English Version
Die deutsche Version
Карта сайта

Поделиться

...
Расширенный поиск

Нашли опечатку?

Пожалуйста, сообщите об этом - просто выделите ошибочное слово или фразу и нажмите **Shift Enter**.

Блог одного Сумасшествия

GCC: inline-функция, работающая так же быстро, как макрос

Добавил(а) microsin

Использованием встраивания (**inline**) для функции Вы можете дать указание для **GCC** сделать вызовы функции быстрее. Один из путей, которым GCC может достичь этого - вставить код тела функции в то место, где функция вызывается. Это быстрее обрабатывается процессором, потому что устраняются расходы на вызов функции и возврат из неё. Кроме того, если любой из действительных аргументов функции константа, то их известные значения могут быть учтены на этапе компиляции, в результате чего может быть встроено не все тело функции. В результате получится, что размер кода будет менее предсказуемым; объектный код может увеличиться или уменьшиться с применением встраивания функций, в зависимости от каждого конкретного случая. Вы можете также указать для GCC пытаться интегрировать все "достаточно простые" функции в места их вызова, если используете опцию -finline-functions. Здесь приведены переводы статей [1, 2].

GCC реализует 3 разные семантики декларирования inline-функций. Одна из них доступна с опцией -std=gnu89 или -fgnu89-inline, или когда присутствует атрибут для всех inline-деклараций, другой при использовании опций -std=c99, -std=c11, -std=gnu99 или -std=gnu11 (без -fgnu89-inline), и третья с использованием компилирования в режиме C++.

Чтобы декларировать встроенную функцию, используйте в её декларации ключевое слово inline, примерно так:

```
static inline intinc (int *a)
{
  return (*a)++;
}
```

Если Вы пишете заголовочный файл, подключаемый в программах ISO C90, вместо ключевого слова inline используйте __inline__ (подробнее см. врезку "Замена ключевых слов").

Замена ключевых слов

Эти три типа встраивания ведут себя подобным образом в двух важных случаях: когда ключевое слово inline используется в статической (static) функции, Наподобие в вышеприведенном примере, и когда функция сначала была декларирована без ключевого слова inline, и затем была определена как inline, примерно так:

```
extern int inc (int *a);

inline int inc (int *a)
{
  return (*a)++;
}
```

В обоих этих случаях программы ведут себя так же, как если бы Вы не использовали ключевое слово inline, за исключением того, что изменится скорость выполнения.

Когда функция определена и как inline и как static, если все вызовы функции в виде её тела интегрированы в вызывающий код, и адреса функции не используются, то нет ссылок к собственному ассемблерному коду функции. Тогда GCC не делает вывод актуального кода для функции, за исключением случая, когда Вы укажете опцию -fkeep-inline-functions. Если здесь сделан вызов без встраивания, то функция будет скомпилирована в ассемблерный код, как обычно. Эта функция также должна быть скомпилирована как обычная функция, если программа обращается к ней по адресу функции, потому что она тогда не может быть встраиваемой.

Обратите внимание, что некоторые использования определений функции делают невозможным inline-подстановки. Например: variadic-функции (функции с переменным количеством аргументов), использование alloca, используемого вычисляемого goto (см. врезку "Метки как значения"), использование не локального goto, использование вложенных (nested) функций, использование setjmp, использование __builtin_longjmp и использование __builtin_return или __builtin_apply_args. Опция -Winline будет предупреждать о ситуациях, когда функция помечена как inline, но для неё не может быть осуществлено встраивание, и также выдает причину этого.

Метки как значения

Как требуется стандартом ISO C++, GCC считает функции - члены класса (т. е. функции, определенные в теле класса) помеченными как inline, даже если они не декларированы явно с ключевым словом inline. Вы можете отменить это поведение использованием опции -fno-default-inline (подробнее см. описание опций, управляющих диалектом C++ [3]).

GCC не делает встраивание любых функций, когда не делается оптимизация, за исключением если Вы укажете атрибут always_inline для функции, примерно так:

```
/* Прототип. */
inline void foo (const char) __attribute__((always_inline));
```

Остальная часть этой секции относится к встраиванию GNU C90.

Когда inline-функция не является статической (без атрибута static), тогда компилятор должен подразумевать, что здесь могут иметь место вызовы из других исходных файлов; поскольку глобальный символ может быть определен только в одном месте программы, эта функция не должна быть определена в других исходных файлах, так что вызовы функции не могут быть встроенными. Таким образом, не статические функции всегда компилируются в выделенный код, как обычно.

