浙江大学实验报告

专业：微电子科学与技术
姓名：焦天晟
学号：3220105664
日期：2023.04.14

课程名称：C 程序设计专题　　指导老师：翁恺　　成绩：

实验名称：作业 1：　ADIF 数据处理

一、实验题目要求：

本题要求实现一个程序，可以导入 ADIF 或 CSV 格式的数据，存储在自己的二进制格式的存储文件中，并可以导出成 ADIF 或 CSV 格式的文件。数据文件为二进制形式，以 fwrite 和 fread 函数来直接读写结构体的内容。

程序使用命令行参数来指定要做的动作和导入/导出的文件名，以下为程序要实现的命令行参数：

- -i <file name>：导入 ADIF 或 CSV 文件，根据文件名后缀来决定导入的文件类型（.adi 或.csv）。所导入的数据用以更新自己的二进制数据文件。qso_date、time_on 两个字段相同的认为是同一条数据。导入数据时，不同的数据作为新数据加入，相同的数据则用以更新已有的数据。
- -o <file name>：导出 ADIF 或 CSV 文件，根据文件名后缀来决定导出的文件类型。
- -s <call>：按照 call 来搜索出有这个 call 的记录，直接输出为 CSV 格式。
- -l <start time> <end time>：时间按照 YYYYMMDDhhmmss 表示，直接输出在这个时间段内的全部记录，格式为 CSV。

二、实验思路

一个 c 语言程序，可以导入 ADIF 或 CSV 格式的数据，存储在自己的二进制格式的存储文件中，并可以导出成 ADIF 或 CSV 格式的文件

数据文件为二进制形式，以 fwrite 和 fread 函数来直接读写结构体的内容

可以用命令行参数来指定要做的动作和导入/导出的文件名

针对这个题目，可以直接读取一组数据到结构体中，然后将结构体内容直接写入到二进制文件中，再在二进制文件中对数据进行比较，用来补充和替代数据。

首先要定义结构体来存储数据：

```
typedef struct _record{
    char qso_date[9]; // 日期，8 位
    char time_on[7]; // 开始时间，6 位
    char freq[11]; // 频率，最多 10 位
    char mode[6]; // 模式，最多 5 位
    char call[16]; // 呼号，最多 15 位
    char rst_rcvd[4]; // 接收报告，最多 3 位
    char rst_sent[4]; // 发送报告，最多 3 位
} record;
```

在结构中采用字符串来储存数据。

1．-i <file name>：导入 ADIF 或 CSV 文件，根据文件名后缀来决定导入的文件类型（.adi 或.csv）

首先要判断文件的类型，选择以后缀为判断依据。

（1）编写一个判断后缀的函数：

```
int ends_with(char *str, char *suffix) {
    int len_str = strlen(str);
    int len_suffix = strlen(suffix);
    if (len_str < len_suffix) return 0; // 如果字符串长度小于后缀长度，返回 0（假）
    return strcmp(str + len_str - len_suffix, suffix) == 0; // 比较字符串末尾和后缀是否相同，返回结果（真或假）
}
```

（2）接下来定义一个函数来读取 adif 文件中的一组数据，首先定义一些系统参数：

```
int read_adif(FILE *fp, record *r) {
    char line[1024]; // 用来存储每组数据的内容
    char *p; // 用来指向每个字段的位置
    int n; // 用来存储每个字段的长度
```

初始化结构体中的所有字段为空字符串：

```
r->qso_date[0] = '\0';
r->time_on[0] = '\0';
r->freq[0] = '\0';
r->mode[0] = '\0';
r->call[0] = '\0';
r->rst_rcvd[0] = '\0';
r->rst_sent[0] = '\0';
```

读取一行内容，如果到达文件末尾，返回-1（失败）

```
if (f_gets(line, sizeof(line), fp) == NULL) return -1;
```

