第1章《正整数个数统计》实验报告

21051117 卢俊成

主要数据结构和变量

- 链表（节点 `Node`）
- 加入数字 `void Add(Node*, int)`
- 链上插入排序 `void Sort(Node*)`

程序主要流程

1. 初始化链表
2. 读入数据直到输入为`0`
3. 链上插入排序
4. 打印链表内容
5. 释放链表

程序主要函数功能

- `void Add(Node*, int)`
  顺序查找链表中的数字，若不存在则插入，若存在则次数加1
- `void Sort(Node*)`
  链上插入排序
- `void Print(Node*)`
  打印链表内容
- `void DelTable(Node*)`
  释放链表空间

已实现功能

全部题目要求

编译与运行信息

编译信息

```
/usr/bin/gcc -std=c17 -fdiagnostics-color=always -Wfatal-errors -Wall -Wextra -g
1.c -o 1 -lm
```

测试数据1

输入

```
1 1 2 2 2 3 4 5 6 3 5 4 0
```

输出

```
2:3
1:2
3:2
4:2
5:2
6:1
```

源代码

```
// Work on stdc17
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int number;
    int count;
    struct node* next;
} Node;

void Add(Node* head, int number) {
    Node* current = head;
    while (current->next != NULL) {
        if (current->next->number == number) {
            current->next->count++;
            return;
        }
        current = current->next;
    }
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }

    newNode->number = number;
    newNode->count = 1;
    newNode->next = NULL;
    current->next = newNode;
}

void Sort(Node* head) {
    Node* current = head->next;
    if (current == NULL) return;
    Node* tmp;
    while (current->next != NULL) {
        tmp = head;
        while (tmp->next->count >= current->next->count && tmp->next != current-
```

```
>next)
            tmp = tmp->next;
        if (tmp->next != current->next) {
            Node* tmp2 = current->next;
            current->next = tmp2->next;
            tmp2->next = tmp->next;
            tmp->next = tmp2;
        } else
            current = current->next;
    }
}

void Print(Node* node) {
    while (node != NULL) {
        printf("%d:%d\n", node->number, node->count);
        node = node->next;
    }
}

void DelTable(Node* node) {
    while (node != NULL) {
        Node* temp = node;
        node = node->next;
        free(temp);
    }
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    freopen("1.err", "w", stderr);
#endif
    int n;
    Node* head = (Node*)malloc(sizeof(Node));
    if (head == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    head->number = -1;
    head->count = 0;
    head->next = NULL;
    while (scanf("%d", &n), n)
        Add(head, n);
    Sort(head);
    Print(head->next);
    DelTable(head);
#ifndef ONLINE_JUDGE
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif
    return 0;
}
```

第1章《英文文件单词词频统计》实验报告

21051117 卢俊成

主要数据结构和变量

- 链表（存储节点 `Word`）
  - 单词（`char*`）
  - 出现次数（`int`）
  - 指针 `next`
- 临时单词 `*word`

程序主要流程

1. 检查传入参数
2. 打开文件
3. 初始化链表
4. 循环读入字符，拼成单词
5. 查表比较停用词
6. 向主表插入单词或更新出现次数
7. 链上插入排序（双因素）
8. 格式化输出
9. 释放链表空间

程序主要函数功能

- `void Add(Word*, char*, int)`
  - 比较停用词表
  - 查主表，若有，更新出现次数
  - 若无，压缩存储单词，插入主表
- `void Sort(Word*)`
  链上插入排序，第一因素为出现次数，第二因素为单词字典序
- `void Print(FILE*, Word*, int maxlen, int sum)`
  格式化输出
- `void DelTable(Word*)`
  释放链表空间

已实现功能

全部要求的功能

编译与运行信息

编译信息

```
/usr/bin/gcc -std=c17 -fdiagnostics-color=always -Wfatal-errors -Wall -Wextra -g
1.c -o 1 -lm
```

