## Пакетный текстовый редактор MSE

Программа MSE (MultiStream Editor — многопоточный редактор текстов) предназначена для проведения замен в текстовом файле в пакетном режиме. Программы такого типа — удобное средство для многих видов работы с текстами, среди них: смена системы маркирования в тексте (например, замена маркеров TEX на маркеры RTF и наоборот); внесение в текст информации о местах разрешенного переноса слов; разного рода преобразования словарей и т. п. MSE можно использовать и для пакетной обработки бинарных данных.

Специальные команды MSE позволяют выполнять некоторые дополнительные действия одновременно с заменами: вводить шаблоны, использовать переменные, организовывать условные переходы и циклы, вносить в таблицы замен комментарии, выполнять арифметические действия, определять и использовать макросы и другое.

В сравнении с другой известной программой пакетной обработки текста, AWK, программа MSE имеет следующие отличия, которые часто можно рассматривать как преимущества: ориентированность на символьные, а не словарные элементы текста; возможность необычайно гибко управлять потоками ввода-вывода; более простой синтаксис для проведения несложной обработки.

## Общая характеристика программы, ее базовые структуры и возможности

Программа MSE при вызове загружает файл с таблицами замен и затем, согласно содержащимся в них правилам, производит преобразования данных входного файла, записывая результаты этих преобразований в выходной файл.

Текстовый файл таблиц замен может состоять из операторов замены вида `ЗАМЕЩАЕМОЕ > ЗАМЕСТИТЕЛЬ` (группа операторов замен, которые используются совместно, образуют одну таблицу замен), комментариев и оператора определения группы, в состав которого не входит символ `">"`. Количество операторов замены ограничивается только размером памяти, но скорость работы программы обратно пропорциональна количеству операторов.

Замены происходят следующим образом. Входной текстовый файл (подлежащий обработке) читается последовательно от начала до конца, этот процесс можно представить как движение читающего устройства вдоль символов текста. Для каждой фиксированной позиции читающего устройства программа пытается найти совпадения ЗАМЕЩАЕМЫХ, входящих в текущие таблицы операторов замены, с порцией текста, начинающейся

с позиции читающего устройства. Если происходит не одно из таких совпадений, а несколько, то выбирается тот оператор замены, ЗАМЕЩАЕМОЕ которого длиннее (в случае совпадения длин выбирается оператор, встретившийся первым). После выбора оператора замены читающее устройство сдвигается на позицию, следующую за концом той порции текста, которая совпала с ЗАМЕЩАЕМЫМ, а в выходной поток (предопределенное направление выходного потока — в выходной файл с результатами замен) записывается текст, определяемый ЗАМЕСТИТЕЛЕМ, и, кроме того, может быть, выполняются некоторые действия, также заданные содержанием ЗАМЕСТИТЕЛЯ. Если не найдется ни одного оператора замены, ЗАМЕЩАЕМОЕ которого совпадало бы с вышеописанной порцией входных данных, то символ с текущей позиции читающего устройства помещается в выходной поток, а само читающее устройство сдвигается на позицию следующего символа, и все повторяется сначала. Работа программы завершается после достижения читающим устройством конца входного файла или по команде endfile.

Как ЗАМЕЩАЕМОЕ, так и ЗАМЕСТИТЕЛЬ в общем случае состоят из набора литералов и специальных команд. Литералы и специальные команды могут как следовать непосредственно один за другим, так и быть в целях наглядности разделенными пробелом или знаком табуляции (одним или несколькими), а в случае ЗАМЕСТИТЕЛЯ — и концами строк (тоже одним или несколькими, т. е. в ЗАМЕСТИТЕЛЕ допускаются пустые строки). Все содержимое ЗАМЕЩАЕМОГО должно располагаться на одной строке со знаком `">"`, а содержимое ЗАМЕСТИТЕЛЯ может занимать сколько угодно строк. Строки, в том числе и с ЗАМЕЩАЕМЫМ, можно разрывать знаком \, непосредственно за которым должен быть конец строки.

Литералы допускают как строковое, так и числовое представление.

Строковый литерал — это последовательность символов, заключенных в двойные или одинарные кавычки, например, “прекрасная погода”, ’синий цвет’. Для отображения кавычек в литерале их заключают в кавычки другого типа, например, телефильм “д’Артаньян и три мушкетера”. Допустимо использование пустого литерала ’’ или `""`.

