

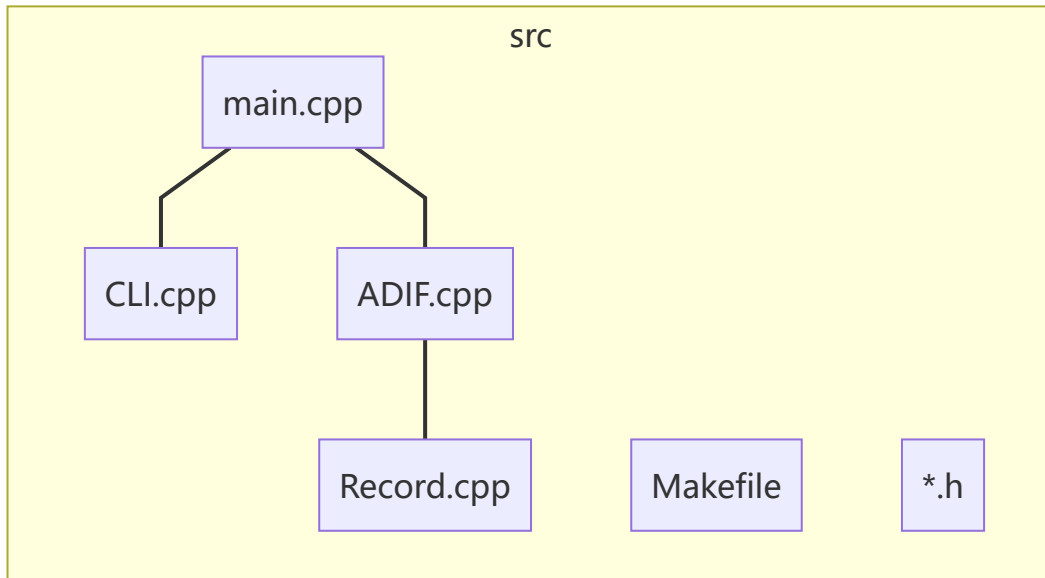
Project 1: ADIF

李秋宇 3220103373

Project 1: ADIF

- 文件结构
- 实现
 - Record类
 - 成员变量
 - 类方法
 - ADIF类
 - 成员变量
 - 类方法
 - 文件解析
 - 记录搜索
 - 记录修改
 - 其它功能
 - CLI类
 - 成员变量
 - 类方法
 - 主程序
- 撰写说明
- 测试
- 源代码
 - main.cpp
 - CLI.h
 - CLI.cpp
 - ADIF.h
 - ADIF.cpp
 - Record.h
 - Record.cpp
 - src/Makefile
 - Makefile
 - main.sh

文件结构



文件架构及组织方式如上，主程序为 `main.cpp`

实现

Record类

Record类是最底层的类，一个 `Record` 的对象代表一条ADIF记录

成员变量

```

1 class Record {
2     std::map<std::string, std::string> fields; // Fields and values.
3     int fieldSize; // Total size of fields.
4     ...
5 }

```

其中使用一个 `std::map<std::string, std::string>` 用来存储ADIF记录的键值对，是整个记录类的核心，再用一个 `fieldSize` 来记录总共存放了多少个键值对

所有的值都以字符串的形式存储

类方法

类方法主要有常规的构造函数，获取键值和设置键值，打印单条记录

设置了 `getPrimaryKey()` 函数用于获取记录的主键，由于ADIF记录的主键是 `<QSO_DATE, TIME_ON>` 一起决定，因此主键只是简单地把两个值进行字符串加法

这里最麻烦的操作就是初始化，对于ADIF格式来说，在传参给本类时需要把所有的字段的键值对转化成 `fieldName:fieldValue\n` 的形式，然后在构造函数对这个传入的字符串进行解析，拆成 `fieldName` 和 `fieldValue` 然后调用函数 `setField(fieldName, fieldValue)`

ADIF类

ADIF类是整个项目的核心部分，主要负责实现题目的功能：

1. 解析ADIF文件并存储数据
2. 多文件合并数据
3. 列出所有记录
4. 转化成 .csv 和 .json 格式
5. 记录搜索
6. 记录修改
7. 记录删除

成员变量

```
1 class ADIF {
2     std::map<std::string, Record> records; // Store all records.
3     std::vector<std::string> fieldNames; // Store all field names.
4     ...
5 };
```

使用一个 `std::vector<std::string>` 存储所有的字段名，用于转化为 .csv 格式输出时使用

使用一个 `std::map<std::string, Record>` 存储所有的记录，以 `<primaryKey, Record>` 形式存储

根据主键的索引建立，方便定位到具体记录

类方法

类方法主要是不同功能对应一条函数

文件解析

最麻烦的也是文件的解析，使用 `parse(filename)` 函数打开一个ADIF文件并解析，这里采用对文件进行逐字符读取，并利用 `<string>` 库的相关字符串操作函数对每个标签进行处理