这里发现如果直接用 fgets 函数来读入数据会导致读入过程在‘\n’就截止了，无法读入在多行的一组数据，故新定义 f_gets 函数来读入一组数据：

首先定义两个长字符宏：

```
#define EOR1 "<eor>"
#define EOR2 "<EOR>"
```

接着编写函数：

```
char *f_gets(char *s, int size, FILE *stream) {
  // 检查参数是否有效
  if (s == NULL || size <= 0 || stream == NULL) {
    return NULL;
  }
  // 定义一个临时缓冲区
  char buffer[size];
  // 初始化缓冲区为空字符串
  buffer[0] = '\0';
  // 定义一个指针，指向缓冲区的末尾
  char *end = buffer;
  // 定义一个变量，记录是否读到了结束符
```

```
13.  int found = 0;
14.  // 循环读取字符，直到遇到文件结束、错误、缓冲区满或者结束符为止
15.  while (!feof(stream) && !ferror(stream) && end - buffer < size - 1 && !found) {
16.   // 读取一个字符
17.   char c = fgetc(stream);
18.   // 如果读取成功
19.   if (c != EOF) {
20.    // 将字符追加到缓冲区末尾
21.    *end++ = c;
22.    // 在缓冲区末尾添加空字符，以便使用字符串函数
23.    *end = '\0';
24.    // 检查缓冲区是否以 EOR1 或 EOR2 结尾
25.    if (strlen(buffer) >= strlen(EOR1) && strcmp(end - strlen(EOR1), EOR1) == 0) {
26.     // 如果是 EOR1，将其替换为空字符，并设置 found 为真
27.     end -= strlen(EOR1);
28.     *end = '\0';
29.     found = 1;
30.    } else if (strlen(buffer) >= strlen(EOR2) && strcmp(end - strlen(EOR2), EOR2) == 0) {
31.     // 如果是 EOR2，将其替换为空字符，并设置 found 为真
32.     end -= strlen(EOR2);
33.     *end = '\0';
34.     found = 1;
35.    }
36.   }
37.   else return NULL;
38.  }
39.  // 如果缓冲区不为空，将其复制到目标字符串中，并返回目标字符串的地址
40.  if (buffer[0] != '\0') {
41.   strcpy(s, buffer);
42.   return s;
43.  }
44.  // 否则，返回 NULL
45.  return NULL;
```

```
46. }
```

经过调试发现没有问题。

接着来查找每一个字段的数据内容：（以第一个数据<QSO_DATE>为例）：

```
1.  p = strstr(line, "<QSO_DATE:");
2.    if (p != NULL) { // 如果找到了
3.      p += 10; // 跳过<QSO_DATE:这 10 个字符
4.      n = atoi(p); // 把后面的数字转换为整数，表示字段的长度
5.      p = strchr(p, '>'); // 找到后面的>字符
6.      if (p != NULL && n == 8) { // 如果找到了，并且长度是 8
7.        p++; // 跳过>字符
8.        strncpy(r->qso_date, p, n); // 把后面 n 个字符复制到结构体中的 qso_date 字段
9.        r->qso_date[n] = '\0'; // 在末尾加上空字符，表示字符串结束
10.     }
11.   }
```

查找其他数据的方法相似。

注意 time_on 数据中要有

```
1.    r->time_on[n] = '\0';
2.    if (n == 4) { // 如果时间只有 4 位，表示没有秒数，需要在后面补两位 00
3.      strcat(r->time_on, "00");
4.    }
5.  }
```

来补全后面的两个 0。

```
1.  最后 return 0; // 返回 0（成功）
```

（3）再定义一个读入 csv 文件中数据的函数：

初始化一些参数

```
1.  int read_csv(FILE *fp, record *r) {
2.    char line[256]; // 用来存储每行的内容
3.    char *p; // 用来指向每个字段的位置
4.
5.    // 初始化结构体中的所有字段为空字符串
6.    r->qso_date[0] = '\0';
```

```
    r->time_on[0] = '\0';
    r->freq[0] = '\0';
    r->mode[0] = '\0';
    r->call[0] = '\0';
    r->rst_rcvd[0] = '\0';
    r->rst_sent[0] = '\0';
```