Числовой литерал — это число, представляющее собой код символа. Числовые литералы дают возможность работать с непечатаемыми (управляющими) символами такими как 0d9 — кодом табуляции в ASCII. Числовые литералы не заключаются в кавычки. Допускаются десятичные, шестнадцатеричные и восьмеричные числовые литералы. Десятичные числовые литералы должны иметь префикс d, D, 0d или 0D; шестнадцатеричные — x, X, 0x, 0X; восьмеричные вводятся без префикса (пример четырех равных друг другу литералов: 0d49, x31, 61,

0X31). Поскольку числовые литералы должны представлять собой код из расширенной таблицы ASCII, то их значения могут лежать лишь в десятичном диапазоне от 0 до 255.

Последовательность шестнадцатеричных литералов (и только их), может быть сгруппирована в строку с единственным префиксом в начале, при этом каждое шестнадцатеричное число должно состоять из двух цифр, например, литерал 0X40444044 эквивалентен строке “@D@D”, а литерал 0x0909 эквивалентен двум кодам табуляции — обе эти эквивалентности есть следствия употребления соглашений ASCII.

### Описание форматов вызова MSE из командной строки

Программу MSE нужно вызывать с параметрами. После опции -o вводят имя выходного файла (не непосредственно за -o, а хотя бы через один пробел, это замечание относится и к остальным опциям); после опции -t — имя файла замен; имена входных файлов (их может быть несколько) вводятся не предваряемые никакими опциями. После опции -l вводят имя файла-отчета. Порядок параметров может быть любой. Дефис вместо имени файла означает ввод или вывод с консоли.

Пример использования программы MSE

```
mse -t my.mse -o result.txt src1.txt -l rpt.html src2.txt
```

При запуске программы MSE с параметром -h на экран выводится подсказка.

## Команды MSE

### Краткий обзор команд MSE

Заглавные и строчные буквы различаются только в литералах.

В следующей далее таблице в первом столбце приводится имя команды с типом параметра на месте параметра; во втором — место команды в операторе замены; в третьем — область приложения команды.

Типы параметров:

ИОХ — Имя Области Хранения ;

ИЛП — Имя Логической Переменной;

ИГ — Имя Группы, в командах с таким параметром допускается использование нескольких параметров ИГ, разделенных запятой;

ИМ — Имя Макрокоманды;

# — десятичное число (без префиксов).

В качестве имени допускается любая последовательность алфавитных символов и цифр, не содержащая символов закрывающей круглой скобки, запятых и пробелов, т. е. именами могут быть и числа. ИОХ, ИЛП, ИГ и ИМ образуют независимые области имен, например, ИОХ и ИЛП могут быть одинаковы, но ссылаться они будут на совершенно различные объекты.

Место команды в операторе замены: L — слева от символа > (в этой позиции допустима только команда, задающая шаблон для сравнения, или команда, задающая строку-определение, define); R — справа от символа >; D — данная команда может быть употреблена только в определенном контексте.

Области приложения команд описаны в последующих разделах.

| | | |
|---|---|---|
| '' | LR | субститут |
| add(ИОХ) | R | арифметика |
| any(ИОХ) | L | субститут |
| append(ИОХ) | R | работа с областями хранения (управление вводом/выводом) |
| back(#) | R | управление вводом/выводом |
| +backi(#) | R | управление вводом/выводом |
| begin | L | субститут |
| begin | DR | управление |
| -caseless | DR | начальная установка |
| clear(ИЛП) | R | логические переменные |
| cont(ИОХ) | L | субститут |
| cont(ИОХ) | DR | субститут |
| define(ИМ) | L | макрокоманды и макроопределения |
| +decr(ИОХ) | R | арифметика |
| div(ИОХ) | R | арифметика |
| do(ИМ) | R | макрокоманды и макроопределения |

| | | |
|---|---|---|
| dup | R | субститут |
| else | DR | управление |
| end | DR | управление |
| endfile | LR | субститут |
| endif | R | управление |
| endstore | R | работа с областями хранения (управление вводом/выводом) |
| excl(ИГ) | R | группы |
| fol(ИОХ) | L | субститут |
| fwd(#) | R | управление вводом/выводом |
| if(ИЛП) | R | управление |
| ifeq(ИОХ) | R | управление |
| ifgt(ИОХ) | R | управление |
| ifn(ИЛП) | R | управление |
| ifneq(ИОХ) | R | управление |
| incl(ИГ) | R | группы |
| incr(ИОХ) | R | арифметика |
| mod(ИОХ) | R | арифметика |
| mul(ИОХ) | R | арифметика |
| next | R | управление |
| nl | LR | субститут |
| +not | R | логические переменные |
| omit(#) | R | управление вводом/выводом |
| out(ИОХ) | R | работа с областями хранения (управление вводом/выводом) |
| outs(ИОХ) | R | работа с областями хранения (управление вводом/выводом) |

