Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №2

По дисциплине: «Естественно-языковой интерфейс ИС»

Тема: «Построение и использование корпусов текстов естественного языка»

Выполнил:

Студент 3 курса

Группы ИИ-23

Романюк А. П.

Проверил:

Булей Е. В.

Брест 2025

Ход работы

Задание: Разработка корпусного менеджера

Разработать приложение, которое предоставляет пользователю возможность обрабатывать фрагменты текста на естественном языке, запрашивая различные частотные и морфологические характеристики словоформ.

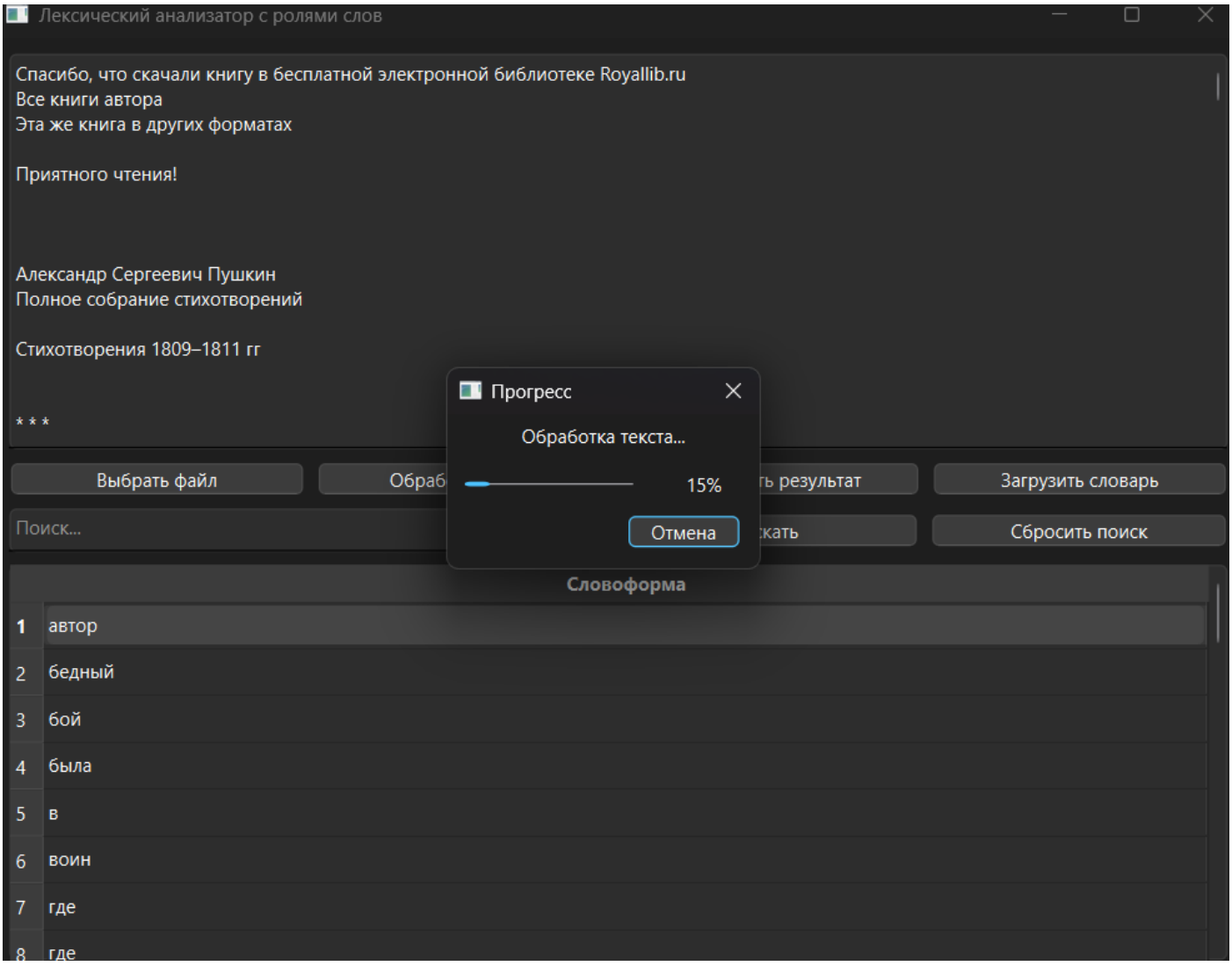
Требования:

Входные данные: Пользователь вводит фрагмент текста (фразу или слово) на естественном языке в качестве запроса к корпусному менеджеру.

