Task 3

Описать программу, которая, пока не получен конец файла, выполняет в цикле следующие действия:

1) считывает из stdin очередную строку,
2) преобразует ее в последовательность (список) слов,
3) выводит в stdout полученную последовательность слов в исходном порядке,
4) выводит слова последовательности в лексикографическом порядке.

При выводе в первой строке напечатать длину списка, в каждой последующей – очередной элемент списка.

Дополнительные условия

1. В строке могут встречаться буквы, цифры, подчеркивание, спецсимволы, пробельные символы (в том числе символы табуляции '\t' и перевода строки '\n' – завершающий строку символ). Слова могут быть двух видов: (1) простые – из букв, цифр и символов $, '_', '/', '.', и (2) специальные. К специальным относятся слова: |, ||, &, &&, ; , >, >>, <, (,). Между словами разных видов, а также между двумя специальными словами пробела (или табуляции) может не быть. Например, последовательность

```
cat|sort&&ls>>file
```

не содержит пробелов, но должна быть разбита на слова: cat, |, sort, &&, ls, >>, file .

Последовательность >>> разбивается на слова >> и >, но не на > и >> или > и > и >, то есть выделяется по возможности максимально длинное специальное слово.

2. Для представления последовательности слов необходимо описать структуру данных "список строк" (список строк представляется массивом, элементами которого являются указатели на char), а также функции добавления строки в конец списка, сортировки списка (любым известным алгоритмом сортировки, – использование библиотечных функций сортировки не допускается), вывода списка на stdout, очистки отведенной под список памяти. Описание структуры и функций работы с ней вынести в отдельный модуль. Для сборки исполняемой программы из модулей использовать утилиту make.

3. Считывание из stdin производится до тех пор, пока не наступит ситуация "конец файла" (EOF). Строки могут быть произвольной длины (длиннее любой наперед заданной константы в программе). Считывание необходимо производить блоками фиксированной длины N (не посимвольно) с помощью функции fscanf. Алгоритм считывания может быть примерно таким:

а) считываем очередные N символов строки в массив str подходящего размера;

б) с помощью отдельной функции getsym() получаем из str очередной символ; если массив str исчерапался, то getsym() считывает в str очередную порцию данных и выдает следующий символ;

в) возвращаемый функцией getsym() символ записывается в буфер текущего слова buf, полученный с помощью malloc(); если buf полностью заполнен, то он увеличивается: с помощью malloc() запрашивается новая память – по объему на константу SIZE бóльшая, чем предыдущий размер буфера, в начало этой памяти переписывается содержимое старого буфера, старый буфер освобождается, новая память становится буфером текущего слова (можно также для этой цели использовать библиотечную функцию realloc);

г) при достижении конца слова добавляем слово, накопленное в буфере buf, в список, предварительно завершив его символом ’\0’; если массив, представляющий список, заполнен, то он увеличивается аналогично буферу в пункте (в);

д) в конец массива, представляющего список, записываем NULL.

Некоторые общие требования к этому и последующим заданиям

1) Для всех используемых в программе библиотечных функций необходимо иметь четкое представление о том, что делает функция, каковы типы и смысл ее параметров и возвращаемого значения (в качестве основного источника информации о функциях рекомендуется использовать утилиту man).

2) Оформление программы должно максимально отражать логику реализуемого алгоритма. Программа должна быть написана так, чтобы ее понимание человеком, впервые видящим код, вызывало минимум усилий. В частности, функциям и переменным должны даваться осмысленные имена; логически связанные последовательности действий, решающие определенную подзадачу, должны быть оформлены в виде функций; не допускается использование глобальных переменных без крайней необходимости; конструкции, находящиеся на одном уровне вложенности, должны иметь одинаковый отступ, а на разных – разный. Обязательно наличие комментариев ко всем описанным в программе функциям (описание смысла и возможных значений параметров функций и ее возвращаемого значения). Также с помощью комментариев должны поясняться выполняемые в коде действия. Можно комментировать по-английски (транслит тоже допустим, если по-английски трудно). Если вы не уверены в том, что умеете правильно оформлять программы, можно ознакомиться и следовать, например, этим рекомендациям по оформлению:
http://ejudge.ru/study/3sem/style.shtml,

полезно также посмотреть:
http://www.stolyarov.info/books/codestyle

3) Любая динамически выделенная память (malloc, calloc, realloc) должна быть освобождена (free). Для каждого вызова malloc необходимо четко понимать, где и когда выполняется соответствующий ему free.

4) В случае неправильных входных данных и ошибок, возникающих во время выполнения программы, необходимо вывести информативное сообщение об ошибке и по возможности продолжить выполнение программы (например, запросить следующую команду или попросить ввести корректные данные и т.п.). Необходима "защита от дурака": программа не должна ломаться (падать) при любых (произвольных) входных данных.