

Макропроцессор m4 - обзор

Последнее обновление: 2021-03-03

В этом разделе описывается макропроцессор **m4**, который в операционной системе играет роль препроцессора для всех языков программирования.

В начале программы вы можете определить символьное имя или символьную константу как строку символов. Затем с помощью макропроцессора **m4** можно заменить все вхождения символьного имени (без кавычек) соответствующей строкой. Помимо замены одной строки текста на другую макропроцессор **m4** может выполнять следующие операции:

- Арифметические операции
- Операции с файлами
- Обработка условных макрокоманд
- Обработка строк и подстрок

Макропроцессор **m4** обрабатывает алфавитно-цифровые строки, которые называются *лексемами*. Макропроцессор **m4** считывает алфавитно-цифровую лексему и проверяет, не совпадает ли она с именем макроопределения. Если да, то программа подставляет вместо имени макрокоманды тело макроопределения и помещает полученную строку обратно во входной поток для повторного просмотра. При вызове макрокоманды можно указывать аргументы; в этом случае значения аргументов подставляются в тело макроопределения перед его повторным просмотром.

Макропроцессор **m4** обрабатывает стандартные макрокоманды, например, **define**. Кроме того, вы можете создать собственные макроопределения. Стандартные и пользовательские макрокоманды обрабатываются одинаково.

Работа с макропроцессором m4

Для вызова макропроцессора **m4** введите следующую команду:

```
m4 [файл]
```



Макропроцессор **m4** обрабатывает все аргументы по порядку. Если аргументы отсутствуют, либо указан аргумент - (дефис), **m4** считывает данные из стандартного ввода. Результаты работы макропроцессора **m4** отправляются в стандартный вывод. Если вы хотите сохранить их для дальнейшего использования, перенаправьте вывод в файл:

```
m4 [file] >файл-вывода
```



Создание пользовательских макроопределений

Макрокоманда	Описание
define (имя, текст)	Создает новое макроопределение с заданным <i>именем</i> и со значением <i>текст-замещения</i> .

Например, если в программе задан оператор

```
define(name, stuff)
```

то макропроцессор **m4** присвоит строке `name` значение `stuff`. Если в файле программы встретится строка `name`, то макропроцессор **m4** подставит вместо нее строку `stuff`. Строка `name` должна быть алфавитно-цифровой строкой ASCII, начинающейся с буквы или символа подчеркивания. Вместо `stuff` можно задать любой текст; однако если этот текст содержит скобки, то число открывающих скобок, должно быть равно числу закрывающих скобок. Для переноса текста **stuff** на другую строку укажите символ `/` (косая черта).

Сразу за словом **define** должна стоять открывающая скобка. Например:

```
define(N, 100)
. . .
if (i > N)
```

определяет, что `N` равно `100`, после чего символьная константа `N` используется в операторе **if**.

Вызов макрокоманды в программе должен быть задан в следующем формате:

```
name(arg1,arg2, . . . argn)
```

Имя макроопределения считается таковым только в том случае, если оно не окружено алфавитно-цифровыми символами. В следующем примере переменная `NNN` никак не связана с макроопределением `N`.

```
define(N, 100)
. . .
if (NNN > 100)
```

В макроопределениях можно использовать имена других макроопределений. Например:

```
define(N, 100)
define(M, N)
```

определяет, что и M, и N равны 100. Если затем изменить определение N и присвоить ей другое значение, то значение M по-прежнему останется равным 100, а не N.

Как только макропроцессор **m4** встречает макрокоманду, он подставляет вместо нее текст замещения, указанный в макроопределении. Строка N заменяется на 100. Затем строка M также заменяется на 100. Результат не изменится, если сначала задать

```
define(M, 100)
```



а затем указать

```
define(M, N)
define(N, 100)
```



Теперь вместо M будет подставляться N, поэтому значение M всегда будет равно текущему значению N (в приведенном примере N равно 100).