采用了异常处理，如果

- 文件扩展名不对，不是 .adi
- 文件无法打开或不存在
- 错误的ADIF标签，如题目所示

并且采取了冲突检查的方式，如果出现两条记录有一样的主键，会提示是否覆盖

记录搜索

这里支持多条件的搜索，对于每个条件都采用一个 `std::pair<std::string, std::string>` 的方式存储要查询的字段及其值 `<fieldName, fieldvalue>`，而且支持模糊查询，因为所有的值都是字符串形式存储，所以只需要调用 `substr` 方法就可以实现

用一个 `std::vector<std::pair<std::string, std::string>>` 来存储多条件，然后传给 `searchRecords(searchConditions)` 进行查询

查询结果返回一个满足条件的记录的集合，并且控制台打印输出对应的记录信息

记录修改

记录修改与记录搜索一致，也是支持多字段修改，传参额外需要一个被修改字段的原始 ID

为什么是原始 ID 呢？因为如果修改了字段是 QSO_DATE 或 TIME_ON 的其中一个或者两个，会导致这条记录的主键变化，这里就是 modifyRecord 函数的处理机制，如果修改后新的主键与已经存在的记录冲突，也会询问是否覆盖

其它功能

- 打印所有记录，只需要遍历记录字典并调用每条记录的打印方法即可
- 导出为 .csv 和 .json，只需要按照对应文件格式将记录转化即可
- 删除记录，只需要找到对应记录并擦除即可

CLI类

最开始设计完上述两个类觉得任务已经基本完成了，后来又着手设计了这个交互的类

主要就是一些字符串的处理

成员变量

```

1  class CLI {
2  public:
3      /**
4       * @brief The ADIF_COMMAND enum defines all supported commands that can
5       * be entered by the user.
6       */
7      enum ADIF_COMMAND {
8          PARSE_ADIF,
9          PRINT_ADIF,
10         EXPORT_ADIF_TO_CSV,
11         EXPORT_ADIF_TO_JSON,
12         SEARCH_ADIF_RECORDS,
13         MODIFY_ADIF_RECORD,
14         DELETE_ADIF_RECORD,
15         EXIT,
16         HELP,
17         INVALID_COMMAND
18     };
19     ...
20 private:
21     ADIF_COMMAND command; // store the command.
22     std::string name; // store the filename or key for search or modify
23     commands.
24     std::vector<std::pair<std::string, std::string>> searchConditions; //
25     store the search conditions.
26 };

```

这里使用了一个枚举类型表示所有的指令，并用成员变量 command 表示

其它成员变量用于调用ADIF类时传参使用

类方法

类方法都是字符串的相关操作，主要是读取用户输入的 `readCommand()`，将用户输入的字符串识别为对应指令，并读取对应需要的参数

错误指令和不支持的指令都有专门的处理方法

为了追求完美的UI，作者将这个CLI类打造成为像bash一样的模式，还设置有help menu，欢迎词和欢送词

主程序

由于之前的类已经很好完成所需功能的封装，因此主程序只需要调用CLI类和ADIF类即可完成操作

撰写说明

这个大程内容确实不少，认真做花了不少时间，虽然难度不大但是要处理的细节很多

个人认为本人所编写的程序规范性较好，也有符合规范的注释说明，也完成了很好的类设计与封装，用户交互也做的很好

写起来非常爽，很沉浸式

不足之处也有，比如没有完成中文字符的识别

查阅相关资料了解到中文字符识别需要用到 `<locale>` 库，以及 `wcin` 等等，就没有做

以及查找的效率可能偏低，是顺序遍历进行的查找，对于小文件来说无伤大雅，但是大文件可能就遭殃一个最初的思路是建立一个索引字典，嵌套的字典

```
1 | std::map<std::string, std::map<std::string, Record>> index;
```

形式为 `fieldName:[<fieldvalue1, Record1>, <fieldvalue2, Record2>, ...]` 可以快速查找，当然这也带来了内存的开销增加和实现上的繁琐，所以没有实现，仅限于一个想法