Если Вы укажете в определении функции и inline и extern, то это определение будет использовано только для встраивания. Функция ни в коем случае не будет скомпилирована как отдельный код, даже если Вы обращаетесь к ней явно по её адресу. Такой адрес становится внешней ссылкой, как будто Вы только декларировали функцию, и не определили её.

Эта комбинация inline и extern дает эффект макроса. Способ использования - поместить определение функции с этими ключевыми словами в заголовочный файл, и поместить другую копию определения (без inline и extern) в библиотечный файл. Определение в заголовочном файле приведет к тому, что большинство вызовов функции будет встроенными. Если остается любое использование функции, то будет обращение к ней как к единственной копии в библиотеке.

[**Встраиваемые функции в языке C в контексте портируемости кода**]

У компилятора GNU C (и некоторых других компиляторов) имелись подставляемые (или, как их иногда называют, встраиваемые, inline) функции задолго до того, как они в языке C стали стандартом (стандарт 1999 года). В этой статье сведены в кучу правила использования встраиваемых (inline) функций, и даны некоторые советы по практическому использованию inline-функций.

В точке применения встраиваемой функции компилятору делается подсказка, что нужно предпринять некие действия для того, чтобы вызвать функцию быстрее, чем это было бы сделано как обычно. Чаще всего эти действия заключаются в том, что в месте вызова будет подставлено тело функции без оператора вызова. Поскольку при такой подстановке уже не требуются инструкции для вызова (call) и возврата (ret), а также не надо сохранять в стеке локальные переменные функции и используемые регистры, то это дает компилятору выполнить определенные оптимизации при объединении кода функции и основной программы.

Иногда необходимо для компилятора сгенерировать автономную копию объектного кода для функции даже тогда, когда эта функция встраиваемая - например, когда нужно взять адрес функции, или если функция не может быть встроена в каком-то частном контексте, или когда (возможно) выключена оптимизация. (И конечно, если используете компилятор, который не понимает inline, Вам нужно иметь отдельную копию объектного кода, чтобы все вызовы действительно могли работать.)

Есть несколько способов определить inline-функции; любой имеющийся вид определения мог бы определенно сгенерировать автономный объектный код, или определенно не генерировать автономный объектный код, или сгенерировать автономный объектный код только если известно, что это необходимо. Иногда это может привести к дублированию объектного кода, который является потенциальной проблемой по следующим причинам:

• Впустую расходуется память.
• Может привести к тому, что указатели на одну и ту же функцию не будут равны друг другу.
• Может снизиться эффективность кэша инструкций (хотя встраивание также могло быть сделано другими способами).

Если любое из перечисленного является для Вас проблемой, то можно использовать стратегию для обхода дублирования кода. Это как раз обсуждается в статье.

Что такое единица трансляции (translation unit)

[**Правила для inline стандарта C99**]

Спецификация для встраивания описана в секции 6.7.4 стандарта C99 (ISO/IEC 9899:1999). К сожалению, эта информация не находится в свободном доступе. Имеются следующие возможности.

C99 inline rules

1. Функция, у которой все её декларации (включая определения) помечены как inline и никогда как extern. Она должна быть определена в пределах одной единицы трансляции (translation unit). Стандарт ссылается на этот вариант как встроенное определение (inline definition). Не будет сгенерирован отдельный объектный код, так что это определение не может быть вызвано из другой единицы трансляции (из другого модуля).

Вы можете[2] иметь отдельное (не inline) определение той же функции в другой единице трансляции, и компилятор может выбрать это отдельное определение или inline-определение.

Такие функции не могут содержать модифицируемых статических переменных, и не могут обращаться к статическим определениям или функциям в другом исходном файле (т. е. не там, где эти inline-функции были объявлены).

В этом примере все декларации и определения используют inline, но не используют extern:

```
// Декларация, помеченная как inline
inline int max(int a, int b);

// Определение, помеченное как inline
```

```
inline int max(int a, int b) {
  return a > b ? a : b;
}
```

Эту функцию нельзя вызывать из других файлов; вместо этого в другом файле должно быть свое определение.

Примечание (2): в стандарте нет четкого описания этого. Он говорит, что inline-определение не запрещает внешнее (external) определение где-то в другом месте, но тогда это предоставляет альтернативу для внешнего определения. К сожалению, это не дает четкого понимания - должно ли это внешнее определение существовать фактически. На практике, если Вы не поставили себе целью замучить компилятор, то будет существовать следующее правило: если хотите сохранить свою встроенную функцию полностью приватной для одного юнита трансляции, то делайте её определение как static inline.

