北京科技大学 计算机与通信工程学院

数据结构实验报告

专　业：计算机科学与技术

班　级：计1503

学生姓名：唐誉源

学　号：41503302

指导教师：齐悦

实验地点：机电楼304

实验时间：2017 年 5 月 8 日~ 6 月 3 日

实验成绩：

一、实验目的与实验要求

1 实验目的

（1）加深对常用数据结构和算法设计基本思路、思考方法及其适用场合的理解，并能运用于解决实际问题；

（2）能根据特定问题需求，分析建立计算模型（包括逻辑结构和物理结构）、设计算法和程序，并在设计中综合考虑多种因素，对结果的有效性进行分析；

（3）训练分析问题、解决问题的能力以及自主学习与程序设计实践能力；

（4）形成将非数值型问题抽象为计算模型到算法设计、程序实现、结果有效性分析的能力。

2 实验要求

（1）由于在有限的实验课内学时难以较好完成所有实验内容，因此要求在实验课前自主完成部分实验或实验的部分内容；

（2）对于每个实验都要针对问题进行分析，设计出有效的数据结构、算法和程序，对实现结果的正确性进行测试验证，给出测试用例和结果，分析算法的时间复杂度、空间复杂度、有效性和不足，在算法设计和实现过程中体现创新意识，并能综合考虑时空权衡、用户的友好性、程序的模块化和扩展性等；（如实验中有优化或改进的设计，请将优化过程进行说明，并给出前后对比分析。）

（3）完成的每个实验需要在实验课内经指导教师现场检查、查看程序代码，回答指导教师提出的问题，以确认实验实际完成的质量；

（4）在实验报告中体现问题分析、算法思路、算法描述、程序实现和验证、算法和结果的有效性分析。

二、实验设备（环境）及要求

实验室提供Windows 7系统下的Visual C++环境。本实验目的是对数据结构知识掌握及应用能力的考察，对编程语言和开发环境不做严格要求。建议采用C语言，在支持C的编译环境下运行。

三、实验内容、步骤与结果分析

1 实验1：链表的应用

1.1 实验内容

输入数据（设为整型）建立单链表，并求相邻k个节点data值之和为最大的第一节点。

要求：

（1）建立链表、求最大值功能采用独立函数实现；

（2）可根据用户需求多次建表；

（3）数据的输入可从键盘输入，也可从txt文件输入；

（4）程序结束时要释放链表空间。

1.2 主要步骤

1.2.1 问题分析与算法思路

设*p[k-1]分别为链表中相邻k个节点的指针，求p[0]->data+p[1]->data+...+p[k-1]->data为最大的那一组值，返回其相应的指针p[0]即可。

1.2.2 算法描述

读取是否建立链表，
if 输入=1，开始读取输入值作为链表节点的data值
  if 输入=‘#’，结束读取
  else 将输入存入链表节点
else if 输入=2，开始读取txt地址作为输入，从txt文件读入数据
else 结束程序
读取k，并建立指针数组*p[k]
将*p[k]依次赋给除去头节点外的前k个节点。
若k<节点数，则报错，并重新录入链表数据
定义整型max来存储最大值，sum来存储p[0]->data+p[1]->data+...+p[k-1]->data的值
for *p[k-1]!=NULL 时
  sum=p[0]->data+p[1]->data+...+p[k-1]->data
  if sum>m, m=sum
  依次将指针后移，指向下一个节点
输出p[0]
读取输入，if 输入=1，建立新的链表
else 释放链表空间，结束程序。

1.2.3 程序实现

```
#include "stdlib.h"
#include "stdio.h"
#include "assert.h"

typedef int datatype;
typedef struct node
#define MAX 1024
{
    datatype data;
    struct node* next;
}linknode,*link;
```

```
int   dtm_stablish()//判断是否开始建立链表
{
   int d;
   do{
      printf("是否建立链表(1/0)?");
      scanf("%d",&d);
      if (!d) {
         printf("\n结束程序");
         return 0;
      }
      else if(d) return 1;
      else printf("\n错误\n");
   }while(d!=0&&d!=1);
   return 0;
}

link CreatList()
{
   int d,a;
   link H,p,r;
   do{
       printf("\n选择输入方式(1键入/2读取txt)：");
      scanf("%d",&d);
   }while(d!=1&&d!=2);
   H=(link)malloc(sizeof(linknode));
   r=H;
   if(d==1){
      scanf("%d",&a);
      while(a!=-1){
         p=(link)malloc(sizeof(linknode));
         p->data=a;
         r->next=p;
         r=p;
         scanf("%d",&a);
      }
   }
   else{
        FILE *f=fopen("/Users/touhomare-gen/Desktop/1.txt","r");
      if (f == NULL)
      {
         printf ("文件打开失败!\n");
          exit (-1); //标记出错位置,需要包含头文件stdlib.h
      }
      fscanf(f,"%d",&a);
      while(a!=-1){
         p=(link)malloc(sizeof(linknode));
         p->data=a;
         r->next=p;
         r=p;
         fscanf(f,"%d",&a);
      }
      fclose(f);
   }
   r->next=NULL;
   return H;
}

link Max(link H){
   int i,k,m,sum=0;
   link q;
   link p[MAX];
   printf("\n输入k的值：");
   do{
      scanf("%d",&k);
      q=H->next;
      for(i=0;i<k&&q->next;i++){
         p[i]=q;
         sum+=q->data;
         q=q->next;}
      p[i]=q;
      sum+=q->data;
```

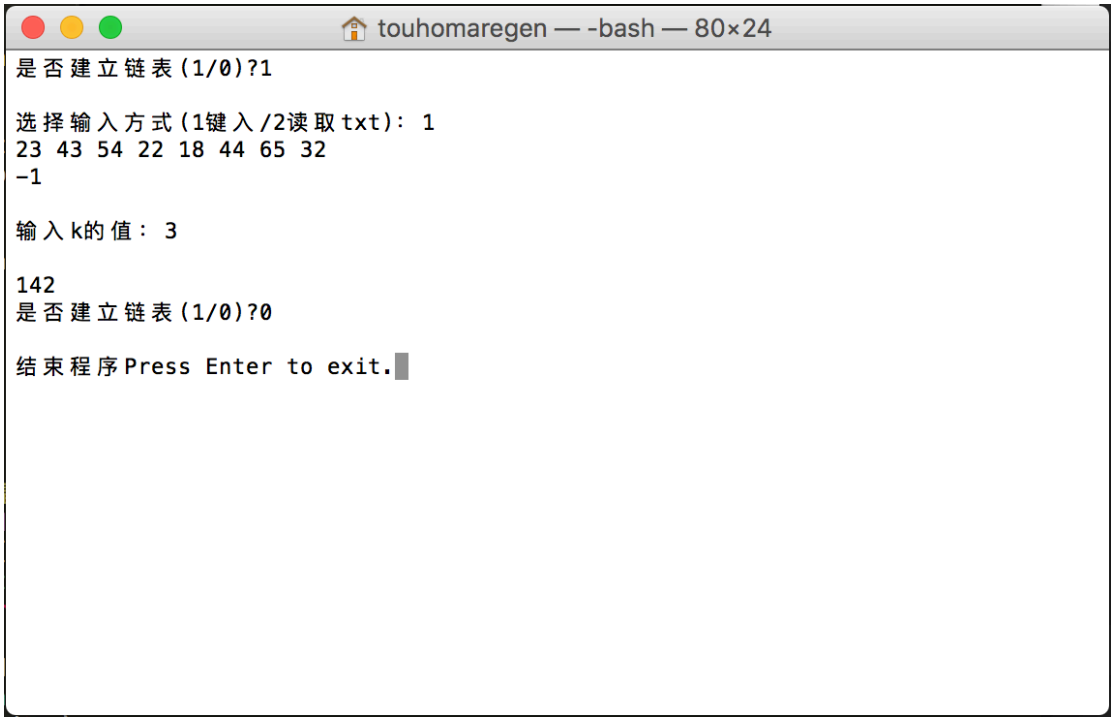
```
        if(i!=k||k>=MAX||k<=0) {printf("k错
误,请重新输入：");sum=0;}
    }while(i!=k);
    while(q->next){
        m=0;
        for(i=0;i<k-1;i++){
            m+=p[i]->data;
            p[i]=p[i+1];
        }
        p[k-1]=q->next;
        q=q->next;
        m+=p[k-1]->data;
        if(m>sum)sum=m;
    }
    printf("\n%d\n",sum);
    return p[0];
}
```