测试

测试均使用PTA提供的3个ADIF文件，均已存放在 `test/` 目录下且通过测试

源代码

main.cpp

```
1 | #include <iostream>
2 | #include "ADIF.h"
3 | #include "CLI.h"
4 |
5 | int main()
6 | {
7 |     CLI command;
8 |     ADIF adif;
9 |     command.welcomeMessage();
```

```

10     while (true) {
11         command.readCommand();
12         try {
13             switch(command.getCommand()) {
14                 case CLI::ADIF_COMMAND::PARSE_ADIF:
15                     adif.parse(command.getName());
16                     break;
17                 case CLI::ADIF_COMMAND::PRINT_ADIF:
18                     adif.print();
19                     break;
20                 case CLI::ADIF_COMMAND::EXPORT_ADIF_TO_CSV:
21                     adif.exportToCSV(command.getName());
22                     break;
23                 case CLI::ADIF_COMMAND::EXPORT_ADIF_TO_JSON:
24                     adif.exportToJSON(command.getName());
25                     break;
26                 case CLI::ADIF_COMMAND::SEARCH_ADIF_RECORDS:
27                     adif.searchRecords(command.getSearchConditions());
28                     break;
29                 case CLI::ADIF_COMMAND::MODIFY_ADIF_RECORD:
30                     adif.modifyRecord(command.getName(),
command.getSearchConditions());
31                     break;
32                 case CLI::ADIF_COMMAND::DELETE_ADIF_RECORD:
33                     adif.deleteRecord(command.getName());
34                     break;
35                 case CLI::ADIF_COMMAND::EXIT:
36                     command.goodbyeMessage();
37                     return 0;
38                 case CLI::ADIF_COMMAND::HELP:
39                     command.printHelp();
40                     break;
41                 default:
42                     command.printInvalidCommand();
43                     break;
44             }
45         } catch (const std::runtime_error e) {
46             std::cerr << e.what() << std::endl;
47         }
48     }
49 }

```

CLI.h

```

1  #ifndef _CLI_H
2  #define _CLI_H
3
4  #include <iostream>
5  #include <string>
6  #include <vector>
7
8  /**
9   * @brief The CLI class is used to read and parse user input commands.
10  */
11  class CLI {

```

```

12 public:
13     /**
14      * @brief The ADIF_COMMAND enum defines all supported commands that can
15      * be entered by the user.
16      */
17     enum ADIF_COMMAND {
18         PARSE_ADIF,
19         PRINT_ADIF,
20         EXPORT_ADIF_TO_CSV,
21         EXPORT_ADIF_TO_JSON,
22         SEARCH_ADIF_RECORDS,
23         MODIFY_ADIF_RECORD,
24         DELETE_ADIF_RECORD,
25         EXIT,
26         HELP,
27         INVALID_COMMAND
28     };
29     /**
30      * Constructor.
31      */
32     CLI();
33
34     /**
35      * Read and parse user input commands.
36      */
37     void readCommand();
38
39     /**
40      * Get the command entered by the user.
41      * @return The command entered by the user.
42      */
43     ADIF_COMMAND getCommand() const;
44
45     /**
46      * Get the filename or key entered by the user for search or modify
47      * commands.
48      * @return The filename or key entered by the user.
49      */
50     std::string getName() const;
51
52     /**
53      * Get the vector of `<fieldName, fieldValue>` pair to search or modify.
54      * @return The vector of `<fieldName, fieldValue>` pair to search or
55      * modify.
56      */
57     std::vector<std::pair<std::string, std::string>> getSearchConditions()
58     const;
59
60     /**
61      * Print the error message for invalid command entered by the user.
62      */
63     void printInvalidCommand(void) const;
64
65     /**

```

```

63     * Get the help mannual for user.
64     */
65     void printHelp(void) const;
66
67     /**
68     * Print welcome message.
69     */
70     void welcomeMessage(void);
71
72     /**
73     * Print goodbye message.
74     */
75     void goodbyeMessage(void);
76
77 private:
78     ADIF_COMMAND command; // Store the command.
79     std::string name; // Store the filename or key for search or modify
80     commands.
81     std::vector <std::pair<std::string, std::string>> searchConditions; //
82     Store the search conditions.
83 };
84 #endif

```

CLI.cpp

```

1  #include <iostream>
2  #include <string>
3  #include "CLI.h"
4
5  CLI::CLI()
6  {
7      command = INVALID_COMMAND;
8      name = "";
9      searchConditions = std::vector<std::pair<std::string, std::string>>();
10 }
11
12 void CLI::readCommand()
13 {
14     std::cout << "user@localhost:~$ ";
15     std::string input;
16     std::getline(std::cin, input);
17     if (input.substr(0, 5) == "parse") {
18         if (input.size() < 7) {
19             command = INVALID_COMMAND;
20         } else {
21             name = input.substr(6);
22             command = PARSE_ADIF;
23         }
24     } else if (input == "print") {
25         command = PRINT_ADIF;
26     } else if (input.substr(0, 12) == "export --csv") {
27         if (input.size() < 14) {
28             command = INVALID_COMMAND;
29         } else {