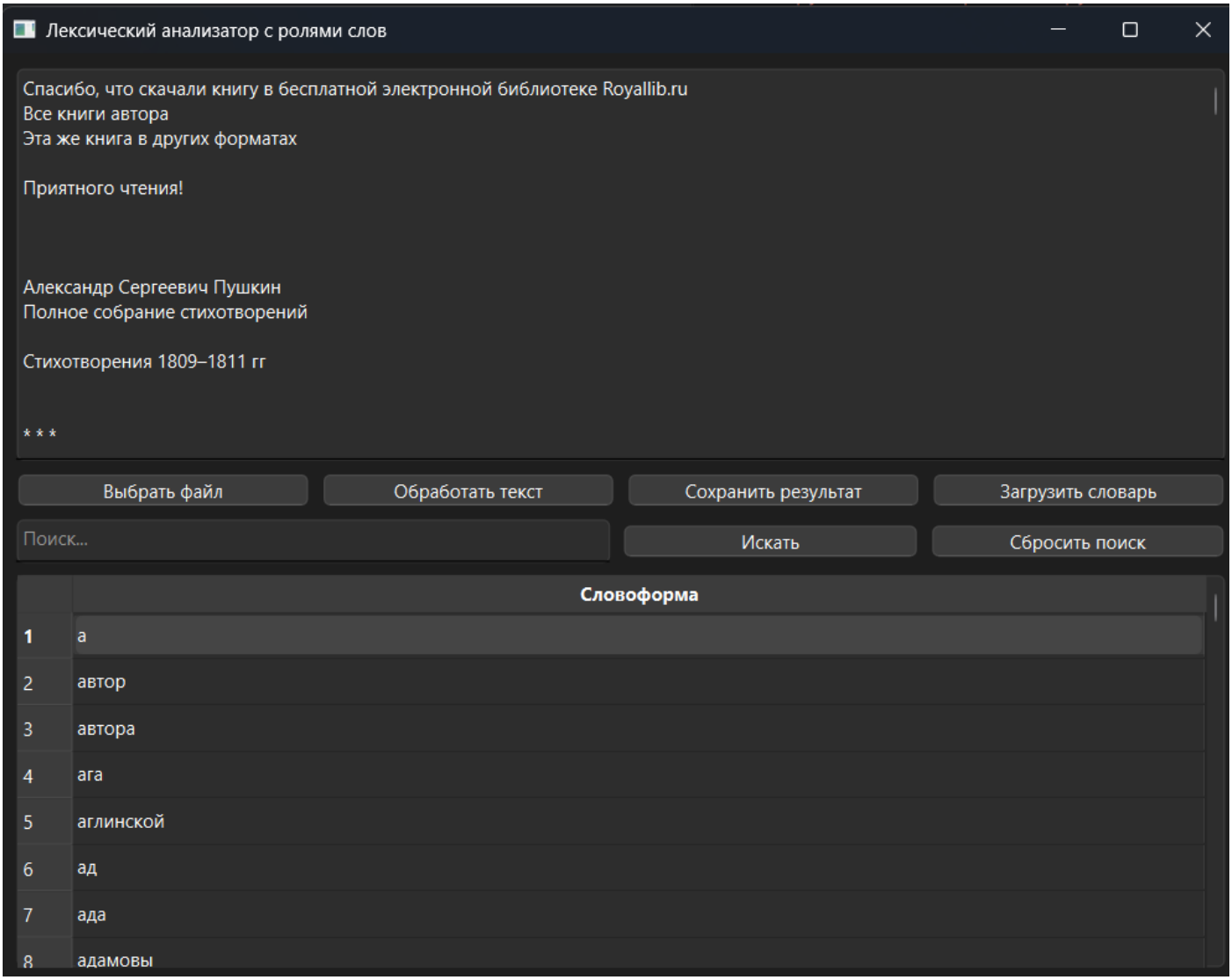
Выходные данные: Приложение предоставляет следующие выходные данные:

- Частотные характеристики словоформ и лексем.
- Грамматические категории.
- Леммы слов.
- Морфологические характеристики словоформ.

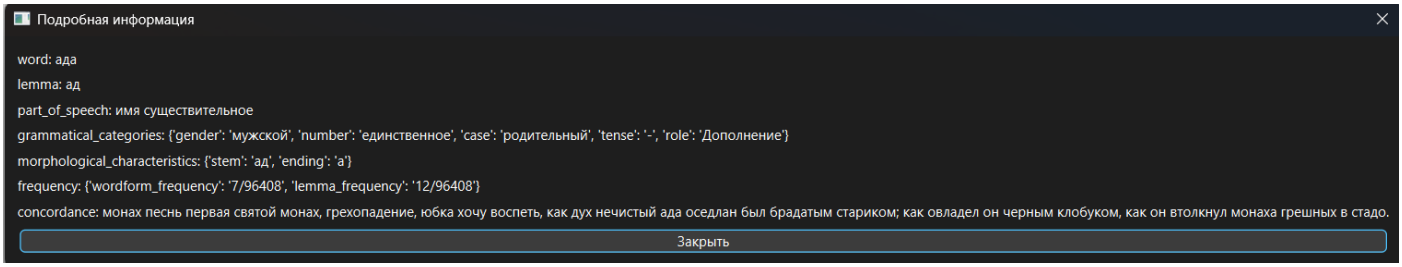
Взаимодействие с пользователем: Пользователь взаимодействует с приложением через графический интерфейс, который должен быть интуитивно-понятным и дружественным для пользователя.



