Project1-C

蒋越 PB16001821

莫定衡 PB16001716

吴昊 PB16001800

文本处理

接口设计

接口为：`static vector<string> process(istream &infile)`

接收输入流 `infile` 的引用（即打开的文件流），然后返回从文件中读取到的单词列表，用 `vector<string>` 存放

功能测试

1. 移除重复单词

```
TEST(preprocessing_test, RemoveDuplicate){
    istringstream test_stream("i have a pen, i have an apple。ah, apple pen.");
    vector<string> vocabulary;
    vocabulary = PreProcessing::process(test_stream);

    vector<string> expected = { "have", "pen", "an", "apple", "ah" };

    ASSERT_EQ(vocabulary, expected);
}
```

2. 大写转换为小写

```
TEST(preprocessing_test, CaseInsensitive){
    istringstream test_stream("i have a PeN, i hAve aN aPplE。ah, appLE pen.");
    vector<string> vocabulary;
    vocabulary = PreProcessing::process(test_stream);

    vector<string> expected = { "have", "pen", "an", "apple", "ah" };

    ASSERT_EQ(vocabulary, expected);
}
```

异常处理

在上次的"功能封装"实验中，我们已经完成了一些异常处理功能。在这次的实验中，我们又做了一些补充，以提高程序的鲁棒性。

我们的异常处理代码主要分布在 文本预处理模块 (PreProcessing.cpp) 和 命令行参数分析模块 (ArgParser.cpp) 中。这两个模块是直接和用户交互的：前者处理用户给出的文本文件（文件名由命令行参数指定）的具体内容中的异常输入，后者处理用户通过命令行调用程序给出的命令行参数中的异常输入。在这两个模块中做异常处理后，能确保输入到核心模块 (Core.cpp) 的各项数据都是合法的。

然而，用户可能不会调用我们的预处理和命令行参数分析模块，而是通过 API 直接调用我们的核心模块。为了应对这种情况，在核心模块中也进行了错误处理和异常抛出。

此外，在主函数中也存在一个异常处理：检查文件是否正常打开，如有异常，输出操作系统给出的异常信息。

1. 文本预处理模块(PreProcessing.cpp)

空文件

应当抛出异常`invalid_argument("no word found in text!")`

```
TEST(preprocessing_test, NullTest){
    istringstream null_stream;
    vector<string> vocabulary;
    try{
        PreProcessing::process(null_stream, vocabulary);
        FAIL(); // 不抛出异常，失败
    }
    catch(invalid_argument const& err){
        EXPECT_EQ(err.what(), string("no word found in text!"));
    }
    catch(...){
        FAIL(); // 抛出其他异常，失败
    }
}
```

2. 命令行参数分析模块(ArgParser.cpp)

头尾字母检查

1. 字母长度检查

头字母

测试样例：`-w test.txt -h aa`

实现原理

```
buffer = argparser.get<string>("head");
if (buffer.length() > 1){
    throw length_error("head character length error!");
    // cerr << "length of head character is greater than 1!" << endl;
}
```

测试方法

测试输入: `vector<string> arguments = {"./a.out", "-w", "test.txt", "-h", "aa"};` 异常测试关键代码:

```
try{
    ArgParser::parse(argv.size() - 1, argv.data(), filename, head, tail, num,
            word_flag, char_flag, num_flag);
    FAIL(); // Exception should be caught!
}
catch(length_error const & err){
    EXPECT_EQ(err.what(), string("head character length error!"));
}
catch(...){
    FAIL(); // Caught wrong type of Exception
}
```

尾字母

实现原理和测试方法与头字母类似

测试样例: `-w test.txt -t aa`

测试数据为 `vector<string> arguments = {"./a.out", "-w", "test.txt", "-t", "aa"};`

2. 字母范围检查

头字母

实现原理

```
if (head_char != '\0'){
    if ('a' <= head_char && head_char <= 'z'){
        head = head_char;
    }
    else if ('A' <= head_char && head_char <= 'Z'){
        head = head_char - 'A' + 'a';
    }
    else{
        throw out_of_range("head character is not in the range(a-z, A-Z)!");
    }
}
```

测试方法

测试样例：`-w test.txt -h 3`

应抛出头字母不在 ascii 范围内异常

测试输入：`{"./a.out", "-w", "test.txt", "-h","3"};`

测试代码：

```
try{
    ArgParser::parse(argv.size() - 1, argv.data(), filename, head, tail, num,
word_flag, char_flag, num_flag);
    FAIL();
}
catch(out_of_range const & err){
    EXPECT_EQ(err.what(), string("head character is not in the range(a-z, A-
Z)!"));
}
catch(...){
    FAIL(); // Caught wrong type of Exception
}
```

