

法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。



关注 小象学院

第九课 文件与C语言工程基础

林沐

内容概述

- 1.文件的引入
- 2.文件的概念
- 3.ASC2码文件的读取
- 4.ASC2码文件的写入
- 5.例1-和谐词汇
- 6.二进制文件的写入
- 7.二进制文件的读取
- 8.文件定位操作
- 9.二进制文件的更新
- 10.例2-快捷方式
- 11.标准的输入输出流
- 12.freopen函数
- 13.随机数生成测试数据
- 14.C语言工程引入
- 15.Linux下的gcc编译工具
- 16.Makefile简介
- 17.外部变量与静态函数
- 18.编译与运行
- 19.编译时可能遇到的问题

文件的引入

计算机学院有**2000多名**学生，输入学生个数n，每个学生的姓名、学号(1-n)(无重复学号)、3门成绩，计算学生的**总成绩**，按照成绩的大小**从大到小**排序(若总成绩相同，学号靠前的排序靠前)，输出并**保存到计算机中**，供后续发奖学金等相关问题使用。

想象一下，如果2000多名学生数据要**键盘输入**，是**多么可怕**的一件事情！

文件的引入_data.in.txt - 记事本

文件(F)

编辑(E)

格式(O)

查看(V)

帮助(H)

5

Bill 4 91 70 66

Tom 1 69 81 77

XiaoFang 5 70 60 95

XiaoMing 3 81 87 85

Mike 2 80 97 71

文件的引入_data.out.txt - 记事本

文件(F)

编辑(E)

格式(O)

查看(V)

帮助(H)

id

name

sum

score1

score2

score3

3

XiaoMing

253

81

87

85

2

Mike

248

80

97

71

1

Tom

227

69

81

77

4

Bill

227

91

70

66

5

XiaoFang

225

70

60

95

```
#include <stdio.h>
#include <stdlib.h>
```

```
struct student{
    char name[20];
    int id;
    int score[3];
    int sum;
};
```

//数据结构体定义

```
typedef struct student student;
```

```
int student_cmp(const void *a, const void *b){
    student *s1 = (student *)a;
    student *s2 = (student *)b;
    if (s1->sum != s2->sum){
        return s2->sum - s1->sum;
    }
    return s1->id - s2->id;
}
```

//比较结构体的回调函数

```
int main() {
```

```
FILE *fp_in = fopen("文件的引入_data.in.txt", "r");
FILE *fp_out = fopen("文件的引入_data.out.txt", "w");
```

```
student s[3000]; //文件的打开
```

```
int n;
```

```
int i, j; //文件的读取
```

```
fscanf(fp_in, "%d", &n);
for (i = 0; i < n; i++){
    fscanf(fp_in, "%s %d %d %d %d", s[i].name, &s[i].id,
        &s[i].score[0], &s[i].score[1], &s[i].score[2]);
    s[i].sum = s[i].score[0] + s[i].score[1] + s[i].score[2];
}
```

```
qsort(s, n, sizeof(s[0]), student_cmp); //文件的输出
```

```
fprintf(fp_out, "%s\t%-15s%s\t%-15s\t%-15s\t%-15s\n", "id", "name", "sum",
    "score1", "score2", "score3");
for (i = 0; i < n; i++){
    fprintf(fp_out, "%d\t%-15s%d\t%-15s\t%-15s\t%-15s\n",
        s[i].id, s[i].name, s[i].sum,
        s[i].score[0], s[i].score[1], s[i].score[2]);
}
```

```
fclose(fp_in); //文件的关闭
```

```
fclose(fp_out);
```

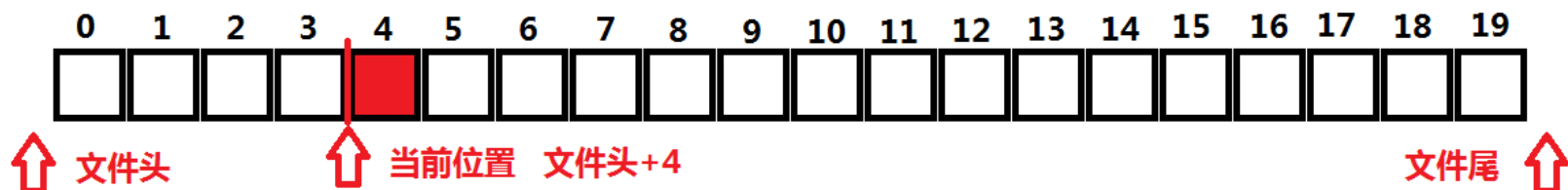
```
return 0;
```

```
}
```

文件的概念

在程序运行**结束**后，我们希望将一些重要的数据存储到一个**即使关掉**计算机，数据也**不会消失**的存储设备中。这个设备就叫做**文件**，文件通常存储到**硬盘**上。文件实际上是一**系列的字节**，文件包括**开头**、**结尾**、**当前位置**，它通常定义为从文件头到当前位置**有多少字节**。当前位置可以移动到文件的任何地方，即为发生文件操作(**读写文件**)的地方。

文件实际上是一系列的字节:



文件包括两种文件，**文本文件(ASC2码文件)**与**二进制文件**。**文本文件**存储的是字符数据类型的文件，通俗的说就是我们**看得懂**的文件。二进制文件存储的是如int、double类型的数据，通常是一些结构体或者数组数据，**没有办法直接理解**。实际上，我们可以将**任意数据**写入文件，无论是**文本数据**或是**二进制数据**，他们只是一系列字节，只是我们的**打开方式不同**。而确认了打开方式，就确定了如何解释文件数据，例如一个20字节的文件，可以理解成20个字符，也可以理解成5个32位整数。

文本打开方式:

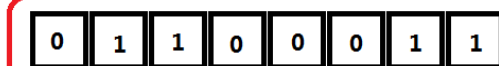
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I		l	o	v	e		c	o	d	i	n	g	.	\0	\0	\0	\0	\0	\0

二进制打开方式:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
73	32	108	111	118	101	32	99	111	100	105	110	103	46	0	0	0	0	0	0

1869357129 1663067510 1852400751 11879 0

互联网:



ASC2码文件的读取

FILE *是**文件指针**类型，它指向打开后的文件。

无论是**读取**还是**写入**文件都需要对文件进行打开，**打开文件**会**耗费**系统的资源，**打开文件**后在**程序结束前**需要对文件进行**关闭**，否则会造成**泄漏**。

FILE * fopen(const char * path, const char * mode);

文件顺利打开后，指向该流的**文件指针**就会被返回，如果文件打开失败则**返回 NULL**。path是文件所在的路径，mode是文件的**打开方式**，"r"方式是文件**读取方式**。

int fclose(FILE *fp);

关闭文件，成功fclose 返回 0，否则返回常量EOF(-1)。

int fgetc(FILE *stream);

fgetc从文件指针stream指向的文件中**读取**一个字符。读取到**文件结束**时，会遇到文件结束符**EOF**。

char *fgets(char *buf, int bufsz, FILE *stream);

fgets从文件结构体指针stream中读取一行。buf最大为bufsz，最多读取bufsz-1个字符。

fscanf

使用方法与scanf完全一样，只是需要传入一个文件指针。

```
ch = a
ch = b
buffer = cdefg
buffer = 100 100 100
num1 = 12345 num2 = 123.200000
请按任意键继续. . .
```

ASC2码的读取_in.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
abcdefg
100 100 100
12345 123.2
```

```
#include <stdio.h> //文件打开
int main(){
    FILE *fp = fopen("ASC2码的读取_in.txt", "r");
```

```
    int ch = fgetc(fp);
    printf("ch = %c\n", ch); //一个字节一个字节读取
    ch = fgetc(fp);
    printf("ch = %c\n", ch);
```

```
    char buffer[50];
    fgets(buffer, 50, fp); //一行一行读取
    printf("buffer = %s", buffer);
    fgets(buffer, 50, fp);
    printf("buffer = %s", buffer);
```

```
    int num1;
    double num2; //按照规则读取
```

```
    fscanf(fp, "%d %lf", &num1, &num2);
```

```
    printf("num1 = %d num2 = %lf\n", num1, num2);
    fclose(fp);
    return 0;
```

