南京大学课程设计报告

高级程序设计项目报告（一）

MarkDown 编辑器

学　　号：181840070

院　　系：计算机科学与技术系

姓　　名：葛睿芃

提交日期：2019/10/13

目录

一、概述

1.1 项目简介

高级程序设计第一次课程设计要求制作一个基于控制台的小程序，从给出的三个可选课题中，经过对设计难度的慎重考虑，最终决定选择了~~相对简单的~~MarkDown 编辑器这一选题。

MarkDown 编辑器可以实现从控制台键入文本，保存成 md 文件，并通过对 md 文件的 MarkDown 语法解析，生成 html 格式的文件输出，最终可以在浏览器上正常显示。项目给出了一些典型 MarkDown 语法的规则，并对这些规则做出了要求。

从本质上来看，本实验项目类似于设计一个解释器，对相应的语法规则进行解析，并转化为目标要求的形式。解释器的核心在于字符串处理，如何对某一遵循相应规则的字符串进行拆解、解释、转化，是这个项目的核心内容，也是难点所在。

作为一个完整的程序设计项目，为了能够实现与用户的交互，本项目采用了数据库结构对所有通过该编辑器生成的文件的管理。项目通过设计文件类和系统类处理数据结构，并设计了相对友好的用户界面，考虑了一系列边界条件，最大限度地保证了程序的高效性和界面的友好型。

1.2 需求分析

MarkDown 编辑器在功能上有如下需求：

- 能够处理键入的字符串，保存为 md 文件；
- 能够对载入的 md 文件作字符串解析，根据 MarkDown 语法生成相应的 html 文件，生成的文件能够用浏览器正常打开。

在对 md 文件进行解析时，有如下需求：

- 生成 html 标签整体框架，包括 title、body 等；
- 能够处理 MarkDown 语法中的标题、加粗、斜体、列表、超链接以及上述语法的组合嵌套，生成对应 html 标签。

为了统一管理编辑器，我们采用数据库结构管理所有 md 文件，数据库系统

需要实现以下功能：

- 能够选中打开已经建立的 md 文件，并能够进行文本编辑操作（鉴于控制台多行控制比较麻烦，所有操作均为单行编辑操作），并作如上解析处理；
- 新建文件后，将文件的信息（对象）插入数据库中，并可以删除已经创建的文件；
- 可以导出已经创建的文件到计算机的其他目录(包括 md 和 html 文件)。

二、设计思路

2.1 数据结构分析

本项目需要处理的数据主要为 md 文件文本内容（字符串），md 文件由多行符合 MarkDown 语法规则的字符串构成，项目主要对这些字符串进行处理。

因此，我们创建 MdFile 类来处理 md 文件内容，MdFile 类的成员变量定义如下：

```
1.  class MdFile{
2.    private:
3.       string title;
4.       vector <string> content;
5.       string filename;
6.       bool trans;
7.       int line_nums;
8.       vector <string> html;
9.    public:
10.      MdFile(string title); //creating new file
11.      MdFile(string title, string filename, bool trans); //loading existing file
12.      //other functions
13.      ~MdFile();
14. };
```

各个成员变量的含义如下：

- title：文件标题；
- content：存放每行文本的字符串数组；
- filename：md 系统文件的存放目录（文件名）；

- trans：是否被转化为 html 文件的标识；
- line_nums：文件的总行数；
- html：生成的 html 文件的字符串存放临时单元。

对于每一个对象，md 文件和 html 文件的文件名是系统自动生成的，都是标题+后缀的形式，如 testfile.md，testfile.html，这些文件将被存放在 ".\ doc" 目录下，为确保程序正常运行，请不要删除、重命名里面的文件。

对于处理 md 文件的数据库，在项目中我们用依赖文件来记录所有已经创建的 md 文件信息，该系统文件目录为 ".\sysfile\table.txt"，每一行为一个文件的信息，每个文件信息以 <标题>’\t’<是否以生成 html 文件(true/false)> 的方式存放文件信息，例如某时刻系统文件中的记录如下：

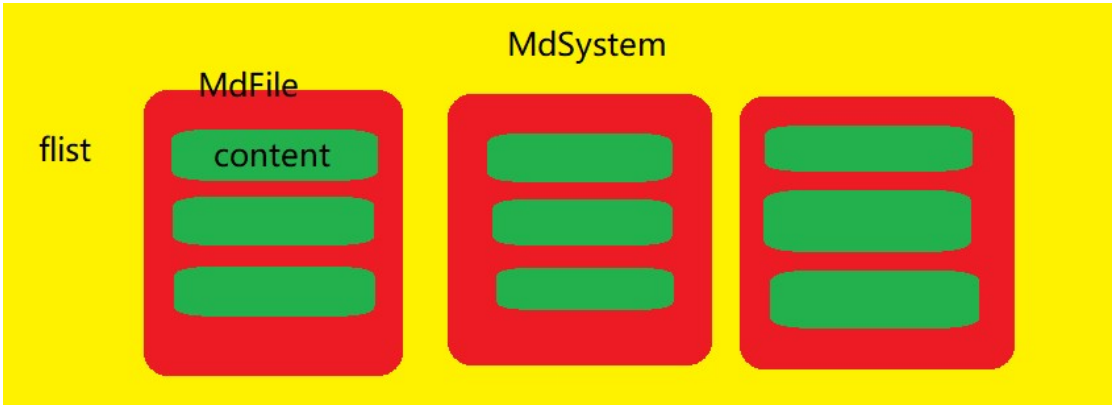
```
ex  true
utf true
testfile   true
testfile1  false
testfile2  false
testfile3  false
testfile4  false
testfile5  false
testfile6  false
testfile7  false
testfile8  false
```