尾字母

测试样例：`-w test.txt -t 3`

抛出尾字母不在 ascii 范围内异常

测试数据: `vector<string> arguments = {"./a.out", "-w", "test.txt", "-t","3"};`

测试代码和 3. 类似

工作模式检查

实现原理

```
if (!argparser.exist("word") &&!argparser.exist("char")){
    throw invalid_argument("Either -w or -c must be selected!");
}
else if (argparser.exist("word") && argparser.exist("char")){
    throw invalid_argument("-w and -c cannot be used together!(Not
Implemented!)");
}
if (num_flag && char_flag){
    throw invalid_argument("Using -c and -n together is not implemented!");
}
```

测试方法

1. `test.txt`

抛出未选定工作模式异常(至少选择 -w 或 -c 的一种)

```
    try{
    ArgParser::parse(argv.size() - 1, argv.data(), filename, head, tail, num,
word_flag, char_flag, num_flag);
    FAIL();
}
catch(invalid_argument const & err){
    EXPECT_EQ(err.what(), string("Either -w or -c must be selected!"));
}
catch(...){
    FAIL();
}
```

2. `test.txt -w -c`

抛出工作模式冲突异常(无法同时 -w 和 -c)

```
try{
    ArgParser::parse(argv.size() - 1, argv.data(), filename, head, tail, num,
            word_flag, char_flag, num_flag);
    FAIL(); // Exception should be caught!
}
catch(invalid_argument const & err){
    EXPECT_EQ(err.what(), string("-w and -c cannot be used together (Not
Implemented!)"));
}
catch(...){
    FAIL(); // Caught wrong type of Exception
}
```

3. `test.txt -c -n 114514`

抛出未实现异常(并未要求实现 -c -n 同时使用的情形)

```
try{
    ArgParser::parse(argv.size() - 1, argv.data(), filename, head, tail, num,
            word_flag, char_flag, num_flag);
    FAIL(); // Exception should be caught!
}
catch(invalid_argument const & err){
    EXPECT_EQ(err.what(), string("Using -c and -n together is not
implemented!"));
}
catch(...){
    FAIL(); // Caught wrong type of Exception
}
```

3. 核心模块 (Core.cpp)

i. words 内包含非 ascii 字符

实现原理

```
  static void check_words(vector<string> words){
    for (auto it = words.begin(); it != words.end(); it++){
      for (auto it_char = it -> begin(); it_char != it -> end(); it_char ++){
        if (!isalpha(*it_char))
          throw out_of_range("char \'" + string(1, *it_char) +
                             "\' out of range in word \"" + string(*it) +
"\"");
      }
    }
}
```

遍历 words 中每一个单词，发现不在 a-z, A-Z 的范围内就抛出异常，并指明出错的字母和对应单词。

测试方法

测试输入为`vector<string> words = {"apple", "26"}`

测试代码如下：

```
TEST(exception_test, check_words){
    vector<string> words = {"apple", "26"};
    vector<string> result;
    try{
        Core::gen_chain(words, result);
        FAIL();
    }
    catch(out_of_range const& err){
        ASSERT_EQ(err.what(), string("char '2' out of range in word \"26\""));
    }
}
```

ii. 头，尾字母为非 ascii 字符

实现原理

```
  static char check_char(char rhs){
    if (rhs != '\0'){
      if ('a' <= rhs && rhs <= 'z')
        return rhs;
      else if ('A' <= rhs && rhs <= 'Z')
        return rhs - 'A' + 'a';
      else
        throw out_of_range("char " + string(1, rhs) + " is not in the range(a-z, A-Z)!");
    }
    else
      return 0;
  }
```

测试方法

输入一个 ascii = 23 的字符

```
try{
    Core::convert_char(23);
    FAIL();
}
catch(out_of_range const& err){
    ASSERT_EQ(err.what(), string("char " + string(1, 23) + " is not in the range(a-z, A-Z)!"));
}
catch(...){
    FAIL();
}
```

4. 主函数 (main.cpp)

打开文件失败异常

实现原理

```
if (infile.fail()){
    throw system_error(errno, std::generic_category());
}
```

测试方法

打开一个不存在的文件，抛出如下异常：`libc++abi.dylib: terminating with uncaught exception of type std::__1::system_error: No such file or directory`

单元测试覆盖率

Filename	Line Coverage		Functions	
ArgParser.cpp	100.0 %	51 / 51	100.0 %	1 / 1
Core.cpp	100.0 %	272 / 272	100.0 %	9 / 9
PreProcessing.cpp	100.0 %	51 / 51	100.0 %	1 / 1
WordNode.h	100.0 %	1 / 1	100.0 %	4 / 4

附录1: 测试使用方法

在 test 文件夹下，输入`make preprocessing`测试 预处理模块 (preprocessing.cpp)

输入 `make argparser` 测试 命令行参数分析模块 (argparser.cpp)

输入 `make core` 测试 核心模块 (core.cpp)

输入 `make all` 或者 `make` 生成覆盖率相关信息（需要安装 LCOV 工具和 GCC 编译器）

附录2:命令行参数分析

参考了

1. 该项目：https://github.com/tanakh/cmdline

和

2. 该文章：https://blog.csdn.net/10km/article/details/50982993