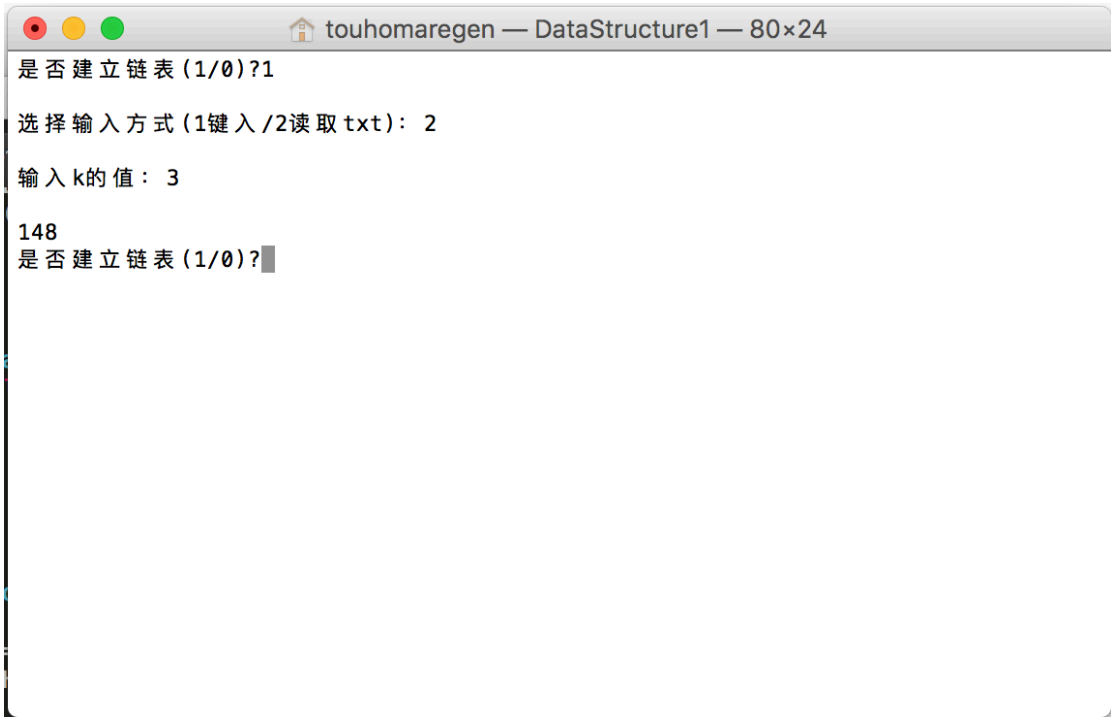
```
int main()
{
    link H,r;
    while(dtm_stablish()) {
        H=CreatList();
        r=Max(H);
    }
    link temp;
    while(H != NULL){
        temp = H;
        H = H->next;
        free(temp);
    }
    return 0;
}
```

本程序中均用函数实现了判断是否建立链表，建立链表和计算最大加和值。而主函数较为简单。简明易读，友好性高，且易于扩展，添加其他功能。

1.3 结果分析

1.3.1 测试





测试结果如图，可以从文本和输入两种方法读入数据，多次建立链表，选择k的值，输出sum的最大值。

1.3.2 算法和结果的有效性分析

时间复杂度为T(n)=O(n^2)。结果正确有效。

可改进的有：增加实现自主输入地址读取文件的功能。

2 实验2：栈的应用

2.1 实验内容

算术表达式求值：输入中缀形式的算术表达式，如：5+(4-2)*3，将其转换成后缀表达式并输出：542-3*+，然后对后缀表达式求值（本例结果为11）并将结果输出。

要求：

（1）“中缀表达式转换后缀表达式”和“后缀表达式求值”采用独立函数实现；

（2）操作数支持多位数和小数；

（3）运算符仅考虑+、-、*、/、（、）、#（#可用作结束符）；

（4）中缀表达式可从键盘输入也可以从文件输入。对输入的中缀表达式要进行合法性检查（表达式头尾以及运算符左右可以包含若干空格）；

（5）可根据用户需求，多次计算不同的表达式；

（6）自行定义栈类型以及需要的栈的基本操作。

2.2 主要步骤

2.2.1 问题分析与算法思路

表达式求值：从左到右扫描，当遇到操作符时，对其左边的两个操作数进行计算，以结果取代之，继续。

中缀表达式到后缀表达式的转换：中缀表达式与后缀表达式操作数的出现的次序是相同的，需要考虑的是操作符的优先级，优先级高的先转换，从左到右扫描表达式，将遇到的操作数输出，设置一个栈存放操作符，遇到的第一个操作符进栈，在扫描的过程中，将遇到的操作符与栈顶进行优先级比较，栈中的操作符满足条件，后进栈的优先级高于先进栈的优先级，当扫描完毕时，若栈非空，则将栈中操作符依次出栈输出。

2.2.2 算法描述

表达式求值：

设置一个栈S；

从左到右依次扫描表达式中的各分量x：

若x是操作数：Push(S,x);

若x是操作符：a=Pop(S);b=Pop(S);Push(b x a的计算结果);

继续，直到表达式扫描完毕；

最后，栈顶就是表达式的值。

中缀表达式到后缀表达式的转换：

从左到右扫描表达式，设置一个栈S存放操作符：

对于遇到的每一个分量x，分以下几种情况处理：

1.x=操作数：输出x；

2.x='('：x进栈；

3.x=其他操作符：

while(1){

如果栈空，跳出；

y=GetsTop，如果y=(，跳出；

如果y优先级小于x，跳出；

y出栈并输出；

}

x进栈；

4.x=')'：之前进站的‘（’和‘）’之间的操作符应该先计算：

反复出栈，输出出栈的操作符，直到遇“（”，退掉；

当扫描完毕时，若栈非空，则将栈中内容一次出栈输出；

2.2.3 程序实现

说明在程序实现中对用户的友好性、程序的模块化和扩展性等是如何考虑的:

本程序中用函数实现了关于栈的基本操作，包括：栈置空（包含了释放非空栈的过程）、判断栈是否为空、进栈、出栈和取栈顶元素等操作。

用函数实现了判断是否（重新）开始计算表达式过程以及输入表达式（可以手动输入和从文件输入）。用函数实现了判断输入表达式是否合理，输入必须以#结尾，可以包含空格，在判断是否合理的过程中，空格会被自动去除，防止影响后续程序处理。