由此，管理 md 文件的数据库类成员变量定义如下：

```
class MdSystem{
  private:
    int filenum;
    string filename; //system file
    vector <MdFile> flist;
    MdFile* current_file;
  public:
    MdSystem(string filename);
    //other functions
    ~MdSystem();
};
```

其中，filenum 表示已经创建的文件数量，flist 存储所有创建的 MdFile 类对象，current_file 用来标记当前正在操作的文件对象（但程序写到最后，发现这个变量好像并没有什么用，在设计 UI 调用操作时并没有用上，因为 UI 设计时我们已经将需要操作的文件的对象地址作为参数传入了，详情请见 3.4 部分）。

因此，用图示表示各个数据之间的关系，可能是这样的：



2.2 主要算法分析

2.2.1 MarkDown 语法解析

下列为与拆解 Token 与 Translate 成 html 文件有关的函数定义（MdFile 类）

```
1.  class MdFile{
2.      //.....
3.      public:
4.          //derive token from a sentence
5.          vector <Token> derive_token(int index);
6.          //check for special circumstances and adjust it
7.          vector <Token> check_token(vector <Token> tlist);
8.          //translate line by line according to the token, save to html, and return linetype
9.          LineType translate_line(int index);
10.         //translate into .html file, return html
11.         vector <string> translate();
12.     //.....
13. };
```

在本项目中，最为重要的算法就是对 md 文件中的字符串作 MarkDown 语法解析，为了简化问题，我们将采用先以行为单位分别拆解 Token，转化为 html

标签，在根据上下行关系，生成其他的过渡标签（主要为列表的头尾标签）。根据语法规则，我们对字符串作拆解 Token 的时候，需要了解 MarkDown 语法规定的语素类型。

其实 MarkDown 文本没有严格的语法错误，所有不合规范的输入都可以视为普通文本，这也是本项目的处理方式。根据标签类型，我们将 Token 语素分类为以下几类：

```
enum MyTokenType{
    NORM, //普通文本
    ITALIC, //斜体文本标签（*）
    BOLD, //加粗文本标签（**）
    BEM, //加粗斜体文本标签 (***)
    HEAD1, //一级标题（# ）
    HEAD2, //二级标题（## ）
    HEAD3, //三级标题（### ）
    HEAD4, //四级标题（#### ）
    HEAD5, //五级标题（##### ）
    HEAD6, //六级标题（###### ）
    ULM, //无序列表（- ）（+ ）（* ）
    OLM, //有序列表（num. ）
    LINK, //超链接标题 （[title]）
    HREF //超链接 url （(url)）
};
```

However，单单通过拆解 Token，然后直接字符串替换的方式是不能符合题目的要求的，因为在 html 标签中，有些是成对出现的，有些需要依赖于上下行，还有些需要多种语素混合搭配才有效，所以对于不同的 Token 类型，需要作不同的处理。面对种种问题，我们对 Token 的结构体定义如下：

```
struct Token{
    string s;
    MyTokenType t;
    int match;//0:no match; 1:left match; 2:right match (only valid for '*' and links)
    Token():match(0){}
};
```

拆解 Token 的函数在 MdFile 中被定义为 `vector <Token> derive_token(int index);`

解析中，我们以行为单位进行分别处理，最后进行边界条件和行间关系的处理，所以在单行解析的过程中，不会涉及到 `<ul>` 和 `<ol>` 标签。因此对于单行

处理，我们分以下情形讨论：

1. MarkDown 语法没有配对，html 标签有配对：适用于标题、列表，在处理时运用**栈空间**，遇到该类 Token 时压栈，解析完成后依次退栈生成结尾标签。

（下为此种情况的示例代码）

```
vector <Token> tlist = derive_token(index); //Token has been derived
vector <string> ending; //Stack
int endnum = 0;
string h;
//...
int nums = tlist.size();

for(int i=0;i<nums;i++){
    if(tlist[i].t==HEAD1){
        h += "<h1>";
        ending.push_back("</h1>"); //push in stack
        endnum++;
        //...
    }
    else if{
        h += "<li>";
        ending.push_back("</li>"); //push in stack
        endnum++;
        //...
    }
    else //...
}
//...
for(int i=endnum-1;i>=0;i--){ //Stack  pop
        h += ending[i];
    }
```

