ТЕХНИЧЕСКИЕ НАУКИ

TECHNICAL SCIENCE

УДК 004.43
ББК 32.973-018.1
Р 27

Рацеев С.М.
Кандидат физико-математических наук, доцент кафедры информационной безопасности и теории управления факультета математики и информационных технологий Ульяновского государственного университета, Ульяновск, тел. (8422) 32-32-47, e-mail: RatseevSM@rambler.ru
Череватенко О.И.
Кандидат физико-математических наук, доцент кафедры высшей математики физико-математического факультета Ульяновского государственного педагогического университета имени И.Н. Ульянова, Ульяновск, тел. (8422) 44-11-09, e-mail: chai@pisem.net

Алгоритмы разбиения строки на лексемы в языке программирования Си

(Рецензирована)

Аннотация
Рассмотрена задача лексического анализа входной последовательности символов, при котором производится распознавание и выделение лексем, где под лексемой подразумевается последовательность символов в некотором алфавите. Предложены алгоритмы разбиения строки на лексемы средствами языка программирования Си. Рассмотрены четыре варианта таких алгоритмов, каждый из которых имеет свои преимущества, исходя из условий поставленной задачи и ее начальных данных.
Ключевые слова: *лексема, разбиение на лексемы, лексический анализатор, алгоритм, сложность алгоритма, язык Си, формальный язык, программа.*

Ratseev S.M.
Candidate of Physics and Mathematics, Associate Professor of Department of Information Security and Control Theory at Faculty of Mathematics and Information Technologies of Ulyanovsk State University, Ulyanovsk, ph. (8422) 32-32-47, e-mail: RatseevSM@rambler.ru
Cherevatenko O.I.
Candidate of Physics and Mathematics, Associate Professor of Department of Higher Mathematics at Faculty of Physics and Mathematics of Ulyanovsk State Pedagogical University named after I.N. Ulyanov, Ulyanovsk, ph. (8422) 44-11-09, e-mail: chai@pisem.net

Algorithms of partition of a string into lexemes in the programming language C

Abstract
The paper deals with the problem of the lexical analysis of entrance sequence of symbols which are related to recognition and allocation of lexemes. Here we conceive of the lexeme as a sequence of symbols in some alphabet. Algorithms of partition of a string into lexemes by means of the programming language C are proposed. Four options of such algorithms are examined, each of which has the advantages, proceeding from conditions of an objective and its initial data.
Keywords: *lexeme, partition into lexemes, lexical analyzer, algorithm, complexity of algorithm, C language, formal language, program.*

Пусть A – некоторый конечный алфавит, в котором записываются конечные последовательности вида $x_1, ..., x_n$, где все $x_i \in A$. Пусть также B – некоторое подмножество в A. Задача состоит в том, чтобы из последовательности $x_1, ..., x_n$ выделить все лексемы, не содержащие символы множества B, то есть множество B является множеством

символов-разделителей, разбивающих последовательность $x_1, ..., x_n$ на лексемы. Данная задача является очень актуальной при обработке большого объема информации [1].

Язык Си является одним из наиболее популярных и мощных языков программирования. Он широко используется при разработке системных и прикладных программ [2, 3]. Поэтому в нашей работе эффективные алгоритмы разбиения строки на лексемы будут записаны именно на Си. В языке Си имеется функция strtok из библиотеки <string.h>, которая выполняет именно эту задачу. При этом хорошо известны негативные стороны функции strtok: портит строку на выходе; имеет относительно невысокую скорость выполнения; работает только со строками, не находящимися в области памяти read-only. В данной работе приводятся алгоритмы разбиения строк на лексемы в зависимости от исходных данных и от поставленных задач. В рассматриваемых ниже алгоритмах в качестве алфавита *A* выступает таблица символов ASCII, а множество символов разделителей *B* будем обозначать как DELIMITERS.

Рассмотрим задачу выделения всех лексем в строке-предложении. Обозначим через sentence исходную строку-предложение, в которой требуется выделить все лексемы, а через word – очередную выделенную лексему в строке sentence. Возможен случай, когда переменная sentence является массивом символов, например, char sentence[1024]. Также возможен случай, когда переменная sentence является указателем на строку: char *sentence. То же самое можно сказать и о переменной word. Данные случаи исходят из той или иной задачи работы со строками.

