6. Условия

В работе препроцессора "условием" называется директива, при выполнении которой часть программы игнорируется во время компиляции после проверки некоторых условий. В C препроцессоре в условии могут использоваться как арифметические выражения, так и имена определенных макросов.

Условие в C препроцессоре в некоторых аспектах имеет сходство с конструкцией 'if' языка C, но важно понимать их отличия. Условие в конструкции 'if' проверяется при выполнении программы. Ее целью служит изменение хода программы в зависимости от обрабатываемых данных. Условие в препроцессоре проверяется при компиляции программы. Оно используется для включения в программу различных частей кода в зависимости от условий, установленных при ее компиляции.

6.1 Для чего используются условия

Существует три основных причины для применения условий.

Для выполнения программы на различных платформах может потребоваться разный исходный код. В некоторых случаях, программа, написанная для одной операционной системы, будет некорректно работать в другой операционной системе. В подобных ситуациях недостаточно уклонения от выполнения ненужных процедур. Если программа их содержит, то часто случается, что невозможно скомпоновать и запустить программу. При использовании условной компиляции, неиспользуемый код может быть исключен из программы.

Иногда требуется скомпилировать один исходный файл в две разные программы. Случается, что различия в программах заключаются в том, что одна из них выполняет постоянную, занимающую много времени, обработку данных или выдает значения этих данных для отладки, в то время как другая не делает этого.

Применение условий, где проверка выдает заведомо ложный результат, используется для исключения кода из программы, который может являться одним из видов комментариев для ссылки на него в будущем.

В большинстве простых программ, предназначенных для выполнения только на одном компьютере, условия обычно не используются.

6.2 Синтаксис условий

Условие в C препроцессоре начинается с директивы условия: '#if', '#ifdef' или '#ifndef'. Далее рассматривается только директива '#if'.

Директива '#if'

Простейшая форма использования директивы '#if' рассмотрена ниже.

```
#if EXPRESSION
CONTROLLED TEXT
#endif /* EXPRESSION */
```

Комментарий, следующий за директивой '#endif' не является обязательным, но помогает при написании и чтении программы. Такие комментарии всегда следует использовать, за исключением небольших конструкций. В действительности, текст в строке после директивы '#endif' игнорируется GNU C препроцессором, но стандарт ANSI C требует применения комментариев.

Выражение EXPRESSION является C выражением типа integer, что представляет собой сильное ограничение. Оно может содержать:

- Целые константы, которые рассматриваются как тип 'long' или 'unsigned long'.
- Символьные константы, которые интерпретируются в соответствии с набором символов и в зависимости от компьютера и операционной системы, на которой установлен препроцессор. Для таких констант GNU C препроцессор использует тип данных 'char'. Поэтому являются ли коды некоторых символов отрицательными значениями, определяется C компилятором, который использовался для компиляции препроцессора. Если тип 'char' является знаковым, то символы, значения которых достаточно велики для установки знакового бита, рассматриваются как отрицательные значения. В противном случае тип 'char' является всегда положительным значением.
- Арифметические операции сложения, вычитания, умножения, деления, операции с битами, сдвиги, сравнения, а также логические операции ('&&' и '||').
- Идентификаторы, не являющиеся макросами и рассматриваемые как нулевое значение.
- Макровызовы. Перед вычислением значения выражения сначала производится макроподстановка.

Следует заметить, что не допускается использовать операторы 'sizeof' и значения типа 'enum'. Все значения типа 'enum', также как и все идентификаторы, не являющиеся макро вызовами, рассматриваются как нулевое значение.

Текст, находящийся внутри условной конструкции, может включать директивы препроцессора, которые обрабатываются при выполнении требуемых условий. Текст может также содержать и другие условные конструкции. Однако директивы '#if' и '#endif' должны образовывать единую конструкцию.

Директива '#else'

Директива '#else' может использоваться в условной конструкции для предоставления альтернативного кода программы втом случае, если условие ложно. Вот как это выглядит:

```
#if EXPRESSION
TEXT-IF-TRUE
#else /* Not EXPRESSION */
TEXT-IF-FALSE
#endif /* Not EXPRESSION */
```

Если значение EXPRESSION является ненулевым и используется код TEXT-IF-TRUE, то директива '#else' рассматривается как ложное условие и код TEXT-IF-FALSE игнорируется. И наоборот, если условие '#if' - ложно, то включается код TEXT-IF-FALSE.

Директива '#elif'

Обычное применение однородных условий связано с проверкой более чем двух возможных вариантов. Например:

```
#if X == 1
...
#else /* X != 1 */
#if X == 2
...
#else /* X != 2 */
...
#endif /* X != 2 */
#endif /* X != 1 */
```

Дополнительная директива '#elif' позволяет это сократить как рассмотрено ниже.

```
#if X == 1
...
#elif X == 2
...
#else /* X != 2 and X != 1*/
...
#endif /* X != 2 and X != 1*/
```

Директива '#elif' означает "иначе если" ("else if"). Также как и '#else', она помещается в середину конструкции '#if'-'#endif' и подразделяет ее. Ей не требуется наличия собственной директивы '#endif'. Также как и '#if', директива '#elif' включает в себя тестируемое выражение.