```



```

30         command = EXPORT_ADIF_TO_CSV;
31         name = input.substr(13);
32     }
33 } else if (input.substr(0, 13) == "export --json") {
34     if (input.size() < 15) {
35         command = INVALID_COMMAND;
36     } else {
37         command = EXPORT_ADIF_TO_JSON;
38         name = input.substr(14);
39     }
40 } else if (input == "search") {
41     searchConditions.clear();
42     command = SEARCH_ADIF_RECORDS;
43     std::cout << "[ADIF] Input search condition(s): " << std::endl;
44     std::string buffer;
45     int num = 1;
46     while (true) {
47         std::cout << "Condition(" << num++ << "): ";
48         getline(std::cin, buffer);
49         if (buffer == "end") {
50             break;
51         }
52         std::string fieldName, fieldValue;
53         int pos = buffer.find(" ");
54         if (pos == std::string::npos) {
55             std::cout << "Invalid input! Please input again." <<
std::endl;
56             num--;
57             continue;
58         }
59         fieldName = buffer.substr(0, pos);
60         fieldValue = buffer.substr(pos + 1);
61         searchConditions.push_back(std::make_pair(fieldName,
fieldValue));
62     }
63 } else if (input.substr(0, 6) == "modify") {
64     if (input.size() < 8) {
65         command = INVALID_COMMAND;
66     } else {
67         name = input.substr(7);
68         std::cout << "[ADIF] Input modify field and value: " <<
std::endl;
69         std::string buffer;
70         command = MODIFY_ADIF_RECORD;
71         int num = 1;
72         while (true) {
73             std::cout << "Pair(" << num++ << "): ";
74             getline(std::cin, buffer);
75             if (buffer == "end") {
76                 break;
77             }
78             std::string fieldName, fieldValue;
79             int pos = buffer.find(" ");
80             if (pos == std::string::npos) {

```

```

81         std::cout << "Invalid input! Please input again." <<
std::endl;
82         num--;
83         continue;
84     }
85     fieldName = buffer.substr(0, pos);
86     fieldValue = buffer.substr(pos + 1);
87     searchConditions.push_back(std::make_pair(fieldName,
fieldvalue));
88     }
89     }
90     } else if (input.substr(0, 6) == "delete") {
91         if (input.size() < 8) {
92             command = INVALID_COMMAND;
93         } else {
94             command = DELETE_ADIF_RECORD;
95             name = input.substr(7);
96         }
97     } else if (input == "exit") {
98         command = EXIT;
99     } else if (input == "help") {
100         command = HELP;
101     } else {
102         command = INVALID_COMMAND;
103     }
104     return ;
105 }
106
107 CLI::ADIF_COMMAND CLI::getCommand() const
108 {
109     return command;
110 }
111
112 std::string CLI::getName() const
113 {
114     return name;
115 }
116
117 std::vector<std::pair<std::string, std::string>> CLI::getSearchConditions()
const
118 {
119     return searchConditions;
120 }
121
122 void CLI::printInvalidCommand(void) const
123 {
124     std::cout << "[ADIF] Invalid command! Type 'help' for help." <<
std::endl;
125 }
126
127 void CLI::printHelp(void) const
128 {
129     std::cout << "#####" << std::endl;
130     std::cout << "##" << std::endl;
131     std::cout << "##          ADIF Parser Help Menu          ##" << std::endl;

```

```

132     std::cout << "##"                                ##" << std::endl;
133     std::cout << "## 1. parse [filename]              ##" << std::endl;
134     std::cout << "##     Parse ADIF file and store    ##" << std::endl;
135     std::cout << "##     it in memory.                ##" << std::endl;
136     std::cout << "##"                                ##" << std::endl;
137     std::cout << "## 2. print                          ##" << std::endl;
138     std::cout << "##     Print all ADIF records in    ##" << std::endl;
139     std::cout << "##     console.                    ##" << std::endl;
140     std::cout << "##"                                ##" << std::endl;
141     std::cout << "## 3. export [--csv|--json] [filename] ##" << std::endl;
142     std::cout << "##     Export ADIF records to CSV    ##" << std::endl;
143     std::cout << "##     or JSON file.                ##" << std::endl;
144     std::cout << "##"                                ##" << std::endl;
145     std::cout << "## 4. search                          ##" << std::endl;
146     std::cout << "##     Search ADIF records by      ##" << std::endl;
147     std::cout << "##     specific conditions in format    ##" << std::endl;
148     std::cout << "##     <fieldName fieldValue>.                ##" << std::endl;
149     std::cout << "##     Type 'End' to start search.            ##" << std::endl;
150     std::cout << "##"                                ##" << std::endl;
151     std::cout << "## 5. modify [recordID]                          ##" << std::endl;
152     std::cout << "##     Modify ADIF record by specific    ##" << std::endl;
153     std::cout << "##     field and value in format                ##" << std::endl;
154     std::cout << "##     <fieldName fieldValue>.                ##" << std::endl;
155     std::cout << "##     Type 'End' to start modify.            ##" << std::endl;
156     std::cout << "##"                                ##" << std::endl;
157     std::cout << "## 6. delete [recordID]                          ##" << std::endl;
158     std::cout << "##     Delete ADIF record by its ID.          ##" << std::endl;
159     std::cout << "##"                                ##" << std::endl;
160     std::cout << "## 7. exit                          ##" << std::endl;
161     std::cout << "##     Exit the program.                        ##" << std::endl;
162     std::cout << "##"                                ##" << std::endl;
163     std::cout << "## 8. help                          ##" << std::endl;
164     std::cout << "##     Show this help menu.                    ##" << std::endl;
165     std::cout << "##"                                ##" << std::endl;
166     std::cout << "##-----EXAMPLES-----" ##" << std::endl;
167     std::cout << "##"                                ##" << std::endl;
168     std::cout << "## 1. parse myadif.adif                          ##" << std::endl;
169     std::cout << "## 2. print                                      ##" << std::endl;
170     std::cout << "## 3. export --csv output.csv                    ##" << std::endl;
171     std::cout << "## 4. search 20240531120503                      ##" << std::endl;
172     std::cout << "##     QSO_DATE 20240601                        ##" << std::endl;
173     std::cout << "##     CALL 111032564533                        ##" << std::endl;
174     std::cout << "##     End                                       ##" << std::endl;
175     std::cout << "## 5. modify 20240531010101                      ##" << std::endl;
176     std::cout << "##     QSO_DATE 20240601                        ##" << std::endl;
177     std::cout << "##     CALL 111032564533                        ##" << std::endl;
178     std::cout << "##     End                                       ##" << std::endl;
179     std::cout << "## 6. delete 20240531010203                      ##" << std::endl;
180     std::cout << "## 7. exit                                      ##" << std::endl;
181     std::cout << "## 8. help                                      ##" << std::endl;
182     std::cout << "##"                                ##" << std::endl;
183     std::cout << "#####" << std::endl;
184     return ;
185 }
186