```
}
```

ASC2码文件的写入

使用fopen打开文件，我们还可以向文件**写入数据**，写入有**多种模式**，最常用的是**建立新文件写入模式**"w"与**追加写入模式**"a"。

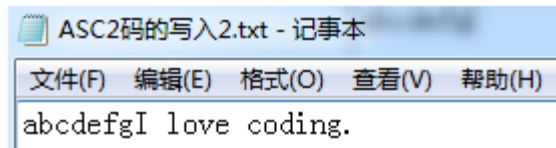
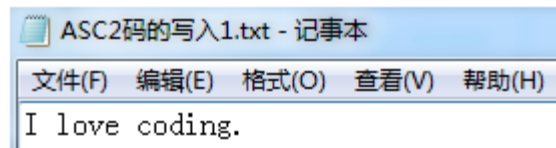
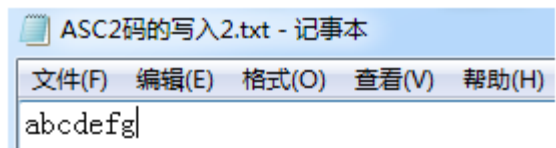
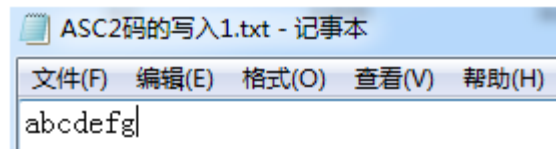
建立新文件写入模式"w": **若文件存在**则将原文件删除，创建一个新的同名文件进行写入，**不存在**直接创建进行写入。

追加写入模式"a": 若文件存在，则在文件**尾部继续写入**，若文件不存在，则**创建一个新文件**。

文件写入有很多函数，**fprintf**是最常用的ASC2码文件写入函数使用方法与printf完全一样，只是多了传入文件指针。

```
#include <stdio.h>

int main() {
    FILE *fp1 = fopen("ASC2码的写入1.txt", "w");
    FILE *fp2 = fopen("ASC2码的写入3.txt", "a");
    fprintf(fp1, "I love coding.\n");
    fprintf(fp2, "I love coding.\n");
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```



例1-和谐词汇

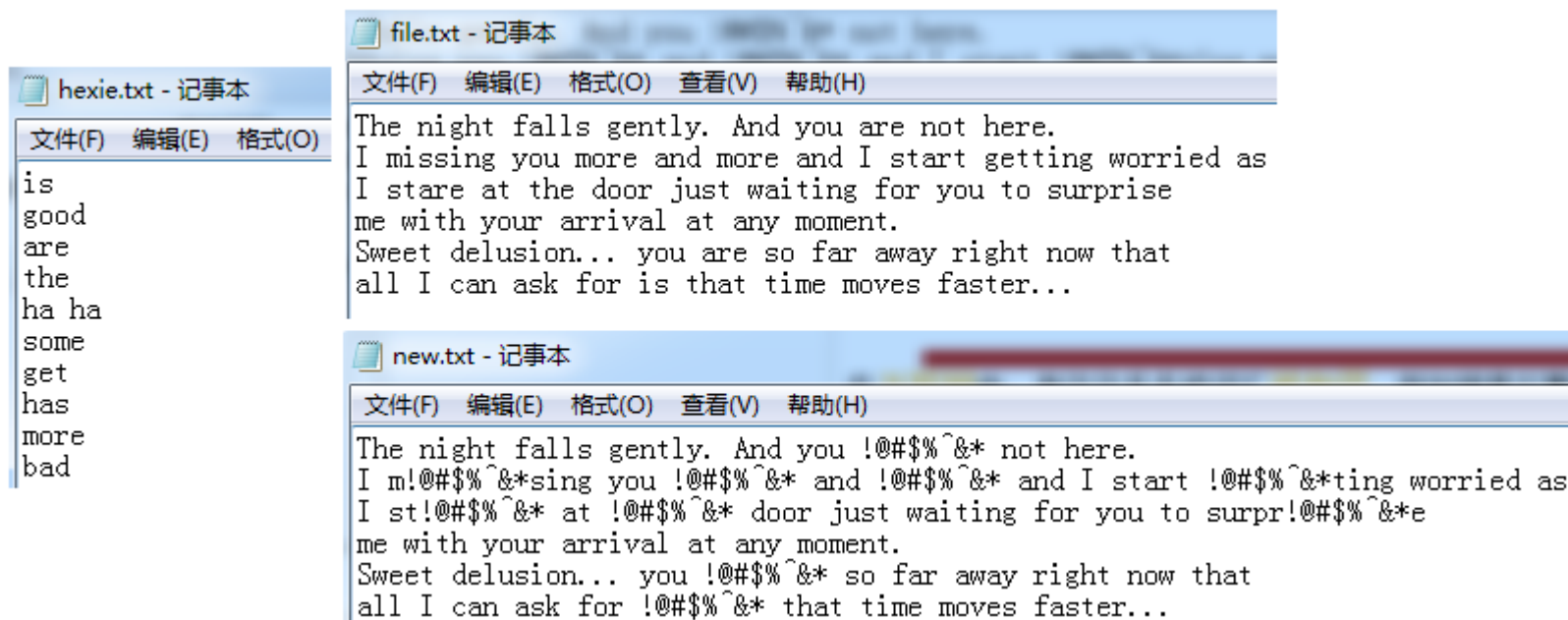
在**互联网**中，有许许多多的词汇**被和谐**，例如搜索引擎经常会出现提示信息“根据相关法律法规和政策，部分搜索结果未予显示”。现在需要设计一个**屏蔽词库**，用来**和谐文章**中的一些**关键词**。屏蔽词库是一个ASCII码文件，这个文件中只含有单词，每个单词占一行，该文件名为“hexie.txt”。屏蔽词语个数**不超过**100个，且每个屏蔽词**小于**50个字节。**待过滤的文件**为(file.txt)，它的大小小于10KB，将里面出现在hexie.txt中的词语全部替换成“!@#\$%^&*”（按住键盘shift和数字1至8），写入一个新文件(new.txt)中，这里要注意，如果一个词语中包含屏蔽词汇，那么只将**屏蔽词汇替换**，例如“hehasAAA”被处理后将得到“he!@#\$%^&*AAA”，注意屏蔽词汇**区分大小写**(aaa与AAA是两个不同的单词)，为了使问题简化，屏蔽词汇中**不会出现互相包含**的情况，如“xabcx”与“abc”不会同时出现在同一个屏蔽词库中。

The screenshot displays three Notepad windows. The top window, 'file.txt - 记事本', contains the following text:
The night falls gently. And you are not here.
I missing you more and more and I start getting worried as
I stare at the door just waiting for you to surprise
me with your arrival at any moment.
Sweet delusion... you are so far away right now that
all I can ask for is that time moves faster...

The bottom-left window, 'hexie.txt - 记事本', shows a list of words to be filtered:
is
good
are
the
ha ha
some
get
has
more
bad

The bottom-right window, 'new.txt - 记事本', shows the result after filtering. The words from hexie.txt have been replaced with '!@#\$%^&*' in the original text:
The night falls gently. And you !@#\$%^&* not here.
I m!@#\$%^&*sing you !@#\$%^&* and !@#\$%^&* and I start !@#\$%^&*ting worried as
I st!@#\$%^&* at !@#\$%^&* door just waiting for you to surpr!@#\$%^&*e
me with your arrival at any moment.
Sweet delusion... you !@#\$%^&* so far away right now that
all I can ask for !@#\$%^&* that time moves faster...

例1-思考



思考:

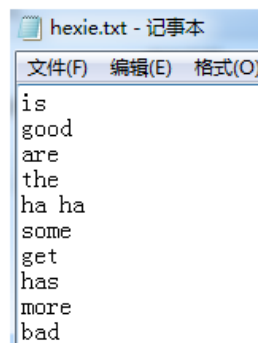
- 1.题目中需要使用**什么模式**打开几个文件？需要**读取几个文件**，**写入几个文件**？
- 2.在读取原始文件file.txt时，应该用**怎样的方式**读取文件中的字符，应该**使用哪个函数**读取文件中内容？
- 3.如何**读取与存储**hexie.txt中的词语列表，后续应该**如何查找**该词表？
- 4.在**读取某个词语**时，应该如何判断这个**词语与词典的关系**？有几种关系？
- 5.如何**设计整体算法**，将file.txt中的出现在hexie.txt中的词语进行替换并写入新文件，没出现的字符直接写入文件中？
- *6.学过数据结构的同学可以思考，该题可以用什么**高级的数据结构**解决？

例1-数据读取，课堂练习

```
#define MAX_WORD_LEN 50
```

//按行读取fp_hexie指向的文件中的词语，存储至二维字符数组hexie中

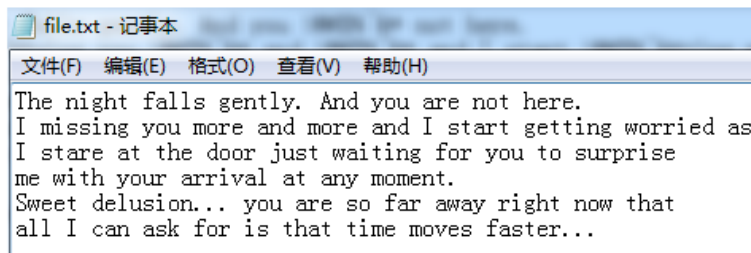
```
int read_hexie_files(FILE *fp_hexie,
                    char hexie[][MAX_WORD_LEN]) {
    int cnt = 0;
    while(fgets(hexie[cnt], MAX_WORD_LEN, fp_hexie)) {
        int len = strlen(hexie[cnt]);
        1
        cnt++;
    }
    return cnt;
}
```