Использование кавычек

Для того чтобы запретить замену аргументов **define**, их следует заключить в кавычки. По умолчанию для этого используются одинарные кавычки ` и ' (левая и правая кавычка). Для имени, указанного в кавычках, макроподстановка не выполняется. При подстановке самого имени кавычки удаляются. Значение строки в кавычках - это строка без кавычек. Например, если ввести

```
define(N, 100)
define(M, `N')
```



то при обработке аргумента удаляются кавычки вокруг N. За счет кавычек M определяется как строка N, а не как строка 100. В общем случае макропроцессор **m4** всегда удаляет один уровень кавычек при обработке лексемы. Это справедливо даже вне макроопределения. Для того чтобы в выводе команды сохранилось слово **define**, заключите его в кавычки:

```
`define' = 1;
```



Другой пример использования кавычек задает переопределение N (N указано в кавычках). Например:

```
define(N, 100)
. . .
define(`N', 200)
```



Для того чтобы избежать ошибок, заключайте в кавычки первый аргумент макроопределения. Например, в следующем фрагменте программы аргумент *N* не будет переопределен:

```
define(N, 100)
. . .
define(N, 200)
```

N во втором определении заменяется на *100*. Результат выполнения этих двух команд будет эквивалентен следующему оператору:

```
define(100, 200)
```

Макропроцессор **m4** игнорирует его, поскольку в качестве аргумента может быть задано только имя, но не число.

Переопределение кавычек

Обычно в качестве служебных символов применяются одинарные кавычки ``` и `'` (левая и правая). Если по каким-то причинам их использовать нельзя, замените их на другие символы с помощью стандартной макрокоманды

Макрокоманда	Описание
changequote (<i>l</i> , <i>r</i>)	Эта команда позволяет заменить левую и правую кавычки символами, указанными вместо переменных <i>l</i> и <i>r</i> .

Для того чтобы восстановить исходные функции кавычек, введите команду **changequote** без аргументов:

```
changequote
```

Аргументы

Простейший способ обработки макрокоманд - замена одной строки другой (фиксированной) строкой. Однако макрокоманда может содержать аргументы, позволяющие влиять на результат макрорасширения. Аргументы указываются в тексте замещения макроопределения (во втором аргументе) в формате **\$*n*** (*n* - номер аргумента). При обработке макрокоманды процессор **m4** подставляет вместо символа значение указанного аргумента. Например, символ

```
$2
```

обозначает второй аргумент макрокоманды. Следовательно, если задать следующее макроопределение с именем `bump`:

```
define(bump, $1 = $1 + 1)
```

то макропроцессор **m4** генерирует код для увеличения первого аргумента на 1. Выражение `bump(x)` равносильно выражению `x = x + 1`.

В макрокоманде можно указывать любое число аргументов. Однако с помощью конструкции `$n` можно задать только 9 аргументов (с `$1` по `$9`). Если вам требуется большее число аргументов, воспользуйтесь макрокомандой **shift**.

Макрокоманда	Описание
shift (<i>список-параметров</i>)	Возвращает все, за исключением первого, элементы списка <i>параметры</i> .

Эта макрокоманда удаляет первый аргумент и переприсваивает оставшиеся аргументы параметрам `$n` (второй аргумент - параметру `$1`, третий - `$2`. . . десятый - `$9`). Выполнив несколько макрокоманд **shift**, можно перебрать все аргументы макроопределения.

Аргумент `$0` возвращает имя макроопределения. Если аргумент не указан в макрокоманде, он заменяется на пустую строку. Например, вы можете задать макроопределение, объединяющее аргументы следующим образом:

```
define(cat, $1$2$3$4$5$6$7$8$9)
```

Так,

```
cat(x, y, z)
```

- то же самое, что и

```
xyz
```

Аргументы с `$4` по `$9` в этом примере представляют собой пустые строки, поскольку их значения не указаны.

Макропроцессор **m4** отбрасывает указанные без кавычек начальные пробелы, символы табуляции и символы новой строки, но сохраняет все остальные непечатаемые символы. Так,

```
define(a, b c)
```

определяет `a` как строку `b c`.

Аргументы разделяются запятыми. Если аргумент содержит запятые, его необходимо заключить в скобки, чтобы запятая не рассматривалась как разделитель. Например:

```
define(a, (b,c))
```

указано только два аргумента. Первый аргумент - `a`, второй - `(b,c)`. Для того чтобы в качестве значения задать запятую или круглую скобку, заключите ее в кавычки.

Стандартные макрокоманды m4

Макропроцессор **m4** предоставляет набор стандартных предопределенных макрокоманд. В этом разделе приведено описание этих макрокоманд.