```

```

187 void CLI::welcomeMessage(void)
188 {
189     std::cout << "#####" << std::endl;
190     std::cout << "##" << std::endl;
191     std::cout << "##    Welcome to my ADIF parser!    ##" << std::endl;
192     std::cout << "##    Written by lqy 3220103373    ##" << std::endl;
193     std::cout << "##    Type 'help' for help menu    ##" << std::endl;
194     std::cout << "##" << std::endl;
195     std::cout << "#####" << std::endl;
196     return ;
197 }
198
199 void CLI::goodbyeMessage(void)
200 {
201     std::cout << "#####" << std::endl;
202     std::cout << "##" << std::endl;
203     std::cout << "##           Goodbye!!!           ##" << std::endl;
204     std::cout << "##" << std::endl;
205     std::cout << "#####" << std::endl;
206     return ;
207 }

```

ADIF.h

```

1  #ifndef _ADIF_H
2  #define _ADIF_H
3
4  #include <iostream>
5  #include <vector>
6  #include <map>
7  #include "Record.h"
8
9  /**
10   * ADIF class to parse ADIF file and store data in main memory.
11   * @exception Throw `std::runtime_error` when error happens.
12   * @note If the ADIF file contains multiple ADIF data, they will be stored
13   together.
14   */
15 class ADIF {
16     std::map<std::string, Record> records; // Store all records.
17     std::vector<std::string> fieldNames; // Store all field names.
18 public:
19     /**
20      * Constructor to initialize ADIF object.
21      */
22     ADIF();
23
24     /**
25      * Get all records
26      * @return A vector of Record objects.
27      */
28     std::vector<Record> getRecords() const;
29
30     /**
31      * Parse ADIF file and store data in main memory.

```

```

31     * @param filename ADIF file name to open.
32     * @exception `std::runtime_error`.
33     * @note Data will be combined if there already exists data in the ADIF
object.
34     */
35     void parse(std::string filename);
36
37     /**
38     * Print the ADIF data.
39     * @note The header will be printed if it exists.
40     */
41     void print(void) const;
42
43     /**
44     * Export the ADIF data to CSV file.
45     * @param filename CSV file name to export.
46     * @exception `std::runtime_error`.
47     * @note ".csv" suffix will be added if not present in the filename.
48     */
49     void exportToCSV(std::string filename) const;
50
51     /**
52     * Export the ADIF data to JSON file.
53     * @param filename JSON file name to export.
54     * @exception `std::runtime_error`.
55     * @note ".json" suffix will be added if not present in the filename.
56     */
57     void exportToJSON(std::string filename) const;
58
59     /**
60     * Search ADIF record(s) with given field value.
61     * @param searchConditions A vector of search conditions with pairs of
`<fieldName, fieldValue>`.
62     * @return A vector of Record objects.
63     */
64     std::vector<Record> searchRecords(std::vector<std::pair<std::string,
std::string>> searchConditions) const;
65
66     /**
67     * Modify one ADIF record.
68     * @param key The primary key of the record to modify.
69     * @param valuePairs A vector of pairs of `<fieldName, fieldValue>` to
modify.
70     * @return True if the record is modified successfully, false otherwise.
71     * @note If the record not exists it will be false.
72     */
73     bool modifyRecord(std::string key, std::vector<std::pair<std::string,
std::string>> valuePairs);
74
75     /**
76     * Delete one ADIF record.
77     * @param key The primary key of the record to delete.
78     * @return True if the record is deleted successfully, false otherwise.
79     * @note If the record not exists it will be false.
80     */