以及用函数实现了将中缀表达式转换为后缀表达式和计算后缀表达式。在后缀表达式中，用空格将各个小数和多位数隔开，方便用户和程序阅读。

主函数较为简单。简明易读，友好性高，且易于扩展，添加其他功能。

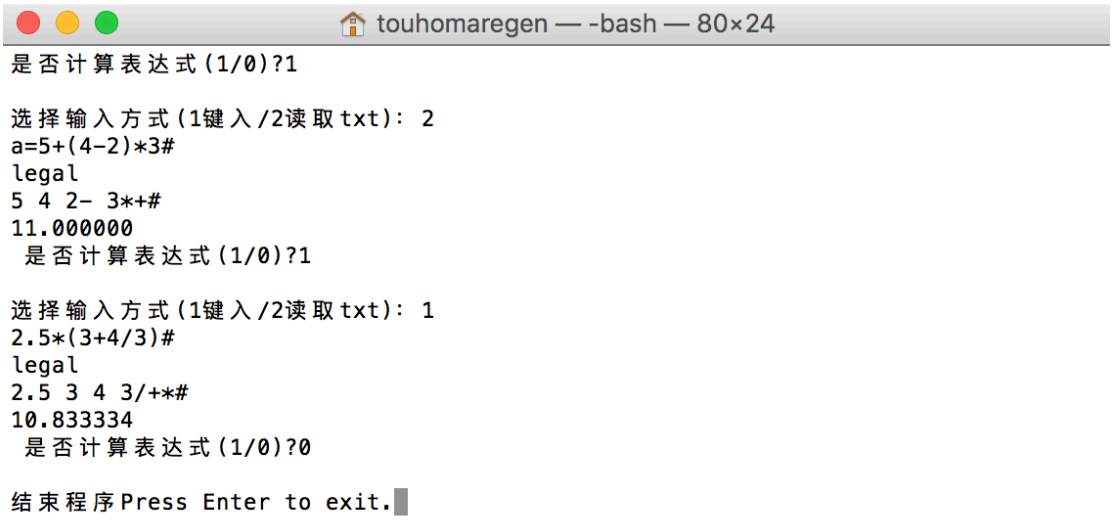
```
#include "stdlib.h"
#include "stdio.h"
#include "string.h"

#define MAXSIZE 1024
typedef struct node{
   char opt;
   float fval;
   struct node* next;
}snode,*slink;//栈定义

void Lclearstack(slink *ptop){
   slink p;
   while(*ptop!=NULL){
      p=*ptop;
      *ptop=(*ptop)->next;
      free(p);
   }
}
int Lemptystack(slink top){//判断栈是否为空
   if(top==NULL) return 1;
   else return 0;
}
void Lpush(slink *ptop,char opt,float fval){
   slink p=(slink)malloc(sizeof(snode));
   p->opt=opt;
   p->fval=fval;
   p->next=*ptop;
   *ptop=p;
}
void Lgetstop(slink top,char* opt,float* fval){//取栈顶元素
   if(Lemptystack(top)) {*opt='#';*fval=-1;}
   else {
      *opt=top->opt;
      *fval=top->fval;
   }
}
void Lpop(slink *ptop,char* opt,float* fval){//退栈
   slink p;
   if(Lemptystack(*ptop)){*opt='#';*fval=-1;}
   else{
      *opt=(*ptop)->opt;
      *fval=(*ptop)->fval;
      p=*ptop;
```

```
        *ptop=(*ptop)->next;
        free(p);}
}

void Compute(char a[],float* result){
    int i=0;
    int j;
    float x,y,k;
    char temp = '\0';
    char p[100];
    slink S=NULL;
    Lclearstack(&S);
    while(a[i]!='#'){
        if(a[i]=='+')
{Lpop(&S,&temp,&x);Lpop(&S,&temp,
&y);Lpush(&S,temp,x+y);i++;}
        else if(a[i]=='-')
{Lpop(&S,&temp,&x);Lpop(&S,&temp,
&y);Lpush(&S,temp,y-x);i++;}
        else if(a[i]=='*')
{Lpop(&S,&temp,&x);Lpop(&S,&temp,
&y);Lpush(&S,temp,y*x);i++;}
        else if(a[i]=='/')
{Lpop(&S,&temp,&x);Lpop(&S,&temp,
&y);Lpush(&S,temp,y/x);i++;}
        else if('0'<=a[i]&&a[i]<='9'){
            j=0;
            while(a[i]!='
'&&a[i]!='+'&&a[i]!='-'&&a[i]!='*'&&a[i]!='/
'&&a[i]!='#'){
                p[j]=a[i];j++;i++;
            }
            if(a[i]==' ')i++;
            p[j]='\0';
            k=atof(p);//将字符串转换为单
精度浮点型值
            //printf("k=%f\n", k);
            Lpush(&S,temp,k);
        }
        else i++;
    }
    Lgetstop(S,&temp,result);
}

void Convert(char a[],char b[]){
    int i,j;
    slink S = NULL;
    char y;
    float temp = 0.0;
    for(i=0,j=0;a[i]!='#';i++){
        if(('0'<a[i]&&a[i]<'9')||a[i]=='.')
{b[j]=a[i];j++;}
        else if(a[i]=='('
Lpush(&S,a[i],temp);
        else if(a[i]=='+'||a[i]=='-
'||a[i]=='*'||a[i]=='/') {
            b[j]=' ';
            j++;
            while(1){
                if(Lemptystack(S)) break;
                else{
                    Lgetstop(S,&y,&temp);
                    if(y=='(') break;
                    if((y=='+'||y=='-
')&&(a[i]=='*'||a[i]=='/')) break;
                    Lpop(&S,&b[j],&temp);
                    j++;
                }
            }
            Lpush(&S,a[i],temp);
        }
        else{
            Lpop(&S,&a[i],&temp);
            while(a[i]!='('){
                b[j]=a[i];
                j++;
                Lpop(&S,&a[i],&temp);
            }
        }
    }
    while(!Lemptystack(S)){
        Lpop(&S,&b[j],&temp);
        j++;
    }
    b[j]='#';
}

int IfLegal(char a[]){
    int i,j;
    if(a[0]=='#') {
        printf("计算表达式出错！");
        return 0;}
    for(i=0;a[i]!='#';i++){
        switch (a[i]) {
            case ' ':{
                for(j=i;a[j]!='#';j++){
                    a[j]=a[j+1];
                }
                i--;
            }// 去掉所有空格
            case '.':break;
            case '+':break;
            case '-':break;
            case '*':break;
            case '/':break;
            case '(':break;
            case ')':break;
            case '0':break;
            case '1':break;
            case '2':break;
            case '3':break;
            case '4':break;
            case '5':break;
            case '6':break;
```

```
            case '7':break;
            case '8':break;
            case '9':break;
            default:{
                printf("The expression is
ilegal, pleace retry.\n");
                return 0;
            }
        }
    }
    printf("legal\n");
    return 1;
}

int start()//判断是否开始
{
    int d;
    do{
        printf("是否计算表达式(1/0)?");
        scanf("%d",&d);
        if (!d) {
            printf("\n结束程序");
            return 0;
        }
        else if(d) return 1;
        else printf("\n错误\n");
    }while(d!=0&&d!=1);
    return 0;
}

void input(char a[])
{
    int d;
    do{
        printf("\n选择输入方式(1键入/2读
取txt)：");
        scanf("%d",&d);
    }while(d!=1&&d!=2);
    if(d==1){
        scanf("%s",a);
    }
    else{
        FILE *f=fopen("/Users/
touhomaregen/Desktop/1.txt","r");
        if (f == NULL)
        {
            printf ("文件打开失败!\n");
            exit (-1); //标记出错位置,需要
包含头文件stdlib.h
        }
        fscanf(f,"%s",a);
        printf("a=%s\n",a);
        fclose(f);
    }
}

int main(){
    char a[100]="0",b[100]="0";
    float c;
    while(start()) {
        input(a);
        //printf("%s\n", a);
        if(IfLegal(a)){
            Convert(a,b);
            printf("%s\n",b);
            Compute(b,&c);
            printf("%f\n ",c);
        }
    }
    return 0;
}
```

2.3 结果分析

2.3.1 测试



验证了手动输入和从文件输入等两个不同输入下的程序，并且包含了小数和多位数的情况。

2.3.2 算法和结果的有效性分析

时间复杂度为O(n)。

实现了题目要求的各个功能且运行正确。

可改进的有：加入更多运算法则，例如开根，乘方等等；增加实现自主输入地址读取文件的功能。

3 实验3：队列的应用

3.1 实验内容

将从键盘输入的一系列字符存储到链式队列中，当输入的字符为’0’时，执行出队操作并将出队元素打印到屏幕上；当输入的字符为’@’时，队列中剩余所有元素依次出队并打印到屏幕上；当输入其他字符时，字符入队。