Удаление макроопределений

Макрокоманда	Описание
<code>undefine</code> (`имя`)	Удаляет определение пользовательской или стандартной макрокоманды с указанным (`именем`)

Например:

```
undefine(`N')
```

удаляет определение `N`. Если с помощью **undefine** удалить стандартную макрокоманду, например:

```
undefine(`define')
```

то вы уже не сможете использовать эту макрокоманду.

Для того чтобы избежать подстановки, необходимо указывать имя в кавычках.

Проверка наличия макроопределения

Макрокоманда	Описание
<code>ifdef</code> (`имя`, аргумент-1,	Если макроопределение с указанным <i>именем</i> существует, и его значение не равно нулю, то возвращает значение <i>аргумента1</i> . В

Макрокоманда	Описание
<i>аргумент-2)</i>	противном случае возвращает <i>аргумент2</i> .

В макрокоманде **ifdef** предусмотрено три аргумента. Если первый аргумент определен, то значение **ifdef** равно второму аргументу. Если первый аргумент не определен, то значение **ifdef** равно третьему аргументу. Если третий аргумент не указан, **ifdef** возвращает пустое значение.

Арифметические операции над целыми числами

Макропроцессор **m4** предоставляет набор встроенных функций для выполнения арифметических действий над целыми числами:

Макрокоманда	Описание
incr (<i>число</i>)	Увеличивает <i>число</i> на 1.
decr (<i>число</i>)	Уменьшает <i>число</i> на 1.
eval	Вычисляет арифметическое выражение.

Следовательно, для того чтобы определить переменную, значение которой на единицу больше, чем заданное *Число*, нужно ввести:

```
define(Number, 100)
define(Number1, `incr(Number)')
```

В этом примере определяется число **Number1**, значение которого на единицу больше, чем значение **Number**.

Функция **eval** может вычислять значения выражений, содержащих следующие операторы (перечислены в порядке убывания приоритета):

- унарные **+** и **-**
- **** и **^** (возведение в степень)
- *** / **%** (модуль)
- +** -
- ==** **!=** **<** **<=** **>** **>=**
- !(не)**
- &** и **&&** (логическое И)
- |** и **||** (логическое ИЛИ)

Для выделения группы операций заключите их в скобки. В качестве операндов должны выступать числа. Истина (например, $1 > 0$) имеет значение 1, ложь - 0. Точность функции **eval** составляет 32 разряда.

Например, с помощью функции **eval** можно создать макроопределение для M со значением $2==N+1$:

```
define(N, 3)
define(M, `eval(2==N+1)`)
```

Любой достаточно сложный текст замещения в макроопределении рекомендуется заключать в кавычки.

Операции с файлами

Для включения нового файла в программу предназначена встроенная функция **include**.

Макрокоманда	Описание
include (<i>файл</i>)	Возвращает содержимое заданного <i>файла</i> .

Например:

```
include(имя-файла)
```

подставляет содержимое файла имя-файла вместо команды **include**.

Если файл, имя которого указано в макрокоманде **include**, недоступен, возникает неустраняемая ошибка. Для того чтобы ее избежать, используйте альтернативную форму **sinclude**.

Макрокоманда	Описание
sinclude (<i>файл</i>)	Возвращает содержимое указанного <i>файла</i> , но не выдает сообщения об ошибке, если <i>файл</i> недоступен.

Если файл недоступен, макрокоманда **sinclude** не выдает сообщение об ошибке, что позволяет программе продолжить свою работу.

Перенаправление вывода

Вывод макропроцессора **m4** можно перенаправить во временные файлы. После этого содержимое этих файлов можно вывести на экран. Макропроцессор **m4** поддерживает до девяти временных файлов (с номерами от 1 до 9). Для перенаправления вывода предназначена встроенная макрокоманда **divert**.

Макрокоманда	Описание
divert (<i>число</i>)	Направляет вывод во временный файл с заданным <i>номером</i> .

После того как макропроцессор **m4** встречает в программе функцию **divert**, все выходные данные записываются в конец временного файла с заданным *номером*. Для возобновления вывода на экран вызовите функцию **divert** или **divert(0)**.

По окончании обработки макропроцессор **m4** записывает перенаправленный вывод во временные файлы в соответствии с их номерами. Если вы перенаправили вывод во временный файл, номер которого не входит в интервал от 0 до 9, то макропроцессор **m4** аннулирует вывод.

