Программирование: Разработка и отладка программ

Глава 15. Макропроцессор m4 - Обзор

Макропроцессор **m4** играет роль препроцессора для всех языков программирования в операционной системе.

Макропроцессор **m4** выполняет несколько функций. В начале программы вы можете определить символьное имя или символьную константу как строку символов. Затем с помощью программы 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 действия
- Операции с файлами
- Обработка условных макрокоманд
- Обработка строк и подстрок

Макропроцессор **m4** обрабатывает алфавитно-цифровые строки, которые называются *лексемами*. Программа **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.

Программа **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]

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

```
m4 [файл] >выходной_файл
```

Создание пользовательских макроопределений

define (*имя*,*текст***)**	Создает новое макроопределение с заданным *именем* и со значением *текст-замещения*.

Например, если в программе задан оператор

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

то программа **m4** присвоит строке name значение stuff. Если в файле программы встретится строка name, то программа **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)
```

определяется, что N равно 100, и затем 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 буквенно-цифровыми символами. В следующем примере:

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

переменная NNN никак не связана с макроопределением N.

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 ему 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 программа **m4** всегда удаляет один уровень кавычек при обработке лексемы. Это справедливо даже вне макроопределения. Для того чтобы в выводе команды сохранилось слово define, возьмите 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)
```

Программа **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

changequote (*l***, *r*)**	Эта команда позволяет заменить левую и правую кавычки символами, указанными вместо переменных *l* и *r*.

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 виде **$***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 программа **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**.

shift (*параметры***)**	Возвращает все, за исключением первого, элементы списка *параметры*.

Эта макрокоманда удаляет первый аргумент и переприсваивает оставшиеся аргументы символам **$***n* (второй аргумент - символу $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.

Программа **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

Программа **m4** предоставляет набор предопределенных макрокоманд. Их описание приводится в последующих разделах.

Удаление макроопределений

undefine (`*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

ifdef (`*имя*',*аргумент-1*,*аргумент-2*)	Если макроопределение с указанным *именем* существует, и его значение не равно нулю, то возвращает значение *аргумента1*. В противном случае возвращает *аргумент2*.

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

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

incr (*число*)	Увеличивает *число* на 1.
decr (*число*)	Уменьшает *число* 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)**

+ -

== != < <= > >=

! (отрицание)

& 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**.

include (*файл*) Возвращает содержимое заданного *файла*.

Например:

include(имя-файла)

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

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

sinclude (*файл*) Возвращает содержимое указанного *файла*, но не выдает сообщения об ошибке, если *файл* недоступен.

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

Перенаправление вывода

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

divert (*номер***)** Направляет вывод во временный файл с заданным *номером*.

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 укажите функцию **divert** или **divert(0)**.

По окончании обработки программа **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 программа **m4** аннулирует вывод.

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

undivert (*номер-1***,***номер-2***...)** 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** программа **m4** не выполняет поиск макроопределений во временных файлах.

Макрокоманда **undivert** не возвращает перенаправленный текст.

divnum Возвращает номер текущего активного временного файла.

Если вы не изменяете файл вывода с помощью макрокоманды **divert**, программа **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.

maketemp (*Строка...nnnnn...Строка***)** — Создает уникальное имя файла, заменяя символы *nnnnn* в строке аргументов идентификатором текущего процесса.

Например, если указана макрокоманда

```
maketemp(myfilennnnn)
```

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

Работа с условными выражениями

ifelse (*строка1***,***строка2***, *аргумент1***, *аргумент2***)** — Если *строка1* совпадает со *строкой2*, возвращается значение *аргумента1*. В противном случае возвращается *аргумент2*.

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:

```
ifelse(a, b, c, d, e, f, g)
```

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

len Возвращает длину строки (аргумента команды) в байтах

Так,

```
len(abcdef)
```

равно 6, а

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

равно 5.

dlen Возвращает длину графических символов в строке

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

substr (*строка***,***позиция***,***длина***)** Возвращает подстроку *строки*, начинающуюся с символа в указанной *позиции* и содержащую заданное *число-символов*.

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

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

возвращает строку

```
now is the time
```

index (*строка1***,***строка2***)** Возвращает позицию первого символа *строки2* в *строке1* (позиции нумеруются с нуля), либо -1, если *строка1* не содержит *строку2*.

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

translit (*строка***,***набор1***,***набор2***)** Поиск в *строке* символов из *набора1* и их замена символами из *набора2*.

В общем случае

translit(s, f, t)

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

translit(s, aeiou, 12345)

заменяет гласные соответствующими цифрами. 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(s, aeiou)

удаляет гласные из строки s.

dnl 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
```

Печать

errprint (*строка***)** Отправляет заданную (*строку*) в стандартный файл вывода сообщений об ошибках.

Например:

```
errprint ('error')
```

dumpdef (`*имя*'...)	Создает дамп текущих имен и определений макрокоманд, указанных в списке *имен*.

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

Список дополнительных макрокоманд m4

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

changecom (*l*, *r*)	Замена левого и правого символа комментария символами, указанными вместо переменных *l* и *r*.
defn (*имя*)	Возвращает заключенное в кавычки макроопределение с указанным *именем*
en (*строка*)	Возвращает число символов в *строке*.
eval (*выражение*)	Возвращает 32-разрядный результат вычисления арифметического *выражения*.
m4exit (*код*)	Выход из **m4** с указанным *кодом* возврата.
m4wrap (*имя*)	Выполняет макрокоманду с указанным *именем* после завершения программы **m4**.
popdef (*имя*)	Замена текущего макроопределения с указанным *именем* предыдущим определением, которое было сохранено с помощью макрокоманды **pushdef**.
pushdef (*имя*,*текст замещения*)	Сохраняет текущее макроопределение с указанным *именем* и заменяет его на указанный *текст-замещения*.
syscmd (*команда*)	Выполняет указанную *системную команду*, не возвращая никакого результата.
sysval	Возвращает код возврата последней выполненной макрокоманды **syscmd**.
traceoff (*список-макроопределений*)	Выключает трассировку макроопределений из указанного *списка*. Если *список* не задан, трассировка выключается полностью.

traceon (*имя*)	Включает трассировку для макроопределения с указанным *именем*. Если *имя* не задано, трассировка включается для всех макроопределений.

[Страница назад | Страница вперед | Содержание | Индекс | Библиотека | Юридическая информация | Поиск]