2. MarkDown 和 html 均有配对，在前语素的 Token.match 记为 1，后语素的记为 2，生成标签时若 match 为 1 生成头标签，为 2 时生成尾标签。
3. 超链接：提取出标题和 url，按照 html 语法直接生成即可。

处理完成后，还要考虑行内边界情况，具体表现为：“*”不配对，超链接语

法不全等等，将非法语素解释为普通文本（尽可能的保留多的特殊语素，如“*****”将被解释为 * ）。此过程在 vector <Token> check_token(vector <Token> tlist); 中定义。

为方便处理行间关系（主要为列表标签的补全），我们给解释行内 Token 函数的返回值定义为 LineType，这是一个枚举类型，返回该行的属性，定义如下：

```
enum LineType{
    NONE,
    NORMAL, //bold, em, href...
    HEAD,
    UL, //unsorted
    OL, //sorted
    ULH, //ul+head
    OLH //ol+head
};
```

函数 vector <string> translate() 为终极 translate 函数，在此函数中，不仅调用了 derive_token，还根据行的属性，自动生成 <ol> 的标签，在普通文本之间加入换行符
 （这里没有采用样例中使用的 <p> 标签，但是显示方面效果差不多，只是行距的问题）。

【遇到的问题】

这样的算法无法处理一个问题：列表嵌套。因为即使是同一类型的列表，也会出现嵌套的情况，而 translate 函数无法做出识别。归根结底，是我们拆解 Token 的时候欠考虑。

因此我们做出如下修改：拆解 Token 时，对于列表标签，我们同时考虑其前一个 Token 的空格数量，每两个空格为一层，将层数记录在 match 变量中（match 变量此时就在配对记录的基础上新增了一个功能）。

作行内处理时，我们不仅返回行类型，还要返回列表层数，因此我们构造如下结构体作为返回值：

```
struct LineInfo{
    LineType t;
    int level;
    LineInfo():level(0),t(NONE){}
};
```

处理行间关系时，列表的嵌套关系我们用**栈空间**管理，期间按列表层数顺序存储行信息 LineInfo，读取该行类型、列表层数时，根据其和栈顶元素列表类型、列表层数的比较，分情况作压栈、退栈处理，或不作处理。如下代码是以处理无序列表为示例的代码：

```
1.  if(info.t==UL||info.t==ULH){
2.      if(tlevel==0){
3.          html.insert(html.end()-1,"<ul>\n");    //html 为生成的 html 标签字符串数组
4.          stack_list.push_back(info);    //stack_list 为栈空间，类型 LineInfo 数组
5.          tlevel++;    //tlevel 为栈中元素
6.      }
7.      else{
8.          LineInfo bk = *(stack_list.end()-1);
9.          if(info.level>bk.level) {
10.             html.insert(html.end()-1,"<ul>\n");
11.             stack_list.push_back(info);    //嵌套列表情形，压栈
12.             tlevel++;
13.         }
14.         else if(info.level<bk.level){
15.             while(!stack_list.empty()){    //依次取出层数大的列表生成尾标签
16.                 LineInfo bk = *(stack_list.end()-1);
17.                 if(bk.level<info.level) {
18.                     html.insert(html.end()-1,"<ul>\n");
19.                     stack_list.push_back(info);
20.                     tlevel++;
21.                     break;
22.                 }
23.                 if(bk.level==info.level&&bk.t==info.t) break;    //同级标签，不作处理
24.                 switch(bk.t){
25.                     case UL: case ULH: html.insert(html.end()-1,"</ul>\n"); break;
26.                     case OL: case OLH: html.insert(html.end()-1,"</ol>\n"); break;
27.                 }
28.                 stack_list.pop_back();
29.                 tlevel--;
30.             }
31.
32.         }
33.     }
34. }
```

【问题解决】

2.2.2 MdFile 类的其他成员函数

MdFile 类中的其他成员函数定义如下：（大部分函数都是为了兼容数据库而编写的）

```
class MdFile{
  //...
  public:
    string prepro_s(string s);//do pre-process to the string, make it compatible to the rules of lines
    void add_line_back(string line); //add a line from the end
    bool add_line_mid(string line, int index); //add a line in the middle
    bool modify_line(int id, string line); //modify an existing line
    bool delete_line(int id);  //delete an existing line
    void display();  //display the whole content
    void display_insert(int index);  //display content in an insert-mode
    void display_chosen(int index);   //display content in modify/delete mode
    void save_file();  //save the md file after modification
    bool export_md(string path); //path is the exported filename of the copy.
    bool export_html(string path);
    string get_title();
    string get_trans_status();
    bool rename_title(string new_title);
    int get_line_nums();
    ~MdFile();
};
```

