Задание DST
Динамические строки

В задаче предлагается написать указанную в варианте функцию для обработки строк, отвечающую заданному прототипу (при этом в самой функции исходные строки-аргументы запрещается менять). Требуется использовать эту функцию в тестовой программе, которая должна в цикле читать строки с клавиатуры и выдавать ответ на экран до тех пор, пока пользователь не введет пустую строку (строку нулевой длины). Заданную функцию можно и **нужно** разбивать на более мелкие и понятные подфункции, обеспечивая читаемость сложного алгоритма.

В этом задании нельзя предполагать, что все вводимые строки имеют длину меньше заранее оговоренной константы (пользователь всегда может ввести строку длиннее, чем вы предполагаете). Для чтения строки и для результата надо использовать динамический буфер и функции динамического распределения памяти (написать отдельную функцию чтения строки заранее неизвестной длины `char* ReadLine(void)`). Для чтения нельзя использовать опасную функцию `gets(s)`, как альтернатива есть `fgets()` или (более медленное) посимвольное чтение через `getchar()`.

В некоторых задачах «строки» языка Си могут содержать один или несколько символов `'\n'`, то есть несколько логических строк текста (абзац). Такие абзацы необходимо читать, склеивая прочитанные логические строки, пока пользователь не обозначит пустую логическую строку (нулевой длины), написав функцию `char* ReadParagraph(void)` вместо `ReadLine`.

Буквами договоримся считать прописные и строчные латинские буквы A–Z и a–z, словами — последовательность букв и цифр 0–9. Все остальные символы в строке будем считать разделителями слов. Подстрокой будем называть произвольную часть строки.

В учебных целях при решении задач данного раздела функций стандартной библиотеки, упрощающих работу со строками и символами (напр. из `<string.h>`), следует избегать, реализуя необходимую функциональность самостоятельно.

Примеры диалога программы и пользователя:

```
Задание DST-1: Выбор слов
```

```
Введите строку: Hello, world!
Результат     : Hello, world
Введите строку: int main(int argc, char* argv[]);
Результат     : int, main, int, argc, char, argv
Введите строку: _
```

План решения

1. Сначала рекомендуется написать основную функцию программы — обработку строки. Для простоты лучше передавать в нее строки, заданные в коде статически. Для написания функции необходимо определить и реализовать ее составные части, проверить их, а затем включить в основной алгоритм.

2. После написания основной функции нужно реализовать чтение строки и выделение памяти. Так как главная функция уже была написана и протестирована, то все ошибки можно будет локализовывать в новом коде. Кроме того, такой подход поможет лучше структурировать программу.

Варианты

Вариант DST-1 (Выбор слов). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую слова исходной, разделенные запятой и пробелом. Например, из «`The good and the EVIL ones.`» должно получиться «`The, good, and, the, EVIL, ones`». Прототип:

```
char* ExtractWords(char const* str);
```

Вариант DST-2 (Выбор уникальных слов). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую слова исходной без повторений, разделенные запятой и пробелом. Например, из «`The good and the EVIL ones.`» должно получиться «`The, good, and, EVIL, ones`». Прототип:

```
char* UniqueWords(char const* str);
```

Вариант DST-3 (Выбор букв). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую буквы исходной (в том же порядке), разделенные запятой и пробелом. Например, из «`The evil ones.`» должно получиться «`T, h, e, e, v, i, l, o, n, e, s`». Прототип:

```
char* ExtractLetters(char const* str);
```

Вариант DST-4 (Выбор уникальных букв). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую буквы исходной без повторений (в том же порядке), разделенные запятой и пробелом. Например, из «`The evil.`» должно получиться «`T, h, e, v, i, l, o, n, s`». Прототип:

```
char* UniqueLetters(char const* str);
```