3分钟，填写代码，有问题提出！

//将fp_file中的全部数据存储在file

```
int read_file_files(FILE *fp_file, char file[]) {
    int ch = fgetc(fp_file);
    int file_len = 0;
    while ( 2 ) {
        3
        ch = fgetc(fp_file);
    }
    return file_len;
}
```

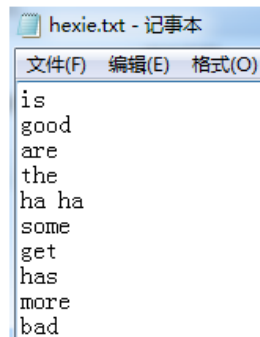


例1-数据读取，实现

```
#define MAX_WORD_LEN 50
```

//按行读取fp_hexie指向的文件中的词语，存储至二维字符数组hexie中

```
int read_hexie_files(FILE *fp_hexie,
                    char hexie[][MAX_WORD_LEN]) {
    int cnt = 0;
    while(fgets(hexie[cnt], MAX_WORD_LEN, fp_hexie)) {
        int len = strlen(hexie[cnt]);
        hexie[cnt][len-1] = '\0';
        cnt++;
    }
    return cnt;
}
```



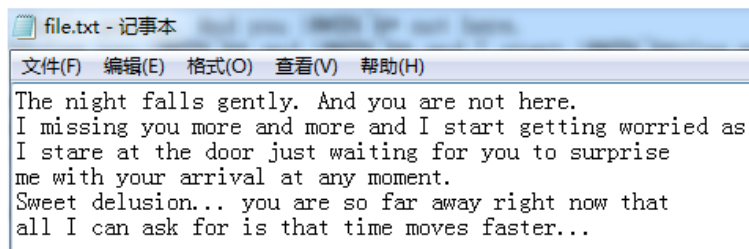
hexie.txt - 记事本

文件(F) 编辑(E) 格式(O)

is
good
are
the
ha ha
some
get
has
more
bad

//将fp_file中的全部数据存储至file

```
int read_file_files(FILE *fp_file, char file[]) {
    int ch = fgetc(fp_file);
    int file_len = 0;
    while (ch != EOF) {
        file[file_len++] = ch;
        ch = fgetc(fp_file);
    }
    return file_len;
}
```



file.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

The night falls gently. And you are not here.
I missing you more and more and I start getting worried as
I stare at the door just waiting for you to surprise
me with your arrival at any moment.
Sweet delusion... you are so far away right now that
all I can ask for is that time moves faster...

例1-读取的词语与词典的关系

在读取过程中遇到的词语在词典中可能会有**3种关系**:

- 1.该词语**在词典中**，**需要被和谐**，即替换为!@#%^&*。
- 2.该词语**不在词典中**，但是它是词典中某个词语的**前缀**，即未来有可能与后面的字符组成的词语被替换为!@#%^&*。
- 3.该词语**不在词典中**，也**不是**词典中某个词的前缀，思考这时应该怎样做？

思考，一个**指针**扫描文章是否可行？

hexie.txt

is
good
are
the
ha ha
some
get
has
more
bad

!@#%^&*

are not here

↑ 未来将扫描到are，会被和谐

某个词的前缀，需要继续向前遍历才知道

away right now that



night falls gently



与任何词都不匹配，也不是某个词的前缀

//hexie字符串数组中有n个单词，查找word单词是否出现，出现返回1，否则返回0

```
int is_in_hexie_dict(const char *word,
                    char hexie[][MAX_WORD_LEN], int n){
    int i;
    for (i = 0; i < n; i++){
        if (strcmp(word, hexie[i]) == 0){
            return 1;
        }
    }
    return 0;
}
```

//判断prefix是str的前缀，是则返回1，否则返回0

```
int is_prefix(const char *str, const char *prefix){
    int i = 0;
    while (1){
        if (2){
            return 0;
        }
        i++;
    }
    if (3){
        return 0;
    }
    return 1;
}
```

//hexie字符串数组中有n个单词，查找word单词是否是其中某个单词的前缀，是则返回1，否则返回0

```
int is_a_prefix_in_hexie_dict(const char *word,
                              char hexie[][MAX_WORD_LEN], int n){
    int i;
    for (i = 0; i < n; i++){
        if (is_prefix(hexie[i], word)){
            return 1;
        }
    }
    return 0;
}
```

例1-课堂练习

3分钟，填写代码，有问题提出！

例1-实现

//hexie字符串数组中有n个单词，查找word单词是否出现，出现返回1，否则返回0

```
int is_in_hexie_dict(const char *word,
                    char hexie[][MAX_WORD_LEN], int n){
    int i;
    for (i = 0; i < n; i++){
        if (strcmp(word, hexie[i]) == 0){
            return 1;
        }
    }
    return 0;
}

//判断prefix是str的前缀，是则返回1，否则返回0
```

```
int is_prefix(const char *str, const char *prefix){
    int i = 0;
    while (str[i] && prefix[i]){
        if (str[i] != prefix[i]){
            return 0;
        }
        i++;
    }
    if (prefix[i]){
        return 0;
    }
    return 1;
}
```

//hexie字符串数组中有n个单词，查找word单词是否是其中某个单词的前缀，是则返回1，否则返回0

```
int is_a_prefix_in_hexie_dict(const char *word,
                              char hexie[][MAX_WORD_LEN], int n){
    int i;
    for (i = 0; i < n; i++){
        if (is_prefix(hexie[i], word)){
            return 1;
        }
    }
    return 0;
}
```

例1-整体分析

begin与end之间的词既不是和谐词，也不是和谐词前缀(没有可能成为和谐词)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
y	o	u		m	o	r	e		a	n	d		m	o	r	e			

is
good
are
the
ha ha
some
get
has
more
bad

begin↑↑end

begin与end之间的词是和谐词前缀时,后面有可能出现和谐词

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
y	o	u		m	o	r	e		a	n	d		m	o	r	e			

begin↑end↑

begin与end之间的词是和谐词时:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
y	o	u		m	o	r	e		a	n	d		m	o	r	e			

begin↑end↑

begin与end之间的词既不是和谐词，也不是和谐词前缀(没有可能成为和谐词)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
y	o	u		m	o	r	e		a	n	d		m	o	r	e			

begin↑↑end

begin与end之间的词既不是和谐词，也不是和谐词前缀(没有可能成为和谐词)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I		m	i	s	s	i	n	g		y	o	u							

