Найти неверные формы слов в предложении	
Внутренняя спецификация	
Студент	Юрасов Р.В.
Преподаватель	доц. Сычев О.А.
Сдано	

1. Назначение

Программа предназначена для нахождения неверных форм слов в предложении.

2. Описание логической структуры

Работа программы логически разделена на несколько частей:
– считывание входных данных;
– деление предложений на составляющие;
– сопоставление слов в строках, пришедших на вход (согласование предложений);
– поиск ошибок;
– вывод результата.

3. Описание используемых в программе структур данных

Перечисление posTags содержит:
notFound, adj, adp, adv, aux, cconj, det, intj, noun, num, part, pron, propn, sconj, sym, verb, x

Перечисление tenseOfVerb содержит:
presentSimple, presentCont, pastSimple, pastCont, presentPerf, presentPerfCont, pastPerf, pastPerfCont, futureSimple, futureCont, futurePerfSimple, futurePerfCont, futureSimpleInThePast, futureContInThePast, futurePerfInThePast, futurePerfContInThePast, undefined

Перечисление formOfVerb содержит:
infinitive, presentFirstOrSecondPerson, present3P, past, preterite, pastParticiple, presentParticiple

Перечисление person содержит:
third, notThird

Перечисление number содержит:
singular, plural, uncountable

Перечисление caseOfNounAndPron содержит:
objective, possessive

Перечисление formOfPron содержит:
nominative, objectiveCasePron, possesiveMain, possesiveAbsolute

Перечисление formOfNumeral содержит:
cardinal, ordinal

Перечисление formOfAdjective содержит:
positive, comparative, superlative

Перечисление auxiliaryForAdj содержит:
notSet, more, most, notNeed

Перечисление sentencesProcessingCodes содержит:
sentencesProcessed, fewerWordsThanTags, fewerTagsThanWords, wrongPosTag, zeroWordsInSentence

Перечисление fileProcessingCodes содержит:
filesProcessed, inputCorrectFileNotExist, inputVerifiedFileNotExist, cantCreateOutputFile, wrongStrCountInCorrectFile, wrongStrCountInVerifiedFile

структура WordForm содержит:
unsigned int firstForm:8
unsigned int secondForm:8
unsigned int thirdForm:8
unsigned int auxForAdj:2

Класс FileDataError содержит:
enum sentencesProcessingCodes sentenceProcessingResult
int strCountInFile - количество строк в файле

class InputDataError {
enum fileProcessingCodes fileProcessingResult
int countOfWordsInFile
int countOfPosTagsInFile
QString wrongPosTag

Класс Word содержит:
QString word
QSet<WordForm> possibleFormsOfWord - все фактически возможные формы слова

Класс Adjective: Word содержит:
int indexOfAuxiliary – вспомогательное слово к данному прилагательному

Класс Verb: Word содержит:
QList<int> indexesOfAuxiliaries

Класс Sentence содержит:
QList<Word> sentence

Класс Pair содержит:
int wordIndexFromCorrectSentence
QList<int> indexesOfAuxiliariesInCorrect
WordForm wordFormFromCorrectSentence
int wordIndexFromVerifiedSentence
QList<int> indexesOfAuxiliariesInVerified
WordForm wordFormFromVerifiedSentence
posTags posTag – часть речи главного слова данной пары
int degreeOfFallacy - степень ошибочности данной пары

Класс LinkedSentences содержит:
QSet<QSet<Pair>> linkedSentences – наборы согласования предложений
int countOfUsingWords - количество используемых слов в согласованиях
int fallacy - ошибочность согласования

Класс ReadFiles содержит:
QString correctSentence
QString posTagSentence
QString verifiedSentence

4. Перечень вызываемых функций:

Считывает правильное предложение и pos-теги из файлов
FileDataError FilesProcessing::readFileWithCorrectSentence(const QString &correctSentenceFilename)

Входные данные:
correctSentenceFilename – имя файла, в котором содержится правильное предложение и Pos-теги

Выходные данные:
QString correctSentence – в поле класса записывается правильное предложение в виде строки
QString posTagSentence – в поле класса записывается предложение с pos-тегами в виде строки
FileDataError information – подробная информация об ошибке

Алгоритм:
1 Если удалось открыть файл с правильным предложением и pos-тегами
  1.1 Если в файле 2 строки
    1.1.1 Считать первую строку в correctSentence
    1.1.2 Считать вторую строку в posTagSentence
  1.2 Иначе
    1.2.1 Запомнить количество строк в файле

1.2.2 Вернуть код ошибки wrongStrCountInCorrectFile

2 Иначе

2.1 Вернуть код ошибки inputCorrectFileNotExist

Считывает проверяемое предложение из файла

InputDataError FilesProcessing::readFileWithVerifiedSentence(const QString &verifiedSentenceFilename);

Входные данные:

verifiedSentenceFilename– имя файла, в котором содержится проверяемое предложение

Выходные данные:

QString verifiedSentence – в поле класса записывается проверяемое предложение

InputDataError information – подробная информация об ошибке

Алгоритм:

1.1 Если удалось открыть файл с проверяемым предложением

1.1.1 Если в файле 1 строка

1.1.1.1 Считать эту строку в verifiedSentence

1.1.2 Иначе

1.1.2.1 Запомнить количество строк в файле

1.1.2.2 Вернуть код ошибки wrongStrCountInVerifiedFile

2 Иначе

2.1 Закрыть файл

2.2 Вернуть код ошибки inputVerifiedFileNotExist

3. Закрыть файл

Согласует 2 предложения. Выбираются варианты с наибольшим количеством согласованных слов и наименьшей ошибочностью
void LinkedSentences::linkSentences(Sentence &correctSentence, Sentence &verifiedSentence)

Входные данные:
Sentence &correctSentence - верное предложение из поля данных класса Sentence.sentence
Sentence &verifiedSentence - ошибочное предложение из поля данных класса Sentence.sentence

Выходные данные:
QList<QList<Pair>> linkedSentences - в поле данных класса записываются все получившиеся варианты согласования предложения

Алгоритм функции:
1. Очистить список согласованных пар
2. Установить формы слов и вспомогательные в правильном предложении
3. Составить все возможные пары слов первого предложения со вторым, рассчитав их правильность
4. Составить все возможные варианты согласования предложений с наибольшим количеством согласованных слов и с наименьшей ошибочностью

Установить времена глаголов и вспомогательные слова глаголов и прилагательных в верном предложении
void Sentence::setFormsInCorrectSentence()

Входные данные:

sentence - поле данных класса, в котором содержится предложение

Выходные данные:

sentence - поле данных класса, в котором содержится предложение, куда записываются времена и вспомогательные слова

Алгоритм:

1. Считать, что индекс предыдущего глагола 0
2. Для каждого слова с тегами adj, noun, pron, num или verb
   2.1 По написанию слова определить его возможные формы
   2.2 Если текущий тег verb
      2.2.1 Между предыдущим глаголом и текущим найти все Auxiliary для глаголов
      2.2.2 Добавить вспомогательные к глаголу
      2.2.3 Для каждой возможной формы глагола
         2.2.3.1 Если данная форма глагола дает в сочетании с вспомогательными какое-либо время
            2.2.3.1.1 Добавить данную форму к слову в предложении
      2.2.4 Если не добавлена ни одна форма, добавить форму undefined
   2.3 ИначеЕсли текущий тег adj
      2.3.1 Определить вспомогательное слово, находящееся слева от прилагательного

      2.3.2 Для каждой возможной формы прилагательного
         2.3.2.1 Если текущей форме требуется more и вспомогательное more
            2.3.2.1.1 Добавить вспомогательное к прилагательному и добавить к нему данную форму слова

2.3.2.2 ИначеЕсли текущей форме требуется most и вспомогательное most

2.3.2.2.1 Добавить вспомогательное к прилагательному и добавить к нему данную форму слова

2.3.2.3 Иначе

2.3.2.3.1 Добавить данную форму к слову в предложении

2.4 Иначе

2.4.1 Добавить все формы к слову в предложении

Создание всех возможных пар слов

QList<Pair> LinkedSentences::generateAllPairs(const Sentence &correctSentence, const Sentence &verifiedSentence)

Входные данные:

correctSentence - правильное предложение

verifiedSentence - проверяемое предложение

Выходные данные:

QSet<Pair> listOfPairs – список всех получившихся пар

Алгоритм:

1. Для каждого слова в правильном предложении, имеющего pos-тег adj, noun, pron, num или verb

1.1 Образовать все формы данного слова

1.2 Для каждой формы, различной по написанию

1.2.1 Пока текущее слово встречается в проверяемом предложении

1.2.1.1 Определить все соответствующие ему наборы вспомогательных слов

1.2.1.2 Создать пары, соединив все формы слова, одинаковых по написанию из правильного предложения, со всеми наборами вспомогательных слов

1.2.1.3 Установить правильность каждой пары

Устанавливает время глагола, вспомогательные слова к указанному глаголу или прилагательному в проверяемом предложении
QList<Word> setFormInVerifiedSentence(int indexOfProcessingWord, QSet<WordForm> formsOfProcessingWord, posTags posTagOfProcessingWord)

Входные данные:
indexOfProcessingWord - индекс обрабатываемого слова
formsOfProcessingWord - формы обрабатываемого слова
posTagOfProcessingWord - часть речи обрабатываемого слова

Выходные данные:
QList<Word> settedForms - все установленные формы глагола/прилагательного со вспомогательными

Алгоритм:
1. Если текущий pos-тег - verb
1.1 Если данные формы глагола без вспомогательных составляют время
1.1.1 Добавить глагол с указанием времени к решению
1.2 Для каждого слова, находящегося слева от глагола и пока текущий набор можно расширить до какого-либо времени
1.2.1 Если данное слово - вспомогательное
1.2.1.1 Если справа от него not
1.2.1.1.1 Добавить not к списку

1.2.1.2 Добавить данное слово к списку вспомогательных

1.2.1.2 Для каждой формы глагола

1.2.1.2.1 Если текущая форма глагола с вспомогательными составляет какое-либо время

1.2.1.2.1.1 Добавить список вспомогательных с текущей формой глагола к решению

1.3 Если не найдено ни одного решения

1.3.1 Добавить к решениям глагол без вспомогательных с указанием времени undefined

2. ИначеЕсли текущий тег - adj

2.1 Для каждой формы

2.1.1 Если данной форме требуется вспомогательное more

2.1.1.1 Если more есть слева от прилагательного

2.1.1.1.1 Добавить more и прилагательное к решению

2.1.2 ИначеЕсли данной форме требуется вспомогательное most

2.1.2.1 Если most есть слева от прилагательного

2.1.2.1.1 Добавить most и прилагательное к решению

2.1.3 Иначе

2.1.3.1 Добавить данную форму к решению

Рекурсивное создание всех возможных вариантов согласования.

