Report for Project 4: myIME

Demand Analysis

预处理

语料库需要自行从pinyin.txt与data.txt中提取

- 过滤汉字外的**无效内容**
- 对记录到的词语进行**词频统计**
- 设计合理高效的方式存储数据，方便后续取用
- 注意对于**多音字词**的处理

打字输入

完成一个功能基本完备的中文拼音输入法

- 支持**全拼与简拼**
- 支持加入用户**自定义**的词语
- 对于有歧义的情形，采取**最长匹配**的策略
- 对结果进行词频排序，且用户输入过的词语**优先级**最高
- 用户可以进行**多次选择**，直至得到期望的结果

Modular & Functional Decomposition

Overall

本次项目采用过程式程序设计，主要分为以下两个文件/模块：

preproc.cpp

- 预处理时，从pinyin.txt与data.txt中提取语料库
- 正常启动时，读取部分信息，完成初始化

type.cpp:

- 完成具体打字操作

preproc.cpp

预处理存储结构

- 将所有的字/词通过统一的**树状目录**结构进行存储
- 单级目录名为对应的**声母**
- 目录内含有：文本文件与子目录
    - 文本文件代表拼音至此结束，文件名为“**全拼.txt**”，里面存储了对应的字/词及词频
    - 子目录代表还存在更长的拼音序列，目录名即为**下一个声母**

举例说明：

现在要存一个词语“`我爱你`”，全拼为 `woaini`

则这个词语储存的路径为 `~/w/a/n/woaini.txt`

打开这个文件，里面就有所有全拼为 `woaini` 的词语（即**可能也有**“`窝哎泥`”），以及对应的**词频**

后续用户输入的词语也用这个规则存储，词频直接设为 `INT_MAX` ，以保证**优先**性

void trans()

- 读取pinyin.txt中的内容
- 将“拼音->汉字”转化为“汉字->拼音”，用 `map<wchar_t, vector<string>> pronunc` 存储
- 存储`pinyin.txt`中的汉字到对应目录

void proc_data()

- 读取并过滤 data.txt 中的信息
- 利用 `regex：L“[\u4e00-\u9fa5]+”` 来判断是否为汉字词语
- 读取和存储汉字类型（属于**宽字符**），应使用`wstring`、`wcin`、`wcout` 、`wfstream`等宽字符专属的类/实例，处理中文字符更加高效

void store_phrase(wstring phrase)

- 将从data.txt中提取到的字词 `phrase` 存入对应目录，并**添加词频**
- 通过上面讲过的 `pronunc` 获得字词的读音，并获取其声母
- 利用 `regex “[zcs]h.+”` 来判断是不是zh、ch、sh
- 遇到 pinyin.txt 中不存在的生僻字（往往是繁体字），忽略
- 遇到含多音字的词语，先存到 duoyinci.txt 中，可借助 `pypinyin` 这个python工具确定其读音，之后再统一存储

void load_info()

从已有的一些记录文件（ignore.txt, duoyinzi.txt, duoyinci.txt等）中读取信息，避免重复处理

void store_duoyinzi()

存储多音字方便使用，同样为了避免重复处理

FYI：以上两者只是**辅助性程序**，对应文件也都是**临时文件**...

type.cpp

About

本次项目采用朴素的过程式程序设计

打字分为**全拼与简拼**两种模式

- 通过 `regex "((([csz]h|[abcdefghjklmnopqrstwxyz])'?)+)"` 判断输入的拼音是否全为声母，若匹配则默认进入简拼模式
- 这里需要注意，有些拼音，如 *fang*，本身既可解释为全拼，也可解释为简拼，存在**歧义**
- 为应对上述情形，可选择输入 `enable-qp/jp` 以**强制开启**全拼/简拼，否则**优先简拼**；再输入 `disable-qp/jp` 以撤销

type_main()

作为打字**入口程序**，也是给main函数的**接口**

- 判断当前是简拼还是全拼
- 全拼时（简拼同理），调用 `quanpin()` 与`qp_choose()` 完成拼音解析

打字输入相关函数

FYI：具体原理在下一个部分详细说明

- 全拼涉及函数

  `void quanpin()`

  `void qp_choose()`

  `bool qp_translate()`
- 简拼涉及函数

  `void jianpin()`

  `void jp_choose()`

  `bool jp_translate()`

Program & Algorithms Implementation

以下以全拼为例，介绍完整算法过程

FYI：由于我的数据存储特点，简拼与全拼的算法**几乎完全一致**，只在个别地方有细微差别，也会一并指出...