读取一行内容，如果到达文件末尾，返回-1（失败）

```
if (fgets(line, sizeof(line), fp) == NULL) return -1;
```

由于 csv 格式中一组数据都在一行，所以用 fgets 即可。

在每行内容中按照逗号分隔出我们需要的字段，并把它们复制到结构体中（以 qso_date 数据为例）：

```
p = strtok(line, ","); // 用 strtok 函数按照逗号分隔字符串，返回第一个字段的位置
if (p != NULL && strlen(p) == 8) { // 如果找到了，并且长度是 8
  strcpy(r->qso_date, p); // 把这个字段复制到结构体中的 qso_date 字段
}
```

注意 tiem_on 数据类型中要补 0：

```
if (strlen(p) == 4) { // 如果时间只有 4 位，表示没有秒数，需要在后面补两位 00
  strcat(r->time_on, "00");
}
```

最后要

```
return 0; // 返回 0（成功）
```

（4）接下来定义一个判断相同的函数：

```
int compare(record *r1, record *r2) {
  return strcmp(r1->qso_date, r2->qso_date) == 0 && strcmp(r1->time_on, r2->time_on)
== 0;
}
```

（5）最后定义一个函数，用来导入 ADIF 或 CSV 文件，根据文件名后缀来决定导入的文件类型，所导入的数据用以更新

自己的二进制数据文件，导入数据时，不同的数据作为新数据加入，相同的数据则用以更新已有的数据：

首先初始化一些参数：

```
int import(char *file_name) {
    FILE *fp_in;
    FILE *fp_out;
    record r_in;
    record r_out;
    int found;

    int file_type;
```

随后用变量 file_type 并调用函数 end_with 来储存文件类型：

```
if (ends_with(file_name, ".adi")) {
    file_type = 1;
} else if (ends_with(file_name, ".csv")) {
    file_type = 2;
} else {
    printf("不支持的文件类型%s\n", file_name);
    return -1;
}
```

用附加方式来打开内容文件与写入文件：

```
fp_in = fopen(file_name, "a+");
if (fp_in == NULL) {
    printf("无法打开文件%s\n", file_name);
    return -1;
}

fp_out = fopen(data_file, "a+");
if (fp_out == NULL) {
```

```
    printf("无法打开文件%s\n", data_file);
    return -1;
}
```

其中：

```
char *data_file = "data.bin";
```

如果内容文件是 adif 格式，需要一些代码来省略<eoh>之前的内容：

```
if(file_type == 1){
    int begin = 0;
    while(!begin){
        while((ch = fgetc(fp_in)) != '<' && ch != EOF);
        begin = 1;
        for(int i = 0;i < 5;i ++){
            ch=(ch >= 'a'&&ch <= 'z')?ch - 'a' + 'A':ch;
            if(ch!= EOH[i]){
                begin = 0;
                break;
            }
            ch=fgetc(fp_in);
        }
    }
}
```

其中实现宏定义了

```
#define EOH "<EOH>"
```

随后调用前面两个 read 函数读出内容并写入二进制文件：

```
while (tag) {
    // 用一个 switch 语句代替 if-else 语句，提高可读性
    switch (file_type) {
```

```
4.      case 1:
5.        if (read_adif(fp_in, &r_in) == -1) tag = 0;
6.        break;
7.      case 2:
8.        if (read_csv(fp_in, &r_in) == -1) tag = 0;
9.        break;
10.     default:
11.       break;
12.   }
13.   if(tag==0)break;
14.   found = 0;
15.   fseek(fp_out, 0, SEEK_SET);
16.   while (fread(&r_out, sizeof(record), 1, fp_out) == 1) {
17.     if (compare(&r_in, &r_out)) {
18.       found = 1;
19.       fseek(fp_out, -(long)sizeof(record), SEEK_CUR);
20.       fwrite(&r_in, sizeof(record), 1, fp_out);
21.       break;
22.     }
23.   }
24.
25.   if (!found) {
26.     fseek(fp_out, 0, SEEK_END);
27.     fwrite(&r_in, sizeof(record), 1, fp_out);
28.     }
29. }
30.
31. fclose(fp_in);
32. fclose(fp_out);
33.
34. return 0;
35. }
```