要求：可根据用户需求多次重复该过程；自行补充所需的队列基本操作。

3.2 主要步骤

3.2.1 问题分析与算法思路

建立队列，将读入的字符进行判断，当输入的字符为’0’时，执行出队操作并将出队元素打印到屏幕上；当输入的字符为’@’时，队列中剩余所有元素依次出队并打印到屏幕上；当输入其他字符时，字符入队。

3.2.2 算法描述

```
//定义队列节点
void Lcreatqueue(linkqueue *q)//创建带头节点的队列
int Lemptyqueue(linkqueue *q)//判断队空
void Lenqueue(linkqueue *q, char e)//入队操作
int Ldequeue(linkqueue *q,char *pe)//退队操作
int start()//判断是否开始
int main(){
  while(开始?  ){
     开始读入字符；
     入队操作；
       while(字符!='@'){
          if(c=='0')出队；
           继续输入；
          入队；
       }
       所有元素出队；
   }
```

3.2.3 程序实现

```
#include "stdlib.h"                                    #include "stdio.h"
```

```
#include <string.h>

typedef struct node//定义队列节点
{
    char data;
    struct node *next;
}Qnode,*Qlink;
typedef struct{
    Qnode *front,*rear;
}linkqueue;

void  Lcreatqueue(linkqueue  *q)//创建带头节点的队列
{
                    q->front=q->rear=(Qlink)malloc(sizeof(Qnode));
    q->front->next=NULL;
}

int Lemptyqueue(linkqueue *q){
    return(q->front==q->rear);
}

void Lenqueue(linkqueue *q, char e){
                                Qlink p=(Qlink)malloc(sizeof(Qnode));
    p->data=e;
    q->rear->next=p;
    q->rear=p;
    p->next=NULL;
}

int Ldequeue(linkqueue *q,char *pe){
    Qlink p;
    if(Lemptyqueue(q)) return(-1);
    p=q->front;
    q->front=p->next;
    free(p);
    *pe=q->front->data;
    return 0;
}

int start()//判断是否开始
{
    int d;
    do{
        printf("是否开始入队(1/0)?");
        scanf("%d",&d);
        if (!d) {
            printf("\n结束程序");
            return 0;
        }
        else if(d) return 1;
        else printf("\n错误\n");
    }while(d!=0&&d!=1);
    return 0;
}


int main(){
    linkqueue Q;
    Lcreatqueue(&Q);
    char c;
    while(start()){
        printf("\n开始输入字符：\n");
        do{
            c=getchar();
        }while(c=='\n');
        Lenqueue(&Q,c);
        while(c!='@'){
            if(c=='0'){
                Ldequeue(&Q,&c);
                printf("出队：%c\n",c);
            }
            do{
```

```
            c=getchar();
        }while(c=='\n');
        Lenqueue(&Q,c);
    }
    printf("所有元素出队：");
        while(Ldequeue(&Q,&c)==0){
            printf("%c ",c);
        }
    }
    return 0;
}
```

给出完整的程序源代码和注释，说明在程序实现中对用户的友好性、程序的模块化和扩展性等是如何考虑的。为了减少篇幅，源代码采用较小的字体打印，也可适当采用分栏排版。

3.3 结果分析

3.3.1 测试

测试结果如上图所示。可以多次建队，可以多次执行单次退队输出，当输入@时执行整队输出。

3.3.2 算法和结果的有效性分析

时间复杂度为O(n)。若输入为零，只需一次操作，退队复杂度为O(1)，若输入为@退队复杂度为O(n)。只需建立一个队列进行储存，数据结构简单。

4 实验4：二叉树的应用

4.1 实验内容

树表的查找：输入一个英文句子，按照字典顺序构造一棵二叉排序树；对此二叉排序树进行中序遍历，并将遍历序列输出到屏幕上。

要求：

（1）英文句子可从键盘输入，也可从txt文件输入；

（2）遍历算法采用非递归遍历算法；

（3）程序结束时需释放树空间。

4.2 主要步骤

4.2.1 问题分析与算法思路

对树表的查找分两部分，首先按照字典顺序建立二叉排序树，之后对二叉排序树进行中序遍历输出。输出结果所有英文单词按字典顺序排列。

4.2.2 算法描述

用函数实现了建立二叉树和中序遍历。主函数较为简单。简明易读，友好性高，且易于扩展，添加其他功能。

用伪代码给出算法描述。

```
//定义树节点结构BSN,*BSP;
//定义栈节点结构
//定义栈基本操作（置空，判断为空，进栈，出栈，取栈顶元素）
void BSTinsert(BSP *T,BSP S)//定义插入树节点的函数
BSP GetLBtree(BSP *T)//按照字典顺序建立二叉排序树
void Inorder(BSP T)//中序遍历，利用栈
int main(){
   BSP T=NULL;
   while(程序开始) {
      建树;
      中序遍历;
   }
   清空树;
}
```

4.2.3 程序实现

```
#include "stdlib.h"                    #include "stdio.h"
```

```
#include "string.h"
typedef struct Bsnode
{
   char word[20];
   struct Bsnode *Lchild,*Rchild;
}BSN,*BSP;

typedef struct node
{
   BSP T;
   struct node*next;
}snode,*slink;

void Lclearstack(slink *ptop){
   slink p;
   while(*ptop!=NULL){
      p=*ptop;
      *ptop=(*ptop)->next;
      free(p);
   }
}

int Lemptystack(slink top){
   if(top==NULL) return 1;
   else return 0;
}

void Lpush(slink *ptop,BSP T){
                                   s l i n k
p=(slink)malloc(sizeof(snode));
      p->T=T;
      p->next=*ptop;
      *ptop=p;
}

void Lpop(slink *ptop,BSP *T){
   slink p;
   if(Lemptystack(*ptop)){T=NULL;}
   else{
      *T=(*ptop)->T;
      p=*ptop;
      *ptop=(*ptop)->next;
      free(p);
   }
}

void Lgetspop(slink top,BSP *Tree){
   if(Lemptystack(top)){*Tree=NULL;}

   else *Tree=top->T;
}

void  BSTinsert(BSP  *T,BSP  S){//T为
根，S为待插入节点的指针
   if(*T==NULL)*T=S;
   else{
      BSP p=*T,q=NULL;
      int i = 0;
      while(p){
         q=p;i=0;
                  while(S->word[i]==p-
>word[i]&&S->word[i]!='\0'){
            i++;
         }
                  if(S->word[i]=='\0'&&p-
>word[i]=='\0'){
                  printf("(repeated word,
not counting.) ");
            S=NULL;
            break;
         }
            else if(S->word[i]=='\0'||S-
>word[i]<p->word[i]) p=p->Lchild;
         else p=p->Rchild;
      }
      if(S!=NULL){
                  if(S->word[i]=='\0'||S-
```