| | | |
|---|---|---|
| prec(ИОХ) | L | субститут |
| +preci(ИОХ) | L | субститут |
| +prevsym(#) | L | субститут |
| read | R | управление вводом/выводом |
| repeat | DR | управление |
| set(ИЛП) | R | логические переменные |
| store(ИОХ) | R | работа с областями хранения |
| | | (управление вводом/выводом) |
| sub(ИОХ) | R | арифметика |
| +symdup(#) | R | субститут |
| unsorted | DR | начальная установка |
| use(ИГ) | R | группы |
| wd(ИОХ) | L | субститут |
| write | R | управление вводом/выводом |
| wrstore(ИОХ) | R | работа с областями хранения |

Вместо последовательности одинаковых операторов с параметрами, в большинстве случаев допускается использование одного оператора с несколькими параметрами, разделенными запятыми, например, последовательность команд `out(first) out(second)` эквивалентна команде `out(first, second)`.

Комментарии вводятся символом c (латинским), остаток строки после этих символов игнорируется программой MSE. Не допускается непосредственного примыкания к c (или C) любых символов кроме пробела, табуляции или конца строки. Можно также вводить комментарий между символами % и концом строки.

Оператор определения группы `GROUP(<имя группы>)` занимает отдельную строку и по существу является просто маркером. Он указывает, что все операторы замены после него и до следующего оператора определения группы или до конца файла замен относятся к группе с вводимым этим оператором именем. Данный оператор не является обязательным. В случае использования в файле таблиц замен только одной группы его употребление излишне (начальная групп имеет имя 1).

### Определение областей хранения и управление потоком ввода/вывода

Как уже отмечалось выше, предопределенное направление потока вывода — это выходной файл. Программа MSE позволяет переопределять это направление в именованные области хранения. Рассмотрим подробно все команды MSE, имеющие отношение к данной возможности.

append(ИОХ) — после этой команды поток вывода будет направлен в область хранения с именем ИОХ, причем содержимое этой области хранения не очищается от данных, которые ранее могли быть туда записаны.

endstore — после этой команды направление потока вывода примет предопределенное значение (в выходной файл).

out(ИОХ) — эта команда сбрасывает содержимое области хранения с именем ИОХ в выходной поток и возвращает направлению потока вывода предопределенное значение (в выходной файл), при этом содержимое самой этой области хранения не изменяется.

outs(ИОХ) — полностью аналогична команде out за исключением того, что она не изменяет текущее направление потока вывода.

store(ИОХ) — полностью аналогична команде append за исключением того, что она удаляет все данные, содержащиеся в данной области хранения до выполнения команды store.

Пример: инвертирование двуязычного словаря

Пусть задан размеченный словарь со следующей структурой статей:

```
\w <слово>
\p <часть речи>
\d <перевод>
\i <иллюстрационное предложение с данным словом>
\t <иллюстрационное предложение с данным переводом>
```

По нему строим инвертированный словарь с следующей структурой каждой статьи:

```
\w <перевод>
\p <часть речи>
\d <слово>
```

```
\i <иллюстрационное предложение с данным переводом>
\t <иллюстрационное предложение с данным словом>
```

Например, статья:

```
\w cat
\p n
\d кот
\i The cat is black.
\t Этот кот --- черный.
```

должна быть трансформирована в статью:

```
\w кот
\p n
\d cat
\i Этот кот --- черный.
\t The cat is black.
```

Следующая последовательность правил совершает требуемое:

```
"\w " > out(def,part,word,trans,ill)
  c выводит инвертированную статью при встрече
  c следующей статьи
store(trans,ill,def,part,word)
"\d "
  c очищает области хранения trans, ill, def, part
  c и word и заносит в последнюю "\d " (входное слово
  c превращается в перевод)
"\p " > store(part) "\p "
"\d " > store(def) "\w "
  c выводит в область сохранения def перевод,
  c снабженные, однако, маркером слова \w
```

```
"\i " > store(ill) "\t "
"\t " > store(trans) "\i "
endfile > out(def,part,word,trans,ill) endfile
  c выводит последнюю инвертированную статью при
  c встрече с концом файла
```