测试命令行

```
./1 1.in 1.out
```

测试数据

输入

```
Our vicar is always raising money for one cause or another, but he has never
managed to get enough money to have the church clock repaired. The big clock
which used to strike the hours day and night was damaged many years ago and has
been silent ever since.

One night, however, our vicar woke up with a start: the clock was striking the
hours! Looking at his watch, he saw that it was one o'clock, but the bell struck
thirteen times before it stopped. Armed with a torch, the vicar went up into the
clock tower to see what was going on. In the torchlight, he caught sight of a
figure whom he immediately recognized as Bill Wilkins, our local grocer.

'Whatever are you doing up here Bill?' asked the vicar in surprise.
'I'm trying to repair the bell,' answered Bill. 'I've been coming up here night
after night for weeks now. You see, I was hoping to give you a surprise.'
'You certainly did give me a surprise!' said the vicar. 'You've probably woken up
everyone in the village as well. Still, I'm glad the bell is working again.'
That's the trouble, vicar,' answered Bill. 'It's working all right, but I'm
afraid that at one o'clock it will strike thirteen times and there's nothing I
can do about it."
We'll get used to that, Bill,' said the vicar. "Thirteen is not as good as one,
but it's better than nothing. Now let's go downstairs and have a cup of tea.'
```

输出

```
    1          to    7  2.64%
    2       vicar    7  2.64%
    3       clock    6  2.26%
    4        bill    5  1.89%
    5         one    5  1.89%
    6          up    5  1.89%
    7         was    5  1.89%
    8         and    4  1.51%
    9          as    4  1.51%
   10       night    4  1.51%
   11        that    4  1.51%
   12        bell    3  1.13%
   13          is    3  1.13%
   14         our    3  1.13%
```

15	surprise	3	1.13%
16	thirteen	3	1.13%
17	answered	2	0.75%
18	been	2	0.75%
19	get	2	0.75%
20	give	2	0.75%
21	has	2	0.75%
22	have	2	0.75%
23	here	2	0.75%
24	hours	2	0.75%
25	money	2	0.75%
26	nothing	2	0.75%
27	now	2	0.75%
28	of	2	0.75%
29	said	2	0.75%
30	see	2	0.75%
31	strike	2	0.75%
32	times	2	0.75%
33	used	2	0.75%
34	ve	2	0.75%
35	with	2	0.75%
36	working	2	0.75%
37	about	1	0.38%
38	afraid	1	0.38%
39	after	1	0.38%
40	again	1	0.38%
41	ago	1	0.38%
42	all	1	0.38%
43	always	1	0.38%
44	another	1	0.38%
45	are	1	0.38%
46	armed	1	0.38%
47	asked	1	0.38%
48	before	1	0.38%
49	better	1	0.38%
50	big	1	0.38%
51	can	1	0.38%
52	caught	1	0.38%
53	cause	1	0.38%
54	certainly	1	0.38%
55	church	1	0.38%
56	coming	1	0.38%
57	cup	1	0.38%
58	damaged	1	0.38%
59	day	1	0.38%
60	did	1	0.38%
61	do	1	0.38%
62	doing	1	0.38%
63	downstairs	1	0.38%
64	enough	1	0.38%
65	ever	1	0.38%
66	everyone	1	0.38%
67	figure	1	0.38%
68	glad	1	0.38%
69	go	1	0.38%

70	going	1	0.38%
71	good	1	0.38%
72	grocer	1	0.38%
73	his	1	0.38%
74	hoping	1	0.38%
75	however	1	0.38%
76	immediately	1	0.38%
77	into	1	0.38%
78	let	1	0.38%
79	ll	1	0.38%
80	local	1	0.38%
81	looking	1	0.38%
82	managed	1	0.38%
83	many	1	0.38%
84	me	1	0.38%
85	never	1	0.38%
86	not	1	0.38%
87	probably	1	0.38%
88	raising	1	0.38%
89	recognized	1	0.38%
90	repair	1	0.38%
91	repaired	1	0.38%
92	right	1	0.38%
93	saw	1	0.38%
94	sight	1	0.38%
95	silent	1	0.38%
96	since	1	0.38%
97	start	1	0.38%
98	still	1	0.38%
99	stopped	1	0.38%
100	striking	1	0.38%
101	struck	1	0.38%
102	tea	1	0.38%
103	than	1	0.38%
104	there	1	0.38%
105	torch	1	0.38%
106	torchlight	1	0.38%
107	tower	1	0.38%
108	trouble	1	0.38%
109	trying	1	0.38%
110	village	1	0.38%
111	watch	1	0.38%
112	we	1	0.38%
113	weeks	1	0.38%
114	well	1	0.38%
115	went	1	0.38%
116	what	1	0.38%
117	whatever	1	0.38%
118	which	1	0.38%
119	whom	1	0.38%
120	wilkins	1	0.38%
121	will	1	0.38%
122	woke	1	0.38%
123	woken	1	0.38%

```
  124       years    1  0.38%
total word count = 265
```