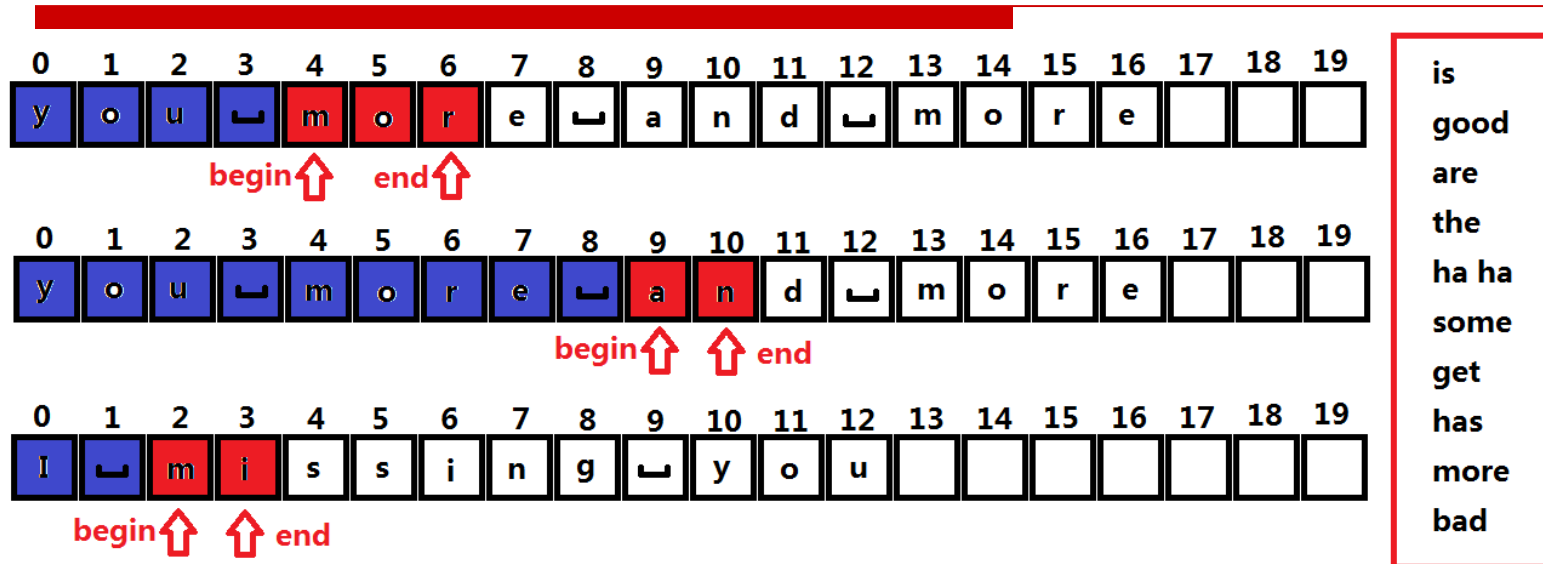
begin↑↑end



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
I		m	i	s	s	i	n	g		y	o	u							

begin↑↑end

例1-整体算法设计



设置两个指针，**begin**指针与**end**指针，**初始时**指向file第一个字符。

循环end指针，每次向前移动一个字符，直到file结束：

begin指针与end指针维护一个**窗口**，判断该窗口中的单词word是否是**hexie词典中单词** 或是
hexie词典中**单词的前缀**。

如果word是**hexie中的单词**，**begin指针指向end+1**，并打印!@#\$%^&*。
否则：

如果word**不是**hexie词典中**单词的前缀**：

输出**字符*begin**，begin++；

如果此时的word是hexie词典中的**单词**，begin指针指向end+1，并打印
!@#\$%^&*。

例1-整体代码

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

#define MAX_WORD_NUM 100
#define MAX_WORD_LEN 50
#define MAX_FILE_LEN 12000
```

```
void word_copy(char *word, char *begin, char *end) {
    while (begin <= end) {
        *word = *begin;    //将begin与end指针区间的
        begin++;           字符拷贝至word指向的空间
        word++;
    }
    *word = '\0';
}
```

```
int main() {
```

```
FILE *fp_hexie = fopen("hexie.txt", "r");
FILE *fp_file = fopen("file.txt", "r");
FILE *fp_new = fopen("new.txt", "w");
```

//文件打开

```
char hexie[MAX_WORD_NUM][MAX_WORD_LEN];
int word_cnt = read_hexie_files(fp_hexie, hexie);

char file[12000] = {0};
int file_len = read_file_files(fp_file, file);
```

核心程序代码

```
fclose(fp_hexie);
fclose(fp_file);
fclose(fp_new);
```

//文件关闭，注意打开的文件都要关闭

```
return 0;
```

例1-核心程序代码，课堂练习

```
char *begin = file; //初识时begin与end指针指向file开始
char *end = file;
char temp_word[100] = {0};
while(*end){
    word_copy(temp_word, begin, end);
    if (is_in_hexie_dict(temp_word, hexie, word_cnt)){
        fprintf(fp_new, " !@#$$%^&*");
        1
    }
    else{
        if (!is_a_prefix_in_hexie_dict(temp_word, hexie, word_cnt)){
            fprintf(fp_new, "%c", *begin);
            2
            word_copy(temp_word, begin, end);
            if (is_in_hexie_dict(temp_word, hexie, word_cnt)){
                3
                4
            }
        }
    }
}
5
while(*begin){
    fprintf(fp_new, "%c", *begin);
    begin++;
}
```

0	1	2	3	4	5	6	7	8	9	10	11
y	o	u	_	m	o	r	e	_	a	n	d

begin ↑ end ↑

0	1	2	3	4	5	6	7	8	9	10	11
y	o	u	_	m	o	r	e	_	a	n	d

begin ↑ end ↑

0	1	2	3	4	5	6	7	8	9	10	11
I	_	m	i	s	s	i	n	g	_	y	o

begin ↑ end ↑

is
good
are
the
ha ha
some
get
has
more
bad

3分钟，填写代码
，有问题提出！

例1-核心程序代码，实现

```
char *begin = file; //初识时begin与end指针指向file开始
char *end = file;
char temp_word[100] = {0};
while(*end){
    word_copy(temp_word, begin, end);
    if (is_in_hexie_dict(temp_word, hexie, word_cnt)){
        fprintf(fp_new, "!@#$$%^&*");
        begin = end + 1;
    }
    else{
        if (!is_a_prefix_in_hexie_dict(temp_word, hexie, word_cnt)){
            fprintf(fp_new, "%c", *begin);
            begin++;
            word_copy(temp_word, begin, end);
            if (is_in_hexie_dict(temp_word, hexie, word_cnt)){
                fprintf(fp_new, "!@#$$%^&*");
                begin = end + 1;
            }
        }
    }
    end++;
}
while(*begin){
    fprintf(fp_new, "%c", *begin);
    begin++;
}
```

0	1	2	3	4	5	6	7	8	9	10	11
y	o	u	_	m	o	r	e	_	a	n	d

begin↑ end↑

0	1	2	3	4	5	6	7	8	9	10	11
y	o	u	_	m	o	r	e	_	a	n	d

begin↑ end↑

0	1	2	3	4	5	6	7	8	9	10	11
I	_	m	i	s	s	i	n	g	_	y	o

begin↑ end↑

is
good
are
the
ha ha
some
get
has
more
bad

课间休息10分钟！

有问题提出！

二进制文件的写入

我们可以把**任意数据**以**二进制**的形式写入文件中，例如整型数据、浮点型数据、用户自定义的结构体数据。即在文件中记录了这些数据在内存中存储的**原始形式**，二进制文件读取与写入比文本文件(ASCII码)**更快**。

二进制文件的打开仍然使用fopen函数，最常用的模式包括，**读取模式**是"rb"；**建立新文件写入模式**"wb"；**追加写入模式**"ab"。二进制文件的写入使用**fwrite函数**，它向指定的文件中写入若干数据块，如**成功**执行则返回实际写入的数据项数目。

函数原型: `size_t fwrite(const void* buffer, size_t size, size_t count, FILE* stream);`

(1)buffer: 指向待写入的**数据内存地址**；(2)size: 待写入的数据项**单位字节数**；(3)count: 写入的**数据项的个数**；(4)stream: **目标文件指针**；(5)返回**实际写入的数据项个数**。

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

```
struct student{
```

```
    char name[20];
```

```
    int score;
```

```
};
```

```
typedef struct student student;
```

```
int main() {
```

```
    student s[20];
```

```
    int n;
```

```
    int i, j;
```

//从键盘中读入数据

```
    scanf("%d", &n);
```

```
    for (i = 0; i < n; i++){
```

```
        scanf("%s %d", s[i].name, &s[i].score);
```

```
    }
```

```
    FILE *fp = fopen("binary_write.dat", "wb");
```

```
    int write_cnt = fwrite(&n, sizeof(int), 1, fp);
```

```
    printf("write_cnt = %d\n", write_cnt);
```

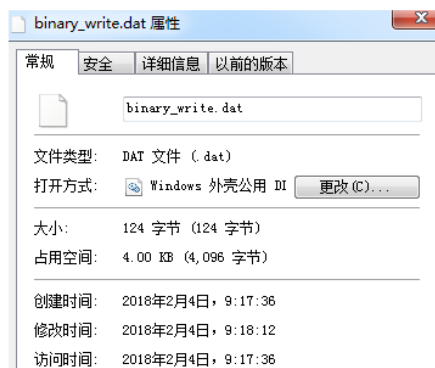
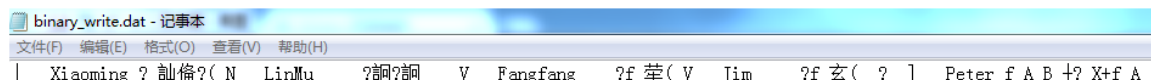
```
    write_cnt = fwrite(s, sizeof(student), n, fp);
```

```
    printf("write_cnt = %d\n", write_cnt);
```

```
    int binary_sum = sizeof(int) + n * sizeof(student);
```

```
    printf("binary_sum = %d\n", binary_sum);
```

```
    return 0;
```



//将读入数据以二进制形式写入文件中

```
5
Xiaoming 78
LinMu 86
Fangfang 86
Jim 93
Peter 65
write_cnt = 1
write_cnt = 5
binary_sum = 124
请按任意键继续. . .
```

二进制文件的读取

实际上，我们若想从某二进制文件中**读入正确的数据**，必须知道这些数据**原来是如何写入的**，或者说这些二进制数据在文件中的**格式**是什么样的。

二进制文件的读取使用**fread函数**，它向指定的文件中**读入**若干数据块，如成功执行则返回**实际读入的数据项数目**。

size_t fread (void *buffer, size_t size, size_t count, FILE *stream);

(1)buffer: 指向**存储读取数据**的内存地址;

(2)size: 待读入的数据项**单位字节数**;

(3)count: 读入的**数据项个数**;

(4)stream: **目标文件指针**;

(5)返回**实际读入**的数据项个数。

```
read_cnt = 1
read_cnt = 5
5
Xiaoming 78
LinMu 86
Fangfang 86
Jim 93
Peter 65
请按任意键继续. . .
```

```
binary_write.dat - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
| Xiaoming ? 訕儻?( N LinMu ?訕?訕 V Fangfang ?f 葦( V Jim ?f 玄(