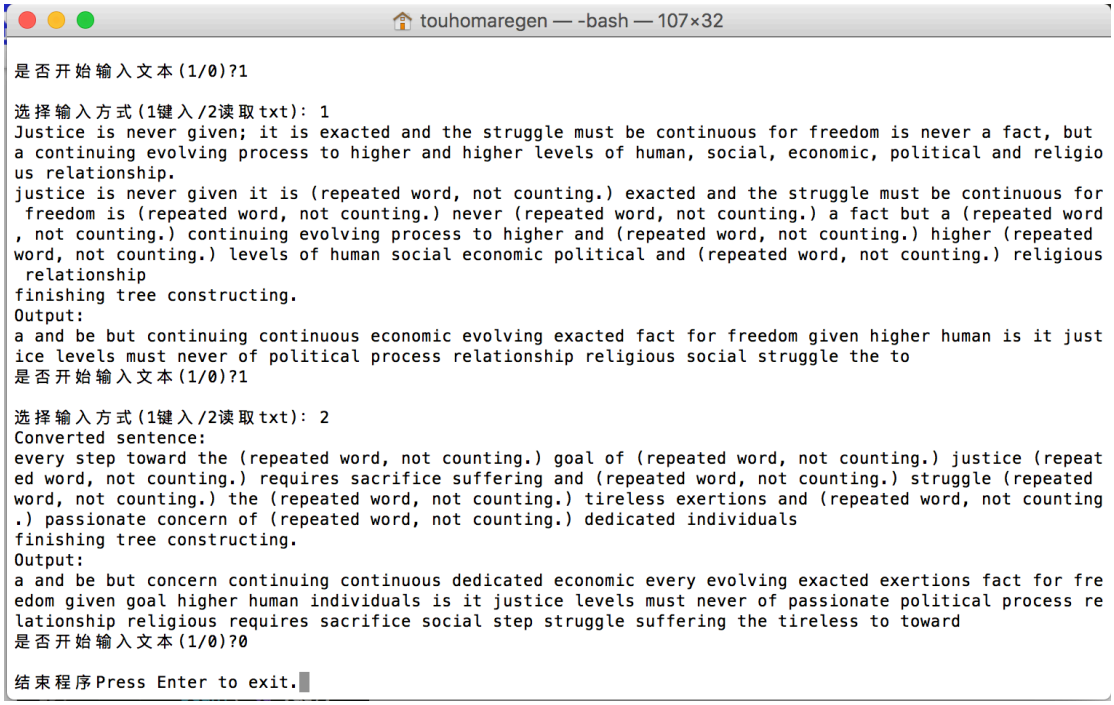
```
>word[i]<q->word[i]) q->Lchild=S;
                else q->Rchild=S;
            }
            else free(S);
        }
    }

    BSP GetLBtree(BSP *T){
        char ch[20];
        BSP p;
        int d,j,k;
        int stop=0;
        do{
            printf("\n选择输入方式(1键入/2
读取txt)： ");
            scanf("%d",&d);
        }while(d!=1&&d!=2);
        if(d==1){
            while(stop==0){
                scanf("%s",ch);

for(j=0;j<20&&ch[j]!='\0';j++){
                    if('A'<=ch[j]&&ch[j]<='Z')
                        ch[j]+=32;//单词大小写
转换

                    if(ch[j]=='.'){
                        ch[j]='\0';
                        stop=1;
                        break;
                    }
                    if(!('a'<=ch[j]&&ch[j]<='z'))

                        for(k=j;ch[k]!='\0';k++)
                            ch[k]=ch[k+1];
                }
                p=(BSP)malloc(sizeof(BSN));
                        strcpy(p->word,ch);p-
>Lchild=p->Rchild=NULL;
                printf("%s ",p->word);
                BSTinsert(T,p);
            }
        }
        else{
                    FILE  *f=fopen("/Users/
touhomaregen/Desktop/4.txt","r");
            if (f == NULL)
            {
                printf ("文件打开失败!\n");
                 exit (-1); //标记出错位置,需
要包含头文件stdlib.h
            }
                 printf("Converted sentence:
\n");
            while(stop==0){
                fscanf(f,"%s",ch);

for(j=0;j<20&&ch[j]!='\0';j++){
                    if('A'<=ch[j]&&ch[j]<='Z')
                        ch[j]+=32;//单词大小写
转换

                    if(ch[j]=='.'){
                        ch[j]='\0';
                        stop=1;
                        break;
                    }
                    if(!('a'<=ch[j]&&ch[j]<='z'))

                        for(k=j;ch[k]!='\0';k++)
                            ch[k]=ch[k+1];
                }
                p=(BSP)malloc(sizeof(BSN));
                        strcpy(p->word,ch);p-
>Lchild=p->Rchild=NULL;
                printf("%s ",p->word);
                BSTinsert(T,p);
```

```
        }
        fclose(f);
    }
      return *T;
}



void Inorder(BSP T){
    BSP p; slink s;
    Lclearstack(&s);
    Lpush(&s,T);
    while(!Lemptystack(s)){
        Lgetspop(s,&p);
        while(p){
            Lpush(&s,p->Lchild);
            Lgetspop(s,&p);
        }
        Lpop(&s,&p);
        if(!Lemptystack(s)){
            Lpop(&s,&p);
            printf("%s ",p->word);
            Lpush(&s,p->Rchild);
        }
    }
}

void ClearTree(BSP *T){
    BSP p,q; slink s=NULL;
    Lclearstack(&s);
    Lpush(&s,*T);
    while(!Lemptystack(s)){
        Lgetspop(s,&p);
        while(p){
            Lpush(&s,p->Lchild);
            Lgetspop(s,&p);
        }
        Lpop(&s,&p);
        if(!Lemptystack(s)){
            Lpop(&s,&p);
            q=p;
            p=p->Rchild;
            free(q);
            Lpush(&s,p);
        }
    }
}

int start()//判断是否开始
{
    int d;
    do{
        printf("是否开始输入文本(1/
0)?");
        scanf("%d",&d);
        if (!d) {
            printf("\n结束程序");
            return 0;
        }
        else if(d) return 1;
        else printf("\n错误\n");
    }while(d!=0&&d!=1);
    return 0;
}

int main(){
    BSP T=NULL;
    while(start()) {
        GetLBtree(&T);
            printf("\nfinishing tree
constructing.\nOutput:\n");
        Inorder(T);
        printf("\n");
    }
    ClearTree(&T);
    return 0;
}
```

//Justice is never given; it is exacted and the struggle must be continuous for freedom is never a fact, but a continuing evolving process to higher and higher levels of human, social, economic, political and religious relationship.

4.3 结果分析

4.3.1 测试



将输入文本全都变成小写字母，去除所有标点符号。建树时忽略所有重复单词。

输出为将所有输入单词进行字母排序后的顺序。

可以多次读入，支持磁盘读入。

4.3.2 算法和结果的有效性分析

读入并处理单词复杂度是O(e^2)，e是单词长度。存储数据结构为二叉排序树，插入一个元素进入二叉排序树的时间复杂度O(log2n)<ASL<(n^2)，遍历输出单词复杂度是O(n)。

5 实验5：图的应用

5.1 实验内容

建立有向图的十字链表存储结构，利用拓扑排序方法判断该图是否为有向无环图。

要求：有向图的顶点信息和弧信息可以从键盘或txt文件输入，输入格式自拟。

5.2 主要步骤

5.2.1 问题分析与算法思路

读入顶点信息，存入顶点表；读入弧信息，即弧头和弧尾所指的顶点，建立十字链表。建立对应的入度表，对其进行拓扑排序。

5.2.2 算法描述

```
//定义弧节点结构
//定义顶点表结构
//定义栈结构和基本栈运算
//定义定位节点的函数
int createorlist(vexnode G[])//建立有向图十字链表
void Creatid(vexnode G[],int n,int id[])//建立顶点入度表id，顶点数为n
void Topsort(vexnode G[],int n)//对网G拓扑排序
int main(){
   while(程序开始) {
     读入顶点和弧的信息，建立十字链表；
     对网进行拓扑排序；
   }
}
```

5.2.3 程序实现

```
#include "stdlib.h"
#include "stdio.h"
#define MAXN 1024
typedef char vtype;
typedef struct Anode//弧节点
{
   int tail,head;
   struct Anode *hlink,*tlink;
}arcnode;
typedef struct Vnode//顶点
{
   vtype data;
   arcnode *fin,*fout;
}vexnode;
vexnode G[MAXN];

typedef struct node
{
   int i;
   struct node*next;
}snode,*slink;
```