源代码

```
// Work on stdc17
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define MAXLEN 1000

typedef struct word {
    char* w;
    int count;
    struct word* next;
} Word;

char* stopword[ ] = { "you","he","she","they","it",
                      "the","an","by","for","mr",
                      "in","on","at","but","be","new",
                      "of","and","or","to","so",
                      "is","are","am","was","were",
                      "as","will","shall","should","can","could",
                      "this","that","these","those",
                      "with","not","then",
                      "myself","yourself","himself","herself","itself",
                      "yourselves","themselves" };


void Add(Word* head, char* w, int len) {
    if (len <= 1) return;
    char* p = *stopword;
    for (long unsigned int i = 0; i < sizeof(stopword) / sizeof(char*); i++, p++)
        if (strcmp(w, p) == 0)
            return;
    Word* current = head;
    while (current->next != NULL) {
        if (strcmp(current->next->w, w) == 0) {
            current->next->count++;
            return;
        }
        current = current->next;
    }
    Word* newWord = (Word*)malloc(sizeof(Word));
    if (newWord == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    newWord->w = (char*)malloc(sizeof(char) * (len + 1));
    if (newWord->w == NULL) {
        fprintf(stderr, "Error for malloc.\n");
```

```
        exit(1);
    }
    strcpy(newWord->w, w);
    newWord->count = 1;
    newWord->next = NULL;
    current->next = newWord;
}

void Sort(Word* head) {
    Word* current = head->next;
    if (current == NULL) return;
    Word* tmp;
    while (current->next != NULL) {
        tmp = head;
        while ((tmp->next->count > current->next->count ||
            (tmp->next->count == current->next->count && strcmp(tmp->next->w,
current->next->w) < 0)) &&
            tmp->next != current->next)
            tmp = tmp->next;
        if (tmp->next != current->next) {
            Word* tmp2 = current->next;
            current->next = tmp2->next;
            tmp2->next = tmp->next;
            tmp->next = tmp2;
        } else
            current = current->next;
    }
}

void Print(FILE* out, Word* node, int maxlen, int sum) {
    char ss[30];
    sprintf(ss, "%%5d %%%ds %%4d %%5.2f%%%%\n", maxlen);
    int i = 0;
    while (node != NULL) {
        fprintf(out, ss, ++i, node->w, node->count, node->count * 100.0 / sum);
        node = node->next;
    }
    fprintf(out, "total word count = %d\n", sum);
}

void DelTable(Word* node) {
    while (node != NULL) {
        Word* temp = node;
        node = node->next;
        free(temp->w);
        free(temp);
    }
}

int main(int argc, char* argv[ ]) {
    if (argc != 3) {
        printf("Usage: %s <inputfile> <outputfile>\n", argv[0]);
        exit(1);
    }
    FILE* in = fopen(argv[1], "r");
```

```
    if (in == NULL) {
        fprintf(stderr, "Error for open %s.\n", argv[1]);
        exit(1);
    }
    FILE* out = fopen(argv[2], "w");
    if (out == NULL) {
        fprintf(stderr, "Error for writing to %s.\n", argv[2]);
        exit(1);
    }
    Word* head = (Word*)malloc(sizeof(Word));
    if (head == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    head->w = NULL;
    head->count = -1;
    head->next = NULL;
    char ch, * word = (char*)malloc(sizeof(char) * MAXLEN);
    int maxlen = 0, cnt = 0;
    if (word == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    while ((ch = tolower(fgetc(in))) != EOF) {
        if (isalpha(ch)) {
            int i = 0;
            while (isalpha(ch)) {
                word[i++] = ch;
                ch = tolower(fgetc(in));
            }
            word[i] = '\0';
            Add(head, word, i);
            maxlen = i > maxlen ? i : maxlen;
            cnt++;
        }
    }
    Sort(head);
    Print(out, head->next, maxlen, cnt);
    DelTable(head);
    free(word);
    fclose(in);
    fclose(out);
    return 0;
}
```

第1章《栈》实验报告

21051117 卢俊成

主要数据结构和变量

- 节点 `Node`
- 栈 `Stack` 及其操作函数 `S_*`
- 队列 `Queue` 及其操作函数 `Q_*`

程序主要流程

1. 初始化栈和队列
2. 读入数字，并将正数和负数分别压入栈和队列
3. 分别输出栈和队列内的数
4. 释放空间

程序主要函数功能

- 栈操作函数 `S_*` 和 队列操作函数 `Q_*`

已实现功能

全部题目要求功能

编译与运行信息

编译信息

```
/usr/bin/gcc -std=c17 -fdiagnostics-color=always -Wfatal-errors -Wall -Wextra -g
1.c -o 1 -lm
```