Следующие команды MSE также позволяют управлять потоком ввода/вывода.

back(#) — данная команда забирает # символов из выходного потока (файла или области хранения) и помещает их во входной поток так, что позиция читающего устройства отходит на # позиций назад и данные # символов ложатся как раз перед читающим устройством.

backi(#) — эта команда помещает во входной поток предшествующие (ушедшие) # символов входного потока.

fwd(#) — после данной команды # символов из входного потока (входного файла) непосредственно, минуя таблицу замен, помещаются в выходной поток.

omit(#) — после данной команды читающее устройство просто перемещается на # позиций вперед во входном файле, т. е. порция входных данных длиной # символов игнорируется.

read — вводит строку с клавиатуры и помещает ее в текущий поток вывода (ввод прекращается после введения символа конца строки, например, клавишей Enter).

write СТРОКА — выводит СТРОКУ на экран дисплея. СТРОКА представляет собой любую последовательность из литералов и nl.

wrstore(ИОХ) — выводит на экран дисплея содержимое области хранения с именем ИОХ.

Примеры

```
"  " > " " back(1)   c заменяет много пробелов на один
" " nl > nl back(1)  c убирает все пробелы перед
         c концом строки
ЗАМЕЩАЕМОЕ > store(4) outs(1) outs(2) outs(3)  c переносит в
 c область хранения 4 содержимое областей хранения 1, 2 и 3.
ЗАМЕЩАЕМОЕ > store(x) endstore  c очистка области хранения x
```

### Субституты

Субститут или шаблон замещает один или множество литералов.

'' — ЗАМЕЩАЕМОЕ, состоящее из одного пустого литерала, совпадает с любыми данными из входного файла; использование пустого литерала в ЗАМЕЩАЕМОМ совместно с другими литералами или в ЗАМЕСТИТЕЛЕ хотя и допустимо, но бессмысленно. Данный субститут должен использоваться (во избежание зацикливания программы) совместно с командами fwd, omit или use. Его можно использовать для проведения замен над всеми символами, не входящими в текущую(ие) таблицу(ы) замен.

Пример

```
"s" > "s"           с ставит перед всеми символами, кроме s,
        с знак дефис
"" > fwd(1) '-'
```

any(ИОХ) — совпадает с любым символом из области хранения с именем ИОХ.

begin — начало файла, используется для начальной установки.

cont(ИОХ) — в ЗАМЕЩАЕМОМ эквивалентен литералу, равному содержимому области хранения с именем ИОХ.

Пример

```
begin > store(1) "xyz" endstore  с замена последовательности
cont(1) > "abc"                  с символов "xyz" на "abc"
```

cont(ИОХ) — в ЗАМЕНИТЕЛЕ может быть использован только в командах ifeq, ifneq, ifgt, add, sub, mul, mod, div как эквивалент литерала, равного содержимому области хранения с именем ИОХ.

Пример

```
"abc" > ifeq(2) cont(4) "xyz" с замена символов "abc" на
else "uvw"           с "xyzf", если область хранения
endif                с 2 равна области хранения 4, и
"f"                  с на "uvwf", если не равна
```

dup — служит для посылки в выходной поток той порции входных данных, которая совпала с ЗАМЕЩАЕМЫМ.

Пример

```
"ab" > dup       с эквивалентно "ab" > "ab"
"big" > "deep"   с замена big на deep, но не
"big bad" > dup  с в сочетании big bad
"big" nl "bad" > dup
```

endfile — выполняется в конце файла последнего входного файла.

fol(ИОХ) — делает ЗАМЕЩАЕМОЕ эквивалентным ЗАМЕЩАЕМОЕ any(ИОХ) при поиске совпадения во входном файле, однако, при выборе данного оператора замены (содержащего fol), читающее устройство после проведения замены становится на позицию символа, совпавшего с этой добавочной командой any, т. е. этот символ читается только для сравнения и не может, например, быть выведен в выходной поток командой dup. При выборе самого длинного ЗАМЕЩАЕМОГО при совпадении нескольких ЗАМЕЩАЕМЫХ с входной порцией данных считается, что fol, как prec и wd, имеет длину меньше единицы (т. е. длины одного символа или приравненных к нему субститутов, например any), но больше нуля.

nl — перенос строки.