2. Функция, где как минимум одна декларация помечена как inline, но где некоторые другие декларации не упоминают inline, или упоминаются как extern. Определение функции должно быть в том же юните трансляции, что и декларации. Будет сгенерирован отдельный объектный код (наподобие, как для обычной функции) и его можно вызывать из других юнитов трансляции Вашей программы.

Здесь также применяется ограничение для статического определения, уже упомянутое выше.

В этом примере все декларации и определения используют inline, но одно добавляет extern:

```
// Декларация, упомянутая как extern и inline
extern inline int max(int a, int b);

// Определение, упомянутое как inline
inline int max(int a, int b) {
  return a > b ? a : b;
}
```

В этом примере одна из деклараций не помечено как inline:

```
// Декларация, не помеченная inline
int max(int a, int b);

// Определение, помеченное inline
inline int max(int a, int b) {
  return a > b ? a : b;
}
```

В любом из этих двух примеров функция может быть вызвана из других файлов.

3. Функция определена как static inline. Может быть выпущено локальное определение, если это необходимо. У Вас может быть несколько определений в программе, в разных юнитах трансляции, и все они будут работать как отдельные функции. Простое отбрасывание inline снижает переносимость программы (опять же, при прочих равных условиях).

Это может быть полезным для маленьких функций, которые иначе могли бы быть определены как макросы. Если функция не всегда встраивается, то тогда Вы получаете дубликаты её кода со всеми уже описанными выше проблемами.

Разумный подход должен был бы поместить static inline функции либо в заголовочный файл, если он должен использоваться широко, либо просто в файлах исходного кода, которые используют эти функции - если они когда-либо используются в этом файле.

В этом примере функция определена как static inline:

```
static inline int max(int a, int b) {
  return a > b ? a : b;
}
```

Первые два варианта естественно сочетаются. Вы либо пишете везде inline и extern в одном месте для запроса автономного определения, или пишете inline почти везде, но опускаете его обязательно один раз, чтобы получить автономное определение.

Функция main не может быть встроенной (inline) функцией.

(Правила C++ строже: функция, которая появилась где-то как inline, должна быть везде определена как inline, и должна быть определена одинаково во всех юнитах трансляции, где используется.)

[**Правила для inline компилятора GNU C**]

Правила GNU C описаны в руководстве по компилятору GNU C, которое поставляется вместе с компилятором. Они свободно доступны на сайте http://gcc.gnu.org. Имеются следующие возможности.

GNU C inline rules

1. Функция, определенная inline самостоятельно. Всегда генерируется автономный объектный код. Вы можете написать только одно определение наподобие этого для всей программы. Если Вы хотите использовать его из других юнитов трансляции, поместите декларацию в файл заголовка; но это не будет делать встраивание в тех модулях трансляции, где используется заголовок.

Этот вариант имеет ограниченное применение: если Вы хотите использовать функцию в одном юните трансляции, больше смысла сделать её static inline, как это показано в варианте 3 - если Вы возможно не хотели бы иметь некую форму, которая позволит функции быть встроенной больше чем в один юнит трансляции.

Однако у этого варианта использования ключевого слова inline действительно есть преимущество, программа уменьшается для портируемой программы с тем же значением (если не используются никакие другие не

переносимые конструкции).

2. Функция, определенная как extern inline. Автономный объектный код не генерируется никогда. У Вас может быть несколько таких определений, и Ваша программа все еще будет работать. Однако Вы должны добавить также где-то и не-inline определение в том случае, если функция где-нибудь не используется как inline.

Это предоставляет разумную семантику (можно избежать дублирования объектного кода функции), но немного неудобно для использования.

Один из способов использования этого варианта - поместить определения в заголовочный файл, окружить оператором препроцессора #if, который будет вычисляться как true либо когда используется GNU C, либо когда заголовок был подключен из файла, который содержит выданные определения (независимо, используется или нет GNU C). В последнем случае extern опущен (например, пишут EXTERN, и определяют это через #define либо как extern, либо как пустоту). Ветка #else содержала бы просто декларации функций, для не GNU компиляторов.

3. Функция, определенная как static inline. Если требуется, будет сгенерирован автономный объектный код. Вы можете иметь несколько определений в своей программе, в разных юнитах трансляции, и это будет работать. Это тот же вариант реализации, что и для правил C99.

С релиза 4.3 компилятор GNU C поддерживает правила встраивания C99, описанные выше, и использует их по умолчанию с опциями -std=c99 или -std=gnu99. Старые правила могут быть запрошены в новых компиляторах опцией -gnu89-inline, или использованием атрибута функции gnu_inline.