测试数据

输入

```
10709
15685
29913
2034
6935
10104
20386
12599
21628
14315
-11833
```

```
-28375
-13005
-29892
-4244
-13470
-5515
-3426
-17944
-31934
0
```

输出

```
14315 21628 12599 20386 10104 6935 2034 29913 15685 10709
-11833 -28375 -13005 -29892 -4244 -13470 -5515 -3426 -17944 -31934
```

源代码

```c
// Work on stdc17
#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    int data;
    struct node* next;
} Node;

typedef struct queue {
    int length;
    Node* head;
    Node* tail;
} Queue;

Queue* Q_Create() {
    Queue* q = (Queue*)malloc(sizeof(Queue));
    if (q == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    q->length = 0;
    q->head = NULL;
    q->tail = NULL;
    return q;
}

void Q_Destroy(Queue* q) {
    Node* current = q->head;
    Node* tmp;
    while (current != NULL) {
```

```
        tmp = current;
        current = current->next;
        free(tmp);
    }
    free(q);
}

int Q_IsEmpty(Queue* q) {
    return q->length == 0;
}

int Q_Length(Queue* q) {
    return q->length;
}

int Q_getTop(Queue* q) {
    if (Q_IsEmpty(q)) {
        fprintf(stderr, "Queue is empty.\n");
        exit(1);
    }
    return q->head->data;
}

void Q_pop(Queue* q) {
    if (Q_IsEmpty(q)) {
        fprintf(stderr, "Queue is empty.\n");
        exit(1);
    }
    Node* tmp = q->head;
    q->head = q->head->next;
    free(tmp);
    q->length--;
    if (q->length == 0) {
        q->tail = NULL;
    }
}

void Q_push(Queue* q, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    newNode->data = data;
    newNode->next = NULL;
    if (Q_IsEmpty(q)) {
        q->head = newNode;
        q->tail = newNode;
    } else {
        q->tail->next = newNode;
        q->tail = newNode;
    }
    q->length++;
}
```

```c
typedef struct Stack {
    int length;
    Node* head;
} Stack;

Stack* S_Create() {
    Stack* s = (Stack*)malloc(sizeof(Stack));
    if (s == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
    }
    s->length = 0;
    s->head = NULL;
    return s;
}

void S_Destroy(Stack* s) {
    Node* current = s->head;
    Node* tmp;
    while (current != NULL) {
        tmp = current;
        current = current->next;
        free(tmp);
    }
    free(s);
}

int S_IsEmpty(Stack* s) {
    return s->length == 0;
}

int S_getTop(Stack* s) {
    if (S_IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(1);
    }
    return s->head->data;
}

void S_pop(Stack* s) {
    if (S_IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(1);
    }
    Node* tmp = s->head;
    s->head = s->head->next;
    free(tmp);
    s->length--;
}

void S_push(Stack* s, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(1);
```

```
    }
    newNode->data = data;
    newNode->next = s->head;
    s->head = newNode;
    s->length++;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    freopen("1.err", "w", stderr);
#endif
    Queue* q = Q_Create();
    Stack* s = S_Create();
    int t;
    while (~scanf("%d", &t), t)
        if (t > 0)
            S_push(s, t);
        else
            Q_push(q, t);
    while (!S_IsEmpty(s)) {
        printf("%d ", S_getTop(s));
        S_pop(s);
    }
    putchar('\n');
    while (!Q_IsEmpty(q)) {
        printf("%d ", Q_getTop(q));
        Q_pop(q);
    }
    Q_Destroy(q);
    S_Destroy(s);
#ifndef ONLINE_JUDGE
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif
    return 0;
}
```

第1章《表达式解释》实验报告

21051117 卢俊成

主要数据结构和变量

- 栈 `Stack` 及其相关操作函数

程序主要流程

与题目提示一致

程序主要函数功能

- 栈 `Stack` 相关操作函数
- `int priority(char)`
  返回运算符优先级
- `int operate(int, int, char)`
  实际运算并返回

已实现功能

全部题目要求功能

编译与运行信息

编译信息

```
/usr/bin/gcc -std=c17 -fdiagnostics-color=always -Wfatal-errors -Wall -Wextra -g
1.c -o 1 -lm
```