其中包括了增、删、改、输出，重命名等操作，以及获取私有成员值的函数。

2.2.3 MdSystem 类

在这个类中，我们定义的接口包括了对文件整体的增删，输出展示等操作，其实现方法均为最基本的数组操作，这里不再赘述，代码可以查看源文件。

MdSystem 的构造函数中，将对已有的文件进行 MdFile 对象的定义并进入数组中，由此可见 MdSystem 和 MdFile 类的嵌套关系。

下为 MdSystem 的构造函数中的关键操作：

```
MdSystem::MdSystem(string filename){
    FILE* pfile = fopen(filename.c_str(),"r");
    this->filename = filename;
    this->filenum = 0;
    this->flist.clear();
    this->current_file=NULL;
    char rc[MAXLEN];
    while(fgets(rc,MAXLEN,pfile)){
        string title;
        string trans;
        //do a series of oprations.....
        MdFile newmd(title,fn,(trans=="true")?true:false); //建立 MdFile 类对象
        flist.push_back(newmd);
        filenum++;
    }
}
```

三、代码框架

3.1 项目文件目录

本项目编写环境：**Windows 下 Dev-C++ 5.11 集成环境**，由于在 Windows 下编程，我们将字符编码集统一规定为 **GBK**。

项目目录如下图所示：

```
MarkDown
    |-- doc  //存储新建的 md，html 文件
    |    |-- XXX.md
    |    |-- XXX.html
    |    |-- ......
    |
    |-- include  //源代码的头文件
    |    |-- mdfile.h
    |    |-- mdsystem.h
    |    |-- ui.h
    |
    |-- src  //源代码的实现部分
    |    |-- main.cpp
```

```
|   |-- mdfile.cpp
|   |-- mdsystem.cpp
|   |-- ui.cpp
|   |-- XXX.o  //系统自动生成的.o 文件
|   |-- ......
|
|-- sysfile  //项目依赖文件
|   |-- table.txt //存储数据库信息
|   |-- writing.log
| //前期编程的进度与自己的感想...仅供记录、参考使用，与本项目运行时无关
|-- MarkDown.exe //可执行文件
|-- MarkDown.dev //工程文件
|-- ......
```

查看完整源代码的方式有：

1. 使用 Dev-C++，直接打开 MarkDown.dev
2. 到相应的目录里自己翻^_^

3.2 模块划分介绍

3.2.1 mdfile.h/mdfile.cpp

在 mdfile.h 中，有关于 TokenType 枚举类型、LineType 枚举类型、Token 结构体和 MdFile 类的定义，在 mdfile.cpp 中予以实现。此模块处理单个文件的修改行为，以及最最最最重要的 html 文件生成行为。

3.2.2 mdsystem.h/mdsystem.cpp

在 mdsystem.h 中，MdSystem 类被定义，在 mdsystem.cpp 中予以实现。此模块处理整个 md 数据库的整体操作。

3.3.3 ui.h/ui.cpp

在 ui.h 中，一系列函数被定义（没有另外定义类），用来设计用户界面，实现跳转，每个函数都以 MdSystem 对象的地址作为形参，部分对文件操作的界面函数还以对应文件 MdFile 对象的地址作为形参，执行操作时，调用对应接口。

设计 UI 时，为了最大限度避免用户误操作，在等待用户操作的地方我们采用按键操作替代键盘输入（即使用函数 getch() ），并用 do-while 结构循环直到按键合法为止，在一些敏感操作（如删除）中要求用户二次确认，才能成功操作。

大部分要求按键都是按照以下结构去写的：

```
1.  char ch;
2.  do{
3.      ch = getch();
4.      switch(ch){
5.          //different cases.....
6.      }
7.  } while(!is_valid(ch));
```

基本的界面设计采用菜单栏结构，界面示例具体可以在本报告的**第四部分**看到，这里给出一些 ui 界面函数的声明：

```
1.  //the main screen
2.  void enter_main(MdSystem* ms);
3.  //the add_new screen
4.  void enter_addnew(MdSystem *ms);
5.  //the view_all_file screen
6.  void enter_display(MdSystem* ms);
7.  //the exit screen
8.  void enter_exit(MdSystem* ms);
9.  //add lines from the last.
10. void edit_file(MdSystem* ms, MdFile* mf);
11. //display file
12. void view_file(MdSystem* ms, MdFile* mf);
13. //add a line in the middle
14. void add_mid_file(MdSystem* ms, MdFile* mf);
15. //modify an existing line
16. void modify_file(MdSystem* ms, MdFile* mf);
17. //delete an existing line
18. void delete_line_file(MdSystem* ms, MdFile* mf);
19. //translate into html
20. void enter_trans(MdSystem* ms, MdFile* mf);
```

