#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_LINES 100

#define MAX\_LENGTH 100

// 分割字符串得到单词数组

void split\_string(char \*str, char \*words[], int \*num\_words) {

char \*token = strtok(str, " ");

\*num\_words = 0;

while (token!= NULL) {

words[\*num\_words] = token;

(\*num\_words)++;

token = strtok(NULL, " ");

}

}

// 处理输入并统计单词相关信息

void process\_input() {

char \*input[MAX\_LINES];

int line\_count = 0;

printf("请输入多行英文字符串，输入END结束：\n");

// 接收用户输入

while (line\_count < MAX\_LINES) {

input[line\_count] = (char \*)malloc(MAX\_LENGTH \* sizeof(char));

fgets(input[line\_count], MAX\_LENGTH, stdin);

// 去除换行符

input[line\_count][strcspn(input[line\_count], "\n")] = '\0';

// 判断是否结束输入

if (strcmp(input[line\_count], "END") == 0) {

break;

}

line\_count++;

}

// 用于存储单词及其相关计数信息

char \*words[MAX\_LINES \* MAX\_LENGTH];

int word\_line\_count[MAX\_LINES \* MAX\_LENGTH][MAX\_LINES];

int word\_total\_count[MAX\_LINES \* MAX\_LENGTH];

int unique\_word\_count = 0;

// 初始化计数数组

for (int i = 0; i < MAX\_LINES \* MAX\_LENGTH; i++) {

for (int j = 0; j < MAX\_LINES; j++) {

word\_line\_count[i][j] = 0;

}

word\_total\_count[i] = 0;

}

// 处理每一行输入，统计单词信息

for (int i = 0; i < line\_count; i++) {

int num\_words;

char \*line\_words[MAX\_LENGTH];

split\_string(input[i], line\_words, &num\_words);

for (int j = 0; j < num\_words; j++) {

int found = 0;

for (int k = 0; k < unique\_word\_count; k++) {

if (strcmp(line\_words[j], words[k]) == 0) {

word\_line\_count[k][i]++;

word\_total\_count[k]++;

found = 1;

break;

}

}

if (!found) {

words[unique\_word\_count] = line\_words[j];

for (int l = 0; l < MAX\_LINES; l++) {

word\_line\_count[unique\_word\_count][l] = 0;

}

word\_line\_count[unique\_word\_count][i] = 1;

word\_total\_count[unique\_word\_count] = 1;

unique\_word\_count++;

}

}

}

// 输出统计结果

printf("\n统计结果如下：\n");

for (int i = 0; i < unique\_word\_count; i++) {

printf("单词：%s\n", words[i]);

printf("在每一行出现的次数：");

for (int j = 0; j < line\_count; j++) {

printf("%d ", word\_line\_count[i][j]);

}

printf("\n在输入中总共出现的次数：%d\n\n", word\_total\_count[i]);

}

// 释放为存储输入字符串分配的内存

for (int i = 0; i < line\_count; i++) {

free(input[i]);

}

}

int main() {

process\_input();

return 0;

}#include <stdio.h>

#include <stdlib.h>

// 链表节点结构体

typedef struct list\_item {

int data;

struct list\_item \*next;

} list\_item;

// 创建新节点函数

list\_item \*create\_node(int value) {

list\_item \*new\_node = (list\_item \*)malloc(sizeof(list\_item));

if (new\_node == NULL) {

perror("Memory allocation failed");

return NULL;

}

new\_node->data = value;

new\_node->next = NULL;

return new\_node;

}

// 在链表末尾添加节点函数

void append\_node(list\_item \*\*head, int value) {

list\_item \*new\_node = create\_node(value);

if (\*head == NULL) {

\*head = new\_node;

return;

}

list\_item \*current = \*head;

while (current->next!= NULL) {

current = current->next;

}

current->next = new\_node;

}

// 非递归版本的链表空间释放函数

void erase\_list(list\_item \*head) {

list\_item \*current = head;

list\_item \*next;

while (current!= NULL) {

next = current->next;

free(current);

current = next;

}

}

int main() {

list\_item \*head = NULL;

// 创建并添加一些节点到链表

append\_node(&head, 10);

append\_node(&head, 20);

append\_node(&head, 30);

// 释放链表空间

erase\_list(head);

// 此时链表已被释放，head应该为NULL

if (head == NULL) {

printf("链表空间已成功释放。\n");

} else {

printf("链表空间释放失败。\n");

}

return 0;

}