```

```

81     bool deleteRecord(std::string key);
82 };
83
84 #endif

```

ADIF.cpp

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <fstream>
5  #include <algorithm>
6  #include "ADIF.h"
7  #include "Record.h"
8
9  #define EndOfHeader "EOH"
10 #define EndOfRecord "EOR"
11
12 ADIF::ADIF()
13 {
14     // Initialize variables.
15     records = std::map<std::string, Record>();
16     fieldNames = std::vector<std::string>();
17 }
18
19 std::vector<Record> ADIF::getRecords() const
20 {
21     std::vector<Record> result;
22     for (const auto& record : records) {
23         result.push_back(record.second);
24     }
25     return result;
26 }
27
28 void ADIF::parse(std::string filename)
29 {
30     if (filename.find(".adi") == std::string::npos) {
31         throw std::runtime_error("[ERROR] Invalid file format: " +
32 filename);
33     }
34     std::cout << "[ADIF] Start parsing ADIF file: " << filename << "..." <<
35 std::endl;
36     // Open ADIF file for reading.
37     std::ifstream file;
38     file.open(filename, std::ios::in);
39     if (!file.is_open()) {
40         throw std::runtime_error("[ERROR] Failed to open file: " +
41 filename);
42     }
43     // Read ADIF file into buffer.
44     std::string buffer = "";
45     while (!file.eof()) {
46         char tmp = file.get();
47         if (tmp == '\n')
48             continue;

```

```

46     buffer += tmp;
47 }
48 // Close ADIF file.
49 file.close();
50 // Get records.
51 int pos1 = 0, pos2 = 0;
52 std::string buf = ""; // Buffer to store data in buffer.
53 while (1) {
54     pos1 = buffer.find('<', pos2);
55     if (pos1 == std::string::npos) // No more tags.
56         break;
57     pos2 = buffer.find('>', pos1);
58     if (pos2 == std::string::npos) // Invalid tag.
59         break;
60     std::string tag = buffer.substr(pos1 + 1, pos2 - pos1 - 1);
61     int sep_pos = tag.find(':');
62     if (sep_pos != std::string::npos) { // B tag.
63         int num = std::stoi(tag.substr(sep_pos + 1));
64         std::string field = tag.substr(0, sep_pos);
65         transform(field.begin(), field.end(), field.begin(),
66 ::toupper); // Convert field name to uppercase.
67         if (std::find(fieldNames.begin(), fieldNames.end(), field) ==
68 fieldNames.end()) {
69             fieldNames.push_back(field);
70         }
71         buf += field + ':' + buffer.substr(pos2 + 1, num) + '\n';
72     } else {
73         transform(tag.begin(), tag.end(), tag.begin(), ::toupper);
74         // A tag.
75         if (tag == EndOfRecord) {
76             Record record(buf);
77             std::string id = record.getPrimaryKey();
78             // Check if record already exists.
79             if (records.count(id)) {
80                 std::cout << "[ADIF] Record with ID = " << id << "
81 already exists. Overwrite it? (Y/n): ";
82                 std::string c;
83                 getline(std::cin, c);
84                 if (c == "y" || c == "Y" || c == "yes" || c == "Yes" ||
85 c == "YES") {
86                     std::cout << "[ADIF] Overwriting record with ID = "
87 << id << std::endl;
88                     records.at(id) = record;
89                 } else {
90                     std::cout << "[ADIF] Skipping record with ID = " <<
91 id << std::endl;
92                     continue;
93                 }
94             } else {
95                 records.insert(std::pair<std::string, Record>(id,
96 record));
97             }
98         } else if (tag == EndOfHeader) {
99             continue;
100         } else { // Invalid tag.

```

```

94         throw std::runtime_error("[ERROR] Invalid tag: " + tag);
95     }
96 }
97 }
98     std::sort(fieldNames.begin(), fieldNames.end());
99     std::cout << "[ADIF] Successfully parsed ADIF file: " << filename <<
std::endl;
100     return ;
101 }
102
103 void ADIF::print(void) const
104 {
105     std::cout << "[ADIF] Start printing " << records.size() << " ADIF
data..." << std::endl;
106     for (const auto& record : records) {
107         record.second.print();
108     }
109     std::cout << "[ADIF] Printing ADIF data done." << std::endl;
110     return ;
111 }
112
113 void ADIF::exportToCSV(std::string filename) const
114 {
115     std::cout << "[ADIF] Start exporting to CSV file... " << filename <<
"... " << std::endl;
116     std::string content = ""; // Content to write to file.
117     // write header.
118     for (const std::string& field : fieldNames) {
119         content += field + ',';
120     }
121     content.erase(content.end() - 1);
122     content += '\n';
123     // write records.
124     for (const auto& record : records) {
125         for (const std::string& field : fieldNames) {
126             content += record.second.getValue(field) + ',';
127         }
128         content.replace(content.length() - 1, 1, "\n");
129     }
130     content.erase(content.end() - 1);
131     // open file for writing.
132     if (filename.find(".csv") == std::string::npos) {
133         filename += ".csv";
134     }
135     if (filename.find("out/") == std::string::npos) {
136         filename = "out/" + filename;
137     }
138     std::ofstream file;
139     file.open(filename, std::ios::out);
140     if (!file.is_open()) {
141         throw std::runtime_error("[ERROR] Failed to export to file: " +
filename);
142     }
143     file << content;
144     file.close();

```