Случай 1. Рассмотрим такой случай: пусть переменная sentence является массивом символов и некоторые ее элементы, например, символы-разделители разрешается заменить на символ '\0', а переменная word является указателем на строку. Составим программу, которая будет заменять первый символ из каждой серии подряд идущих символов-разделителей на '\0', а указатель word будет каждый раз содержать адрес первого символа очередной лексемы в предложении sentence. Для эффективной реализации данного алгоритма введем логический массив int flag[256], в котором порядковые номера (индексы) элементов соответствуют кодам символов из таблицы символов ASCII. Элементы данного массива инициализируем следующим образом: если символ с кодом i является символом-разделителем, то полагаем flag[i] = 1, иначе flag[i] = 0. И так для всех символов из таблицы символов ASCII (i = 0, 1, …, 255):

```
int flag[256] = {0};
    for (I = 0; DELIMITERS[i]; i++)
       flag[DELIMITERS[i]] = 1;
```

Теперь проверка, является ли i-й символ строки sentence символом-разделителем, будет выглядеть следующим образом: if (flag[sentence[i]]).

Ниже приводится очень эффективный алгоритм выделения лексем из строки sentence. Заметим, что сложность данного алгоритма равна m+n, где m – длина строки DELIMITERS, n – длина строки sentence, при этом данный метод разбиения строки на лексемы работает быстрее, чем функция strtok().

```
#include <stdio.h>
#define DELIMITERS  " .,:;?!\n\t"     /* символы-разделители */
#define N 1024

int main( )
{
    char sentence[N];       /* исходная строка */
```

```
    char *word; /* адрес начала очередной лексемы в предложении */
    int i, j, flag[256] = {0};
    fgets(sentence, N, stdin);    /* вводим строку с клавиатуры */
    /* если символ с кодом i является символом-разделителем,
       то полагаем flag[i] = 1: */
    for (i = 0; DELIMITERS[i]; i++)
        flag[DELIMITERS[i]] = 1;
    /* пробегаем символы-разделители до первой лексемы в строке: */
    for (i = 0; sentence[i] && flag[sentence[i]]; i++)
        ;
    /* выделяем лексемы из строки: */
    while (sentence[i])
    {
        word = &sentence[i];  /* позиция начала новой лексемы  */
        /* пробегаем символы очередной лексемы  */
        while (sentence[i] && !flag[sentence[i]])
            i++;
        j = i;  /* запоминаем позицию начала серии разделителей */
        /* пробегаем по всем символам-разделителям: */
        while (sentence[i] && flag[sentence[i]])
            i++;
        sentence[j] = '\0';  /* отмечаем конец очередной лексемы */
        puts(word);   /* выводим на экран очередную лексему */
    }
    return 0;
}
```

Заметим, что данный алгоритм очень легко переделать в функцию, которую можно вызывать последовательно, как и функцию strtok. Только за счет использования массива int flag[256] наша функция будет работать быстрее, чем strtok. Назовем нашу функцию разбиения строки на лексемы my_strtok и для большей скорости используем в ней адресную арифметику.

```
#include<stdio.h>
#define DELIMITERS  " .,:;\n\t"      /* символы-разделители */
#define N 1024

/* разбиение строки на лексемы: */
char *my_strtok1(char *s, const int *flag)
{
   static char *beg = NULL;
   char *pword, *pbuf;
   if (s != NULL)
   {
      for(pword = s; *pword && flag[*pword]; ++pword)
         ;
      beg = pword;
   }
   else
      pword = beg;
   for( ; *beg && !flag[*beg]; ++beg)
```

```
        ;
    pbuf = beg;
    for( ; *beg && flag[*beg]; ++beg)
        ;
    *pbuf = '\0';
    return *pword ? pword : NULL;
}

int main()
{
    char s[N], *word;
    int flag[256];
    fgets(s, N, stdin);
    /* инициализируем массив flag: */
    Init(flag, DELIMITERS);
    /* разбиваем строку на лексемы: */
    word = my_strtok1(s, flag);
    while(word != NULL)
    {
        puts(word);
        word = my_strtok1(NULL, flag);
    }
    return 0;
}
```

Случай 2. Рассмотрим следующий случай. Пусть имеется некоторая строка-предложение sentence, причем переменная sentence является либо указателем на строку (то есть, в общем случае, элементы данной строки нельзя изменять, так как строка может находиться в области памяти только для чтения), либо массивом символов, значения элементов которого по условию задачи менять нельзя. Также пусть переменная word является динамическим массивом символов, и все лексемы из строки sentence требуется поочередно скопировать в переменную word.