3.3.4 main.cpp 以及程序运行调用结构

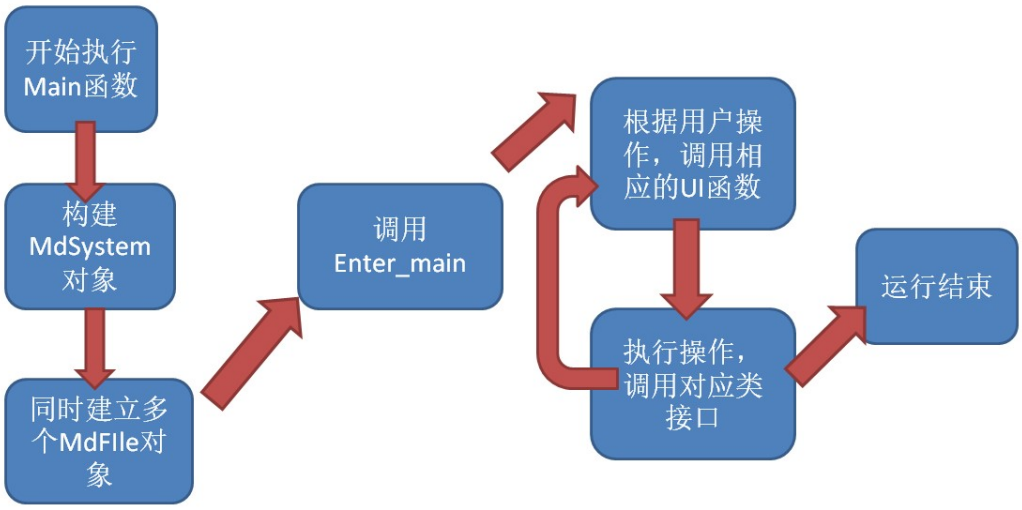
在 main.cpp 中，一个 MdSystem 类的对象将被定义，在 MdSystem 的构造函

数中定义一系列 MdFile 对象，随后我们直接调用 ui 中的 enter_main 函数，剩下的事情就是不断调用各种 ui 函数转换界面，并执行相应的操作接口。

下为 main 函数的定义：

```
1.  int main(int argc, char** argv) {
2.      MdSystem ms("sysfile\\table.txt");
3.      MdSystem *Ms = &ms;
4.      enter_main(Ms);
5.      return 0;
6.  }
```