Примеры

```
"дерево" > nl         с замена слова "дерево" во входном
"  *  " nl   с тексте на пиктограмму ёлки
" *** " nl
"*****" nl

"горячая пища" > "каша"       с замена через перенос строки
"горячая" nl "пища" > "каша"
```

prec(ИОХ) — делает ЗАМЕЩАЕМОЕ эквивалентным any(ИОХ) ЗАМЕЩАЕМОЕ при поиске совпадения во входном файле, однако при выборе данного оператора замены (содержащего prec) читающее устройство после проведения замены становится на позицию символа, совпавшего с этой добавочной командой any (с исключением

остальных совпавших символов из входного потока), т. е. этот символ читается только для сравнения, и не может, например, быть выведен в выходной поток командой dup.

preci(ИОХ) — отличается от prec только тем, что символ для сравнения берется из неизмененного предыдущими командами потока.

prevsym(#) — соответствует заданному позицией символу в выходном потоке. Номер позиции считается от позиции prevsym (равной 0).

Пример

```
begin > store(1) 'ab' endstore
any(1) 'x' prevsym(2) 'z' > symdup(0) '*'
                          с замена слов "axaz" и "bxbz"
                          с на "a*" и "b*" соответственно
```

symdup(#) — служит для посылки в выходной поток заданного символа из ЗАМЕЩАЕМОГО. Позиции считаются с нуля.

wd(ИОХ) — делает ЗАМЕЩАЕМОЕ эквивалентным any(ИОХ) ЗАМЕЩАЕМОЕ any(ИОХ) при поиске совпадения во входном файле, однако при выборе данного оператора замены (содержащего wd) читающее устройство после проведения замены становится на позицию символа, совпавшего с первой из этих добавочных команд any (с исключением символов совпавших с ЗАМЕЩАЕМЫМ из входного потока), т. е. эти символы (совпавшие с добавочными командами any) читаются только для сравнения, и не могут, например, быть выведены в выходной поток командой dup.

Пример

```
begin > store(delim) ' ' nl '.,";:?!([])' endstore
  с запись границ слов (разделителей) в область
  с хранения delim
"man" wd(delim) > "person"
  с замена выделенных разделителями слов
"men" wd(delim) > "people"
  с man и men на person и people соответственно
```

### Логические переменные

MSE позволяет использовать логические переменные, которые затем можно использовать в командах if и ifn.

set(ИЛП) — установка значения логической переменной с именем ИЛП, равным true (истина).

clear(ИЛП) — установка значения логической переменной с именем ИЛП, равным false (ложь).

not(ИЛП) — инвертация значения логической переменной с именем ИЛП.

### Команды управления

Данная группа команд предназначена для организации замен зависимых от некоторых условий, а также для организации ветвлений, циклов и переходов при выполнении замен.

begin — начало блока. Эта команда дает возможность вложения операторов if, ifeq, ifgt, ifn друг в друга, а также отделять литералы друг от друга.

Пример

```
begin > store(1) '10000' endstore
'a' > add(1) '1000' begin '*' end out(1) dup
```

else — после этой команды замены и/или команды выполняются только, если условие при предшествующем if, ifeq, ifgt, ifn или ifneq не истинно.

end — конец блока, начало которого обозначено командой begin.

endif — означает конец действия операторов типа if (if, ifeq, ifgt, ifn, ifneq, else). Такое же действие имеет начало следующего оператора замены или конец блока begin end.

if(ИЛП) — эта команда сравнивает значение логической переменной с именем ИЛП со значением true (истина). Если сравнение успешно, то выполняются все команды и/или замены, следующие за ней до команд else или endif или, если сама команда if находится в блоке (begin end), end.