Текст, следующий за директивой '#elif' включается только в том случае, когда значение изходящей директивы '#if' - ложно, а условие '#elif' - верно. В одной конструкции '#if'-'#endif' может использоваться более чем одна директива '#elif'. Текст после директивы '#elif' включается только в том случае, когда условие '#elif' - верно и находится после условия '#if' или предшествующего '#elif', значения которых - ложь. '#else' является эквивалентом директивы '#elif 1', а '#else' может следовать после любого количества директив '#elif', в то время как '#elif' не может следовать за '#else'.

6.3 Сохранение удаленного кода для дальнейших ссылок

Если часть программы была перемещена или удалена, но есть необходимость в сохранении старого кода в качестве комментария для дальнейших ссылок к нему, то простейший способ реализации этого заключается в использовании конструкции '#if 0'-'#endif', внутри которой находится этот код. Это рациональнее применения обычных комментариев, так как это не всегда помогает, если этот код также содержит комментарии.

Такая конструкция в любом случае будет безошибочной, даже если заключенный в нее текст также содержит условия (полные конструкции '#if'- '#endif').

Однако не следует применять такую конструкцию, если комментируемый текст не является C кодом. Для этого используются обычные C комментарии. Директива '#if 0' должна состоять из правильных лексем.

6.4 Условия и макросы

Условия часто используются вместе с макросами или утверждениями, так как они являются единственными выражениями, чьи значения могут варьироваться при компиляции. Директива '#if', не использующая макросы или утверждения, является эквивалентом директиве '#if 1' или '#if 0'.

Например, рассмотрим условие, проверяющее выражение 'BUFSIZE == 1020', где 'BUFSIZE' является макросом.

```
#if BUFSIZE == 1020
  printf ("Large buffers!\n");
#endif /* BUFSIZE is large */
```

При программировании часто требуется определить размер переменной или тип данных в директиве '#if', но препроцессор не обрабатыват такие операторы как 'sizeof' или ключевые слова как 'int'.

В директиве '#if' применяется специальный оператор 'defined', используемый для проверки соответствия указанного имени существующему макросу. В любом случае, значением выражения 'defined NAME' или 'defined (NAME)' является 1, если в данном месте программы определен макрос с именем NAME, в противном случае значением выражения будет 0. Для оператора 'defined' имеет значение не определение макроса, а то что оно есть. Рассмотрим пример:

```
#if defined (vax) || defined (ns16000)
```

Здесь значением выражения будет истина, если как имя 'vax', так и 'ns16000' определены как макросы. То же самое можно выполнить с помощью утверждений:

```
#if #cpu (vax) || #cpu (ns16000)
```

Если макрос был определен, а затем уничтожен с помощью директивы '#undef', то последующее применение оператора 'defined' возвратит значение 0, так как это имя больше не определено. Если же макрос заново определен директивой '#define', то оператор 'defined' возвратит значение 1.

Условия, проверяющие определение одного имени довольно часто используются, поэтому для этой цели существует две дополнительные условные директивы.

'#ifdef NAME'

что является эквивалентом '#if defined (NAME)'.

'#ifndef NAME'

что является эквивалентом '#if ! defined (NAME)'.

Макроопределения могут меняться при разных процессах компиляции по некоторым причинам.

Некоторые макросы являются заранее определенными, в зависимости от типа используемого компьютера. Например, на компьютерах Vax, имя 'vax' является заранее определенным макросом. На других компьютерах оно не определено.

Большое количество макросов определяется системными подключаемыми файлами. На различных системах и компьютерах определяются разные макросы с разными значениями. Очень часто полезно проверять эти макросы условными конструкциями во избежание использования аппаратных возможностей на компьютере, где они не реализованы.

Макросы являются простым способом настройки пользователями программы для различных систем или приложений. Например, макрос 'BUFSIZE' может быть определен в конфигурационном файле программы, который включается в качестве подключаемого файла в каждый исходный файл. Можно использовать макрос 'BUFSIZE' в условии препроцессора для генерации кода, зависящего от выбранной конфигурации.

Макросы могут определяться или уничтожаться с помощью опций препроцессора '-D' и '-U' при компиляции программы. Можно сделать так, что один и тот же исходный файл будет скомпилирован в две различные программы путем определения нужного макроса, использования условий для проверки значения этого макроса и передачи значения макроса через опции компилятора.

Утверждения обычно являются заранее определенными, но они также могут быть определены с помощью директив или опций препроцессора.

6.5 Утверждения

"Утверждения" являются более систематической альтернативой макросам при создании условий на проверку типа компьютера или системы, используемой при компиляции программы. Утверждения обычно определены заранее, хотя они могут быть также определены директивами препроцессора или с помощью опций.