程序的调用结构可以用下面的框图来表示：



四、界面展示&用户手册（简略版）

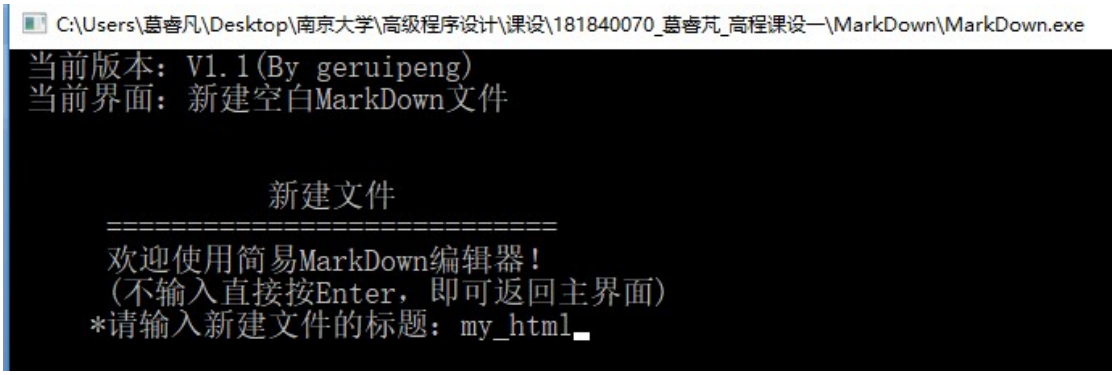
4.1 进入主界面

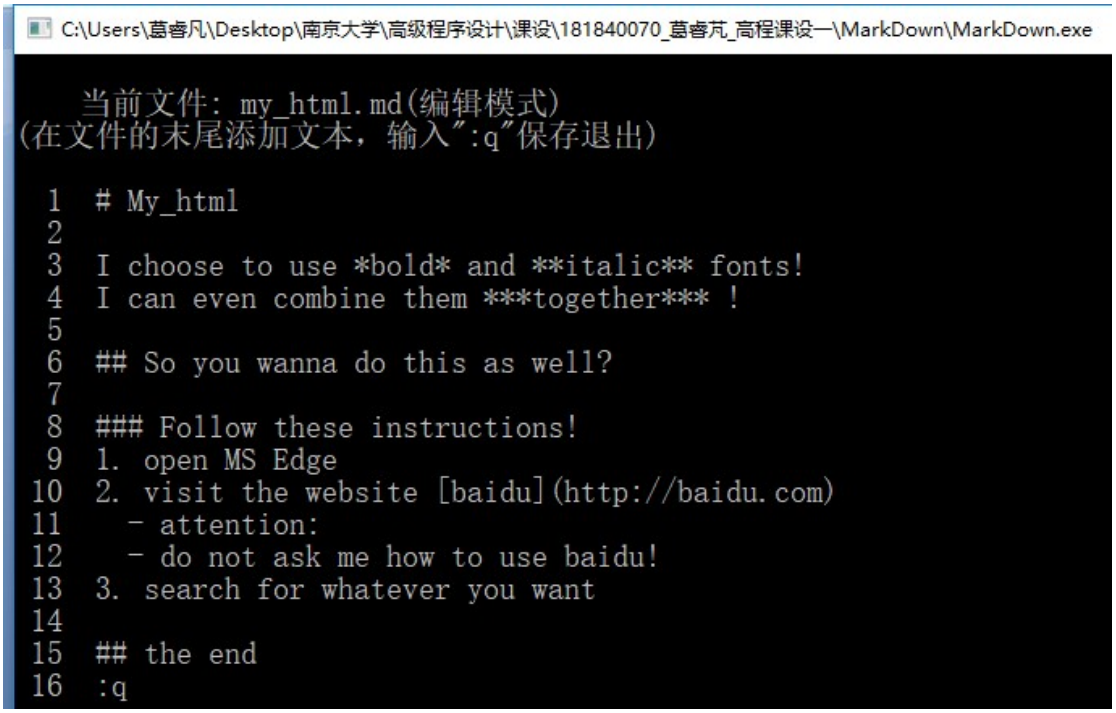
按“↑”“↓”移动光标，按 Enter 进入相应子界面。

【设计亮点】这里没有采用用户输入序号的形式，为了防止一些 dalao 输入一些非法字符导致程序崩溃，同时方便用户操作。在需要用户操作的地方，我们都会输出一段提示字符，帮助用户使用文档。

4.2 新建空白 MarkDown 文档

在主界面选择 1 菜单，进入新建界面，输入标题后自动进入编辑模式，编辑完成输入“:q”保存退出，即可进入文档展示界面。





4.3 浏览文档列表，显示文档

在主界面选择 2 菜单，即可查看所有文档的信息，移动光标选择文档，按下 Enter 进入文档展示界面。

【设计亮点】显示文档列表时，采用分页输出，体现了友好的用户界面。

4.4 文档处理

在文档展示界面按下 Enter 呼出菜单，按下相应按钮进入相应文档操作：

‘W’：进入编辑模式，在行尾继续键入文本；

‘I’：进入插入模式，选择插入位置后，在某一行插入想要插入的文本；

‘M’：进入修改模式，选择对应行后，修改该行的文本；

‘D’：进入删除模式，选择对应行后，删除该行的文本；

‘E’：导出创建的 md 文件至计算机其他目录；

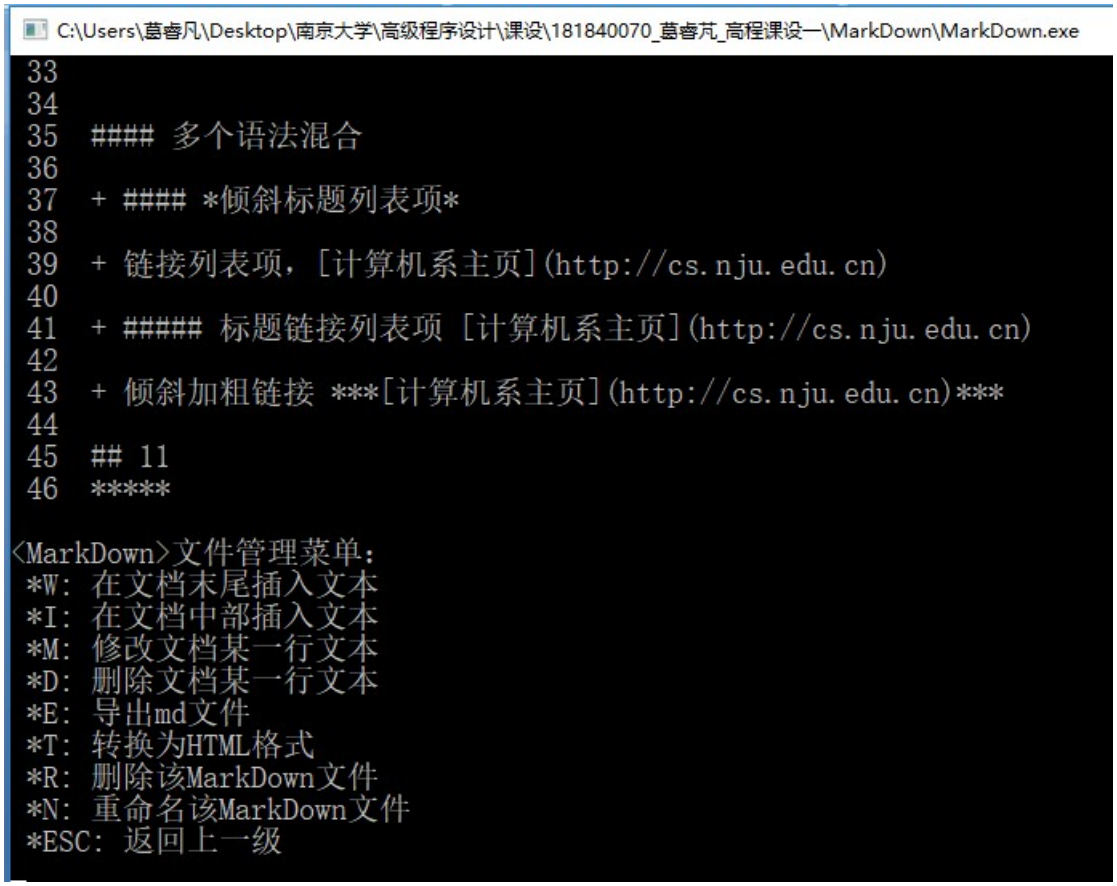
‘T’：生成 html 标签文件，并自动打开浏览器显示网页，同时可以设置导出 html 文件至指定计算机其他目录。