Вариант DST-5 (Поиск зеркальных подстрок). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую список без повторений всех подстрок длиной больше одного символа, зеркальное отражение которых также содержится в строке. Все разделители при этом следует игнорировать. Например, из «`a=babc+ba`» должно получиться «`ab, aba, ba, bab, abc, abcb, abcba, bc, bcb, bcba, cb, cba`». Прототип:

```
char* FindMirror(char const* str);
```

Вариант DST-6 (Упрощённый поиск по маске). В рамках общего условия задачи написать функцию, которая по заданной строке и строковой же «маске» создает в динамической памяти другую, содержащую список всех слов исходной строки (через запятую), удовлетворяющих маске. Маска может содержать буквы, цифры и ровно один знак `'*'`, обозначающий совпадение с любой последовательностью букв, в том числе и пустой (`"c*p"` даёт совпадение с `cp, cap, clip, creep...`). Прототип:

```
char* FindMaskWords(char const* str, char const* mask);
```

Вариант DST-7 (Поиск по маске). В рамках общего условия задачи написать функцию, которая по заданной строке и строковой же «маске» создает в динамической памяти другую, содержащую список всех слов исходной строки (через запятую), удовлетворяющих маске. Маска может содержать буквы, цифры и знаки `'*'`, обозначающие совпадение с любой последовательностью букв, в том числе и пустой (`"c*p*"` даёт совпадение с `cp, cap, clip, couple, champion...`). Прототип:

```
char* FindMaskWords(char const* str, char const* mask);
```

Вариант DST-8 (Поиск цепочек). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую список новых слов (через запятую), образованных по следующему правилу. Если одно слово исходной строки заканчивается, а какое-то другое начинается с одинакового сочетания букв (длины > 1), то эта пара образует новое слово результирующей строки (*tab**le*** + ***le**ngth* $\rightarrow$ *tab**le**ngth*). Например, из «`This is the ordinary password`» должно получиться «`This, passwordinary`». Прототип:

```
char* FindChains(char const* str);
```

Вариант DST-9 (Удалена). К сожалению, текст данной задачи был изъят.

Вариант DST-10 (Транслитерация строк). В рамках общего условия задачи написать функцию, которая по заданной строке создает в динамической памяти другую, содержащую транслитерированную строчку в соответствие со стандартами МИД РФ (www.mid.ru). В этой строке оригинальные русские буквы должны быть заменены на последовательности английских букв.

Пример: `Alyosha` («Алёша»), `narushitel' zakona` («нарушитель закона»), `zhivotnoye` («животное»). Прототип:

```
char* ConvertRussian(char const* str);
```

Вариант DST-11 (Перенос текста). В рамках общего условия задачи написать функцию, которая по заданной строковой переменной (возможно, содержащей несколько строк текста, разделенных символом перевода строки `'\n'`) создает в динамической памяти другую, содержащую исходный текст, отформатированный в один абзац так, чтобы длина каждой строки не превышала заданной ширины (вводится единожды в начале тестовой программы). Несколько подряд идущих пробелов следует заменять одним пробелом. Короткие строки надо дополнять словами со следующей строки, а слишком длинные слова переносить целиком на следующую. Например, при ширине 10 получаются такие переносы:

```
ab, k dh
asd ah
hhss ssss
```

Прототип:

```
char* FormatText(char const* str, int width);
```

Вариант DST-12 (Выравнивание текста). В рамках общего условия задачи написать функцию, которая по заданной строковой переменной (возможно, содержащей несколько строк текста, разделенных символом перевода строки `'\n'`) создает в динамической памяти другую, содержащую исходный текст, отформатированный в один абзац так, чтобы длина каждой строки была равна заданной ширине (вводится в начале тестовой программы). Делать это надо за счет изменения числа подряд идущих пробелов. Слова разбивать нельзя. Короткие строки надо дополнять словами со следующей строки, а слишком длинные слова переносить целиком на следующую. Например, при ширине 10 получается такое выравнивание:

```
ab,  k  dh
asd     ah
hhss  ssss
```

Прототип:

```
char* JustifyText(char const* str, int width);
```