Задача

Из входного потока вводится произвольное число строк. Длина строки не ограничена. Каждая строка содержит последовательность слов, разделённых пробелами и/или знаками табуляции. Строка представлена списком (первое поле элемента списка - символ строки; второе - указатель на следующий элемент списка или NULL в конце списка). Конец ввода определяется концом файла.

Для каждой строки сформировать новую строку, поместив в нее каждое второе слово исходной последовательности. Слова в новой строке должны быть разделены только одним пробелом. Полученную строку вывести в выходной поток.

Примечания:

1. Для ввода строк неопределённой длины посимвольный ввод не использовать!
2. Логически законченные части алгоритма решения задачи должны быть оформлены отдельными функциями с параметрами (например, ввод строки – списка, вывод списка, пропуск разделителей и др.).
3. Целесообразно обработку строки оформить отдельной функцией; структура программы должна быть следующей: пока не обнаружен конец файла {ввести строку, представленную списком; обработать список в соответствии с условием задачи; вывести результат; освободить память, выделенную под список}.
4. Новый список формировать, модифицируя исходный список.

```
#include <stdio.h>
#include <malloc.h>
#include <string.h>

typedef struct Item {
      char c;
      struct Item *next;
} Item;

/* Ввод строки - списка */
int getList(Item **);

/* Вывод строки - списка */
void putList(const char *, Item *);

/* Освобождение памяти */
Item *deleteList(Item *);

/* Формирование новой строки - списка */
Item *newString(Item *);

/* удаление пробелов из строки - списка */
Item *delSpace(Item *);

/* пропуск слова в строке - списке */
Item *skipWord(Item *);

/* удаление слова из строки - списка */
Item *delWord(Item *);

/* Функция для ввода строки - списка.
** Функция возвращает указатель на список, в котором размещена введенная строка.
** По условиям задачи, может быть введена пустая строка, тогда
** функция возвращает значение NULL - пустой список.
** Функция должна отслеживать состояние "Конец файла", и это состояние нельзя
```

```
** связать со значением указателя NULL - пустая строка.
** Поэтому предлагаемая функция с прототипом int getList(Item **) возвращает
** результат 1, если строка введена, или 0, если обнаружен конец файла,
** а саму строку возвращает через параметр функции.
*/
int getList(Item **pptr)
{
      char buf[21], *str;
      Item head = {'*', NULL};
            Item *last = &head;
      int n, rc = 1;

      do{
            n = scanf("%20[^\n]", buf);
            if(n < 0){
                  deleteList(head.next);
                  head.next = NULL;
                  rc = 0;
                  continue;
            }
            if(n > 0){
                  for(str = buf; *str != '\0'; ++str){
                        last->next = (Item *)malloc(sizeof(Item));
                        last = last->next;
                        last->c = *str;
                  }
                  last->next = NULL;
            }
            else
                  scanf("%*c");
      } while(n > 0);
      *pptr = head.next;
      return rc;
}

/* Вывод строки - списка */
void putList(const char *msg, Item *ptr)
{
      printf("%s: \"", msg);
      for(; ptr != NULL; ptr = ptr->next)
            printf("%c", ptr->c);
      printf("\"\n");
}

/* Освобождение памяти */
Item *deleteList(Item *ptr)
{
      Item *tmp = NULL;
      while(ptr != NULL){
            tmp = ptr;
            ptr = ptr->next;
            free(tmp);
      }
      return ptr;
}

int main()
```

```
{
      Item *ptr = NULL;
      while(puts("enter string"), getList(&ptr)){
            putList("Entered string", ptr);
            ptr = newString(ptr);
            putList("Result string", ptr);
            ptr = deleteList(ptr);
      }
      return 0;
}

/* Формирование новой строки - списка */
Item *newString(Item *p)
{
      Item head = {'*', p};
      Item *cur = &head, *prev = &head; /* prev - для корректной обработки конца
списка */
      int fl = 0;

      /* Используется список с головным элементом; поэтому cur указывает
      ** на элемент списка, предшествующий тому, который будет анализироваться
      */
      while(cur && (cur->next = delSpace(cur->next))){
            /* здесь надо анализировать и cur, так как в результате пропуска
            ** слова cur указывает на элемент списка, следующий за последним
символом
            ** слова, а его может и не быть - последний символ слова есть последний
            ** символ списка
            */
            if(fl){
                  cur = skipWord(cur->next);
                  prev = cur; /* сохраняем последний символ слова */
                  cur = cur->next; /* теперь cur указывает на элемент после слова
*/
                  if(cur){ /* есть какое-то продолжение списка, cur указывает на
                              ** пробельный элемент списка */
                        cur->c = ' ';
                  }
            }
            else
                  cur->next = delWord(cur->next);

            fl = !fl;
      }
      /* возможно, в конце строки - списка остался лишний пробел */
      if(prev->next){
            free(prev->next);
            prev->next = NULL;
      }
      return head.next;
}

/* удаление пробелов из строки - списка
** результат - указатель на элемент списка, содержащего
** непробельный символ, или NULL
*/
Item *delSpace(Item *p)
```

```
{
      Item *tmp;
      while(p && (p->c == ' ' || p->c == '\t')){
            tmp = p;
            p = p->next;
            free(tmp);
      }
      return p;
}

/* пропуск слова в строке - списке
** результат - указатель в исходном списке на
** послений символ слова
*/
Item *skipWord(Item *p)
{
      while(p->next && p->next->c != ' ' && p->next->c != '\t')
            p = p->next;
      return p;
}

/* удаление слова из строки - списка
** результат - указатель на элемент списка, содержащего
** пробельный символ, или NULL
*/
Item *delWord(Item *p)
{
      Item *tmp;
      while(p && p->c != ' ' && p->c != '\t'){
            tmp = p;
            p = p->next;
            free(tmp);
      }
      return p;
}
```