлу исходного текста программы. Это более легкий путь, который к тому же приводит к меньшему количеству ошибок, чем повторение этих объявлений вручную в каждом файле.

На заметку	Спецификатор `extern` можно применять в объявлении функций, но в этом нет необходимости.

Спецификатор static

Переменные, объявленные со спецификатором `static`, хранятся постоянно внутри своей функции или файла. В отличие от глобальных переменных они невидимы за пределами своей функции или файла, но они сохраняют свое значение между вызовами. Эта особенность делает их полезными в общих и библиотечных функциях, которые будут использоваться другими программистами. Спецификатор `static` воздействует на локальные и глобальные переменные по-разному.

Локальные статические переменные

Для локальной переменной, описанной со спецификатором static, компилятор выделяет в постоянное пользование участок памяти, точно так же, как и для глобальных переменных. Коренное отличие статических локальных от глобальных переменных заключается в том, что статические локальные переменные видны только внутри блока, в котором они объявлены. Говоря коротко, статические локальные переменные – это локальные переменные, сохраняющие свое значение между вызовами функции.

Статические локальные переменные очень важны при создании функций, работающих отдельно, так как многие процедуры требуют сохранения некоторых значений между вызовами. Если бы не было статических переменных, вместо них пришлось бы использовать глобальные, подвергая их риску непреднамеренного изменения другими участками программы. Рассмотрим пример функции, в которой особенно уместно применение статической локальной переменной. Это – генератор последовательности чисел, каждое из которых зависит только от предыдущего. Для хранения числа между вызовами можно использовать глобальную переменную. Однако тогда при каждом использовании функции придется объявлять эту переменную и, что особенно неудобно, постоянно следить за тем, чтобы ее объявление не конфликтовало с объявлениями других глобальных переменных. Значительно лучшее решение – объявить эту переменную со спецификатором static:

```
int series(void)
{
  static int series_num;

  series_num = series_num+23;
  return series_num;
}
```

В этом примере переменная series_num продолжает существовать между вызовами функций, в то время как обычная локальная переменная создается заново при каждом вызове, а затем уничтожается. Поэтому в данном примере каждый вызов series() генерирует новое число, зависящее от предыдущего, причем удается обойтись без глобальных переменных.

Статическую локальную переменную можно инициализировать. Это значение присваивается ей только один раз – в начале работы всей программы, но не при каждом входе в блок программы, как обычной локальной переменной. В следующей версии функции series() статическая локальная переменная инициализируется числом 100:

```
int series(void)
{
  static int series_num = 100;

  series_num = series_num+23;
  return series_num;
}
```

Теперь эта функция всегда будет генерировать последовательность, начинающуюся с числа 123. Однако во многих случаях необходимо дать пользователю программы возможность ввести первое число вручную. Для этого переменную series_num можно сделать глобальной и предусмотреть возможность задания начального значения. Если же отказаться от объявления переменной series_num в качестве глобальной, то необходимо ее объявить со спецификатором static.

Глобальные статические переменные

Спецификатор static в объявлении глобальной переменной заставляет компилятор создать глобальную переменную, видимую только в том файле, в котором она объявлена. Статическая глобальная переменная, таким образом, подвергается внутреннему связыванию, как описано ранее в разделе "Спецификатор extern". Это значит, что хоть эта переменная и глобальная, тем не менее процедуры в других файлах не увидят ее и не смогут случайно изменить ее значение. Этим снижается риск нежелательных побочных эффектов. А в тех относительно редких случаях, когда для выполнения задачи статическая локальная переменная не подойдет, можно создать небольшой отдельный файл, который содержит только функции, в которых используется эта статическая глобальная переменная. Затем этот файл необходимо откомпилировать отдельно; тогда можно быть уверенным, что побочных эффектов не будет.

В следующем примере иллюстрируется применение статической глобальной переменной. Здесь генератор последовательности чисел переделан так, что начальное число задается вызовом другой функции, series_start():

```
/* Это должно быть в одном файле
     отдельно от всего остального. */

static int series_num;
void series_start(int seed);
int series(void);
```

```
int series(void)
{
  series_num = series_num+23;
  return series_num;
}

/* инициализирует переменную series_num */
void series_start(int seed)
{
  series_num = seed;
}
```

Вызов функции series_start() с некоторым целым числом в качестве параметра инициализирует генератор series(). После этого можно генерировать последовательность чисел путем многократного вызова series().

