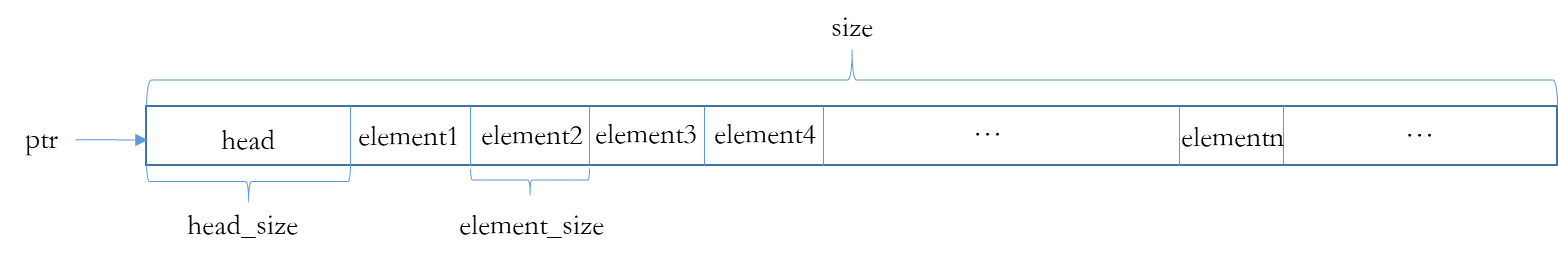
#### 第一层Interface

###### 线性表List操作



描述线性表在ram中的结构

typedef struct List{

char \*ptr; //初始化为NULL

uint32\_t head\_size; //需初始化

uint32\_t element\_size; //需初始化

uint32\_t size; //需初始化

uint32\_t flag; //初始化为0（具体的线性表具体使用）

uint32\_t element\_num;

uint32\_t element\_real\_num; //初始化为0（新定义的List是空的，往里面append后才增加num）

}List;//对应的结构变量初始化是强制的，否则极可能出错

**int List\_init(List \*list\_hd);**

**功能：** 申请list\_hd->size大小的内存并由list\_hd->ptr指向 + 将片内存区memset为0；

**输入参数：** 线性表控制结构变量指针List \*；

**返回值：** 成功返回OK，错误返回-1（ptr != NULL）；

**int List\_get\_item\_num(List \*list\_hd, uint32\_t flag);**

**功能：** 如果flag为0，则返回list\_hd->element\_num，否则返回list\_hd->element\_real\_num（这2个成员变量和线性表元素变化是同步的：改变线性表项（删除或增加），这2个成员变量也相应变化 🡺不要纠结它们会不会和实际项数目不一致，它们可以实现它们所代表的含义）；

**输入参数：** （略）

**返回值：** （list\_hd->ptr等于NULL则返回-1）

**int List\_append\_items(List \*list\_hd, char \*buf, uint32\_t num);**

**功能：** 往线性表ram后面添加buf里面的num个元素（num \* list\_hd->element\_size个字节）；

**int List\_get\_item\_index(List \*list\_hd, char \*name, element\_cmp cmp);**

如果ptr是NULL，（加一个回调比较函数，输入参数是指向两个比较元素的指针，调用者要对指针合法性负责，指针指向的数据类型以及数据大小，回调函数都是默认知道并直接使用的）

**int List\_delete\_item(List \*list\_hd, uint32\_t index, element\_delete de\_qualify);**

如果ptr是NULL，

**int List\_clean(List \*list\_hd, element\_qualify is\_qualify);**

如果ptr是NULL，

**void List\_uinit(List \*list\_hd);**

free掉分配的空间；

###### 2、线性文件linear\_file操作

（已完成）

#### 第二层Interface

###### 1、3个重要全局变量的初始化

**2个linear\_file变量初始化**

#define WORD\_LEN 20

#define EXP\_LEN 400 //5 \* ( 400)

#define DEFINITION 0

#define EXP\_SENTENCE 200

#define RESERVE\_LEN 1000

#define VOC\_FILE "voc.dic"

#define INDEX\_FILE "index.dic"

typedef struct Word{

char name[WORD\_LEN];

char exp1[EXP\_LEN];

char exp2[EXP\_LEN];

char exp3[EXP\_LEN]

char exp4[EXP\_LEN]

char exp5[EXP\_LEN]

char reserve[RESERVE\_LEN]

}word\_content;//Becaerful of Pack problem (Keep it at Multiple of 4 OR use pragma pack(1) macro)

char word\_name[WORD\_LEN];//index.dic的element数据类型

**1个List变量的初始化**：

###### 2、函数接口：

退出码是YELLOW表示需要重新加载ram，为RED表示源文件需要删除并重新创建；

**int dic\_init(void);//返回值错误，则需要检查错误码是否发生改变**

打开index.dic和voc.dic，并将index.dic文件内容导入到缓存中（涉及到3个全局变量）；如果打开文件出错则依情况设置退出码（退出码为RED，表示需要“重启”，index.dic和voc.dic需要删除）；