void recoursivePairsLink(const QList<Pair> &listOfPairs, int numberOfPair, QList<Pair> &currentLinkedPairsLis, long long int bitCartForCorrectSentence, long long int bitCartForVerifiedSentence)

Входные данные:

listOfPairs – список всех получившихся пар

numberOfPair– номер пары, с которой продолжать поиск

currentLinkedPairsList – текущий вариант согласования предложений

bitCartForCorrectSentence - битовая карта занятости слов в правильном предложении
bitCartForVerifiedSentence - битовая карта занятости слов в правильном предложении

Выходные данные:
linkedSentences - в поле класса записываются все получившиеся варианты согласования

Алгоритм функции:
1. Считать, что цикл не заходил в рекурсию
2. Для каждой пары
    2.1 Если ни одно из слов данной пары (включая вспомогательные) не содержится в текущем сочетании пар
        2.1.1 Добавить пару к текущему сочетанию пар
        2.1.2 Отметить на картах занятости слова из пары как использованные
        2.1.3 Составить с текущем сочетанием пар все возможные последующие сочетания пар
        2.1.4 Считать, что цикл заходил в рекурсию
        2.1.5 Удалить пару из текущего сочетания пар
        2.1.6 Отметить на картах занятости слова из пары как неиспользованные

3. Если цикл не заходил в рекурсию
    3.1 Если в текущем сочетании пар во втором предложении задействовано больше слов, чем в сохраненном решении ИЛИ задействовано столько же слов и меньше ошибочность
        3.1.1 Считать данное данное сочетание пар единственным в решении

3.2 ИначеЕсли в текущем сочетании пар во втором предложении задействовано столько же слов, сколько во всех решениях И такая же ошибочность

3.2.1 Добавить данное сочетание пар к решению

Считает количество слогов в слове
int Word::countSyllables()

Считает количество слогов в слове
int Word::countSyllables()

Входные данные:
word – поле класса, слово, в котором необходимо подсчитать количество слогов

Выходные данные:
int countOfSyllables – количество слогов в слове
Алгоритм:
... Гласными считаются буквы a, e, i, o, u
1. Считать количество слогов 0
2. Считать, что предыдущий символ не был гласной
3. Для каждой буквы в слове
3.1 Если текущая буква - гласная ИЛИ y, которая является гласной (до нее согласная)
3.1.1 Если предыдущая буква не была гласной
3.1.1.1 Инкрементировать счетчик слогов
3.1.1.2 Считать, что предыдущая буква была гласной
3.2 Иначе
3.2.1 Считать, что предыдущая буква была не была гласной

4. Если последняя буква е И предпоследняя буква не е

   4.1 Уменьшить количество слогов на 1

5. ИначеЕсли последняя буква у и предпоследняя не является гласной

   5.1 Увеличить количество слогов на 1

6. Вернуть максимальное из количества слогов в слове и единицы

Образует все формы прилагательного

QList<Word> Adjective::formAllForms()

Входные данные:

word – слово в элементе данных класса

Выходные данные:

QList<Word> allFormsOfWord – все формы данного слова

Алгоритм функции:

1. Образовать начальную форму прилагательного
2. Добавить слово в список
3. Образовать сравнительную форму прилагательного
4. Добавить слово в список
5. Образовать превосходную форму прилагательного
6. Добавить слово в список
7. Вернуть получившийся список

Образует начальную форму прилагательного

Adjective Adjective::formInitial()

Входные данные:

word – слово в элементе данных класса

Выходные данные:

Adjective initial – начальная форма данного слова

Алгоритм:

1. Образовать начальную форму прилагательного
2. Присвоить данной форме в первое поле метку positive
3. Присвоить в четвертое поле метку notNeed

Образует сравнительную форму прилагательного

Adjective Adjective::formComparative(const QString &adjective)

Входные данные:

adjective - начальная форма прилагательного

Выходные данные:

Adjective comparativeForm - сравнительная форма прилагательного

Алгоритм:

1. Если слово исключение
   1.1 Сравнительная форма данного слова - слово из 2 столбца таблицы исключений
   1.2 Присвоить в четвертое поле метку more
2. ИначеЕсли прилагательное имеет 1 слог и заканчивается на -e
   2.1 Добавить к нему окончание -r
3. ИначеЕсли односложное прилагательное заканчивается сочетанием согласная + гласная + согласная
   3.1 Удвоить последнюю букву
   3.2 Добавить окончание -er
4. ИначеЕсли прилагательное односложное

4.1 Добавить к нему окончание -er

5. ИначеЕсли в прилагательном 2 слога и последняя буква -y

5.1 Заменить -y на -i

5.2 Добавить окончание -er

6. Иначе

6.1 Слово не меняется

7. Присвоить получившемуся слову в первое поле метку comparative

8. Вернуть слово

Образует превосходную степень прилагательного

Adjective Adjective::form Superlative(const QString &adjective)

Входные данные:

adjective - начальная форма прилагательного

Выходные данные:

Adjective superlativeForm

Алгоритм функции:

1. Если слово исключение

1.1 Превосходная форма данного слова - слово из 3 столбца таблицы исключений

1.2 Присвоить в четвертое поле метку most

2. ИначеЕсли прилагательное имеет 1 слог и заканчивается на -e

2.1 Добавить к нему окончание -st

3. ИначеЕсли односложное прилагательное заканчивается сочетанием согласная + гласная + согласная

3.1 Удвоить последнюю букву

3.2 Добавить окончание -est

4. ИначеЕсли прилагательное односложное

4.1 Добавить к нему окончание -est

5. ИначеЕсли в прилагательном 2 слога и последняя буква -y

5.1 Заменить -y на -i

5.2 Добавить окончание -est

6. Иначе

6.1 Слово не меняется

7. Присвоить получившемуся слову в первое поле метку superlative

Считает ошибочность

int Adjective::calculateFallacy(Word& other)

Входные данные:

word – текущее слово, поле класса

other – слово, с которым сравниваем

Выходные данные:

int fallacy – ошибочность пары

Алгоритм:

1. Если формы не совпадают

1.1 Прибавить к ошибочности 100

Образует все формы существительного

QList<Word> Noun::formAllForms()

Входные данные:

word – существительное в элементе данных класса

Выходные данные:

QSet<Word> allFormsOfWord – все формы данного слова

Алгоритм функции:

1. Если слово имеет только множественную форму
   1.1 Присвоить ему метку plural
   1.2 Присвоить ему метку objectiveCase
   1.3 Добавить слово в список
   1.4 Образовать форму притяжательного падежа множественного числа
   1.5 Добавить ее в список
2. ИначеЕсли слово неисчисляемое
   2.1 Присвоить ему метку incountable
   2.2 Образовать форму притяжательного падежа
3. Иначе
   3.1 Образовать начальную форму
   3.2 Присвоить ей метку objectiveCase
   3.3 Присвоить ей метку singular
   3.4 Образовать форму множественного числа
   3.5 Добавить ее в список
   3.6 Образовать форму притяжательного падежа единственного числа
   3.7 Добавить ее в список
   3.8 Образовать форму притяжательного падежа множественного числа
   3.9 Добавить ее в список

Образует начальную форму существительного

Noun formInitial()

Входные данные:

noun - поле класса, существительное

Выходные данные:

Noun initial - начальная форма слова

Алгоритм:

1. Удалить все символы после апострофа, включая его, если он имеется
2. Образовать начальную форму

Образует множественную форму существительного

Noun Noun::formPlural(const QString &noun)

Входные данные:

noun - существительное в начальной форме

Выходные данные:

Noun plural - существительное во множественной форме

Алгоритм:

1. Если слово является исключением
   1.1 Взять из таблицы форму его множественного числа
2. Иначе
   2.1 Если существительное оканчивается на -s, -ss, -ch, -sh, -x, -z
      2.1.1 Добавить к слову окончание -es
   2.2 Если существительное оканчивается на согласный + -y
      2.2.1 Заменить -y на -i
      2.2.2 Добавить к слову окончание -es
   2.3 Если существительное оканчивается на -f или -fe
      2.3.1 Заменить -f/-fe на -v
      2.3.2 Добавить к слову окончание -es

2.4 Иначе

2.4.1 Добавить к слову окончание -s

3. Присвоить слову в первое поле метку plural

4. Вернуть получившееся слово

Образует притяжательный падеж существительного

Noun Noun::formCaseOfNoun(const QString &noun)

Входные данные:

noun - существительное

Выходные данные:

Noun possesiveNoun - существительное в притяжательном падеже

Алгоритм:

1. Если существительное во множественном числе и оно существует не только во множественном числе

1.1 Добавить к существительному '

2. Иначе

2.1 Добавить к существительному 's

3. Присвоить слову во второе поле метку possesiveCase

Считает ошибочность

int Noun::calculateFallacy(Word& other)

Входные данные:

word – текущее слово, поле класса

other – слово, с которым сравниваем

Выходные данные:

int fallacy – ошибочность пары

Алгоритм:

1. Если не совпадает число
   1.1 Прибавить к ошибочности 50
2. Если не совпадает падеж
   2.1 Прибавить к ошибочности 50

Образует все формы числительного

QList<Word> Numeral::formAllForms()

Входные данные:

word – слово в элементе данных класса

Выходные данные:

QSet<Word> allFormsOfWord – все формы данного слова

Алгоритм:

1. Образовать количественную форму числительного
2. Добавить ее в список
3. Образовать порядковую форму числительного
4. Добавить ее в список

Образует количественное числительное

Numeral Numeral::formCardinal()

Входные данные:

word – слово в элементе данных класса

Выходные данные:

Numeral cardinalNumeral – количественное числительное

Алгоритм:

1. Если слово заканчивается на th или оно first, second, third
   1.1 Если слово first
      1.1.1 Его количественная форма one
   1.2 ИначеЕсли слово second
      1.2.1 Его количественная форма two
   1.3 ИначеЕсли слово third
      1.3.1 Его количественная форма three
   1.4 ИначеЕсли слово заканчивается на -fth
      1.4.1 Заменить -fth на -ve
   1.5 ИначеЕсли слово заканчивается на -ieth
      1.5.1 Заменить -ieth на -y
   1.6 ИначеЕсли слово ninth
      1.6.1 Заменить его на nine
   1.7 Иначе
      1.7.1 Удалить окончание -th
2. Присвоить слову в первое поле метку cardinal
3. Вернуть слово