Для того чтобы алгоритм работал очень быстро, в данном случае используем массив flag, как и в случае 1. При этом сложность данного алгоритма не превосходит числа m+2n, где m – длина строки DELIMITERS, n – длина строки sentence.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define DELIMITERS  " .,:;?!\n\t"     /* символы-разделители */
#define N 1024

int main( )
{
    char sentence[N];       /* исходная строка */
    char *word;             /* очередная лексема в предложении */
    int i, j, flag[256] = {0};
    for (i = 0; DELIMITERS[i]; i++)
        flag[DELIMITERS[i]] = 1;
    fgets(sentence, N, stdin);     /* вводим строку с клавиатуры */
    /* пробегаем символы-разделители до первой лексемы в строке */
```

```
    for (i = 0; sentence[i] && flag[sentence[i]]; i++)
        ;
    while (sentence[i])
    {
        j = i;          /* позиция начала новой лексемы */
        /* определяем позицию окончания очередной лексемы в строке */
        while (sentence[i] && !flag[sentence[i]])
            i++;
        /* выделяем память для очередной лексемы: */
        word = (char *)malloc((i - j + 1) * sizeof(char));
        /* копируем в переменную word символы очередной лексемы */
        strncpy(word, &sentence[j], i - j);
        word[i - j] = '\0';
        puts(word);
        free(word);     /* освобождаем динамическую память */
        word = NULL;
        /* пропускаем все разделители */
        while (sentence[i] && flag[sentence[i]])
            i++;
    }
    return 0;
}
```

Как и в случае 1, напишем функцию my_strtok, которую можно вызывать последовательно. Теперь функция my_strtok не будет портить строку, а будет выделять динамическую память для каждой новой лексемы и копировать выделенную лексему в эту область памяти. После копирования новой лексемы функция my_strtok будет возвращать адрес выделенной области памяти, которую после использования каждой новой лексемы необходимо очищать с помощью функции free.

```
#include<stdio.h>
#include<stdlib.h>
#define DELIMITERS  " .,:;\n\t"    /* символы-разделители */
#define N 1024

/* инициализация массива flag размера 256 */
void Init(int *flag, const char *s)
{
    int i;
    for (i = 0; i < 256; i++)
        flag[i] = 0;
    for(i = 0; s[i]; i++)
        flag[s[i]] = 1;
}

/* разбиение строки на лексемы: */
char *my_strtok2(char *s, const int *flag)
{
    static char *beg = NULL;
    char *pstr, *pword = NULL;
    int len;
```

```
    if (s != NULL)
    {
        for(pstr = s; *pstr && flag[*pstr]; ++pstr)
            ;
        beg = pstr;
    }
    else
        pstr = beg;
    for( ; *beg && !flag[*beg]; ++beg)
        ;
    if (*pstr)
    {
        pword = (char *)malloc(beg - pstr + 1);
        if (pword != NULL)
        {
            len = (beg - pstr) / sizeof(char);
            strncpy(pword, pstr, len);
            pword[len] = '\0';
        }
    }
    for( ; *beg && flag[*beg]; ++beg)
        ;
    return pword;
}

int main()
{
    char s[N], *word;
    int flag[256];
    fgets(s, N, stdin);
    /* инициализируем массив flag: */
    Init(flag, DELIMITERS);
    /* разбиваем строку на лексемы: */
    word = my_strtok2(s, flag);
    while(word != NULL)
    {
        puts(word);
        free(word);
        word = my_strtok2(NULL, flag);
    }
    return 0;
}
```

Случай 3. Алгоритм из случая 1 имел сложность m+n. Это означает, что данный алгоритм работает очень быстро. Недостаток данного алгоритма заключается в том, что, как и функция strtok(), после выделения всех лексем строка sentence будет испорчена (так как некоторые символы-разделители будут заменены символом '\0'). Алгоритм из случая 2 лишен этого недостатка, так как в данном случае лексемы копируются в массив (динамический либо статический) word. При этом сложность алгоритма оценивается сверху числом m+2n. На время работы алгоритма будет также влиять функция динамического выделения памяти malloc (поэтому иногда лучше использовать статиче-

ский массив word) и функция копирования strncpy. Алгоритм со сложностью, не превышающей число m+2n, по-прежнему является очень быстрым, но возникает вопрос, можно ли написать алгоритм выделения всех лексем из строки-предложения со сложностью как в случае 1 (m+n), но лишенный отмеченного выше недостатка. При этом рассмотрим случай, когда символы строки-предложения менять нельзя даже на время (например, строка-предложение может находиться в области памяти read only). На этот вопрос можно дать положительный ответ, правда, в этом случае каждая лексема уже будет рассматриваться не как строка, заканчивающаяся символом '\0' (как в двух предыдущих случаях), а просто как массив символов определенной размерности. Иными словами, в случаях 1 и 2 каждая очередная выделенная лексема word из строки sentence может, например, быть передана в качестве параметра функциям, принимающим указатель на начало строки (char *s) и работающим с переменной s как со строкой. Например, можно вызвать функцию определения длины строки strlen(word), функцию поиска stschr(word, c) символа c в строке word и т.д.

В данном случае лексемой будем считать последовательность символов некоторой длины. Тогда придется прописывать собственные пользовательские функции работы с такими строками, принимающими, как минимум, в качестве своих аргументов адрес начала строки и ее длину. Например, в следующем алгоритме (сложности m+n) выделения всех лексем пропишем собственную функцию вывода лексем на экран, так как функции puts() и printf() здесь применять уже нельзя.