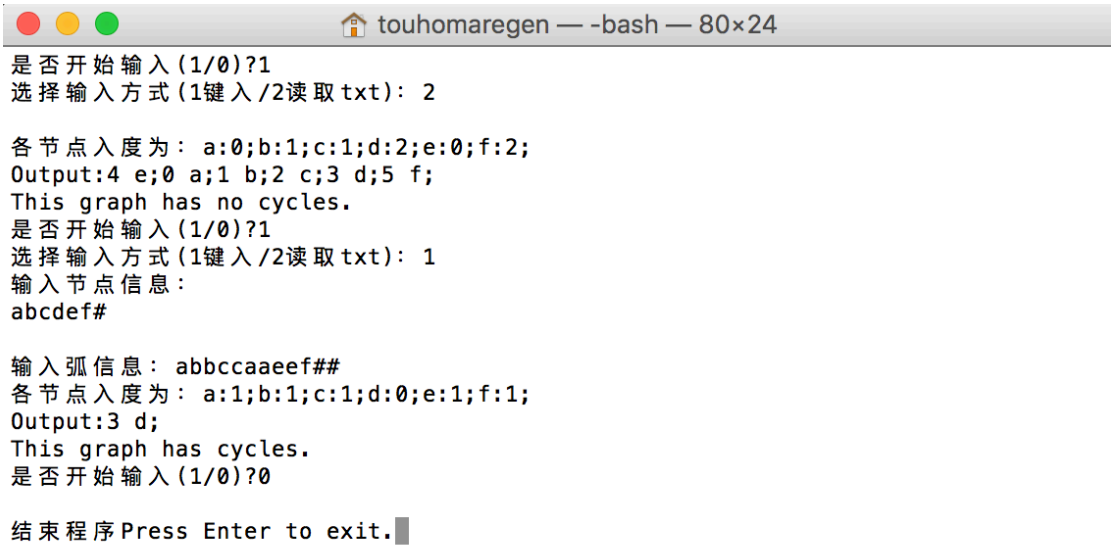
```
void Lclearstack(slink *ptop){
    slink p;
    while(*ptop!=NULL){
        p=*ptop;
        *ptop=(*ptop)->next;
        free(p);
    }
}
int Lemptystack(slink top){
    if(top==NULL) return 1;
    else return 0;
}

void Lpush(slink *ptop,int i){
    slink p=(slink)malloc(sizeof(snode));
    p->i=i;
    p->next=*ptop;
    *ptop=p;
}

void Lpop(slink *ptop,int *i){
    slink p;
    if(Lemptystack(*ptop)){*i=-1;}
    else{
        *i=(*ptop)->i;
        p=*ptop;
        *ptop=(*ptop)->next;
        free(p);
    }
}

void Lgetspop(slink top,int *i){
    if(Lemptystack(top)){*i=-1;}
    else *i=top->i;
}


int locatevex(vexnode G[],int n,char c){
    int i;
    for(i=0;i<n;i++){
        if(G[i].data==c) return i;
    }
    return -1;
}

//建立有向图十字链表
int createorlist(vexnode G[])
{
    int d,i=0,n=0,j;
    arcnode *p;
    vtype ch,u='\0',v = '\0';
    do{
            printf("选择输入方式(1键入/2读取
txt)：");
        scanf("%d",&d);
    }while(d!=1&&d!=2);
    getchar();
    if(d==1){
        printf("输入节点信息：\n");
        scanf("%c",&ch);
        while(ch!='#'){
            n++;
            G[i].data=ch;
            G[i].fin=G[i].fout=NULL;
            i++;
            scanf("%c",&ch);
        }
        printf("\n输入弧信息：");
        getchar();
        scanf("%c%c",&u,&v);
        while(u!='#'&&v!='#'){
                        if(locatevex(G,n,u)!=-1)
i=locatevex(G,n,u);
            else{
                printf("error");
```

```
            return -1;
        }
                    if(locatevex(G,n,v)!=-1)
j=locatevex(G,n,v);
        else{
            printf("error");
            return -1;
        }

p=(arcnode*)malloc(sizeof(arcnode));
        p->tail=i;p->head=j;
        p->tlink=G[i].fout;
        G[i].fout=p;
        p->hlink=G[j].fin;
        G[j].fin=p;
        scanf("%c%c",&u,&v);
      }
   }
   else{
                    FILE  *f=fopen("/Users/
touhomaregen/Desktop/5.txt","r");
      if (f == NULL)
      {
         printf ("文件打开失败!\n");
          exit (-1); //标记出错位置,需要包
含头文件stdlib.h
      }
      printf("输入节点信息：\n");
      fscanf(f,"%c",&ch);
      while(u!='#'&&v!='#'){
         n++;
         G[i].data=ch;
         G[i].fin=G[i].fout=NULL;
         i++;
         fscanf(f,"%c",&ch);
      }
      printf("\n输入弧信息：");
      getchar();
      fscanf(f,"%c%c",&u,&v);
      while(u!='#'){
                    if(locatevex(G,n,u)!=-1)
i=locatevex(G,n,u);
        else{
            printf("error");
            return -1;
        }
                    if(locatevex(G,n,v)!=-1)
j=locatevex(G,n,v);
        else{
            printf("error");
            return -1;
        }

p=(arcnode*)malloc(sizeof(arcnode));
        p->tail=i;p->head=j;
        p->tlink=G[i].fout;
        G[i].fout=p;
        p->hlink=G[j].fin;
        G[j].fin=p;
        fscanf(f,"%c%c",&u,&v);
      }
      fclose(f);
   }
   return n;
}

//建立顶点入度表id，顶点数为n
void Creatid(vexnode G[],int n,int id[]){
   int count,i; arcnode *p;
   printf("各节点入度为：");
   for(i=0;i<n;i++){
      id[i]=0;count=0;
      p=G[i].fin;
      while(p){count++;p=p->hlink;}
                    id[i]=count;  printf("%c:
```

```
%d;",G[i].data,id[i]);
    }
}
//对网G拓扑排序
void Topsort(vexnode G[],int n){
    int i,j=0,k,count,id[n];
    arcnode *p;slink s=NULL;
    Creatid(G,n,id);
    Lclearstack(&s);
    for(i=0;i<n;i++)
        if(id[i]==0) Lpush(&s,i);
    count=0;
    printf("\nOutput:");
    while(!Lemptystack(s)){
        Lpop(&s,&j);
        printf("%d %c;",j,G[j].data);
        count++;p=G[j].fout;
        while(p){
            k=p->head;
            id[k]--;
            if(id[k]==0)Lpush(&s,k);
            p=p->tlink;
        }
    }
      if(count==n) printf("\nThis graph has no cycles.\n");
       else printf("\nThis graph has cycles. \n");
}

int start()//判断是否开始
{
    int d;
    do{
        printf("是否开始输入(1/0)?");
        scanf("%d",&d);
        if (!d) {
            printf("\n结束程序");
            return 0;
        }
        else if(d) return 1;
        else printf("\n错误\n");
    }while(d!=0&&d!=1);
    return 0;
}

int main(){
    while(start()) {
        int n;
        vexnode G[MAXN];
        n=createorlist(G);
        Topsort(G,n);
        return 0;
    }
}
```

5.3 结果分析

5.3.1 测试



可以多次输入建立网结构；可以判断出有环和无环的情况。为了使程序过程更有可读性，添在输入弧信息之后，自动输出每个节点的入度和每次输出入度为0的节点的信息。最后给出是否有环的信息。

5.3.2 算法和结果的有效性分析

拓扑排序用到了十字链表和栈两种数据结构，时间复杂度为O(n)。

顶点类型为字符型数据，能够处理常见的输入异常。

6 实验6：综合应用

6.1 实验内容

统计若干个大型英文txt文件（如英文小说）中所有单词出现的次数，并输出出现次数最多的前10个单词及其出现次数。假设单词字符定义为大小写字母、数字和下划线，其他字符均看作单词分隔符。