2．-o <file name>：导出 ADIF 或 CSV 文件，根据文件名后缀来决定导出的文件类型

（1）定义一个函数，用来把结构体中的一条数据写入 ADIF 文件中

```
（2）int write_adif(FILE *fp, record *r) {
（3）    // 把结构体中的每个字段按照 ADIF 的格式写入文件中
（4）    // 格式为<字段名:长度>字段值
（5）    // 每行以换行符结束
（6）
（7）    // 写入 qso_date 字段
（8）    fprintf(fp, "<QSO_DATE:8>%s", r->qso_date);
（9）
（10）    // 写入 time_on 字段，去掉末尾的两位秒数
（11）    fprintf(fp, "<TIME_ON:4>%s", r->time_on);
（12）
（13）    // 写入 freq 字段
（14）    fprintf(fp, "<FREQ:%d>%s", strlen(r->freq), r->freq);
（15）
（16）    // 写入 mode 字段
（17）    fprintf(fp, "<MODE:%d>%s", strlen(r->mode), r->mode);
（18）
（19）    // 写入 call 字段
（20）    fprintf(fp, "<CALL:%d>%s", strlen(r->call), r->call);
（21）
（22）    // 写入 rst_rcvd 字段
（23）    fprintf(fp, "<RST_RCVD:%d>%s", strlen(r->rst_rcvd), r->rst_rcvd);
（24）
```

```
（25）    // 写入 rst_sent 字段
（26）    fprintf(fp, "<RST_SENT:%d>%s", strlen(r->rst_sent), r->rst_sent);
（27）
（28）    fprintf(fp,"<EOR>\n");
（29）
（30）    return 0; // 返回 0（成功）
（31）}
```

（2）定义一个函数，用来把结构体中的一条数据写入 CSV 文件中

```
（3）int write_csv(FILE *fp, record *r) {
（4）    // 把结构体中的每个字段按照 CSV 的格式写入文件中
（5）    // 格式为字段值之间用逗号分隔，每行以换行符结束
（6）
（7）    // 写入 qso_date 字段
（8）    fprintf(fp, "%s,", r->qso_date);
（9）
（10）    // 写入 time_on 字段，去掉末尾的两位秒数
（11）    fprintf(fp, "%s,", r->time_on);
（12）
（13）    // 写入 freq 字段
（14）    fprintf(fp, "%s,", r->freq);
（15）
（16）    // 写入 mode 字段
（17）    fprintf(fp, "%s,", r->mode);
（18）
（19）    // 写入 call 字段```
（20）    fprintf(fp, "%s,", r->call);
（21）
（22）    // 写入 rst_rcvd 字段
（23）    fprintf(fp, "%s,", r->rst_rcvd);
（24）
```

```
（25）    // 写入 rst_sent 字段
（26）    fprintf(fp, "%s\n", r->rst_sent);
（27）
（28）    return 0; // 返回 0（成功）
（29）}
```

