Project 1: Philspel

实验概览

我们需要实现一个函数`philspel.c`其需要接受一个文本文件作为参数：

```
$ ./philspel dictionary.txt
```

现在我们输入一个单词`word`，那么如果：

- `word`本身在`dictionary`里面
- `word`的字母全部换成小写所形成的那个单词在`dictionary`里面
- `word`的第一个字母换成小写其余不变的那个单词在`dictionary`里面

如果以上三种情况之一发生，那么我们直接把`word`输出， 否则我们输出`word`加上`" [sic]"`

> If any of the three variations are found in the dictionary, the word is copied directly to standard output. Otherwise, the word is copied to standard output with the string `" [sic]"` (without the quotation marks, but *with the spaces*) appended. All other input (e.g. whitespace) is copied to standard output unchanged.

下面是我们需要完成的函数：

1. `void insertData(HashTable *table, void *key, void *data)` in `hashtable.c`
2. `void *findData(HashTable *table, void *key)` in `hashtable.c`
3. `unsigned int stringHash(void *s)` in `philspel.c`
4. `int stringEquals(void *s1, void *s2)` in `philspel.c`
5. `void readDictionary(char *dictName)` in `philspel.c`
6. `void processInput()` in `philspel.c`

那么其实我们可以发现，这道题如果放在算法竞赛里面是非常简单的，不就是`hash`表吗？但现在我们需要造一个`hash`表，然后还要处理一下比较复杂的输入输出。

实验开始

hash.c

首先我们看看官方给我们的`hash`表模板：

```
/*
 * This creates a new hash table of the specified size and with
 * the given hash function and comparison function.
 */
struct HashBucket {
  void *key;
  void *data;
  struct HashBucket *next;
```

```
};

HashTable *createHashTable(int size, unsigned int (*hashFunction)(void *),
                           int (*equalFunction)(void *, void *)) {
  int i = 0;
  HashTable *newTable = malloc(sizeof(HashTable));
  newTable->size = size;
  newTable->data = malloc(sizeof(struct HashBucket *) * size);
  for (i = 0; i < size; ++i) {
    newTable->data[i] = NULL;
  }
  newTable->hashFunction = hashFunction;
  newTable->equalFunction = equalFunction;
  return newTable;
}
```

其中`hashfuction`和`equalfunction`分别代表把字符串转化为哈希值以及判断两个字符串是否相等的函数。

insertData

现在新进来一个单词，我们怎么把其插入到我们的`hash`表中?

```
void insertData(HashTable *table, void *key, void *data) {
  // -- TODO --
  // HINT:
  // 1. Find the right hash bucket location with table->hashFunction.
  // 2. Allocate a new hash bucket struct.
  // 3. Append to the linked list or create it if it does not yet exist.
  int location = table->hashFunction(key);
  struct HashBucket* bucket = malloc(sizeof(struct HashBucket));
  bucket->key = key;
  bucket->data = data;
  bucket->next = NULL;
  if(table->data[location] == NULL) {
    table->data[location] = bucket;
  } else {
    bucket->next = table->data[location]->next;
    table->data[location]->next = bucket;
  }
}
```

findData

如何查询一个给定的单词是否出现在我们的`hash`表中?

```
void *findData(HashTable *table, void *key) {
  // -- TODO --
  // HINT:
  // 1. Find the right hash bucket with table->hashFunction.
```

```
  // 2. Walk the linked list and check for equality with table->equalFunction.
  int location = table->hashFunction(key);
  struct HashBucket* head = table->data[location];
  while(head != NULL) {
    if(table->equalFunction(head->key, key) == 0) {
      return head->data;
    }
    head = head->next;
  }
  return NULL;
}
```

philspel.c

stringHash

把字符串转化为对应`hash`值

```
unsigned int stringHash(void *s) {
  char *string = (char *)s;
  unsigned int hash = 5381;
  char c;
  while((c = *string++)) {
    hash = ((hash << 5) + hash) + c;
  }
  return hash % 2255;
}
```

stringEquals

判断两个字符串是否相等

```
int stringEquals(void *s1, void *s2) {
  char *string1 = (char *)s1;
  char *string2 = (char *)s2;
  while(*string1 != 0 && *string1++ == *string2++) ;
  return *string1 < * string2 ? -1 : *string1 > *string2;
}
```

readDictionary

读取`dictionary`文件，并把所有单词加到`hash`表中：

```
void readDictionary(char *dictName) {
  FILE *file = fopen(dictName, "r");
  if(file == NULL) {
    fprintf(stderr, "Open %s error", dictName);
    exit(1);
  }
```

```
  char buf[1024];
  while(fgets(buf, 1024, file)) {
    buf[strlen(buf) - 1] = '\0';
    char* dict = malloc(strlen(buf) + 1);
    strcpy(dict, buf);
    insertData(dictionary, (void*)dict, (void*)dict);
  }

  fclose(file);
}
```

findInDict

判断我们所给的单词`word`是否满足那三个条件

```
static int findInDict(char* str) {
  char str1[strlen(str) + 1];
  char str2[strlen(str) + 1];
  for(int i = 0; str[i] != '\0'; ++i) {
    if(isupper(str[i]))
      str1[i] = str[i] + 'a' - 'A';
    else
      str1[i] = str[i];
  }
  str1[strlen(str)] = '\0';

  str2[0] = str[0];
  for(int i = 1; str1[i] != '\0'; ++i) {
      str2[i] = str1[i];
  }
  str2[strlen(str)] = '\0';

  if(findData(dictionary, (void*)str)  ||
     findData(dictionary, (void*)str1) ||
     findData(dictionary, (void*)str2))
     return 1;
  else return 0;
}
```

processInput

```
void processInput() {
  char buf[1024];
  int index = 0;
  char c = fgetc(stdin);
  while(c != EOF) {
    if(!isalpha(c)) {
      if(index != 0) {
        buf[index] = '\0';
```

```
        if(findInDict(buf)) {
          printf("%s", buf);
        } else {
          printf("%s [sic]",buf);
        }
        index = 0;
      }
      printf("%c", c);
    } else {
      buf[index++] = c;
    }
    c = fgetc(stdin);
  }
  if(index != 0) {
    buf[index] = '\0';
    if(findInDict(buf)) {
      printf("%s", buf);
    } else {
      printf("%s [sic]",buf);
    }
  }
}
```