Обычно макросы не классифицируются каким-либо образом по их определению. Они могут указывать на архитектуру модели компьютера, отдельную модель компьютера, на операционную систему, ее версию или на специфические возможности конфигурации. Все это может сочетаться в одном макросе. В отличие от макросов, утверждения состоят из четко поставленного вопроса и ответа на него. Вопрос обычно называется "утверждением". Утверждение выглядит следующим образом:

```
#PREDICATE (ANSWER)
```

Для имени PREDICATE следует использовать правильно сформированный идентификатор. Значением ANSWER может быть любая последоватльность слов. Здесь все символы являются значимыми, за исключением пробелов, расположенных в начале и в конце ANSWER. Различия в пробелах в середине значения игнорируются. Не разрешается использовать символ ')' в значении ANSWER.

Далее приведен пример условия, проверяющего является ли ответ ANSWER утверждением PREDICATE:

```
#if #PREDICATE (ANSWER)
```

Для одного утверждения может существовать несколько ответов. Если ответ упущен при определении утверждения, то следует проверять, существует ли у данного утверждения какой-нибудь ответ:

```
#if #PREDICATE
```

Большинство проверяемых утверждений являются заранее определенными. GNU C предоставляет три заранее определенных утверждения: 'system', 'cpu' и 'machine'. Утверждение 'system' используется для описания типа операционной системы, 'cpu' - для описания архитектуры компьютера, а 'machine' предоставляет дополнительную информацию о компьютере. Например, в системе GNU будут верны следующие утверждения:

```
#system (gnu)
#system (mach)
#system (mach 3)
#system (mach 3.SUBVERSION)
#system (hurd)
#system (hurd VERSION)
```

а также возможно и другие. Альтернативные утверждения с более или менее подробной информацией о версии системы помогут получить ответ на вопрос о типе операционной системы.

В системе Unix существует утверждение '#system (unix)', а возможно одно из следующих: `#system (aix)', `#system (bsd)', `#system (hpux)', `#system (lynx)', `#system (mach)', `#system (posix)', `#system (svr3)', `#system (svr4)', или `#system (xpg4)' вероятно с последующей информацией о версии системы.

Другие значения для 'system' это '#system (mvs)' и '#system (vms)'.

Многие Unix C компиляторы предоставляют только один ответ на утверждение 'system': '#system (unix)', если они вообще используют утверждения.

Утверждение, ответ которого сосотоит из нескольких слов сильно отличается от утверждений с ответом из одного слова. Например, утверждение 'system (mach 3.0)' не означает, что 'system (3.0)' – верно. Это также не всегда означает, что 'system (mach)' тоже верно, но в GNU C последнее утверждение может быть использовано.

В настоящий момент возможные значения утверждений для 'cpu' являются '#cpu (a29k)', `#cpu (alpha)', `#cpu (arm)', `#cpu (clipper)', `#cpu (convex)', `#cpu (elxsi)', `#cpu (tron)', `#cpu (h8300)', `#cpu (i370)', `#cpu (i386)', `#cpu (i860)', `#cpu (i960)', `#cpu (m68k)', `#cpu (m88k)', `#cpu (mips)', `#cpu (ns32k)', `#cpu (hppa)', `#cpu (pyr)', `#cpu (ibm032)', `#cpu (rs6000)', `#cpu (sh)', `#cpu (sparc)', `#cpu (spur)', `#cpu (tahoe)', `#cpu (vax)', `#cpu (we32000)'.

В C программе можно создавать свои утверждения с помощью директивы '#assert' следующим образом:

```
#assert PREDICATE (ANSWER)
```

(следует заметить отсутствие симола '#' перед PREDICATE.)

При каждом выполнении этой директивы создается новый правильный ответ для PREDICATE. При утверждении одного ответа предыдущие значения остаются в силе. Единственный способ удалить утверждение – использовать директиву '#unassert'. Эта директива имеет такой же систаксис как и '#assert'. Можно удалить все утверждения для PREDICATE следующим образом:

```
#unassert PREDICATE
```

Также имеется возможность добавления или удаления утверждений с помощью опций при вызове 'gcc' или 'cpp'.

6.6 Директивы '#error' и '#warning'

Директива '#error' вынуждает препроцессор сделать отчет о фатальной ошибке. Все что следует после '#error' используется для сообщения.

Директива '#error' в теле условия, проверяющего комбинацию параметров, не до конца поддерживаемых программой, используется для сообщения о возможной ошибке. Например, если известно, что программа не совсем корректно выполняется на системе Vax, то можно написать:

```
#ifdef __vax__
#error Won't work on Vaxen.  See comments at get_last_object.
#endif
```

Если имеется несколько конфигурационных параметров, которые должны быть указаны соответствующим образом при установке, можно использовать условия для определения несоответствия и выдать сообщение об ошибке. Например,

```
#if HASH_TABLE_SIZE % 2 == 0 || HASH_TABLE_SIZE % 3 == 0 \
    || HASH_TABLE_SIZE % 5 == 0
#error HASH_TABLE_SIZE should not be divisible by a small prime
#endif
```

Директива '#warning' аналогична директиве '#error', но приводит к тому, что препроцессор выдает предупреждающее сообщение и продолжает обработку. Все что, что следует после '#warning' используется для сообщения.

Эту директиву можно использовать в устаревших подключаемых файлах с указанием на новую версию файла.

Вперед Назад Содержание