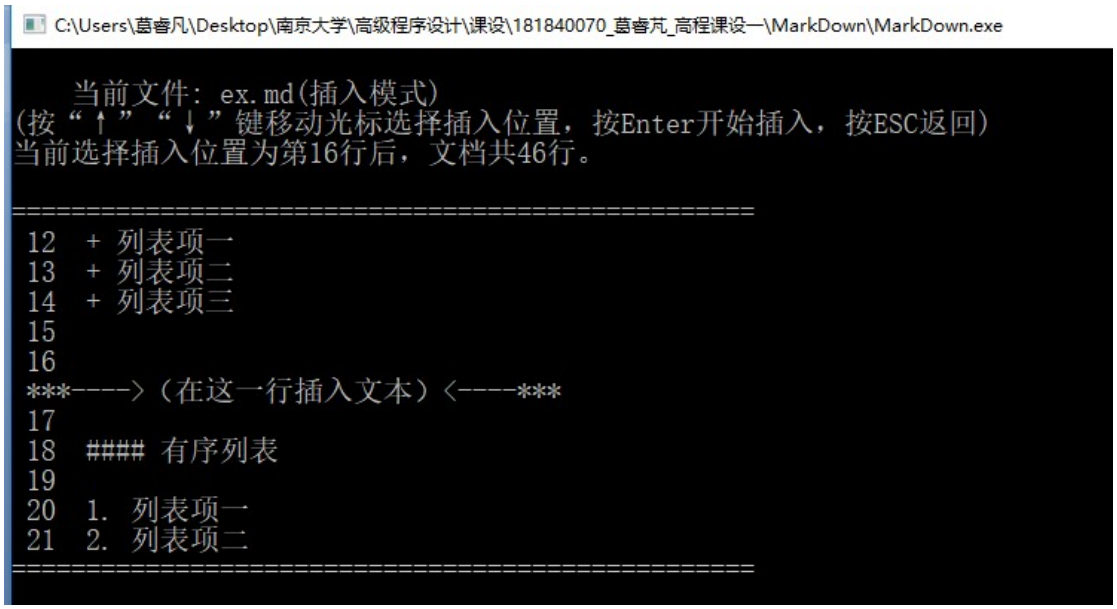
‘R’：删除创建的文档；

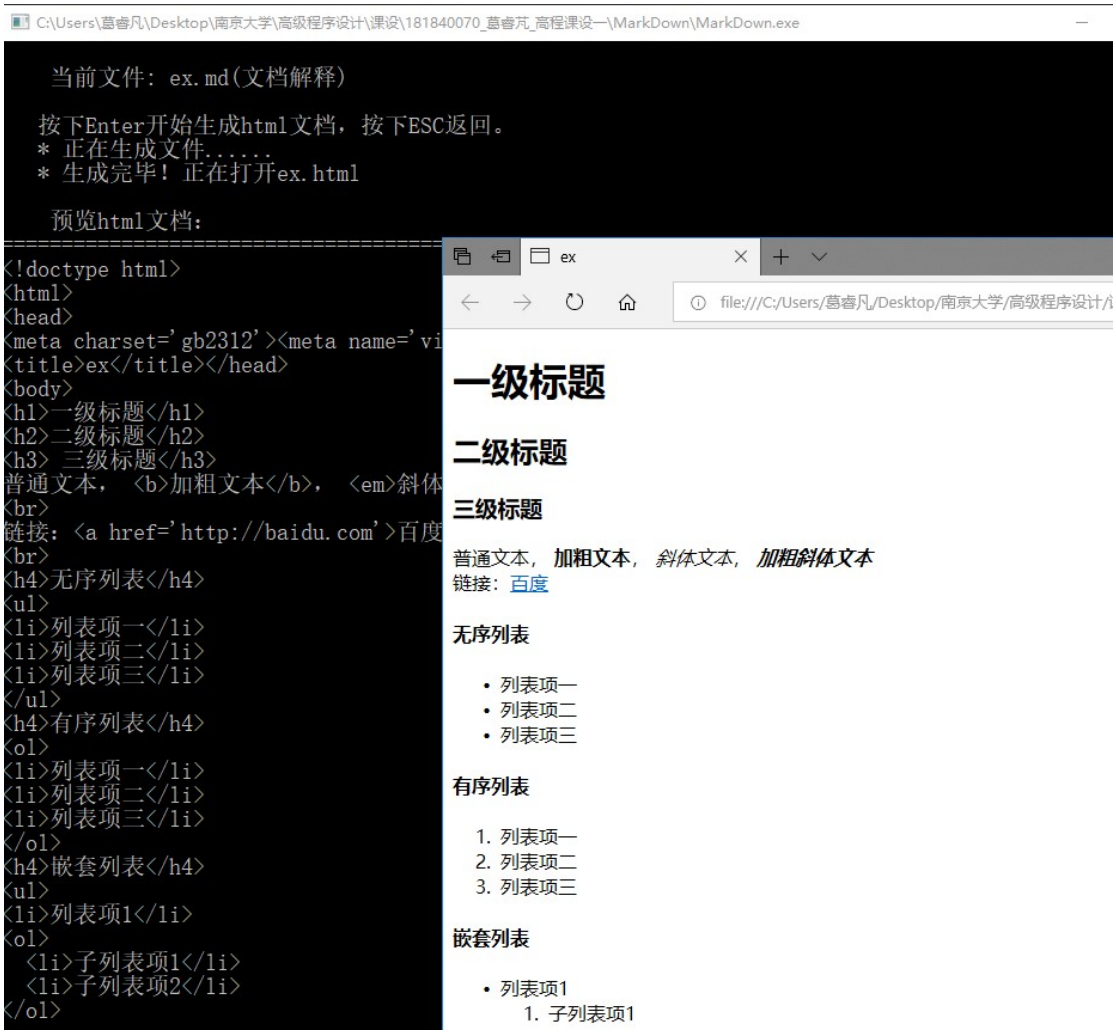
‘N’：重命名该文档；

‘ESC’：返回上一层菜单。

【设计亮点】 在插入、修改、删除时，只显示待插入行上下 5 行，用“↑”“↓”键选定对应行时，会实时更新输出视角。


C:\Users\葛睿凡\Desktop\南京大学\高级程序设计\课设\181840070_葛睿芃_高程课设一\MarkDown\MarkDown.exe
33
34
35 #### 多个语法混合
36
37 + #### *倾斜标题列表项*
38
39 + 链接列表项，[计算机系主页](http://cs.nju.edu.cn)
40
41 + ##### 标题链接列表项 [计算机系主页](http://cs.nju.edu.cn)
42
43 + 倾斜加粗链接 ***[计算机系主页](http://cs.nju.edu.cn)***
44
45 ## 11
46 *****
<MarkDown>文件管理菜单:
*W: 在文档末尾插入文本
*I: 在文档中部插入文本
*M: 修改文档某一行文本
*D: 删除文档某一行文本
*E: 导出md文件
*T: 转换为HTML格式
*R: 删除该MarkDown文件
*N: 重命名该MarkDown文件
*ESC: 返回上一级


C:\Users\葛睿凡\Desktop\南京大学\高级程序设计\课设\181840070_葛睿芃_高程课设一\MarkDown\MarkDown.exe