#include <stdio.h>
struct student{
    char name[20];
    int score;
};
typedef struct student student;

int main() {
    student s[20];
    int n;
    int i;

    FILE *fp = fopen("binary_write.dat", "rb");
    int read_cnt = fread(&n, sizeof(int), 1, fp);
    printf("read_cnt = %d\n", read_cnt);
    read_cnt = fread(s, sizeof(student), n, fp);
    printf("read_cnt = %d\n", read_cnt);

    printf("%d\n", n);
    for (i = 0; i < n; i++){
        printf("%s %d\n", s[i].name, s[i].score);
    }

    return 0;
}
```

//二进制数据的读取

//将读取的数据打印到屏幕

文件定位操作

由于文件在读写时，**文件指针是向后移动的**，有时我们希望知道当前文件指针距离初始时的**偏移量**或**重新定义**文件指针的位置。这时可以用**ftell**与**fseek**函数。

函数原型: `int ftell(FILE *stream);`

函数 `ftell` 用于得到文件指针**当前位置**相对于文件首的**偏移字节数**。注意:该函数对大于 $2^{31}-1$ 的文件，即：2.1G以上的文件操作时可能出错。

函数原型: `int fseek(FILE *stream, int offset, int fromwhere);`

函数 `fseek` 用于根据文件的**某一位置**，**移动**文件指针。从文件的**fromwhere**位置**向后移动** `stream` 指针 `offset` 个位置，执行成功返回0,失败返回非0。有三个**常量**可作为**fromwhere**参数传入函数，**SEEK_SET**代表文件的**开头**、**SEEK_CUR**代表文件当前位置，**SEEK_END**代表文件的**结尾**。

```
sizeof student = 24
pos = 0
n = 5
pos = 4
pos = 76
sum = 76
n = 5
Xiaoming 78
LinMu 86
Fangfang 86
Jim 93
Peter 65
请按任意键继续...
```

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

```
struct student{
    char name[20];
    int score;
```

```
};
typedef struct student student;
```

```
int main(){
    student s[20];
    int n;
    int i;
```

//从二进制文件中读取一个整数，观察文件指针的移动

```
printf("sizeof student = %d\n", sizeof(student));
```

```
FILE *fp = fopen("binary_write.dat", "rb");
int pos = ftell(fp);
printf("pos = %d\n", pos);
fread(&n, sizeof(int), 1, fp);
printf("n = %d\n", n);
pos = ftell(fp);
printf("pos = %d\n", pos);
```

```
int m = n - 2;
fread(s, sizeof(student), m, fp);
pos = ftell(fp);
printf("pos = %d\n", pos);
```

//从二进制文件中读取n-2个学生数据(即剩下两个学生数据)，观察指针的移动

```
int sum = sizeof(int) + sizeof(student) * 3;
printf("sum = %d\n", sum);
```

//计算已读取的数据大小与pos比较一下看看

```
fseek(fp, 0, SEEK_SET);
fread(&n, sizeof(int), 1, fp);
printf("n = %d\n", n);
fseek(fp, pos, SEEK_SET);
fread(s + m, sizeof(student), 2, fp);
```

**//重新定位文件指针到开始位置读取n
//重新定位文件到pos位置读取剩下的两个学生数据**

```
for (i = 0; i < n; i++){
    printf("%s %d\n", s[i].name, s[i].score);
}
```

```
fclose(fp);
return 0;
```

```
}
```

二进制文件的更新

二进制文件的**更新最常用的模式**有“rb+”与“ab”：

“rb+”代表打开二进制文件，对该文件**进行读写**，但写入**不能超过**原文件大小。

“ab”代表为二进制文件**追加写**内容，所有内容都必须**追加写**入文件的**尾部**。

将存储了n个学生数据的二进制文件进行**更新**，从键盘读入需要**增加**的m个学生数据，写入**文件尾部**，最终原文件存储了**n+m**个学生数据。

```
#include <stdio.h>

struct student{
    char name[20];
    int score;
};
typedef struct student student;

int main(){
    student s[20];
    int n, m;
    int i;
    scanf("%d", &m);
    for (i = 0; i < m; i++){
        scanf("%s %d", s[i].name, &s[i].score);
    }
    //修改原文件学生个数的值

    FILE *fp = fopen("binary_write.dat", "rb+");
    fread(&n, sizeof(int), 1, fp);
    fseek(fp, 0, SEEK_SET);
    n += m;
    fwrite(&n, sizeof(int), 1, fp);
    fclose(fp);

    fp = fopen("binary_write.dat", "ab");
    fwrite(s, sizeof(student), m, fp);
    fclose(fp);

    return 0;
}
```

//追加写入新的m个学生的数据

```
3
s1 99
s2 88
s3 77
请按任意键继续. . .

read_cnt = 1
read_cnt = 8
8
Xiaoming 78
LinMu 86
Fangfang 86
Jim 93
Peter 65
s1 99
s2 88
s3 77
请按任意键继续. . .
```


例2-快捷方式

windows系统中大部分的文件都是二进制文件，例如可执行程序的快捷方式文件就是一个二进制文件。已知快捷方式的**文件头**的结构，设计一个**快捷方式分析程序**，分析快捷方式的**创建时间**。快捷方式的创建时间使用windows.h中的**FILETIME结构体**解析，可以使用FileTimeToSystemTime将FILETIME转换为系统时间**SYSTEMTIME结构体**。windows系统的一个WORD是一个unsigned short，一个DWORD是一个unsigned int，一个QWORD是一个unsigned long long。

文件头结构，参照下面表格：

Offset	Size/Type	Description
0h	1 dword	值常为00000004CH，为字符"L"
4h	16 bytes	GUID
①14h	1 dword	Flags，用来标识快捷方式文件中有哪些可选属性，后面有表单独解释每一位的意义。
②18h	1 dword	目标文件属性，后面解释。
1ch	1 qword	文件创建时间 //1个qword FILETIME结构
24h	1 qword	文件修改时间
2ch	1 qword	文件最后一次访问时间
34h	1 dword	目标文件长度
38h	1 dword	自定义图标个数，
3ch	1 dword	目标文件执行时窗口显示方式： 1、正常显示 2、最小化 3、最大化
40h	1 dword	热键
44h	2 dword	暂时还不清楚用途值常为0

```
struct FILETIME{
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
};
```

```
struct SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
};

BOOL WINAPI FileTimeToSystemTime(
    _In_ const FILETIME *lpFileTime,
    _Out_ LPSYSTEMTIME lpSystemTime
);
```

//上述结构体与函数定义在windows.h中

思考：

- 1.如何将**关键信息**读出？设计**二进制数据**读入结构体。
- 2.如何调用**系统函数**进行时间的转换？

例2-课堂练习

```
C:\C语言程序设计>例2-快捷方式.exe sublime.lnk
year = 2016
month = 1
day = 12
```

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
```

```
struct LinkHead{
```

1

2

```
};
```

```
typedef struct LinkHead LinkHead;
```

```
int main(int argc, char *argv[]){
```

```
FILE *fp = fopen(argv[1], 3 );
```

```
LinkHead head;
```

```
fread( 4 , 1, sizeof(LinkHead), fp);
```

```
SYSTEMTIME system_time;
```

```
FileTimeToSystemTime(&head.create_time, 5 );
```

```
printf("year = %d\n", system_time.wYear);
```

```
printf("month = %d\n", system_time.wMonth);
```

```
printf("day = %d\n", system_time.wDay);
```

```
fclose(fp);
```

```
return 0;
```

```
}
```

DWORD是4字节

Offset	Size/Type	
0h	1 dword	值常为00000004CH, 为字符
4h	16 bytes	GUID
①14h	1 dword	Flags, 用来标识快捷方式
②18h	1 dword	目标文件属性, 后面解释。
1ch	1 qword	文件创建时间

```
struct FILETIME{
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
};
```

//提示

3分钟, 填写代码
有问题提出!