Обзор: Имена локальных статических переменных видимы только внутри блока, в котором они объявлены; имена глобальных статических переменных видимы только внутри файла, в котором они объявлены.

Если поместить функции series() и series_num() в библиотеку, то уже нельзя будет сослаться на переменную series_num, она оказалась спрятанной от любых операторов всей остальной программы. При этом в программе (конечно, в других файлах) можно объявить и использовать другую переменную под именем series_num. Иными словами, спецификатор static позволяет создать переменную, видимую только для функций, в которых она нужна, что исключает нежелательные побочные эффекты.

Таким образом, при разработке больших и сложных программ для "сокрытия" переменных можно применять спецификатор static.

Спецификатор register

Первоначально спецификатор класса памяти register применялся только к переменным типа int, char и для указателей. Однако стандарт C расширил использование спецификатора register, теперь он может применяться к переменным любых типов.

В первых версиях компиляторов C спецификатор register сообщал компилятору, что переменная должна храниться в регистре процессора, а не в оперативной памяти, как все остальные переменные. Это приводит к тому, что операции с переменной register осуществляются намного быстрее, чем с обычными переменными, потому такая переменная уже находится в процессоре и не нужно тратить время на выборку ее значения из оперативной памяти (и на запись в память).

В настоящее время определение спецификатора register существенно расширено. Стандарты C89 и C99 попросту декларируют "доступ к объекту так быстро, как только возможно". Практически при этом символьные и целые переменные по-прежнему размещаются в регистрах процессора. Конечно, большие объекты (например, массивы) не могут поместиться в регистры процессора, однако компилятор получает указание "позаботиться" о быстродействии операций с ними. В зависимости от конкретной реализации компилятора и операционной системы переменные register обрабатываются по-разному. Иногда спецификатор register попросту игнорируется, а переменная обрабатывается как обычная, однако на практике это бывает редко.

Спецификатор register можно применить только к локальным переменным и формальным параметрам функций. В объявлении глобальных переменных применение спецификатора register не допускается. Ниже приведен пример использования переменной, в объявлении которой применен спецификатор register; эта переменная используется в функции возведения целого числа m в степень. (Степень – натуральное число – представлена идентификатором e.)

```
int int_pwr(register int m,  register int e)
{
  register int temp;

  temp = 1;

  for(; e; e--) temp = temp * m;
  return temp;
}
```

В этом примере в объявлениях к переменным e, m и temp применен спецификатор register потому, что они используются внутри цикла. Переменные register идеально подходят для оптимизации скорости работы цикла. Как правило, переменные register используются там, где от них больше всего пользы, а именно, когда процесс многократно обращается к одной и той же переменной. Это существенно потому, что в объявлении можно применить спецификатор register к любой переменной, но средства оптимизации быстродействия могут быть применены далеко не ко всем переменным в равной степени.

Максимальное количество переменных `register`, оптимизирующихся по быстродействию, зависит от среды программирования и конкретной реализации компилятора. Если таких переменных окажется слишком много, то компилятор автоматически преобразует регистровые переменные в нерегистровые. Этим обеспечивается переносимость программы в широком диапазоне процессоров.

Обычно в регистры процессора можно поместить как минимум две переменные типа `char` или `int`. Однако в различных средах программирования режимы оптимизации могут очень отличаться, поэтому выбор режима оптимизации необходимо осуществлять с учетом особенностей конкретного компилятора.

В языке C с помощью оператора `&` (рассматривается в этой главе далее) нельзя получить адрес регистровой переменной, потому что она может храниться в регистре процессора, который обычно не имеет адреса.

Хотя в настоящее время применение спецификатора `register` в значительной мере вышло за его традиционные рамки, практически ощутимый эффект от его применения по-прежнему может быть получен только для переменных целого и символьного типа. Не следует ожидать заметного повышения скорости от объявления регистровыми переменных других типов.

---

[1] Синонимы: *определение*, *дефиниция*.

Содержание | <<< | >>>
[Главная] [Гостевая]