**int get\_word\_index(const char \*name);**

根据单词全名查找单词，返回单词位置（索引号）；不存在则返回ERROR；

**int add\_word(const char \*buf); //返回值错误，则需要检查错误码是否发生改变**

将buf中的内容添加到voc.dic中（末尾），并将buf中的name部分存入ram和index.dic中；如果这一步出现错误（无法存储到dic文件），程序需要退出，退出码为RED（退出码放在common.h中，定义为全局变量exit\_code，正常情况下退出码为GREEN即正常退出）；

**void print\_word\_content(uint32\_t index);**

根据索引号（单词位置打印单词内容）；

**int delete(const char \*name); //返回值错误，则需要检查错误码是否发生改变**

根据单词全名删除单词，同步从3个全局变量中删除，如果从2个文件中删除出错则程序需要退出，退出码为RED；如果不能从ram中删除，则程序需要退出，退出码为YELLOW

**int dic\_clean(void); //返回值错误，则需要检查错误码是否发生改变**

将index.dic，voc.dic和index\_ram进行clean，如果前两者clean失败则均设置退出码为RED，并返回-1；index\_ram如果clean失败则退出码设置为YELLOW，并返回-1；

**void dic\_exit(void);**

根据退出码执行相关操作后退出，如果是退出码RED则删除两个dic再退出，如果是YELLOW则直接退出（记得free掉申请的内存）；

#### 第3层Interface

（可在第二层和主程序功能实现之间进一步封装接口，不过应该不用封装了）

#### 主程序功能

###### voc68（打开程序）

**please enter command:\n**输入这3个search [word] / add [word] / delete [word]命令其中之一（在实现这3个命令时注意降低耦合性）

给出互动提示：

please enter one of commands as follow:

search [word]

add [word]

delete [word]

###### add [word]

1. add [word]之后程序先检查word是否存在，如果存在则提示已存在并输出word的相关内容；
2. 如果不存在，那么进入互动模式：
   1. please enter explanation1:\n输入英语释义1（一共10项，每项最大199个字节，第200个字节手动置'\0'），如果输入0（ASCII字符）则该项为空；
   2. please enter example sentence1:\n…
   3. …
   4. please enter example sentence1:\n…
   5. confirm your information:[all explanation]并进入互动模式
      1. 输入y则存储到文件中（包括目录文件和词汇文件）；
      2. 输入n则退回到**please enter command:\n**

###### search [word]

查找所有以word为前缀的单词及其序号（先从目录文件中查找）如果：

1. 一个都没用，给出相关提示并退回**please enter command:\n**；
2. 至少有1个，则进入互动模式：

提示：输入序号查找单词解释，输入q退出

* 1. 输入序号[n]则根据序号索引到词汇文件并给出对应的词汇项内容，然后回到互动模式可继续输入序号或者q；
  2. 输入q则退回到**please enter command:\n**

###### delete [word]

1. 如果没有唯一匹配的单词则给出提示并退回到**please enter command:\n**；
2. 如果有唯一匹配的单词，则输入相关内容，并通过y/n互动决定是否删除，然后退回到**please enter command:\n**（删除包括目录文件和词汇文件的删除）

issue：允许“文件空洞，索引号不连续”的存在，然后实现一个刷新功能，把两个文件的空洞都去掉，索引号不连续的修成连续（该功能可以手动调用，也可以在进程正常退出后自动调用，后期还可以设置成进程收到相关signal后自动调用）；实现一个进程正常退出的功能调用；

消除空洞的做法是复制两个文件（除了空洞之外其他都复制），将原来的文件删除，然后用lseek连续存储目录文件的索引号（将索引号修成连续的）；

#### 完善和改进

先不要追求所谓的完美架构，只要保存好原始的词汇文件，后期可以大改大修，不断地优化架构；

###### 功能拓展

1. 可以通过记录文件修改的时间戳判断（以及设置文件为不共享），如果文件被修改了则直接报错（本地数据被corrupt，需要从头再来，对应的要实现一个重头再来的功能：删除原来的文件，重新初始化现在的文件）；
2. 再实现一个本地备份功能，备份的文件设置为仅可读（用system调用一系列git命令完成备份）；
3. “再添加一个文件头，表面这是词汇文件以及目录文件（而不是其他内容的但名字一样的文本文件）”；
4. 现在只有记录功能，后期有时间可以添加文章收藏功能，笔记功能，写作锻炼功能，单词学习功能（多进程，多终端等等）；