例2-实现

```
C:\C语言程序设计>例2-快捷方式.exe sublime.lnk
year = 2016
month = 1
day = 12
```

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
```

```
struct LinkHead{
```

```
    int temp[7];
```

```
    FILETIME create_time;
```

```
};
typedef struct LinkHead LinkHead;
```

```
int main(int argc, char *argv[]){
```

```
    FILE *fp = fopen(argv[1], "rb");
```

```
    LinkHead head;
```

```
    fread(&head, 1, sizeof(LinkHead), fp);
```

```
    SYSTEMTIME system_time;
```

```
    FileTimeToSystemTime(&head.create_time, &system_time);
```

```
    printf("year = %d\n", system_time.wYear);
```

```
    printf("month = %d\n", system_time.wMonth);
```

```
    printf("day = %d\n", system_time.wDay);
```

```
    fclose(fp);
```

```
    return 0;
```

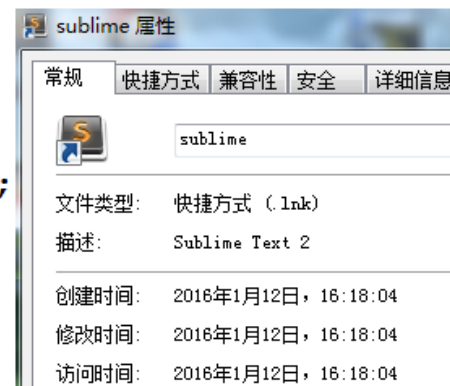
```
}
```

dword是4字节

Offset	Size/Type	
0h	1 dword	值常为00000004CH, 为字符
4h	16 bytes	GUID
①14h	1 dword	Flags, 用来标识快捷方式一位的意义。
②18h	1 dword	目标文件属性, 后面解释。
1ch	1 qword	文件创建时间

```
struct FILETIME{
    DWORD dwLowDateTime;
    DWORD dwHighDateTime;
};
```

//提示



标准的输入输出流

C语言有**3个**预定义的**标准流**，他们**预定义**在<stdio.h>头文件中，分别是**stdin**、**stdout**、**stderr**；分别对应用于**键盘**、**命令行上的正常输出**和**命令行上的错误输出**；使用这些流不需要任何初始化，只需要将它们传入适当的库函数中。标准输入流主要是用于**重定向使用**，使用操作系统命令(例如linux为<与>)，stdin、stdout都可以重定向到文件上。

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

int main() {
    char buffer[1024] = {0};
    fgets(buffer, 1024, stdin);
    while(buffer[0] != '#') {
        int len = strlen(buffer);
        fprintf(stdout, "len = %d\n", len);
        fgets(buffer, 1024, stdin);
    }
    return 0;
}
```

```
abcdefg
len = 8
111
len = 4
#
请按任意键继续. . .
```

freopen函数

freopen是被包含于C标准库头文件<stdio.h>中的一个函数，用于**重定向**输入输出流。该函数可以在**不改变代码原貌**的情况下改变输入输出环境。

函数原型:FILE *freopen(const char *filename, const char *mode, FILE *stream);

1)filename是**重定向**的文件。2)mode是**打开方式**。3)stream是流，经常是**标注输入stdin、输出stdout流**。

**//将文件的引入_data.in.txt的内容，重定向至
stdin；即这个文件的内容当作是键盘输入给程序**

```
int main() {
```

```
    freopen("文件的引入_data.in.txt", "r", stdin);  
    freopen("文件的引入_data.out.txt", "w", stdout);
```

```
    student s[3000]; //将屏幕打印的内容重定向至文件的引入  
    int n;  
    int i;  
    _data.out.txt; 即将屏幕打印的内容写入该文件
```

```
    scanf("%d", &n); //利用标准输入函数，实际上读取的是文件内容  
    for (i = 0; i < n; i++) {  
        scanf("%s %d %d %d %d", s[i].name, &s[i].id,  
            &s[i].score[0], &s[i].score[1], &s[i].score[2]);  
        s[i].sum = s[i].score[0] + s[i].score[1] + s[i].score[2];  
    }
```

```
    qsort(s, n, sizeof(s[0]), student_cmp);
```

```
    printf("%s\t%-15s%s\t%-15s\t%-15s\t%-15s\t%-15s\n", "id", "name", "sum",  
        "score1", "score2", "score3");  
    for (i = 0; i < n; i++) {  
        printf("%d\t%-15s%d\t%-15s\t%-15s\t%-15s\t%-15s\n",  
            s[i].id, s[i].name, s[i].sum,  
            s[i].score[0], s[i].score[1], s[i].score[2]);  
    }
```

```
    return 0;
```

//利用标准输出函数，实际上是往文件里写

文件的引入_data.in.txt - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)
5			
Bill 4 91 70 66			
Tom 1 69 81 77			
XiaoFang 5 70 60 95			
XiaoMing 3 81 87 85			
Mike 2 80 97 71			

文件的引入_data.out.txt - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)	
id	name	sum	score1	score2	score3
3	XiaoMing	253	81	87	85
2	Mike	248	80	97	71
1	Tom	227	69	81	77
4	Bill	227	91	70	66
5	XiaoFang	225	70	60	95

随机数生成测试数据

rand是**随机数**函数，但它不是真正的随机数生成器，srand()会设置供rand()使用的**随机数种子**。最常见的用法是，指定**系统时间**作为**随机种子**，**time(NULL)**可以获取系统时间。

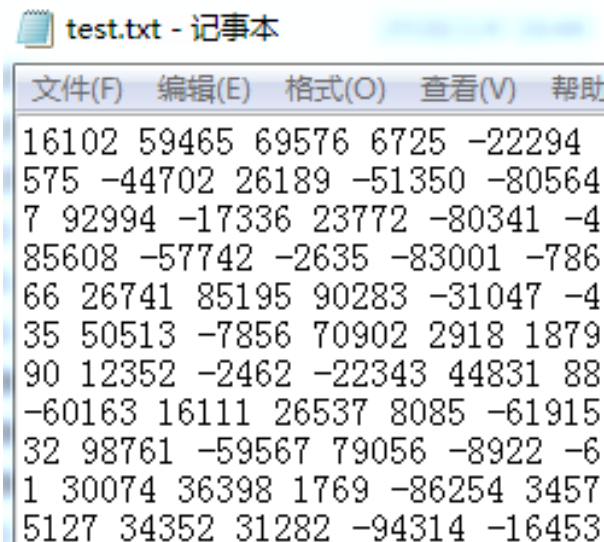
例如srand(time(NULL));

指定当前系统时间作为随机种子。由于**每个种子**对应一组根据算法预先生成的随机数，所以在相同的平台环境下，**不同时间**产生的**随机数**会是**不同的**，相应的，若将time(NULL)改为**任一常量**，则**无论何时运行**、**运行多少次**得到的“随机数”都会是一组**固定的序列**，因此srand生成的随机数是**伪随机数**。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define MAX_NUMBER 10000

int main() { //生成10000个随机正数，该整数在100003内，并且有符号
    freopen("test.txt", "w", stdout);
    srand(time(NULL));
    int i;
    for (i = 0; i < MAX_NUMBER; i++){
        int num = rand() * rand() * rand() % 100003;
        if (rand() % 2){
            printf("%d ", num);
        }
        else{
            printf("%d ", -num);
        }
    }
    return 0;
}
```



test.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助

```
16102 59465 69576 6725 -22294
575 -44702 26189 -51350 -80564
7 92994 -17336 23772 -80341 -4
85608 -57742 -2635 -83001 -786
66 26741 85195 90283 -31047 -4
35 50513 -7856 70902 2918 1879
90 12352 -2462 -22343 44831 88
-60163 16111 26537 8085 -61915
32 98761 -59567 79056 -8922 -6
1 30074 36398 1769 -86254 3457
5127 34352 31282 -94314 -16453
```

C语言工程引入

实际应用的**C语言工程**一般包含多个**c源文件**与**h头文件**。c源文件保存源程序，h头文件保存源程序中的结构体定义或函数声明等。我们一般使用**#include** 指令包含头文件(通常不会使用include指令包含源文件，那样可能会出现重复代码定义)，从而将各个c源文件**组织**在一起，然后进行整体的**编译、链接并生成**一个**可执行程序**(bin或者.exe文件)在操作系统中运行。**编译工程**我们可以使用**集成ide环境**(例如VS)或**Makefile脚本**调用编译工具gcc进行编译，这与开发环境相关。

sort.h

```
1
2 void sort_array(void *base, int num, int width,
3               int (*compare)(const void *, const void *));
```

sort.c

```
1
2 #include <stdlib.h>
3 #include "sort.h"
4
5 static void swap(char *a, char *b, int width){
6     char tmp;
7     while(width){
8         tmp = *a;
9         *a = *b;
10        *b = tmp;
11        a++;
12        b++;
13        width--;
14    }
15 }
16
17 void sort_array(void *base, int num, int width,
18               int (*compare)(const void *, const void *)){
19     int i, j;
20     void *temp = malloc(width);
21     for (i = 0; i < num; i++){
22         for (j = i + 1; j < num; j++){
23             void *element1 = base + i * width;
24             void *element2 = base + j * width;
25             if (compare(element1, element2) > 0){
26                 swap(temp, element1, width);
27                 swap(element1, element2, width);
28                 swap(element2, temp, width);
29             }
30         }
31     }
32     free(temp);
33 }
```

student.h

```
1
2 #define MAX_STUDENT_BUFFER 1024
3
4 struct student{
5     char name[20];
6     int score;
7 };
8
9 typedef struct student student;
10
11 int student_cmp(const void *a, const void *b);
```

student.h

student.c