Пример

```
'x' > if(1)       с замена 'x' на
begin      c 'a', если значения 1 и 2 true (истина)
  if(2) 'a' c 'b', если значения 1 true (истина)
```

```
  else 'b'  с и 2 false (ложь)
end else begin      с 'c', если значения 1 false (ложь)
  if(2) 'c' с и 2 true (истина)
  else 'd'  с 'd', если значения 1 и 2 false (ложь)
end
```

ifeq(ИОХ) СТРОКА — эквивалентна команде if во всем, кроме того, что сравнение считается успешным, если содержимое области хранения с именем ИОХ совпадает с содержимым СТРОКИ, СТРОКА — это либо литерал, либо команда cont(ИОХ); в последнем случае сравнивается содержимое двух областей хранения. Если необходимо числовое сравнение без учета лидирующих 0 в ИОХ, то перед сравнением сложите ИОХ с 0 командой add(ИОХ)'0'. Последнее замечание касается также команд ifneq и ifgt.

ifgt(ИОХ) СТРОКА — эквивалентна команде ifeq во всем, кроме того, что сравнение считается успешным, если содержимое области хранения с именем ИОХ больше СТРОКИ. Содержимое строки-1 больше строки-2 в смысле MSE, если при побайтовом сравнении обеих строк с их начал, в строке-1 первым встретится символ, ASCII которого больше ASCII соответствующего ему символа из строки-2, или если строка-1 длиннее строки-2.

ifn(ИЛП) — эквивалентна команде if во всем, кроме того, что сравнение считается успешным, если значение переменной с именем ИЛП ложно.

ifneq(ИОХ) СТРОКА — эквивалентна команде ifeq во всем, кроме того, что сравнение считается успешным, если содержимое области хранения с именем ИОХ не совпадает с содержимым СТРОКИ.

repeat — позволяет организовывать циклы; после этой команды вновь выполняются все замены и/или команды, следующие после команды begin, находящейся перед repeat.

Пример

```
begin > store(count)
 '00'
endstore
'x' > begin              с увеличивает число в области
 incr(count)        с хранения с именем count до тех
 ifneq(count) '80'  с пор, пока оно не станет равным
```

```
 repeat              с 80 (простейший цикл)
end
```

next — выполняет правую часть (ЗАМЕСТИТЕЛЬ) следующего оператора замены.

Использование данной команды полезно в тех случаях, когда для различных замещаемых нужно выполнить одинаковые действия, определяемые ЗАМЕСТИТЕЛЕМ.

Пример

```
'1' > next   %выводит символ 'x' после каждой цифры
'2' > next
'3' > next
'4' > next
'5' > next
'6' > next
'7' > next
'8' > next
'9' > next
'0' > dup 'x'
```

**Арифметика**

Следующие команды позволяют производить арифметические действия над содержимым областей хранения.

add(ИОХ) ЧИСЛО — прибавляет к содержимому области хранения ЧИСЛО (литерал или последовательность литералов, составленный(ые) только из символов десятичных цифр 0, 1, . . ., 9), и записывает результат обратно в область хранения.

Пример

```
begin > store(test) '22' endstore
add(test) '34'  с в результате в области хранения
            с с именем test будет храниться '56'
```

decr(ИОХ) — уменьшает содержимое области хранения на единицу. В всем остальном ведет себя подобно incr.

div(ИОХ) ЧИСЛО — целочисленно делит содержимое области хранения на ЧИСЛО и записывает частное обратно в область хранения.

incr(ИОХ) — увеличивает содержимое области хранения на единицу. При этой операции не происходит добавления разрядов к растущему числу, поэтому нужно заранее записать в начало области хранения достаточное количество символов 0. В отличие от add, div, mod, mul и sub данная команда сохраняет начальные символы 0 в области хранения и может работать с нечисловым содержимым области хранения.

Пример

```
begin > store(zork) 'x' incr(zork) out(zork)  с выведет 'y'
begin > store(num) 'b9' incr(num) incr(num) out(num)
                                  с выведет 'c1'
```

mod(ИОХ) ЧИСЛО — целочисленно делит содержимое области хранения на ЧИСЛО и записывает остаток обратно в область хранения.

mul(ИОХ) ЧИСЛО — умножает содержимое области хранения на ЧИСЛО и записывает результат обратно в область хранения.

sub(ИОХ) ЧИСЛО — вычитает от содержимого области хранения ЧИСЛО и записывает результат обратно в область хранения.

### Команды использования групп

Эти команды позволяют выбирать текущие таблицы замен из общей совокупности таблиц файла замен.

excl(ИГ1, ИГ2, ...) — исключает из совокупности текущих таблиц замены группы с именами ИГ1, ИГ2 и т. д.