（3）定义一个函数，用来导出 ADIF 或 CSV 文件，根据文件名后缀来决定导出的文件类型

```
（4）int export(char *file_name) {
（5）    FILE *fp_in; // 用来打开自己的二进制数据文件
（6）    FILE *fp_out; // 用来打开导出的文件
（7）    record r; // 用来存储从自己的二进制数据文件中读取的一条数据
（8）
（9）    // 根据文件名后缀判断导出的文件类型，并以只写模式打开
（10）    if (ends_with(file_name, ".adi")) { // 如果是 ADIF 文件
（11）        fp_out = fopen(file_name, "w"); // 以文本模式打开
（12）        if (fp_out == NULL) { // 如果打开失败，返回-1（失败）
（13）            printf("无法打开文件%s\n", file_name);
（14）            return -1;
（15）        }
（16）    } else if (ends_with(file_name, ".csv")) { // 如果是 CSV 文件
（17）        fp_out = fopen(file_name, "w"); // 以文本模式打开
（18）        if (fp_out == NULL) { // 如果打开失败，返回-1（失败）
（19）            printf("无法打开文件%s\n", file_name);
（20）            return -1;
（21）        }
（22）    } else { // 如果不是 ADIF 或 CSV 文件，返回-1（失败）
（23）        printf("不支持的文件类型%s\n", file_name);
（24）        return -1;
（25）    }
（26）
（27）    // 以附加模式打开自己的二进制数据文件
```

```
（28）    fp_in = fopen(data_file, "rb"); // 以二进制模式打开
（29）    if (fp_in == NULL) { // 如果打开失败，返回-1（失败）
（30）        printf("无法打开文件%s\n", data_file);
（31）        return -1;
（32）    }
（33）
（34）    // 从自己的二进制数据文件中循环读取每条数据，直到文件末尾
（35）    if (ends_with(file_name, ".adi"))fprintf(fp_out,"<EOH>\n");
（36）    else if (ends_with(file_name, ".csv"))fprintf(fp_out,"QSO_DATE,TIME_ON,FREQ,MODE,CALL,RST_R
CVD,RST_SENT\n");
（37）    while (fread(&r, sizeof(record), 1, fp_in) == 1) { // 如果成功读取了一个结构体的内容
（38）        // 根据文件名后缀判断导出的文件类型，并调用相应的函数来写入一条数据
（39）        if (ends_with(file_name, ".adi")) { // 如果是 ADIF 文件
（40）            write_adif(fp_out, &r); // 调用 write_adif 函数
（41）        } else if (ends_with(file_name, ".csv")) { // 如果是 CSV 文件
（42）            write_csv(fp_out, &r); // 调用 write_csv 函数
（43）        }
（44）    }
（45）
（46）    // 关闭打开的文件
（47）    fclose(fp_in);
（48）    fclose(fp_out);
（49）
（50）    return 0; // 返回 0（成功）
（51）}
```

-s <call>：按照 call 来搜索出有这个 call 的记录，直接输出为 CSV 格式

定义一个函数，用来按照 call 来搜索出有这个 call 的记录，直接输出为 CSV 格式

```
1.  int search(char *call) {
2.      FILE *fp; // 用来打开自己的二进制数据文件
3.      record r; // 用来存储从自己的二进制数据文件中读取的一条数据
4.
```

```
    // 以附加模式打开自己的二进制数据文件
    fp = fopen(data_file, "rb"); // 以二进制模式打开```
    if (fp == NULL) { // 如果打开失败，返回-1（失败）
        printf("无法打开文件%s\n", data_file);
        return -1;
    }

    // 从自己的二进制数据文件中循环读取每条数据，直到文件末尾
    while (fread(&r, sizeof(record), 1, fp) == 1) { // 如果成功读取了一个结构体的内容
        // 如果结构体中的 call 字段和给定的 call 相同，就把这条数据以 CSV 格式输出
        if (strcmp(r.call, call) == 0) {
            write_csv(stdout, &r); // 调用 write_csv 函数，把 stdout 作为参数，表示输出到标准输出
        }
    }

    // 关闭打开的文件
    fclose(fp);

    return 0; // 返回 0（成功）
}
```