```
1 #include <string.h>
2 #include "student.h"
3
4 int student_cmp(const void *a, const void *b){
5     student *s1 = (student *)a;
6     student *s2 = (student *)b;
7     if (s1->score != s2->score){
8         return s2->score - s1->score;
9     }
10    return strcmp(s1->name, s2->name);
11 }
12
13 student g_student[MAX_STUDENT_BUFFER];
```

student.c

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -L./ -lstudent
$ ./sort_student_tool < test_in.txt
Jim 93
Fangfang 86
LinMu 86
Xiaoming 78
Peter 65
```

main.c

```
1 #include <stdio.h>
2 #include "student.h"
3 #include "sort.h"
4
5 extern student g_student[MAX_STUDENT_BUFFER];
6
7 int main(){
8     int n;
9     int i;
10    scanf("%d", &n);
11    for (i = 0; i < n; i++){
12        scanf("%s %d", g_student[i].name,
13              &g_student[i].score);
14    }
15    sort_array(g_student, n, sizeof(g_student[0]));
16    for (i = 0; i < n; i++){
17        printf("%s %d\n", g_student[i].name,
18              g_student[i].score);
19    }
20    return 0;
21 }
```

main.c

Makefile

```
1 CC := gcc
2 INCLUDE := -I./
3 LDFLAGS := -L./ -lstudent
4 CFLAGS := -Wall $(INCLUDE)
5 LIB := libstudent.a
6 src := student.c sort.c
7 obj := $(src:%.c=%.o)
8
9 default : $(LIB)
10 all : $(LIB) sort_student_tool
11
12 $(LIB) : $(obj)
13     ar rc $@ $^
14 sort_student_tool : main.c
15     $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
16 clean :
17     rm -rf *.o $(LIB)
18     rm sort_student_tool
19
```

Makefile

Linux下的gcc编译工具

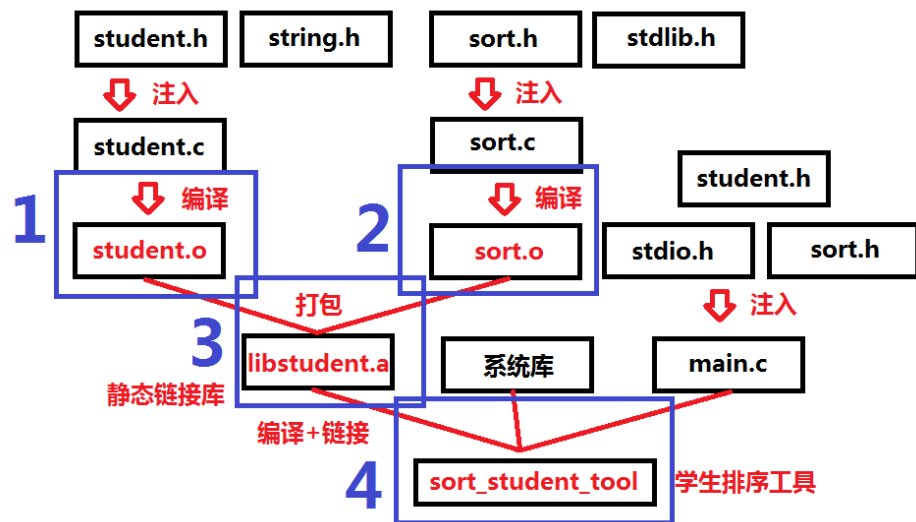
GCC最初是**C语言编译器**，原名为(GNU C Compiler)，现在为**GNU编译器套件**(GNU Compiler Collection)，除了C语言，又拓展了C++、Objective-C、Fortran、Java、Ada和Go等语言编译。GCC功能强大，并附带了如**GDB**等**编译调试工具**。

编译的目的是将源程序进行**预处理、编译、汇编**生成**目标文件**(windows下为.obj，linux下为.o)，再将这些目标文件**链接**为**可执行程序**(windows下为.exe，linux一般无扩展名)，gcc可直接编译.c文件生成可执行文件，该工具有很多常用的参数。

静态链接库(windows下为.lib，linux下为.a)，将目标obj(或o)文件可直接打包为静态链接库。**编译阶段**生成可执行程序时，静态链接库会直接链接到可执行程序中，所以使用静态链接库的可执行程序**看起来比较大**。

动态链接库(windows下为.dll，linux下为.so)，**不同于**静态链接，动态链接是可执行程序**执行时**链接的，有了动态链接，我们**不必要**将数百MB甚至数GB的代码都编译到一个程序中(那样后续维护起来会非常不方便)。

将sort.c、student.c编译为**静态链接库**，并与main.c编译为**可执行文件**sort_student_tool main.c。



```
1 $ gcc -Wall -I./ -c -o student.o student.c
```

```
2 $ gcc -Wall -I./ -c -o sort.o sort.c
```

```
3 $ ar rc libstudent.a student.o sort.o
```

```
4 $ gcc -Wall -I./ -o sort_student_tool main.c -L./ -lstudent
```

-Wall，生成所有警告信息。

-I，指定头文件路径。

-o，制定目标名称。

-c，只激活预处理、编译和汇编，即只把原程序编译为obj文件(不生成可执行文件)

-l，指定程序要链接的库，-l参数紧接着就是库名，例如student库的库名为student，库文件名是libstudent.a，即将lib与尾.a去掉就是库名了。

-L，库文件所在的目录名。

ar，将.o文件打包为建静态库.a文件。rc是它的参数。

Makefile简介

直接使用GCC命令进行工程编译较为复杂并且不具备自动化，一个现实中的工程源文件不计其数，其按类型、功能、模块分别放在若干个目录中,如果全部使用手动gcc编译工作量非常大，并且难以维护。通过Makefile文件制定一系列规则，从而指定源文件的编译顺序、编译依赖或进行更复杂的工程操作，使得整个工程完全自动编译，极大提高了效率。

make是一个命令工具，它解释Makefile中的指令。类似C语言，Makefile有自己的书写格式、关键字、函数，并且在Makefile中可以使用系统shell所提供的任何命令来完成想要的工作。

Makefile规则:

目标target : 先决条件(或是说依赖)prerequisites

指令command1

指令command2

...

目标target:

目标文件、可执行文件或者是标签。

先决条件prerequisites:

生成target所需要的文件或目标。

指令command:

make需要执行的命令，任意的Shell命令。

```
1 CC := gcc
2 INCLUDE := -I./
3 LDFLAGS := -L./ -lstudent
4 CFLAGS := -Wall $(INCLUDE)
5 LIB := libstudent.a
6 src := student.c sort.c
7 obj := $(src:%.c=%.o)
```

//变量赋值

//src:%.c=%.o

变量替换函数，将src中的所有*.c文件替换为*.o文件。

```
9 default : $(LIB)
10 all : $(LIB) sort_student_tool
```

//标签

```
12 $(LIB) : $(obj)
13     ar rc $@ $^
14 sort_student_tool : main.c
15     $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
16 clean :
17     rm -rf *.o $(LIB)
18     rm sort_student_tool
```

//Makefile

\$符号是取变量值，@与^是自动化变量，

\$@指目标文件

\$^指去除重复的依赖文件

整体来说，target这一个或多个的目标文件依赖于prerequisites中的文件，其生成规则定义在command中。

prerequisites中如果有一个以上的文件比target文件要新的话，command所定义的命令就会被执行，上述即为Makefile规则。

外部变量与静态函数

一个由几个源文件组成的程序通常需要使用在其他文件中定义的**全局变量**。使用**关键字extern**将变量声明为**外部变量**，则编译器不会**创建**这些变量(分配内存)，只是通知编译器，这些变量在**文件外定义**，通过**extern**修饰声明变量，就可以使用其他文件中定义的**全局变量**了。

源文件中的所有函数都默认为**隐式的extern**，即在**链接器**处理文件对象时，它们在所有**文件**中都可见，即如果知道某函数的定义，即使不include该函数头，直接在文件中写出**正确的函数头**就可以正常使用该函数了。有时希望函数**不可被**本文件外的其他代码访问，确保该函数**仅在定义它的源文件中可见**，即把函数声明为**static**。全局变量若声明为static，其他文件中也无法通过extern使用到该变量了。

```
sort.h
1
2 void sort_array(void *base, int num, int width,
3               int (*compare)(const void *, const void *));

sort.c
1
2 #include <stdlib.h>
3 #include "sort.h"
4
5 static void swap(char *a, char *b, int width){
6     char tmp;
7     while(width){
8         tmp = *a;
9         *a = *b;
10        *b = tmp;
11        a++;
12        b++;
13        width--;
14    }
15 }
16
17 void sort_array(void *base, int num, int width,
18               int (*compare)(const void *, const void *)){
19     int i, j;
20     void *temp = malloc(width);
21     for (i = 0; i < num; i++){
22         for (j = i + 1; j < num; j++){
23             void *element1 = base + i * width;
24             void *element2 = base + j * width;
25             if (compare(element1, element2) > 0){
26                 swap(temp, element1, width);
27                 swap(element1, element2, width);
28                 swap(element2, temp, width);
29             }
30         }
31     }
32     free(temp);
33 }
```

sort.h

sort.c

static void swap(char *a, char *b, int width){

//静态函数，别的文件不能使用该函数

```
student.h
1
2 #define MAX_STUDENT_BUFFER 1024
3
4 struct student{
5     char name[20];
6     int score;
7 };
8
9 typedef struct student student;
10
11 int student_cmp(const void *a, const void *b);

student.c
1 #include <string.h>
2 #include "student.h"
3
4 int student_cmp(const void *a, const void *b){
5     student *s1 = (student *)a;
6     student *s2 = (student *)b;
7     if (s1->score != s2->score){
8         return s2->score - s1->score;
9     }
10    return strcmp(s1->name, s2->name);
11 }
12
13 student g_student[MAX_STUDENT_BUFFER];
```

student.h

student.c

student g_student[MAX_STUDENT_BUFFER];