Для получения данных из временных файлов предназначена встроенная макрокоманда **undivert**.

Макрокоманда	Описание
undivert (<i>номер-1, номер-2... </i>)	Добавляет содержимое указанных временных файлов к текущему временному файлу.

Для восстановления выбранных временных файлов в указанном порядке выполните команду **undivert** с аргументами. При выполнении макрокоманды **undivert** макропроцессор **m4** отбрасывает обнаруженные временные файлы и не выполняет в них поиск макроопределений.

Макрокоманда **undivert** не возвращает перенаправленный текст.

Макрокоманда **divnum** позволяет определить временный файл, используемый в данный момент.

Макрокоманда	Описание
divnum	Возвращает номер текущего активного временного файла.

Если вы не изменяете файл вывода с помощью макрокоманды **divert**, то макропроцессор **m4** направляет весь вывод во временный файл с номером 0.

Запуск системных команд из программы

С помощью встроенной макрокоманды **syscmd** вы можете запустить из программы любую команду операционной системы. Например, для выполнения команды **date** необходимо указать следующий оператор:

```
syscmd(date)
```



Создание уникальных имен файлов

Встроенная макрокоманда **maketemp** позволяет сформировать в программе уникальное имя файла.

Макрокоманда	Описание
maketemp (<i>Строка...nnnn...Строка</i>)	Создает уникальное имя файла, заменяя символы <i>nnnn</i> в строке аргументов на идентификатор текущего процесса.

Например, если указана макрокоманда

```
maketemp(myfilennnnn)
```

то макропроцессор **m4** возвращает строку, состоящую из символов `myfile` и идентификатора процесса. Эту строку можно использовать в качестве имени временного файла.

Работа с условными выражениями

Оценка условных выражений позволяет определять выражения макрокоманд.

Выражение	Описание
ifelse (<i>строка1, строка2, аргумент1, аргумент2</i>)	Если <i>строка1</i> совпадает со <i>строкой2</i> , возвращается значение <i>аргумента1</i> . В противном случае возвращается <i>аргумент2</i> .

Встроенная макрокоманда **ifelse** представляет собой условный оператор, выполняющий проверку. В простейшем случае:

```
ifelse(a, b, c, d)
```

сравнивает две строки - `a` и `b`.

Если строки `a` и `b` одинаковы, стандартная макрокоманда **ifelse** возвращает строку `c`, если нет - строку `d`. Например, вы можете задать макроопределение для команды `compare`, которая будет сравнивать две строки и возвращать слово `yes`, если они совпадают, и `no`, если нет:

```
define(compare, `ifelse($1, $2, yes, no)`)
```

Кавычки позволяют избежать преждевременной подстановки значений вместо аргументов **ifelse**. Если четвертый аргумент отсутствует, он считается пустым.

В команде **ifelse** может быть любое число аргументов, что позволяет использовать ее вместо громоздкой последовательности условных операторов, реализующих процедуру ветвления. Например:

Эта команда эквивалентна следующей конструкции:

```
if(a == b) x = c;  
else if(d == e) x = f;  
else x = g;  
return(x);
```

Если последний аргумент отсутствует, то результат будет нулевым, поэтому

```
ifelse(a, b, c)
```

возвращает c, если a совпадает с b, и пустое значение в противном случае.

Работа со строками

Макроопределение, приведенное в этом разделе, позволяет преобразовать входные строки в выходные.

Макрокоманда	Описание
len	Возвращает длину строки (аргумента команды) в байтах

Так,

```
len(abcdef)
```

равно 6, а

```
len((a,b))
```

равно 5.

Макрокоманда	Описание
dlen	Возвращает длину графических символов в строке

31.01.2024, 02:52

Макропроцессор m4 - обзор - IBM Documentation

Символ, которому соответствует двухбайтовый код, на экране выглядит как один символ. Следовательно, результат работы команд **dlen** и **len** для двухбайтовой строки будет различным.

Макрокоманда	Описание
substr (<i>строка</i> , <i>позиция</i> , <i>длина</i>)	Возвращает подстроку <i>строки</i> , начинающуюся с символа в указанной <i>позиции</i> и содержащую заданное <i>число-символов</i> .