当前文件: ex.md(插入模式)
(按“↑”“↓”键移动光标选择插入位置，按Enter开始插入，按ESC返回)
当前选择插入位置为第16行后，文档共46行。
12 + 列表项一
13 + 列表项二
14 + 列表项三
15
16
---->（在这一行插入文本）<----
17
18 #### 有序列表
19
20 1. 列表项一
21 2. 列表项二



程序操作提示：在大部分要求选择相应菜单、文档、行的情况下，使用键盘的“↑”“↓”（翻页可以使用“←”“→”）即可，遇到需要自行输入的地方请输入一行字符串，程序的大部分操作都是按照这个规则的。

五、自我检讨（程序中的不足）

1、考虑不够周全：在设计类时，缺乏整体性考虑，拉起一个类就开始写，导致了出现类的成员变量定义不合理（比如 MdSystem 类中的 current_file 最后根本没有用到）的问题；

2、算法设计的全局观：在 Token 拆解时修改了许多次，浪费了很多时间，为了兼容各个边界条件，其中包括一次大规模的代码重做。

3、项目开始太晚，DDL 太赶，很多附加功能没有时间去实现。

六、后记

经历了去年一学期的程序设计实验课的训练，虽然我有感觉的我的项目构建能力有了提升，但是在算法考虑，数据设计等方面依然存在很多问题。这个报告中，我基本阐述了所有我在设计项目时候的想法，如果各部分安排不当，或有表述问题，还请助教和老师谅解。

十分感谢！

181840070 葛睿芃

2019/10/12