要求：

（1）自行设计合适的数据结构及相关算法；

（2）程序运行结束时将txt文件名以及统计结果写入磁盘；

（3）每次程序启动时（除了首次运行）将上次的结果读入内存、显示；

（4）能根据用户选择实现重新初始化、查找某单词出现次数、追加统计、退出等功能。

6.2 主要步骤

6.2.1 问题分析与算法思路

采用二叉排序树的数据结构，按字典顺序进行单词排序。多设置一个整型的num来记录出现的次数。第一次读入单词时，建立节点，num为1，之后每次读入num都加1。存入二叉树时以字符串形式存取，但接收字符以字符形式接收，将所有大写字母转化为小写字母，将所有符号字符（除下划线和数字）都替换为\0，作为分隔符。建立一个长度为10的顺序表，遍历二叉树，将出现次数前十的单词和出现次数录入顺序表中。复杂度为O(n)。

6.2.2 算法描述

```
//定义树节点结构，顺序表结构和栈结构
typedef struct Bsnode{
    char word[50];
    int num;
    struct Bsnode *Lchild,*Rchild;
}BSN,*BSP;
typedef struct node{
    BSP T;
    struct node*next;
}snode,*slink;
struct topwords{
    char word[50];
    int num;
};
```

void BSTinsert(BSP *T,BSP S){//T为根，S为待插入节点的指针

if(S->word[i]=='\0'&&p->word[i]=='\0'){

p->num++;//重复单词，出现次数加一

否则根据单词字母顺序插入二叉树；

BSP GetLBtree(BSP *T){//从多个文件中读取单词，并构建二叉排序树

读入文件，将句子转化，去除标点，并将大写全部转化成小写字母

假设单词字符定义为大小写字母、数字和下划线，其他字符均看作单词分隔符。

void visit(BSP T, struct topwords word[])中序遍历二叉树，取前十的num计入顺序表中

void Inorder(BSP T, struct topwords word[])

中序遍历二叉树，取前十的num计入顺序表中

int main(){

循环：{

选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)\n");

当选择初始化：

置空表；

选择输入文件建立树GetLBtree(&T);

中序遍历二叉树生成出现次数前十的单词表并输出，将结果写入文件。

当选择查找某单词出现次数时：

读入单词，并遍历二叉树查找；

当选择追加统计时：

读入文件，在原有二叉树基础上修改树。

当选择退出时，退出循环。

}

6.2.3 程序实现

```
#include "stdlib.h"
#include "stdio.h"
#include "string.h"
typedef struct Bsnode
{
   char word[50];
   int num;
   struct Bsnode *Lchild,*Rchild;
}BSN,*BSP;
```

```
typedef struct node
{
   BSP T;
   struct node*next;
}snode,*slink;

struct topwords{
   char word[50];
```

```
    int num;
};

void Lclearstack(slink *ptop){
    slink p;
    while(*ptop!=NULL){
        p=*ptop;
        *ptop=(*ptop)->next;
        free(p);
    }
}

int Lemptystack(slink top){
    if(top==NULL) return 1;
    else return 0;
}

void Lpush(slink *ptop,BSP T){
                                    slink
p=(slink)malloc(sizeof(snode));
    p->T=T;
    p->next=*ptop;
    *ptop=p;
}

void Lpop(slink *ptop,BSP *T){
    slink p;
    if(Lemptystack(*ptop)){T=NULL;}
    else{
        *T=(*ptop)->T;
        p=*ptop;
        *ptop=(*ptop)->next;
        free(p);
    }
}

void Lgetspop(slink top,BSP *Tree){
    if(Lemptystack(top)){*Tree=NULL;}
    else *Tree=top->T;
}

void BSTinsert(BSP *T,BSP S){//T为
根，S为待插入节点的指针
    if(*T==NULL)*T=S;
    else{
        BSP p=*T,q=NULL;
        int i = 0;
        while(p){
            q=p;i=0;
                    while(S->word[i]==p-
>word[i]&&S->word[i]!='\0'){
                i++;
            }
                    if(S->word[i]=='\0'&&p-
>word[i]=='\0'){
                p->num++;//重复单词，出
现次数加一
                    //printf("%c:%d; ",p-
>word[i],p->num);
                S=NULL;
                break;
            }
                else if(S->word[i]=='\0'||S-
>word[i]<p->word[i]) p=p->Lchild;
            else p=p->Rchild;
        }
        if(S!=NULL){
                    if(S->word[i]=='\0'||S-
>word[i]<q->word[i]) q->Lchild=S;
            else q->Rchild=S;
        }
        else free(S);
    }
}
```

```
BSP GetLBtree(BSP *T){//从多个文件中读取单词，并构建二叉排序树
    char ch[50],chr[50]="\0";
    BSP p;
    int j;
    char str[1024];
    printf("输入打开文件的地址：");
    scanf("%s",str);
    while(str[0]!='#'){
      FILE *f=fopen(str,"r");
      if (f == NULL)
      {
        printf ("文件打开失败!\n");
          printf("\n继续输入打开文件的地址：");
        getchar();
        scanf("%s",str);
        continue;
      }
       //printf("Converted sentence:\n");//将句子转化，去除标点，并将大写全部转化成小写字母
        //假设单词字符定义为大小写字母、数字和下划线，其他字符均看作单词分隔符。
            if(fscanf(f,"%s",ch)==EOF){break;}
        while(1){

for(j=0;j<20&&ch[j]!='\0';j++){
            if('A'<=ch[j]&&ch[j]<='Z')
              ch[j]+=32;//单词大小写转换

if(!(('a'<=ch[j]&&ch[j]<='z')||'a'=='_'||('0'<=ch[j]&&ch[j]<='9'))){
              ch[j]='\0';j++;
              //for(k=0;j<50;j++)
              //    chr[k]=ch[j];
              //break;
            }
          }
          if(ch[0]!='\0'){

p=(BSP)malloc(sizeof(BSN));
            strcpy(p->word,ch);
           p->Lchild=p->Rchild=NULL;

            p->num=1;//设置初始出现次数为1
            //printf("%s ",p->word);
            BSTinsert(T,p);
          }
          if(chr[0]=='\0'){

if(fscanf(f,"%s",ch)==EOF)break;
          }
          else strcpy(ch,chr);
        }
        fclose(f);
         printf("\n继续输入打开文件的地址：");
        getchar();
        scanf("%s",str);
      }
      return *T;
    }

    void visit(BSP T, struct topwords word[]){
      int i,j;
      for(i=0;i<10;i++){
        if(T->num>word[i].num){
```

```
            for(j=9;j>i;j--){
                    word[j].num=word[j-
1].num;
strcpy(word[j].word,word[j-1].word);
            }
            word[i].num=T->num;
                strcpy(word[i].word,T-
>word);
            break;
          }
        }
    }

    void Inorder(BSP T, struct topwords
word[]){
        BSP p; slink s=NULL;
        Lclearstack(&s);
        Lpush(&s,T);
        while(!Lemptystack(s)){
          Lgetspop(s,&p);
          while(p){
            Lpush(&s,p->Lchild);
            Lgetspop(s,&p);
          }
          Lpop(&s,&p);
          if(!Lemptystack(s)){
            Lpop(&s,&p);
            visit(p,word);
            Lpush(&s,p->Rchild);
          }
        }
    }

    void ClearTree(BSP *T){
        BSP p,q; slink s=NULL;
        Lclearstack(&s);
        Lpush(&s,*T);
        while(!Lemptystack(s)){
          Lgetspop(s,&p);
          while(p){
            Lpush(&s,p->Lchild);
            Lgetspop(s,&p);
          }
          Lpop(&s,&p);
          if(!Lemptystack(s)){
            Lpop(&s,&p);
            q=p;
            p=p->Rchild;
            free(q);
            Lpush(&s,p);
          }
        }
    }

    void load(struct topwords word[]){
        int i;
            FILE *f=fopen("/Users/
touhomaregen/Desktop/6.txt","r");
        if (f == NULL) {
            printf ("首次运行，文件不存
在!\n");
          for(i=0;i<10;i++){
            word[i].num=0;
            word[i].word[0]='\0';
          }//初始化最大出现次数的单词
序列
        }
        else {
          for(i=0;i<10;i++){
                        fscanf(f,"%s
%d\n",word[i].word,&word[i].num);
          }
          fclose(f);
          printf("读取上次统计结果：\n");
          for(i=0;i<10;i++){
```

```
printf("%s:%d\n",word[i].word,word[i].num);
            }
        }
    }

    int sel(){
        int i;
        printf("选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)\n");
        scanf("%d",&i);
        return i;
    }//判断是否开始//

    int lookup(BSP T,char str[]){
        BSP p; slink s=NULL;
        Lclearstack(&s);
        Lpush(&s,T);
        while(!Lemptystack(s)){
            Lgetspop(s,&p);
            while(p){
                Lpush(&s,p->Lchild);
                Lgetspop(s,&p);
            }
            Lpop(&s,&p);
            if(!Lemptystack(s)){
                Lpop(&s,&p);
                    if(!strcmp(p->word,str)) return p->num;
                Lpush(&s,p->Rchild);
            }
        }
        return 0;
    }

    int main(){
        struct topwords word[10];
        int i,j;
        load(word);
        BSP T=NULL;
        char str[50]="\0";
        do{
            i=sel();
            switch (i) {
                case 1:
                    printf("重新初始化：\n");
                    for(j=0;j<10;j++){
                        word[j].num=0;
                        word[j].word[0]='\0';
                    }
                    GetLBtree(&T);
                        printf("\nfinishing tree constructing.\n");
                    Inorder(T,word);
                    printf("\nOutput:\n");
                    for(j=0;j<10;j++){

printf("%s:%d\n",word[j].word,word[j].num);
                    }
                        FILE *f=fopen("/Users/touhomaregen/Desktop/6.txt","wt+");
                    for(j=0;j<10;j++){
                                fprintf(f,"%s %d\n",word[j].word,word[j].num);
                    }
                    fclose(f);
                    break;
                case 2:
                    printf("查找某单词出现次数:\n");
                    scanf("%s",str);
                        printf("出现的次数：%d\n",lookup(T,str));
```

```
            break;
        case 3:
            printf("追加统计:\n");
            BSP p; slink s=NULL;
            Lclearstack(&s);
            Lpush(&s,T);
            while(!Lemptystack(s)){
                Lgetspop(s,&p);
                while(p){
                    Lpush(&s,p->Lchild);
                    Lgetspop(s,&p);
                }
                Lpop(&s,&p);
                if(!Lemptystack(s)){
                    Lpop(&s,&p);
                    for(j=0;j<10;j++){
                            if(strcmp(T->word,word[j].word)){
                                T->num=word[j].num;
strcpy(word[j].word,T->word);
                                break;
                            }
                    }
                    Lpush(&s,p->Rchild);
                }
            }
            for(j=0;j<10;j++){
                word[j].num=0;
                word[j].word[0]='\0';
            }
```