Образует порядковое числительное

Numeral Numeral::formOrdinal(const QString &numeral)

Входные данные:

numeral – количественное числительное

Выходные данные:

Numeral ordinalNumeral – порядковое числительное

Алгоритм:

1. Если слово one
   1.1 Его количественная форма first
2. ИначеЕсли слово two
   2.1 Его количественная форма second
3. ИначеЕсли слово three
   3.1 Его количественная форма third
4. ИначеЕсли слово заканчивается на -ve
   4.1 Заменить -ve на -fth
5. ИначеЕсли слово заканчивается на -y
   5.1 Заменить -y на -ieth
6. ИначеЕсли слово nine
   6.1 Заменить его на ninth
7. Иначе
   7.1 Добавить окончание -th
8. Добавить слову в первое поле метку ordinal
9. Вернуть слово

Считает ошибочность

int Numeral::calculateFallacy(Word& other)

Входные данные:

word – текущее слово, поле класса

other – слово, с которым сравниваем

Выходные данные:
int fallacy – ошибочность пары

Алгоритм:
1. Если формы не совпадают
   1.1 Прибавить к ошибочности 100

Образует все формы местоимения
QList<Word> Pronoun::formAllForms()

Входные данные:
word – слово в элементе данных класса

Выходные данные:
QSet<Word> allFormsOfWord – все формы данного слова

Алгоритм:
1. Создать все формы местоимения
2. Перенести вперед все формы местоимения, совпадающие с текущим

Считает ошибочность
int Pronoun::calculateFallacy(Word& other)

Входные данные:
word – текущее слово, поле класса
other – слово, с которым сравниваем

Выходные данные:
int fallacy – ошибочность пары

Алгоритм:

1. Если не совпадает число
   1.1 Прибавить к ошибочности 40
2. Если не совпадает падеж
   2.1 Прибавить к ошибочности 30
3. Если не совпадает форма
   3.1 Прибавить к ошибочности 30

Образует все формы глагола

QList<Word> Verb::formAllForms()

Входные данные:

word – слово в элементе данных класса

Выходные данные:

QSet<Word> allFormsOfWord – все формы данного слова

Алгоритм:

1. Образовать начальную форму глагола
2. Добавить ее в список с пометкой infinitive
3. Если глагол be
   3.1 Образовать все формы глагола be
4. Иначе
   4.1 Образовать форму глагола 3-го лица
   4.2 Добавить ее в список
   4.3 Образовать форму preterite
   4.4 Добавить ее в список
   4.5 Образовать форму pastParticiple

4.6 Добавить ее в список

4.7 Образовать форму presentParticiple

4.8 Добавить ее в список

5. Вернуть получившийся список

Образовать все формы глагола be

QList<Verb> Verb::formAllFormsOfVerbBe()

Выходные данные:

QList<Verb> allFormsOfVerbBe - все формы глагола be

Алгоритм:

1. Образовать формы первого и второго лица: am, are
2. Присвоить им во первое поле метку presentFirstOrSecondPerson
3. Образовать форму третьего лица: is
4. Присвоить ей во первое поле метку present3P
5. Образовать формы preterite: was, were
6. Присвоить им в третье поле метку preterite
7. Образовать форму прошедшего времени: been
8. Присвоить ей в третье поле метку pastParticiple
9. Образовать форму presentParticiple: being
10. Присвоить ей в третье поле метку presentParticiple

Образовать глагол третьего лица

Verb Verb::formThirdPersonVerb(const QString &verb)

Входные данные:

verb - глагол в начальной форме

Выходные данные:

Verb thirdPersonVerb - глагол в 3 лице

Алгоритм функции:

1. Если глагол оканчивается на (s, ss, sh, ch, x, o)
   1.1 Добавить к его окончанию -es
2. ИначеЕсли глагол оканчивается на -y с предшествующей согласной
   2.1 Заменить -y на -i
   2.2 Добавить к его окончанию -es
3. Иначе
   3.1 Добавить -s к окончанию
4. Присвоить глаголу в третье поле метку present3P
5. Вернуть получившийся глагол

Образовать форму претерита глагола

Verb Verb::formPreterite(const QString &verb)

Входные данные:

verb - глагол

Выходные данные:

Verb preteriteVerb - глагол в форме претерита

Алгоритм функции:

1. Если глагол находится в таблице неправильных
   1.1 Считать его формой претерита форму из 2-го столбца неправильных глаголов
2. Иначе
   2.1 Образовать форму правильного глагола прошедшего времени

3. Присвоить глаголу в третье поле метку preterite
4. Вернуть получившийся глагол

Образовать форму past participle
Verb Verb::formPastParticiple(const QString &verb)

Входные данные:
verb - глагол

Выходные данные:
Verb pastParticipleVerb - глагол в форме past participle

Алгоритм функции:
1. Если глагол находится в таблице неправильных
   1.1 Считать его формой претерита форму из 3-го столбца неправильных глаголов
2. Иначе
   2.1 Образовать форму правильного глагола прошедшего времени
3. Присвоить глаголу в третье поле метку pastParticiple
4. Вернуть получившийся глагол

Образует форму прошедшего времени правильного глагола
Verb Verb::formPastRightVerb(const QString &verb)

Входные данные:
verb - глагол

Выходные данные:
Verb pastRightVerb - глагол в прошедшем времени

Алгоритм функции:

1. Если слово заканчивается на e
   1.1 Добавить к нему окончание -d
2. ИначеЕсли в слове 1 слог и оно оканчивается на согласная-гласная-согласная
   2.1 Удвоить последнюю букву
   2.2 Добавить к нему окончание -ed
3. ИначеЕсли заканчивается на согласную + y
   3.1 Заменить окончание -y на окончание -ied
4. Иначе
   4.1 Добавить окончание -ed
5. Присвоить глаголу в третье поле метку past
6. Вернуть получившийся глагол

Образует форму present participle глагола
Verb Verb::formPresentParticiple(const QString &verb)

Входные данные:
verb - глагол

Выходные данные:
Verb presentParticipleVerb - глагол в форме present participle

Алгоритм функции:

1. Если слово заканчивается на e
   1.1 Заменить окончание -e на окончание -ing
2. ИначеЕсли слово (оканчивается на согласная-гласная-согласная И оно не cover И оно не remember) ИЛИ (оканчивается на гласную + согласную L)

2.1 Удвоить последнюю букву

2.2 Добавить к слову окончание -ing

3. Иначе

3.1 Добавить к слову окончание -ing

4. Присвоить глаголу в третье поле метку presentParticiple

Считает ошибочность

int Verb::calculateFallacy(Word& other)

Входные данные:

word – текущее слово, поле класса

other – слово, с которым сравниваем

Выходные данные:

int fallacy – ошибочность пары

Алгоритм:

1. ...Считать ошибочность пары 0
2. Если не совпадают лица глагола

2.1 Прибавить к ошибочности 20

3. Если времена совпадают

3.1 Вернуть степень ошибочности

4. ИначеЕсли времена Present Simple И Present Continuous

4.1 Прибавить к ошибочности 30

5. ИначеЕсли времена Past Simple И Present Perfect

5.1 Прибавить к ошибочности 31

6. ИначеЕсли времена Past Simple И Past Continuous

6.1 Прибавить к ошибочности 32

7. ИначеЕсли времена Present Perfect И Present Perfect Continuous

7.1 Прибавить к ошибочности 33

8. ИначеЕсли времена Future Simple И Future Continuous

8.1 Прибавить к ошибочности 34

9. ИначеЕсли времена Future Simple И Present Simple

9.1 Прибавить к ошибочности 35

10. ИначеЕсли времена Past Simple И Past Perfect

10.1 Прибавить к ошибочности 50

11. ИначеЕсли времена Present Perfect И Past Perfect

11.1 Прибавить к ошибочности 50

12. ИначеЕсли времена Past Continuous И Past Perfect Continuous

12.1 Прибавить к ошибочности 50

13. ИначеЕсли времена Future Perfect И Future Perfect Continuous

13.1 Прибавить к ошибочности 50

14. ИначеЕсли времена Present Perfect Continuous И Past Perfect Continuous

14.1 Прибавить к ошибочности 50

15. ИначеЕсли времена Past Perfect И Past Perfect Continuous

15.1 Прибавить к ошибочности 50

16. ИначеЕсли времена Future Continuous И Future Perfect Continuous

16.1 Прибавить к ошибочности 50

17. ИначеЕсли времена Present Simple И Past Simple

17.1 Прибавить к ошибочности 50

18. ИначеЕсли времена Future Perfect И Future Simple

18.1 Прибавить к ошибочности 50

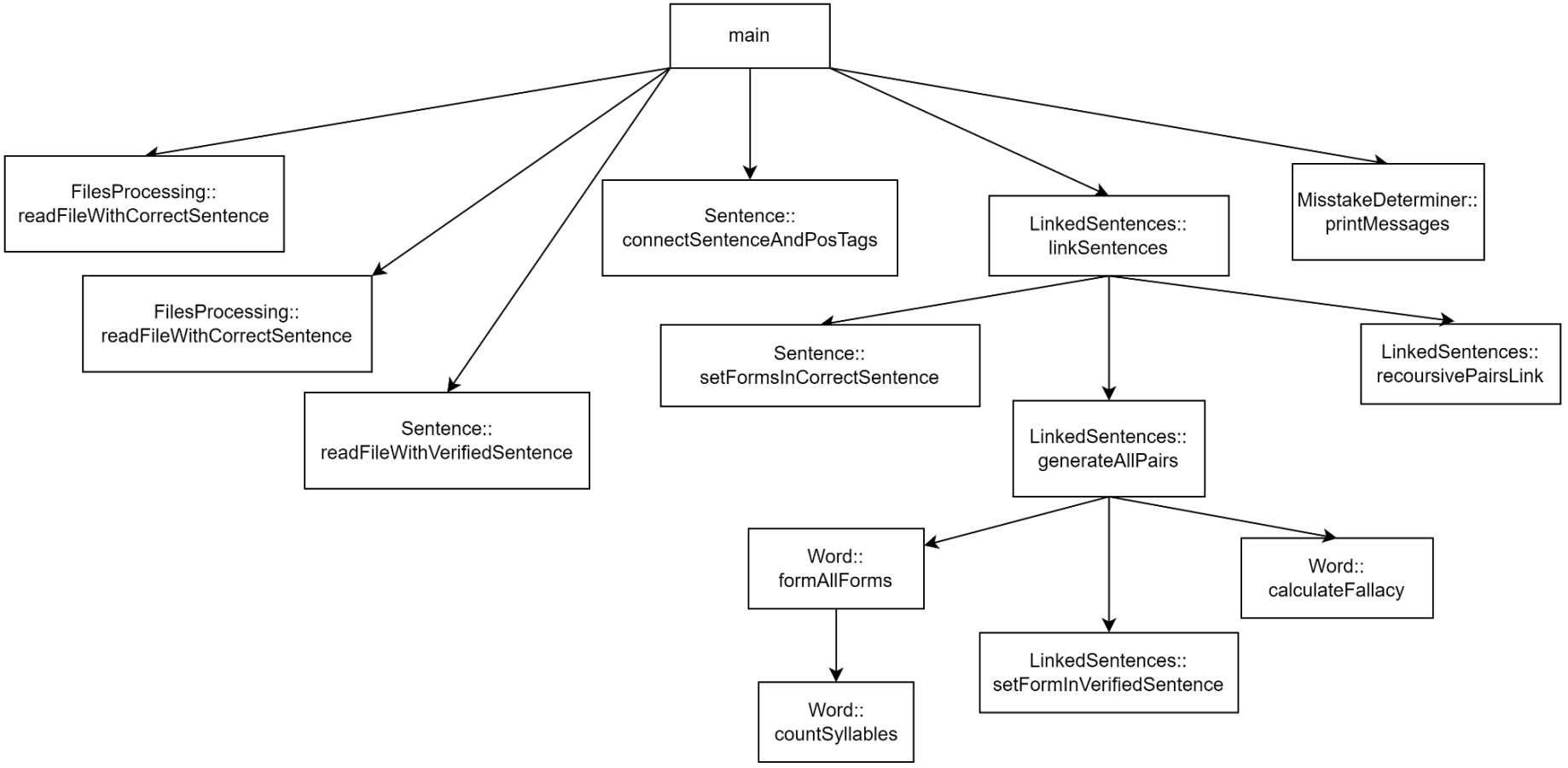
19. ИначеЕсли времена Present Perfect Continuous И Future Perfect Continuous