测试数据1

输入

```
((2+3)*6/8+9*20)#
```

输出

```
183
```

测试数据2

输入

```
#
```

输出

```
0
```

测试数据3

输入

```
1+2+3+4+5+6+7+8+9+10#
```

输出

```
55
```

测试数据4

输入

```
100-9-8-7-6-5-4-3-2-1#
```

输出

```
55
```

测试数据5

输入

```
2*3*5*7#
```

输出

```
210
```

测试数据6

输入

```
441/5/7#
```

输出

```
12
```

测试数据7

输入

```
20*4+15/9#
```

输出

```
81
```

测试数据8

输入

```
()#
```

输出

```
0
```

测试数据9

输入

```
1/0#
```

输出

```
Divide by zero.
```

测试数据10

输入

```
8*2/9*9-1/8+5/8*5#
```

输出

```
9
```

源代码

```
// Work on stdc17
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

typedef struct node {
    int data;
    struct node* next;
} Node;

typedef struct Stack {
    int length;
    Node* head;
} Stack;

Stack* Create() {
    Stack* s = (Stack*)malloc(sizeof(Stack));
    if (s == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(-1);
    }
    s->length = 0;
    s->head = NULL;
    return s;
}
```

```
void Destroy(Stack* s) {
    Node* current = s->head;
    Node* tmp;
    while (current != NULL) {
        tmp = current;
        current = current->next;
        free(tmp);
    }
    free(s);
}

int IsEmpty(Stack* s) {
    return s->length == 0;
}

int getTop(Stack* s) {
    if (IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(-1);
    }
    return s->head->data;
}

void pop(Stack* s) {
    if (IsEmpty(s)) {
        fprintf(stderr, "Stack is empty.\n");
        exit(-1);
    }
    Node* tmp = s->head;
    s->head = s->head->next;
    free(tmp);
    s->length--;
}

void push(Stack* s, int data) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    if (newNode == NULL) {
        fprintf(stderr, "Error for malloc.\n");
        exit(-1);
    }
    newNode->data = data;
    newNode->next = s->head;
    s->head = newNode;
    s->length++;
}

int priority(char c) {
    if (c == ')') return 3;
    if (c == '*' || c == '/') return 2;
    if (c == '+' || c == '-') return 1;
    return 0;
}

int operate(int a, int b, char c) {
```

```
    if (c == '+') return a + b;
    if (c == '-') return a - b;
    if (c == '*') return a * b;
    if (c == '/') {
        if (b == 0) {
            fprintf(stderr, "Divide by zero.\n");
            exit(-1);
        }
        return a / b;
    }
    return 0;
}

int main() {
#ifndef ONLINE_JUDGE
    freopen("1.in", "r", stdin);
    freopen("1.out", "w", stdout);
    freopen("1.err", "w", stderr);
#endif
    Stack* num = Create();
    Stack* op = Create();
    char ch;
    int n, num1, num2;
    push(op, '(');
    while ((ch = getchar())) {
        if (isdigit(ch)) {
            n = ch - '0';
            while (isdigit(ch = getchar()))
                n = n * 10 + ch - '0';
            push(num, n);
        }
        if (ch == '(') {
            push(op, ch);
            continue;
        }
        if (ch == ')') {
            while (getTop(op) != '(') {
                num1 = getTop(num);
                pop(num);
                num2 = getTop(num);
                pop(num);
                push(num, operate(num2, num1, getTop(op)));
                pop(op);
            }
            pop(op);
            continue;
        }
        if (ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            if (IsEmpty(op) || (priority(ch) > priority(getTop(op)))) {
                push(op, ch);
                continue;
            }
            num1 = getTop(num);
            pop(num);
            num2 = getTop(num);
```

```
            pop(num);
            push(num, operate(num2, num1, getTop(op)));
            pop(op);
            push(op, ch);
            continue;
        }
        if (ch == '#') break;
    }
    while (getTop(op) != '(') {
        num1 = getTop(num);
        pop(num);
        num2 = getTop(num);
        pop(num);
        push(num, operate(num2, num1, getTop(op)));
        pop(op);
    }
    if (IsEmpty(num)) {
        printf("0\n");
    } else {
        printf("%d\n", getTop(num));
    }
    Destroy(num);
    Destroy(op);
#ifndef ONLINE_JUDGE
    fclose(stdin);
    fclose(stdout);
    fclose(stderr);
#endif
    return 0;
}
```