Пример

```
begin > use(1,2,4)
use(1,2)     с эта строка может быть заменена
         с командой excl(4)
```

incl(ИГ1, ИГ2, ...) — включает в совокупность текущих таблиц замены новые группы с именами ИГ1, ИГ2 и т. д.

use(ИГ1, ИГ2, ...) — отменяет все текущие таблицы замен и устанавливает новую совокупность текущих таблиц замены ИГ1, ИГ2 и т. д. Таким образом, use(ИГ1, ИГ2, ...) заменяется не на use(ИГ1) use(ИГ2) use..., а на use(ИГ1) incl(ИГ2) incl..., являясь единственным исключением из правила.

### Макрокоманды

Использование макрокоманд позволяет сократить размер файла замен и сделать его более понятным.

define(ИМ) — позволяет определить последовательность замен и/или команд, которую затем можно будет вызывать по имени ИМ.

Пример: define(macro1) » '**' dup '**'

do(ИМ) — вызов последовательности команд и/или замен, ассоциируемых с именем ИМ.

Пример

```
define(macro2) > dup '*' dup
'a' > do(macro2)
'b' > do(macro2)
```

### Начальная установка

Дает возможность определить некоторые глобальные характеристики проводимых замен. Команды начальной установки можно использовать только совместно с командой begin в ЗАМЕЩАЕМОМ.

unsorted — используется для указания того, что порядок замен должен быть такой же самый, как и последовательность правил в файле замен. Без использования этой команды замены происходят прежде всего для самых длинных заменяемых.

## ИСТОРИЯ

Программа MSE является модифицированной версией программы CC (Consistent Changes — последовательные замены), созданной в Летнем институте лингвистики (Summer Institute of Linguistics — SIL), США. Код

MSE полностью независим от кода CC. Их различия заключаются в следующем, в MSE в отличие от CC версии 7.5:

* не поддерживается специальной обработки заглавной буквы, начинающей слово, и команды начальной установки caseless;

* замены всегда проходят в бинарном режиме;

* добавлены операторы backi, decr, not, prevsym, preci и symdup;

* операторы fol, prec и wd всегда работают правильно и могут помещаться в любой позиции ЗАМЕЩАЕМОГО;

* нет никаких ограничений на количество используемых prec, fol и wd в субститутах;

* можно определять (командой define) и использовать сколько угодно макросов;

* нет ограничений на числовые значения для back, omit, fwd;

* нет никаких ограничений на количество используемых групп;

* можно использовать %-комментарии;

* нет ограничений на глубину вложенности макрокоманд;

* можно разрывать строку в таблицах замен на знаке \, после которого следует символ конца строки;

* можно использовать как фильтр и вводить таблицу замен в пакетном файле, например, используя bash,

```
(mse -t - -o - in.txt <<END
'a' > 'b'
'ab' > 'ac'
END
)| wc;
```

* вывод можно направлять в файл с именем входного файла, например,

```
mse -t my.mse -o data.txt data.txt;
```

* есть возможность генерировать файл-отчет в формате HTML;

* нет ограничений на использование кодов 0 (пусто), 10 (новая строка), 13 (возврат каретки) и 26 (конец файла в CP/M и MS-DOS);

* нет режима пошагового исполнения замен и возможности вводить параметры в диалоговом режиме;

* субститут begin не обязательно должен быть в первой команде таблицы замен;

* нет необходимости использовать команду endfile или dup после субститута endfile;
* ifeq, ifneq, ifgt всегда используют строчное посимвольное сравнение, сравнивая сначала длины строк;
* порядок групп в выбранном активном множестве групп не имеет значения;
* endfile, nl считаются командами и могут завершать строковый литерал;
* incr не увеличивает размер области хранения в случае ее переполнения;
* в define можно использовать только один аргумент;
* опции командной строки можно вводить в любом порядке;
* нет опций командной строки -i, ?, -m, -w, -a, -n, -q и -s;
* добавлены опции командной строки -h, –help, -V и -l;
* нет ограничений на длины строк во всех входных файлах;
* выполнение команд замен всегда начинается с группы с именем 1;
* возможно командой excl убрать все активные группы;
* группа не обязательно должна быть непрерывна, т. е. команды одной группы могут определяться в разных частях таблицы замен, чередуясь с другими группами;
* размер областей хранения не ограничен;
* команды левой части begin, define и endfile глобальны, т. е. принадлежат всем группам сразу.
* все неопределенные командой define макросы при использовании в команде do выполняются как пустая операция;
* строчные и заглавные буквы различаются только в литералах.