//全局变量，如果它定义为static，别的地方就无法使用到了

```
main.c
1 #include <stdio.h>
2 #include "student.h"
3 #include "sort.h"
4
5 extern student g_student[MAX_STUDENT_BUFFER];
6
7 int main(){
8     int n;
9     int i;
10    scanf("%d", &n);
11    for (i = 0; i < n; i++){
12        scanf("%s %d", g_student[i].name, &g_student[i].score);
13    }
14    sort_array(g_student, n, sizeof(g_student[0]), student_cmp);
15    for (i = 0; i < n; i++){
16        printf("%s %d\n", g_student[i].name, g_student[i].score);
17    }
18    return 0;
19 }
```

main.c

//使用外部变量，实际使用了student.c中的全局变量

```
Makefile
1 CC := gcc
2 INCLUDE := -I./
3 LDFLAGS := -L./ -lstudent
4 CFLAGS := -Wall $(INCLUDE)
5 LIB := libstudent.a
6 src := student.c sort.c
7 obj := $(src:%.c=%.o)
8
9 default : $(LIB)
10 all : $(LIB) sort_student_tool
11
12 $(LIB) : $(obj)
13     ar rc $@ $^
14 sort_student_tool : main.c
15     $(CC) $(CFLAGS) -o $@ $^ $(LDFLAGS)
16 clean :
17     rm -rf *.o $(LIB)
18     rm sort_student_tool
19
```

Makefile

编译与运行

```
$ ll
total 28
-rwxr--r-- 464 Feb  8 10:32 main.c
-rwxr--r-- 340 Feb  8 10:04 Makefile
-rwxr--r-- 825 Feb  7 11:29 sort.c
-rwxr--r-- 115 Feb  7 11:28 sort.h
-rwxr--r-- 324 Feb  8 10:30 student.c
-rwxr--r-- 179 Feb  7 11:36 student.h
-rwxr--r--  57 Feb  7 12:40 test_in.txt
```

//显示文件

```
$ cat test_in.txt
```

```
5
Xiaoming 78
LinMu 86
Fangfang 86
Jim 93
Peter 65
```

//打印test_in.txt文件中的内容

//编译静态库

```
$ make
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
```

```
$ ll
total 40
-rw-rw-r-- 3470 Feb  8 10:57 libstudent.a
-rwxr--r-- 464 Feb  8 10:32 main.c
-rwxr--r-- 340 Feb  8 10:04 Makefile
-rwxr--r-- 825 Feb  7 11:29 sort.c
-rwxr--r-- 115 Feb  7 11:28 sort.h
-rw-rw-r-- 1760 Feb  8 10:57 sort.o
-rwxr--r-- 324 Feb  8 10:30 student.c
-rwxr--r-- 179 Feb  7 11:36 student.h
-rw-rw-r-- 1472 Feb  8 10:57 student.o
-rwxr--r--  57 Feb  7 12:40 test_in.txt
```

//显示文件

//编译全部内容

```
$ make all
gcc -Wall -I./ -o sort_student_tool main.c -L./ -lstudent
```

```
$ ll
total 52
-rw-rw-r-- 3470 Feb  8 10:41 libstudent.a
-rwxr--r-- 464 Feb  8 10:32 main.c
-rwxr--r-- 340 Feb  8 10:04 Makefile
-rwxr--r-- 825 Feb  7 11:29 sort.c
-rwxr--r-- 115 Feb  7 11:28 sort.h
-rw-rw-r-- 1760 Feb  8 10:41 sort.o
-rwxrwxr-x 8295 Feb  8 10:41 sort_student_tool
-rwxr--r-- 324 Feb  8 10:30 student.c
-rwxr--r-- 179 Feb  7 11:36 student.h
-rw-rw-r-- 1472 Feb  8 10:41 student.o
-rwxr--r--  57 Feb  7 12:40 test_in.txt
```

//显示文件

```
$ ./sort_student_tool < test_in.txt > test_out.txt
```

```
$ cat test_out.txt
Jim 93
Fangfang 86
LinMu 86
Xiaoming 78
Peter 65
```

//将test_in.txt作为标准输入输入至工具
并将标准输出的结果保存至test_out.txt

```
$ make clean
rm -rf *.o libstudent.a
rm sort_student_tool
```

//清理结果

```
$ ll
total 32
-rwxr--r-- 464 Feb  8 10:32 main.c
-rwxr--r-- 340 Feb  8 10:04 Makefile
-rwxr--r-- 825 Feb  7 11:29 sort.c
-rwxr--r-- 115 Feb  7 11:28 sort.h
-rwxr--r-- 324 Feb  8 10:30 student.c
-rwxr--r-- 179 Feb  7 11:36 student.h
-rwxr--r--  57 Feb  7 12:40 test_in.txt
-rw-rw-r-- 49 Feb  8 10:47 test_out.txt
```

//显示文件

编译时可能遇到的问题，课堂练习

问题1：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -lstudent
/usr/bin/ld: cannot find -lstudent
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

问题2：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -L./
/tmp/cc0RQdxG.o(.text+0x42): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x64): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x86): In function 'main':
: undefined reference to 'student_cmp'
/tmp/cc0RQdxG.o(.text+0x90): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x95): In function 'main':
: undefined reference to 'sort_array'
/tmp/cc0RQdxG.o(.text+0xbd): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0xd7): In function 'main':
: undefined reference to 'g_student'
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

问题3：

```
$ make all
gcc -Wall -c -o student.o student.c
student.c:2:21: student.h: No such file or directory
```

问题4：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -L./ -lstudent
/tmp/ccyURkIs.o(.text+0x86): In function 'main':
: undefined reference to 'student_cmp'
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

方案1：

检查int student_cmp(const void *a, const void *b)
函数的实现是否成功编译并成功打包至静态库libstudent.a中，
或该实现是否真的在源文件中存在

方案2：

检查-I include文件目录是否或头文件.h是否在指定目录下

方案3：

检查-L目录是否写正确或静态库libstudent.a是否在指定-L目录中

方案4：

检查-L目录是否写正确或静态库libstudent.a的名称是否写正确

3分钟，将问题与解决方案连线

编译时可能遇到的问题，解答

问题1：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -lstudent
/usr/bin/ld: cannot find -lstudent
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

问题2：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -L./
/tmp/cc0RQdxG.o(.text+0x42): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x64): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x86): In function 'main':
: undefined reference to 'student_cmp'
/tmp/cc0RQdxG.o(.text+0x90): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0x95): In function 'main':
: undefined reference to 'sort_array'
/tmp/cc0RQdxG.o(.text+0xbd): In function 'main':
: undefined reference to 'g_student'
/tmp/cc0RQdxG.o(.text+0xd7): In function 'main':
: undefined reference to 'g_student'
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

问题3：

```
$ make all
gcc -Wall -c -o student.o student.c
student.c:2:21: student.h: No such file or directory
```

问题4：

```
$ make all
gcc -Wall -I./ -c -o student.o student.c
gcc -Wall -I./ -c -o sort.o sort.c
ar rc libstudent.a student.o sort.o
gcc -Wall -I./ -o sort_student_tool main.c -L./ -lstudent
/tmp/ccYurKis.o(.text+0x86): In function 'main':
: undefined reference to 'student_cmp'
collect2: ld returned 1 exit status
make: *** [sort_student_tool] Error 1
```

方案1：

检查int student_cmp(const void *a, const void *b)

函数的实现是否成功编译并成功打包至静态库libstudent.a中，
或该实现是否真的在源文件中存在

方案2：

检查-I include文件目录是否或头文件.h是否在指定目录下

方案3：

检查-L目录是否写正确或静态库libstudent.a是否在指定-L目录中

方案4：

检查-L目录是否写正确或静态库libstudent.a的名称是否写正确

结束

非常感谢大家！

林沐

问答互动

在所报课的课程页面，

- 1、点击“全部问题”显示本课程所有学员提问的问题。
- 2、点击“提问”即可向该课程的老师 and 助教提问问题。



联系我们

小象学院：互联网新技术在线教育领航者

— 微信公众号：**小象学院**