Если действуют правила C99, то GCC определит макрос __GNUC_STDC_INLINE__. Начиная с GCC 4.1.3 будет определен макрос __GNUC_GNU_INLINE__, если используются только правила GCC, но старые компиляторы используют эти же правила без определения какого-либо макроса. Вы можете разрулить ситуацию, используя фрагмент кода наподобие следующего:

```
#if defined __GNUC__ && !defined __GNUC_STDC_INLINE__ && !defined __GNUC_GNU_INLINE__
# define __GNUC_GNU_INLINE__ 1
#endif
```

[**Стратегии использования inline-функций**]

Следующие правила советуют возможные модели использования inline-функций, более или менее влияя на портируемость.

Простая модель для портирования. Используйте static inline (либо в общем заголовочном файле, или просто в одном файле). Если компилятору нужно сгенерировать определение (например, чтобы получить его адрес, или потому, что он не хочет делать некоторые вызовы встроенными) то Вы потеряете некоторое место под код; если Вы возьмете адрес функции в двух юнитах трансляции, то взятые адреса не будут одинаковыми при сравнении.

Например, в заголовочном файле:

```
static inline int max(int a, int b) {
  return a > b ? a : b;
}
```

Вы можете поддерживать устаревшие компиляторы (например не имеющие inline) через опцию -Dinline="", хотя это приведет к потерям памяти, если компилятор не делает оптимизацию по неиспользуемым функциям.

Модель GNU C. Используйте extern inline в общем заголовочном файле и предоставьте определение где-нибудь в файле .c, возможно используя макрос - чтобы гарантировать появления везде одинакового кода. Например, в заголовочном файле:

```
#ifndef INLINE
# define INLINE extern inline
#endif
INLINE int max(int a, int b) {
  return a > b ? a : b;
}
```

... и точно только в одном файле исходного кода:

```
#define INLINE

#include "header.h"
```

Поддержка устаревших компиляторов будет затруднительной за исключением случая, когда Вас не заботит лишняя трата памяти, и можно иметь несколько адресов одной и той же функции; Вам нужно ограничить определения пределами одного юнита трансляции (с ключевым словом INLINE, преобразованном препроцессором в пустую строку), и добавить некоторые внешние декларации в заголовочный файл.

Модель C99. Используйте inline в общем заголовочном файле, и предоставьте определения где-нибудь в файле .c, через декларации extern. Например, в файле заголовка:

```
inline int max(int a, int b) {
  return a > b ? a : b;
}
```

... и только одном каком-нибудь файле исходного кода:

```
#include "header.h"

extern int max(int a, int b);
```

Чтобы добавить поддержку устаревших компиляторов, Вы должны все это обратить кодом для препроцессора так, чтобы декларации были видны в общем заголовке, и определения были ограничены одним юнитом трансляции, в котором определена функция через inline.

Сложная модель портирования. Используйте макрос для выбора определения либо с использованием extern inline для GNU C, inline для C99, либо без ничего из этого. Например, в заголовке:

```
#ifndef INLINE
# if __GNUC__ && !__GNUC_STDC_INLINE__
#  define INLINE extern inline
# else
#  define INLINE inline
# endif
#endif

INLINE int max(int a, int b) {
  return a > b ? a : b;
}
```

... и только в одном файле исходного кода:

```
#define INLINE

#include "header.h"
```

У поддержки устаревших компиляторов есть те же проблемы, что и с моделью GNU C.

Если Вы заметили какие-либо ошибки, пожалуйста сообщите об этом в комментариях.

[**Ссылки**]

1. Inline Functions In C site:greenend.org.uk.
2. An Inline Function is As Fast As a Macro site:gcc.gnu.org.
3. Options Controlling C++ Dialect site:gcc.gnu.org.

Комментарии

#1 **APh** 19.10.2020 14:03 **0**

зачем в Си наворотили вокруг этого столько малопонятного, когда у коллег из C++ с подстановками функций полная ясность и однозначность?

microsin: "ясности" и "однозначности" C++ это повод для очередного холивара. Всяк имеет право использовать то, что нравится, к чему привык. В реальной жизни приходится использовать как C, так и C++, так и смесь кода на этих языках.

Цитировать

Обновить список комментариев
RSS лента комментариев этой записи

Добавить комментарий

Имя (обязательное)

E-Mail (обязательное)

Сайт

Осталось: 1000 символов

Подписаться на уведомления о новых комментариях

Обновить

Отправить

JComments