Команда **substr** (*s*, *i*, *n*) возвращает подстроку строки *s*, которая начинается в *i*-той позиции (позиции нумеруются с нуля), и длина которой составляет *n* символов. Если аргумент *n* отсутствует, возвращается весь остаток строки. Например, функция

```
substr('now is the time',1)
```

возвращает строку

```
ow is the time
```

Макрокоманда	Описание
index (<i>строка1</i> , <i>строка2</i>)	Возвращает позицию первого символа <i>строки2</i> в <i>строке1</i> (позиции нумеруются с нуля), либо -1, если <i>строка1</i> не содержит <i>строку2</i> .

Как и в команде **substr**, нумерация символов строки начинается с нуля.

Макрокоманда	Описание
translit (<i>строка</i> , <i>набор1</i> , <i>набор2</i>)	Поиск в <i>строке</i> символов из <i>набора1</i> и их замена символами из <i>набора2</i> .

В общем случае

```
translit(s, f, t)
```

изменяет в *s* все символы из *f* на соответствующие символы из *t*. Например, функция:

```
translit('little', aeiou, 12345)
```

заменяет гласные соответствующими цифрами и возвращает следующее значение:

```
l3ttl2
```

Если *t* короче, чем *f*, то символы, для которых нет соответствующего символа в *t*, удаляются. Если набор *t* не задан, то символы, входящие в *f*, удаляются из *s*. Так,

```
translit('little', aeiou)
```

удаляет гласные из строки `little` и возвращает следующее значение:

```
lttl
```

Макрокоманда	Описание
<code>dnl</code>	Удаляет все символы, следующие за этой функцией до символа новой строки, включая сам символ.

Эту макрокоманду можно использовать для удаления пустых строк. Например, функция

```
define(N, 100)
define(M, 200)
define(L, 300)
```

добавляет символ новой строки ко всем строкам, не являющимся частью определения. Символы новой строки передаются на вывод. Для того чтобы избавиться от пустых строк, добавьте после каждого макроопределения встроенную макрокоманду `dnl`:

```
define(N, 100) dnl
define(M, 200) dnl
define(L, 300) dnl
```

Отладка макрокоманд M4

Макрокоманды, описанные в этом разделе, позволяют создать отчет об ошибках, в который также будет включена информация об обработке.

Макрокоманда	Описание
<code>errprint</code> (строка)	Отправляет заданную (<i>строку</i>) в стандартный файл вывода сообщений об ошибках.

Например:

Макрокоманда	Описание
dumpdef (<i>`имя'...</i> <i>)</i>	Создает дамп текущих имен и определений макрокоманд, указанных в списке <i>имен</i> .

Макрокоманда **dumpdef** без аргументов печатает все текущие имена и определения. Не забудьте заключать имена в кавычки.

Дополнительные макрокоманды m4

В таблице перечислены дополнительные макрокоманды **m4** и приведено их краткое описание:

Макрокоманда	Описание
changecom (<i>l, r</i>)	Замена левого и правого символа комментария символами, указанными вместо переменных <i>l</i> и <i>г</i> .
defn (<i>имя</i>)	Возвращает заключенное в кавычки макроопределение с указанным <i>именем</i>
en (<i>строка</i>)	Возвращает число символов в <i>строке</i> .
m4exit (<i>код</i>)	Выход из макропроцессора m4 с указанным <i>кодом</i> возврата.
m4wrap (<i>имя</i>)	Выполняет макрокоманду с заданным <i>именем</i> при завершении работы макропроцессора m4 .
popdef (<i>имя</i>)	Замена текущего макроопределения с указанным <i>именем</i> предыдущим определением, которое было сохранено с помощью макрокоманды pushdef .
pushdef (<i>имя, текст замещения</i>)	Сохраняет текущее <i>макроопределение</i> с указанным именем и заменяет его на указанный <i>текст-замещения</i> .
sysval	Выдает код возврата последней выполненной макрокоманды syscmd .
traceoff (<i>список- макроопределений</i>)	Выключает трассировку макроопределений из указанного <i>списка</i> . Если <i>список</i> не задан, трассировка выключается полностью.
traceon (<i>имя</i>)	Включает трассировку для макроопределения с указанным <i>именем</i> . Если <i>имя</i> не задано, трассировка включается для всех макроопределений.

На уровень выше:

→ [Общие принципы программирования](#)