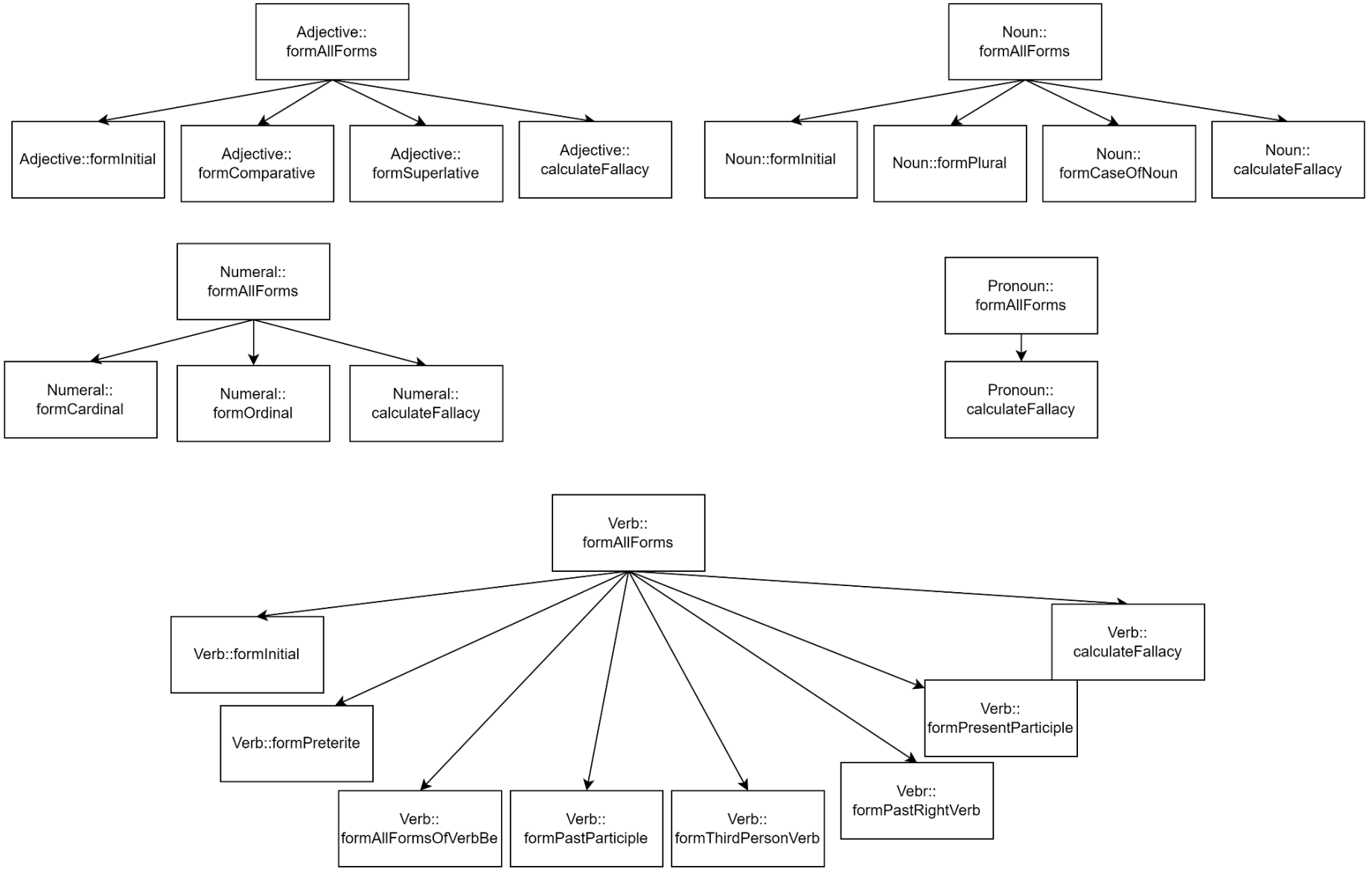
19.1 Прибавить к ошибочности 50

20. Иначе

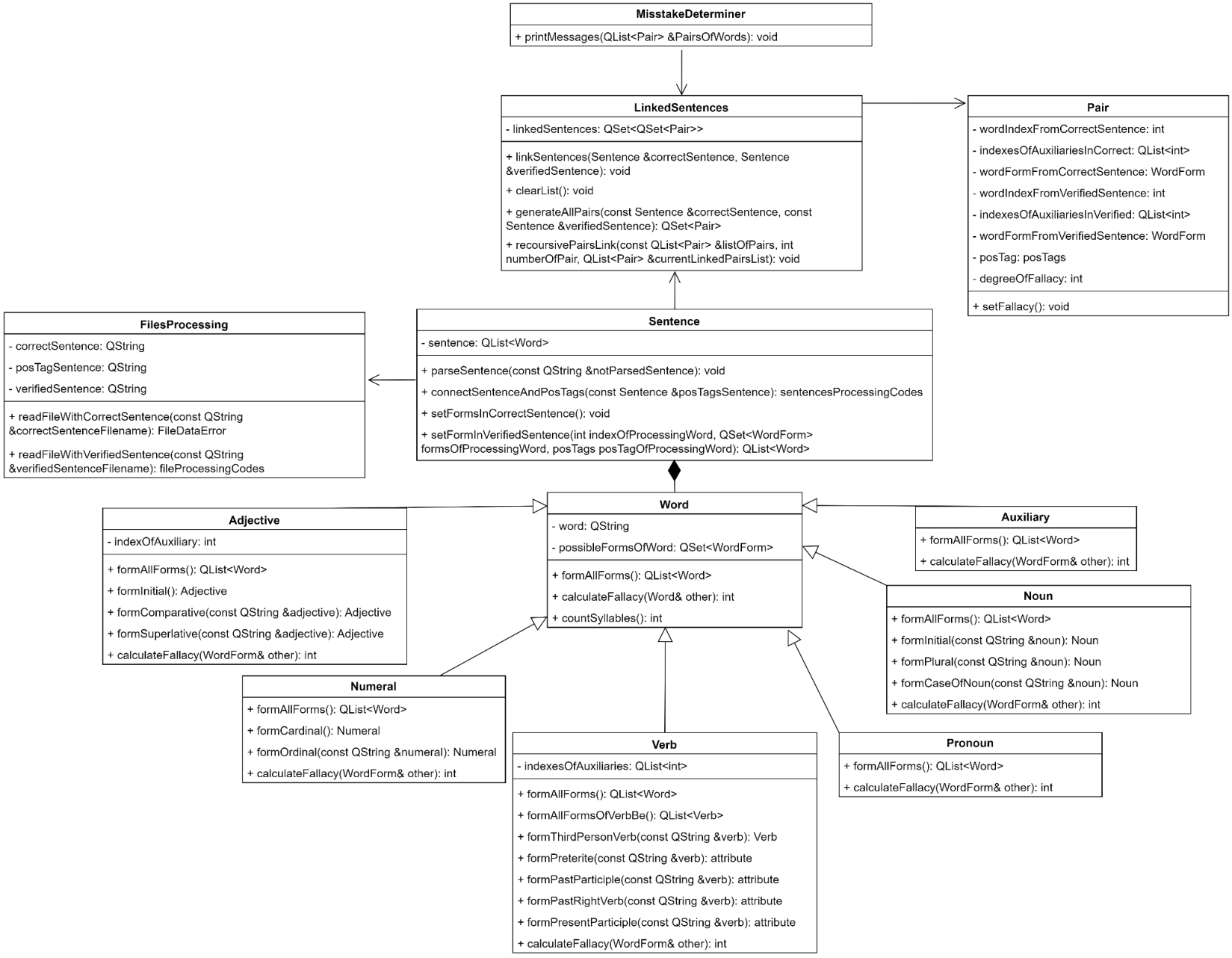
20.1 Прибавить к ошибочности 80

5. Диаграмма вызовов функций




Adjective::
formAllForms
Adjective::formInitial
Adjective::
formComparative
Adjective::
formSuperlative
Adjective::
calculateFallacy
Noun::
formAllForms
Noun::formInitial
Noun::formPlural
Noun::
formCaseOfNoun
Noun::
calculateFallacy
Numeral::
formAllForms
Numeral::
formCardinal
Numeral::
formOrdinal
Numeral::
calculateFallacy
Pronoun::
formAllForms
Pronoun::
calculateFallacy
Verb::
formAllForms
Verb::formInitial
Verb::formPreterite
Verb::
formAllFormsOfVerbBe
Verb::
formPastParticiple
Verb::
formThirdPersonVerb
Vebr::
formPastRightVerb
Verb::
formPresentParticiple
Verb::
calculateFallacy

6. UML-диаграмма классов



7. Диаграмма потоков данных