6.3 结果分析

6.3.1 测试

```
            GetLBtree(&T);
                printf("\nfinishing tree constructing.\n");
            Inorder(T,word);
            printf("\nOutput:\n");
            for(j=0;j<10;j++){
printf("%s:%d\n",word[j].word,word[j].num);
            }
                FILE *fx=fopen("/Users/touhomaregen/Desktop/6.txt","at");
            for(j=0;j<10;j++){
                        fprintf(fx,"%s %d\n",word[j].word,word[j].num);
            }
            fclose(fx);
            break;
        default:
            break;
    }
  }while(i==1||i==2||i==3);
  ClearTree(&T);
  return 0;
}
//"/Users/touhomaregen/Desktop/6.1.txt"
//"/Users/touhomaregen/Desktop/6.2.txt"
//"/Users/touhomaregen/Desktop/6.3.txt"
```

```
首次运行，文件不存在！
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
1
重新初始化：
输入打开文件的地址：/Users/touhomaregen/Desktop/6.1.txt

继续输入打开文件的地址：/Users/touhomaregen/Desktop/6.2.txt

继续输入打开文件的地址：#

finishing tree constructing.

Output:
the:43
is:20
and:15
in:15
question:15
answer:14
a:13
to:12
qa:11
this:11
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
4
Program ended with exit code: 0
```

首次运行，文件不存在，重新初始化，可以多次从文件输入并进行统计，输出出现次数最多的前十个单词。结束时将结果写入磁盘，并清空树。

```
读取上次统计结果：
the:43
is:20
and:15
in:15
question:15
answer:14
a:13
to:12
qa:11
this:11
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
1
重新初始化：
输入打开文件的地址：/Users/touhomaregen/Desktop/6.2.txt

继续输入打开文件的地址：/Users/touhomaregen/Desktop/6.3.txt

继续输入打开文件的地址：#

finishing tree constructing.

Output:
the:48
is:20
and:17
in:15
answer:13
of:12
question:12
to:12
this:11
a:8
```

```
a:8
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
2
查找某单词出现次数：
value
value出现的次数：0
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
2
查找某单词出现次数：
released
released出现的次数：4
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
3
追加统计：
输入打开文件的地址：/Users/touhomaregen/Desktop/6.1.txt
```

```
继续输入打开文件的地址：#

finishing tree constructing.

Output:
the:48
is:28
the:25
and:21
in:21
question:21
the:20
answer:20
of:18
the:17
选择进行的操作：重新初始化(1)、查找某单词出现次数(2)、追加统计(3)、退出(4)
4
Program ended with exit code: 0
```

再次运行程序，可以成功读取上次运行写入磁盘的文件并输出。之后重新建立二叉树，并进行读取文件统计，可以成功查找单词出现的次数，可以成功追加统计。

6.3.2 算法和结果的有效性分析

运用了二叉排序树的存储结构。插入一个元素进入二叉排序树的时间复杂度O(log2n)<ASL<(n^2)。

输出长度前十的单词时，建立一个长度为10的链表，采用直接插入排序。复杂度为O(n)。

改进：可以将二叉排序树改进为平衡二叉树，这样能使查找时间复杂度为O(log2n)。

四、结论与讨论

对照实验内容，简要总结完成的主要工作；对照实验目的，着重说明通过实验所取得的收获和能力的达成情况；存在的问题及可能的改进方向；其他需要说明的问题。

另：五教师评审请保持单独一页。

本学期主要工作内容：

1.设计实现输入数据（设为整型）建立单链表，并求相邻k个节点data值之和为最大的第一节点。

2.设计实现算术表达式求值：输入中缀形式的算术表达式，将其转换成后缀表达式并输出，然后对后缀表达式求值并将结果输出。

3.设计实现队列基本操作：将从键盘输入的一系列字符存储到链式队列中，当输入的字符为'0'时，执行出队操作并将出队元素打印到屏幕上；当输入的字符为'@'时，队列中剩余所有元素依次出队并打印到屏幕上；当输入其他字符时，字符入队。

4.设计实现树表的查找：输入一个英文句子，按照字典顺序构造一棵二叉排序树；对此二叉排序树进行中序遍历，并将遍历序列输出到屏幕上。

5.设计实现有向图的十字链表存储结构，利用拓扑排序方法判断该图是否为有向无环图。

6.实现统计若干个大型英文txt文件（如英文小说）中所有单词出现的次数，并输出出现次数最多的前10个单词及其出现次数。假设单词字符定义为大小写字母、数字和下划线，其他字符均看作单词分隔符。

收获：在本次实验中我加深了对常用数据结构和算法设计基本思路、思考方法及其适用场合的理解，并能运用于解决实际问题能根据特定问题需求，分析建立计算模型（包括逻辑结构和物理结构）、设计算法和程序，并在设计中综合考虑多种因素，对结果的有效性进行分析；训练了分析问题、解决问题的能力以及自主学习与程序设计实践能力；形成了将非数值型问题抽象为计算模型到算法设计、程序实现、结果有效性分析的能力。

五、教师评审

教师评语	实验成绩（25分）
签名: 日期:	