-l <start time> <end time>：时间按照 YYYYMMDDhhmmss 表示，直接输出在这个时间段内的全部记录，格式为 CSV

定义一个函数，用来按照时间段来搜索出在这个时间段内的全部记录，直接输出为 CSV 格式

时间按照 YYYYMMDDhhmmss 表示：

```
int list(char *start_time, char *end_time) {
    FILE *fp; // 用来打开自己的二进制数据文件
    record r; // 用来存储从自己的二进制数据文件中读取的一条数据
    char datetime[15]; // 用来存储每条数据的日期和时间

    fp = fopen(data_file, "rb"); // 以二进制模式打开
```

```
    if (fp == NULL) { // 如果打开失败，返回-1（失败）
        printf("无法打开文件%s\n", data_file);
        return -1;
    }

    // 从自己的二进制数据文件中循环读取每条数据，直到文件末尾
    while (fread(&r, sizeof(record), 1, fp) == 1) { // 如果成功读取了一个结构体的内容
        // 把结构体中的 qso_date 和 time_on 字段拼接成一个字符串，表示日期和时间
        strcpy(datetime, r.qso_date);
        strcat(datetime, r.time_on);

        // 如果日期和时间在给定的时间段内，就把这条数据以 CSV 格式输出
        if (strcmp(datetime, start_time) >= 0 && strcmp(datetime, end_time) <= 0) {
            write_csv(stdout, &r); // 调用 write_csv 函数，把 stdout 作为参数，表示输出到标准输出
        }
    }

    // 关闭打开的文件
    fclose(fp);

    return 0; // 返回 0（成功）
}
```

以上功能完成后编写 main 函数：

```
nt main(int argc,char *argv[]) {
    if(argc==1)printf("请输入命令行命令");
    // 如果给出了-i 参数，就调用 import 函数来导入文件
    else if (strcmp(argv[1], "-i") == 0) {
        // 如果没有给出文件名，就输出提示信息，并返回-1（失败）
        // 调用 import 函数，并返回结果
        import(argv[2]);
```

```
    }

    // 如果给出了-o 参数，就调用 export 函数来导出文件
    else if (strcmp(argv[1], "-o") == 0) {
        // 如果没有给出文件名，就输出提示信息，并返回-1（失败）
        // 调用 export 函数，并返回结果
        export(argv[2]);
    }

    // 如果给出了-s 参数，就调用 search 函数来搜索记录
    else if (strcmp(argv[1], "-s") == 0) {
        // 如果没有给出 call，就输出提示信息，并返回-1（失败）
        // 调用 search 函数，并返回结果
        search(argv[2]);
    }

    // 如果给出了-l 参数，就调用 list 函数来搜索记录
    else if (strcmp(argv[1], "-l") == 0) {
        // 如果没有给出 start time 和 end time，就输出提示信息，并返回-1（失败）
        // 调用 list 函数，并返回结果
        list(argv[2], argv[3]);
    }

    // 如果给出了其他参数，就输出提示信息，并返回-1（失败）
    else printf("不支持的参数%s\n", argv[1]);
    return -1;
}
```

四、实验体会与心得：

在实现代码的过程中，我遇到了很多困难。首先，我对命令行参数的使用一无所知，因此在编写代码之前，我会在各个

论坛上寻找资料，了解命令行参数的使用方法。此外，我对于文件的读写操作十分生疏，特别是在处理二进制文件时，我常常需要在网上查找函数原型、使用方法等。这些经历使我坚定了自主学习的能力。

在遇到一些难以描述的问题时，我也会寻求同学的帮助。我的成功完成任务与同学们的协作密切相关，同时，在调试过程中，我更深入地掌握了 VSCode 的调试功能与变量监控方法。

从我最初看到这个任务时的恐惧和困惑，到编写代码时思路的逐渐清晰，再到最终完成时的欢呼雀跃，这是对我的一次全新体验。这次作业不仅对我所学知识进行了全面整合，还拓宽了我的知识面，是难忘的挑战，特别是作为我第一次编写超过 500 行的工程。