```

145     std::cout << "[ADIF] Successfully exported to file: " << filename <<
std::endl;
146     return ;
147 }
148
149 void ADIF::exportToJson(std::string filename) const
150 {
151     std::cout << "[ADIF] Start exporting to JSON file... " << filename <<
"... " << std::endl;
152     std::string content = "{\n"; // Content to write to file.
153     content += "\t\"Number of Records\": " + std::to_string(records.size())
+ ",\n";
154     // Write records.
155     content += "\t\"Records Info\": [\n";
156     for (const auto& record : records) {
157         std::vector<std::string> fields = record.second.getFields();
158         content += "\t\t{\n";
159         for (const std::string& field : fields) {
160             content += "\t\t\t\"" + field + "\": \"" +
record.second.getValue(field) + "\",\n";
161         }
162         content.erase(content.end() - 2, content.end());
163         content += "\n\t\t},\n";
164     }
165     content.erase(content.end() - 2, content.end());
166     content += "\n\t]\n"; // End of content.
167     // Open file for writing.
168     if (filename.find(".json") == std::string::npos) {
169         filename += ".json";
170     }
171     if (filename.find("out/") == std::string::npos) {
172         filename = "out/" + filename;
173     }
174     std::ofstream file;
175     file.open(filename, std::ios::out);
176     if (!file.is_open()) {
177         throw std::runtime_error("[ERROR] Failed to export to file: " +
filename);
178     }
179     file << content;
180     file.close();
181     std::cout << "[ADIF] Successfully exported to file: " << filename <<
std::endl;
182     return ;
183 }
184
185 std::vector<Record> ADIF::searchRecords(std::vector<std::pair<std::string,
std::string>> searchConditions) const
186 {
187     std::cout << "[ADIF] Start searching records satisfying search
conditions..." << std::endl;
188     for (auto& condition : searchConditions) {
189         transform(condition.first.begin(), condition.first.end(),
condition.first.begin(), ::toupper);
190     }

```

```

191     std::vector<Record> result;
192     for (const auto& record : records) {
193         bool match = true;
194         for (const auto& condition : searchConditions) {
195             if
196             (record.second.getValue(condition.first).find(condition.second) ==
197             std::string::npos) {
198                 match = false;
199                 break;
200             }
201         }
202         if (match) {
203             result.push_back(record.second);
204         }
205     }
206     std::cout << "[ADIF] Searching records done, found " << result.size()
207     << " records." << std::endl;
208     for (const Record& record : result) {
209         record.print();
210     }
211     return result;
212 }
213
214 bool ADIF::modifyRecord(std::string key, std::vector<std::pair<std::string,
215 std::string>> valuePairs)
216 {
217     if (!records.count(key)) {
218         std::cout << "[ADIF] Record with ID = " << key << " does not exist,
219 no modification is made." << std::endl;
220         return false;
221     }
222     Record record = records.at(key);
223     for (const auto& valuePair : valuePairs) {
224         std::string fieldName = valuePair.first;
225         std::string fieldValue = valuePair.second;
226         transform(fieldName.begin(), fieldName.end(), fieldName.begin(),
227         ::toupper);
228         record.setField(fieldName, fieldValue);
229     }
230     std::string newKey = record.getPrimarykey();
231     // Check if primary key is changed.
232     if (newKey != key) {
233         // Check if new primary key already exists.
234         if (records.count(newKey)) {
235             std::cout << "[ADIF] Modified record with ID = " << newKey << "
236 already exists. Overwrite it? (Y/n): ";
237             std::string c;
238             getline(std::cin, c);
239             if (c == "y" || c == "Y" || c == "yes" || c == "Yes" || c ==
240 "YES") {
241                 std::cout << "[ADIF] Overwriting record with ID = " <<
242 newKey << std::endl;
243                 records.at(newKey) = record;
244                 std::cout << "[ADIF] Modified record with ID = " << key <<
245 " into new ID = " << newKey << " by overwriting the old one." << std::endl;

```

```

236         } else {
237             std::cout << "[ADIF] Skipping to modify record with ID = "
<< key << ", no modification is made." << std::endl;
238         }
239     } else {
240         records.insert(std::pair<std::string, Record>(newKey, record));
241         records.erase(key);
242         std::cout << "[ADIF] Modified record with ID = " << key << "
into new ID = " << newKey << "." << std::endl;
243     }
244 } else {
245     records.at(key) = record;
246     std::cout << "[ADIF] Modified record with ID = " << key << "." <<
std::endl;
247 }
248 return true;
249 }
250
251 bool ADIF::deleteRecord(std::string key)
252 {
253     if (!records.count(key)) {
254         std::cout << "[ADIF] Record with ID = " << key << " does not exist,
no deletion is made." << std::endl;
255         return false;
256     }
257     records.erase(key);
258     std::cout << "[ADIF] Deleted record with ID = " << key << "." <<
std::endl;
259     return true;
260 }

```

Record.h