拼音分割

- 写出正确的表示单个全拼的正则表达式，且须满足最长匹配原则，具体实现会在问题解决部分给出
- 首先调用 `quanpin()`，通过 `regex_search()` 分割词语中每个字的拼音，存储在 `vector<string> pinyin` 中，顺便得到整个词语的全拼 `string full_pinyin`
- 对于简拼，就要简单得多，只要每次取一个声母即可

用户选择

- 递归地向后查找，若被选中则前进，否则后退
- 调用 `qp_choose()`

伪代码如下：

```
void qp_choose(pinyin)
    left = 0 //当前起点
    right = pinyin.size() - 1 //当前终点
    cur_end = 0 //当前已完成的终点
    while cur_end < pinyin.size() - 1:
        qp_translate(pinyin, res, left, right, 0, cur_end)
        sort results based on frequency and show them to user
        if user does not choose any:
            if cur_end > 0: //结果太长
                right = cur_end - 1
            else:
                show no results available
                break
        else:
            record current result
            left = cur_end + 1
            right = pinyin.size() - 1
```

单次查找

- 递归地从词库目录中查找
- 被 `qp_choose()` 调用

伪代码如下：

```
//left 起点 right 终点
//cur 当前递归位置 end 成功返回的递归终点
bool qp_translate(pinyin[...], &res, int left, int right, int cur, int& end):
    if cur < right AND current path contains directory named $the next shengmu$:
        //go to that path
        if qp_translate(pinyin, res, left, right, cur+1, end) is true:
            return true
    if current path contains file "pinyin[left~cur].txt":
        get res from that file
        end = cur
        return true
    return false
```

Novelty

实现了附加的候选词排序与简拼功能

Problems & Solutions

P：

个人感觉预处理时真正的难点在于读取和存储汉字类型（属于宽字符），对于 pinyin.txt 中的单个汉字尚可采取读入2个char再转化为 `w_char` 的方式，但此处只能以`wstring`读入。

S：

根据踩过的坑，`wstring`、`wcin`、`wcout` 、`wfstream`等类/实例均需经过类似以下操作流程才能正确处理中文字符：

```
setlocale(LC_ALL, "chs");
wfstream wfout;
wfout.imbue(std::locale("chs"));
```

P：

得到的语料中含有大量的**多音字词**，在特定的搭配中无法判断其读音。

S：

遇到含多音字的词语，先存到 duoyinci.txt 中，可借助 `pypinyin` 这个python工具确定其读音，之后再对文件中的词语统一存储。`data/duoyin_proc.py` 即为这部分的 Python 代码。

P：

要实现拼音字母的按字分割，且遵循**最长匹配原则**，难点在于写出正确的 regex （可以搜到很多表示拼音的正则，但不保证满足最长匹配原则）。我经过了很多的修改完善后，发现既要尽可能使相同的拼音放在一起，**减少冗余**，又要注意**不能把前缀相同但较短的拼音放在前面，以免被覆盖**。在此给出我的实现。

S：

```
(ang?|a[io]?|ou?|e[inr]?|pou|m[io]u|[bmp](ia[no]|[aei]ng?|a[io]?|ei|ie?
|o|u)|me|fou|wai|[fw]([ae]ng?|a|ei|o|u)|dei|diu|[dt]([aeio]ng|an|a[io]?
|e|ia[no]|ie?|ou|uan|u[ino]?)|lun|[nl](i?ang?|a[io]?|
[eio]ng|ei|iao|in|i[eu]|ou|uan|u[eo]?|ve?)|ni|nen?|le|lia?|[ghk]([ae]ng?
|a[io]?|ong|ou|uang?|uai|u[aino]?)|ke|[gh]ei?|[jqx](i(ang?|ao?|e|ng?
|ong|u)?|uan|u[en]?)|([csz]h?|r)([ae]ng?|ao|e|i|ou|uan|u[ino]?)|[csz](ai?
|ong)|[csz]h(ai?|uai|uang)|zei|[sz]hua|([cz]h|r)ong|y([ai]ng?|ao?
|e|i|ong|ou|uan|u[en]?))
```

Testing Snapshots

```
请输入拼音，回车结束：
qingdajiajianchilajifenlei
1. 请  2. 轻  3. 清  4. 轻  5. 清  6. 氢  7. 轻  8. 清  9. 氢  10. 轻  11.
7. 清  18. 氢  19. 轻  20. 清  21. 氢  22. 轻  23. 清  24. 氢  25. 轻  26.
2. 清  33. 氢  34. 情  35. 轻  36. 清  37. 氢  38. 情  39. 青  40. 轻  41.
7. 氢  48. 情  49. 青  50. 轻  51. 清  52. 氢  53. 情  54. 青  55. 轻  56.
2. 轻  63. 调情  64. 清  65. 氢  66. 晴  67. 情  68. 鲭  69. 青  70. 轻  7
鲭  77. 青  78. 轻  79. 调情  80. 氰  81. 氢  82. 晴  83. 情  84. 庆  85.
91. 蜻  92. 苘  93. 罄  94. 綮  95. 箐  96. 磬  97. 氰  98. 檠  99. 晴  10
. 倾
请输入对应标号：
（输入0表示没有符合的选项）
1
1. 大家  2. 打架  3. 打假
请输入对应标号：
（输入0表示没有符合的选项）
1
1. 坚持  2. 尖齿  3. 减持
请输入对应标号：
（输入0表示没有符合的选项）
1
1. 垃圾  2. 拉吉
请输入对应标号：
（输入0表示没有符合的选项）
1
1. 分类  2. 酚类
请输入对应标号：
（输入0表示没有符合的选项）
1

请大家坚持垃圾分类
```

```
请输入拼音，回车结束：
jqr
1. 机器人  2. 几千人
请输入对应标号：
（输入0表示没有符合的选项）
1

机器人
```