```
#include <stdio.h>
#define DELIMITERS  " .,:;?!\n\t"     /* символы-разделители */
#define N 1024

/* вывод массива символов на экран */
void Print(char *s, int len)
{
   int i;
   for (i = 0; i < len; i++)
      putchar(s[i]);
   putchar('\n');
}

int main( )
{
   char sentence[N];       /* исходная строка */
   int i, j, flag[256] = {0};
   for (i = 0; DELIMITERS[i]; i++)
      flag[DELIMITERS[i]] = 1;
   fgets(sentence, N, stdin);     /* вводим строку с клавиатуры */
   /* пробегаем символы-разделители до первой лексемы в строке */
   for (i = 0; sentence[i] && flag[sentence[i]]; i++)
      ;
   while (sentence[i])
   {
      j = i;      /* позиция начала новой лексемы */
      /* определяем позицию окончания очередной лексемы в строке */
      while (sentence[i] && !flag[sentence[i]])
         i++;
```

```
        Print(&sentence[j], i - j);
        /* пропускаем все разделители */
        while (sentence[i] && flag[sentence[i]])
            i++;
    }
    return 0;
}
```

Как и в двух предыдущих случаях, напишем функцию my_strtok, которую можно будет вызывать последовательно. Теперь функция my_strtok будет иметь уже три параметра, при этом третий параметр будет являться указателем на целочисленную переменную, отвечающую за количество символов в очередной лексеме.

```
#include<stdio.h>
#define DELIMITERS  " .,:;\n\t"      /* символы-разделители */
#define N 1024

/* печать строки s размера n */
void Print(const char *s, const int n)
{
    int i;
    for(i = 0; i < n; i++)
        putchar(s[i]);
    putchar('\n');
}

/* инициализация массива flag размера 256 */
void Init(int *flag, const char *s)
{
    int i;
    for (i = 0; i < 256; i++)
        flag[i] = 0;
    for(i = 0; s[i]; i++)
        flag[s[i]] = 1;
}

/* разбиение строки на лексемы: */
char *my_strtok3(char *s, const int *flag, int *psize)
{
    static char *beg = NULL;
    char *pword;
    if (s != NULL)
    {
        for(pword = s; *pword && flag[*pword]; ++pword)
            ;
        beg = pword;
    }
    else
        pword = beg;
    for( ; *beg && !flag[*beg]; ++beg)
        ;
```

```
    *psize = (beg - pword) / sizeof(char);
    for( ; *beg && flag[*beg]; ++beg)
        ;
    return *pword ? pword : NULL;
}

int main()
{
    char s[N], *word;
    int flag[256], size;
    fgets(s, N, stdin);
    /* инициализируем массив flag: */
    Init(flag, DELIMITERS);
    /* разбиваем строку на лексемы: */
    word = my_strtok3(s, flag, &size);
    while(word != NULL)
    {
        Print(word, size);
        word = my_strtok3(NULL, flag, &size);
    }
    return 0;
}
```

Случай 4. Алгоритмы из случаев 1 и 3 работают очень быстро, но в первом случае исходная строка-предложение на выходе получается испорченной, а в третьем случае под лексемой подразумевается массив символов определенной размерности (не используется символ '\0'), но зато исходная строка-предложение после действия алгоритма не изменятся (не портится). Алгоритм из случая 2 лишен отмеченных недостатков, но использует функции malloc и strncpy, за счет чего проигрывает в скорости алгоритмам из случаев 1 и 3. Попробуем достигнуть компромисса, если заранее известно, что строка-предложение является массивом символов (то есть ее элементы можно менять), следующим образом. Как и в замечании к случаю 1, будем заменять первый символ из серии подряд идущих символов-разделителей на '\0', предварительно сохранив его в буферной переменной char buf. После того, как очередная выделенная лексема word станет ненужной, поместим обратно замененный символ и перейдем к выделению следующей лексемы. После действия алгоритма исходная строка-предложение будет иметь прежнее состояние, то есть не будет испорчена.