```

1  #ifndef _RECORD_H
2  #define _RECORD_H
3
4  #include <iostream>
5  #include <vector>
6  #include <map>
7
8  /**
9   * A class to represent a single ADIF record.
10  * @exception `std::runtime_error` if the input string has no primary key.
11  * @note 1. All the values are stored in `string` format.
12  * @note 2. `` is the primary key of the record.
13  * @note 3. All the fields' name are stored in uppercase.
14  */
15  class Record {
16      std::map<std::string, std::string> fields; // Fields and values.
17      int fieldSize; // Total size of fields.
18  public:
19      /**
20       * Constructor.
21       * @param input A string of a ADIF record.

```

```

22     * @note 1. The `input` string should be in one string and each record
seperated by a newline.
23     * @note 2. Each field and value pair should be in format
`fieldName:value` and field name in uppercase.
24     * @note 3. Invalid fields and records should be ignored and not passed
to the constructor.
25     */
26     Record(std::string input);
27
28     /**
29     * Set a field with a value string.
30     * @param field The name of the field in uppercase.
31     * @param value The value of the field.
32     * @note If the field already exists, the value will be updated.
33     */
34     void setField(std::string field, std::string value);
35
36     /**
37     * Get the value of a field.
38     * @param field The name of the field.
39     * @return The value of the field.
40     * @note If the field does not exist, an empty string will be returned.
41     */
42     std::string getValue(std::string field) const;
43
44     /**
45     * Get the field names with valid data of the record.
46     * @return A vector of field names.
47     */
48     std::vector<std::string> getFields(void) const;
49
50     /**
51     * Get the primary key of a record.
52     * @return The primary key of the record in `string` format.
53     */
54     std::string getPrimaryKey(void) const;
55
56     /**
57     * Print the record to the console.
58     * @note Each field and its value are printed in one line.
59     */
60     void print(void) const;
61 };
62
63 #endif

```

Record.cpp

```

1  #include <iostream>
2  #include <string>
3  #include <vector>
4  #include <map>
5  #include "Record.h"
6
7  Record::Record(std::string input)

```

```

8  {
9      fields = std::map<std::string, std::string>();
10     fieldSize = 0;
11     int pos;
12     while (1) {
13         // Find field.
14         pos = input.find(":");
15         if (pos == std::string::npos)
16             break;
17         std::string field = input.substr(0, pos);
18         input = input.substr(pos + 1);
19         // Find value.
20         pos = input.find('\n');
21         std::string value = input.substr(0, pos);
22         input = input.substr(pos + 1);
23         setField(field, value);
24     }
25     // No primary key.
26     if (!(fields.count("QSO_DATE") && fields.count("TIME_ON"))) {
27         throw std::runtime_error("Invalid record: no primary key.");
28     }
29 }
30
31 void Record::setField(std::string field, std::string value)
32 {
33     if (!fields.count(field)) // New field.
34         fieldSize++;
35     fields[field] = value;
36 }
37
38 std::string Record::getValue(std::string field) const
39 {
40     if (fields.count(field))
41         return fields.at(field);
42     else
43         return "";
44 }
45
46 std::vector<std::string> Record::getFields(void) const
47 {
48     std::vector<std::string> result;
49     for (const auto& field: fields)
50         result.push_back(field.first);
51     return result;
52 }
53
54 std::string Record::getPrimaryKey(void) const
55 {
56     return fields.at("QSO_DATE") + fields.at("TIME_ON");
57 }
58
59 void Record::print(void) const
60 {
61     std::cout << "----- Record Info -----" << std::endl;
62     std::cout << "(0) ID: " << getPrimaryKey() << std::endl;

```

```

63     int index = 1;
64     for (const auto& field: fields)
65         std::cout << "(" << index++ << ") " << field.first << ": " <<
field.second << std::endl;
66     std::cout << "----- END -----" << std::endl;
67     return ;
68 }

```

src/Makefile

```

1  CC = g++
2  CFLAGS = -std=c++11 -I.
3  SRC = .
4  OD = ..
5  OUT = $(OD)/out
6  RM = rm -f
7  OBJS = $(SRC)/main.o $(SRC)/ADIF.o $(SRC)/Record.o $(SRC)/CLI.o
8  TARGET = $(OD)/ADIF
9
10 $(TARGET): $(OBJS)
11     $(CC) $(CFLAGS) -o $@ $^ -g
12
13 .cpp.o:
14     $(CC) $(CFLAGS) -c $< -o $@
15
16 clean:
17     $(RM) $(TARGET) $(OBJS) $(OUT)/*

```

Makefile

```

1  SRC = src
2  MAKE = make
3
4  all:
5      cd $(SRC) && $(MAKE)
6
7  clean:
8      cd $(SRC) && $(MAKE) clean

```

main.sh

```

1  #!/usr/bin/fish
2  make clean
3  make
4  clear
5  ./ADIF
6  echo "Run Done!"

```