```
#include <stdio.h>
#define DELIMITERS  " .,:;?!\n\t"     /* символы-разделители */
#define N 1024

int main( )
{
    char sentence[N];       /* исходная строка */
    char *word; /* адрес начала очередной лексемы в предложении */
    char buf;
    int i, flag[256] = {0};
    fgets(sentence, N, stdin);    /* вводим строку с клавиатуры */
    /* если символ с кодом i является символом-разделителем,
       то полагаем flag[i] = 1:
```

```
    */
    for (i = 0; DELIMITERS[i]; i++)
        flag[DELIMITERS[i]] = 1;
    /* пробегаем символы-разделители до первой лексемы в строке: */
    for (i = 0; sentence[i] && flag[sentence[i]]; i++)
        ;
    /* выделяем лексемы из строки: */
    while (sentence[i])
    {
        word = &sentence[i];  /* позиция начала новой лексемы  */
        /* пробегаем символы очередной лексемы  */
        while (sentence[i] && !flag[sentence[i]])
            i++;
        /* запоминаем первый символ разделитель из серии
           разделителей: */
        buf = sentence[i];
        /* заменяем разделитель на '\0': */
        sentence[i] = '\0';
        puts(word);   /* выводим на экран очередную лексему */
        /* возвращаем обратно символ-разделитель: */
        sentence[i] = buf;
        /* пробегаем по всем символам-разделителям: */
        while (sentence[i] && flag[sentence[i]])
            i++;
    }
    return 0;
}
```

Для этого случая также можно преобразовать данный алгоритм в функцию my_strtok таким образом, чтобы ее можно было вызывать последовательно:

```
#include<stdio.h>
#define DELIMITERS  " .,:;\n\t"      /* символы-разделители */
#define N 1024

/* инициализация массива flag размера 256 */
void Init(int *flag, const char *s)
{
    int i;
    for (i = 0; i < 256; i++)
        flag[i] = 0;
    for(i = 0; s[i]; i++)
        flag[s[i]] = 1;
}

/* разбиение строки на лексемы: */
char *my_strtok4(char *s, const int *flag)
{
    static char *beg = NULL;
    static char buf = '\0';
    static char *pbuf = NULL;
```

```
    char *pword;
    if (s != NULL)
    {
        for(pword = s; *pword && flag[*pword]; ++pword)
            ;
        beg = pword;
    }
    else
    {
        pword = beg;
        *pbuf = buf;
    }
    for( ; *beg && !flag[*beg]; ++beg)
        ;
    pbuf = beg;
    buf = *beg;
    for( ; *beg && flag[*beg]; ++beg)
        ;
    *pbuf = '\0';
    return *pword ? pword : NULL;
}

int main()
{
    char s[N], *word;
    int flag[256];
    fgets(s, N, stdin);
    /* инициализируем массив flag: */
    Init(flag, DELIMITERS);
    /* разбиваем строку на лексемы: */
    word = my_strtok4(s, flag);
    while(word != NULL)
    {
        puts(word);
        word = my_strtok4(NULL, flag);
    }
    return 0;
}
```

Примечания:

1. Хантер Р. Основные концепции компиляторов. М.: Вильямс. 2002. 256 с.
2. Керниган Б., Ритчи Д. Язык программирования Си. М.: Вильямс. 2009. 304 с.
3. Прата С. Язык программирования С. Лекции и упражнения. М.: Вильямс, 2006. 960 с.

References:

1. Hunter R. Main concepts of compilers. M.: Williams. 2002. 256 pp.
2. Kernighan B., Ritchi D. The C programming language. M.: Williams. 2009. 304 pp.
3. Prata S. The programming language C. Lectures and exercises. M.: Williams, 2006. 960 pp.