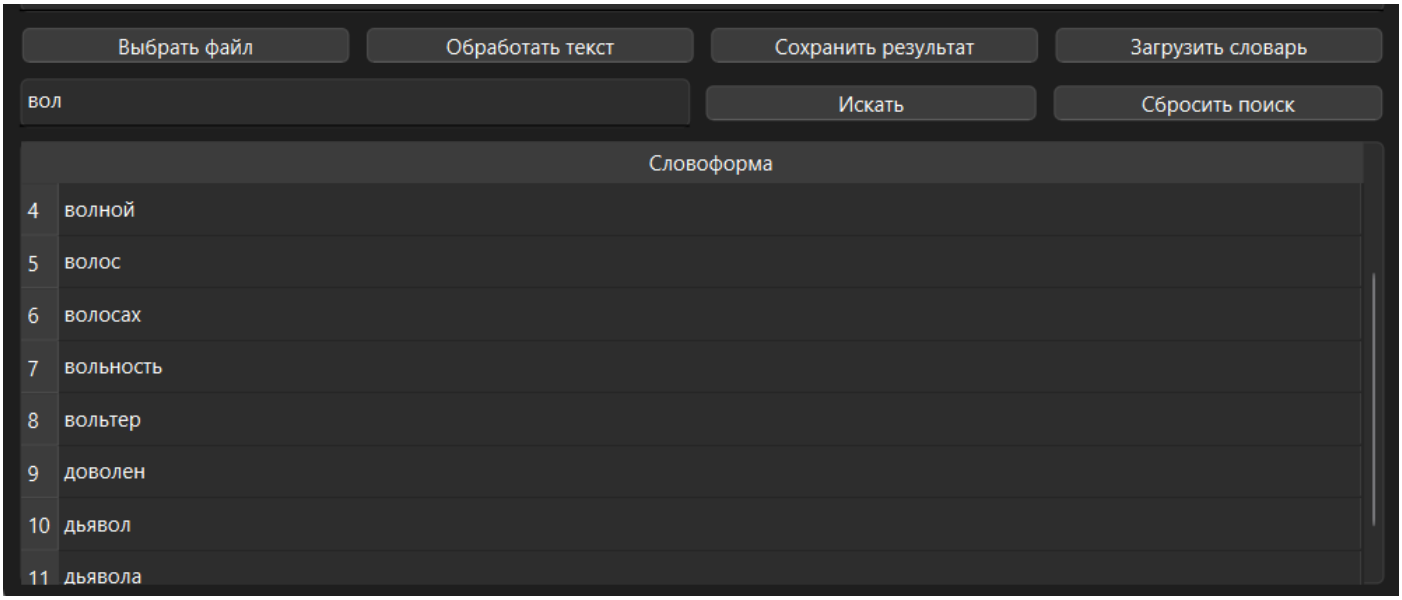
Процесс обработки корпуса текста



Приложение с загруженным корпусом текста



Просмотр характеристик слова



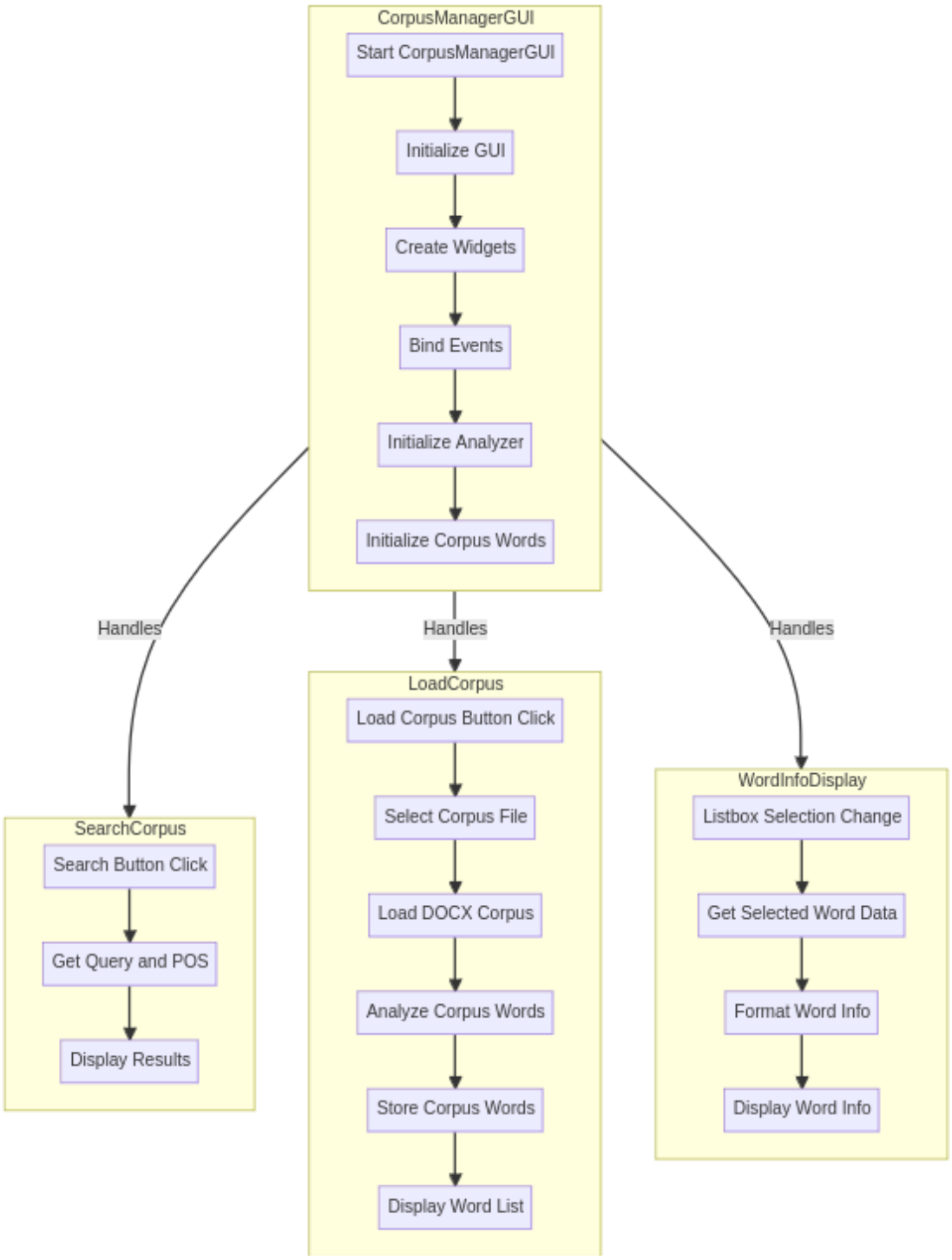
Поиск по вхождению подстроки в лексеме

Код программы:

```
from collections import defaultdict
import re
import pymorphy3
from PySide6.QtCore import Qt
from PySide6.QtWidgets import QProgressDialog

class TextProcessor:
    def __init__(self, parent=None):
        self.morph = pymorphy3.MorphAnalyzer()
        self.parent = parent  # Для взаимодействия с UI

    @staticmethod
    def clean_text(text):
        """Удаляет лишние пробельные символы (\n, \t и т. д.)"""
        return re.sub(r'\s+', ' ', text).strip()

    def parse_word(self, word):
        """Кэшированный анализ слова"""
        return self.morph.parse(word)[0]

    def generate_concordance(self, word, corpus):
        """Генерирует первое вхождение конкордансного списка для слова"""
        corpus = self.clean_text(corpus)
        pattern = r'\b' + re.escape(word) + r'\b'
        sentences = re.split(r'(?<=[.!?])\s+', corpus)
        return next((s for s in sentences if re.search(pattern, s)), None)

    def process_text(self, words, corpus):
        """Обрабатывает текст без многопоточности, с прогресс-баром"""
        word_info_dict = {}  # Храним информацию о словах
        parsed_words = {}  # Сохраняем результаты морфологического анализа
        word_freq, lemma_freq = defaultdict(int), defaultdict(int)

        # Прогресс-бар
        progress = QProgressDialog("Обработка текста...", "Отмена", 0, len(words)*2, self.parent)
        progress.setWindowTitle("Прогресс")
        progress.setWindowModality(Qt.WindowModal)
        progress.show()

        # Первый проход: сохраняем результаты морфологического анализа и подсчитываем частоты
        for word in words:
            parsed_word = self.parse_word(word)
            parsed_words[word] = parsed_word  # Сохраняем результаты для каждого слова
            word_freq[word] += 1
            lemma_freq[parsed_word.normal_form] += 1
            progress.setValue(progress.value() + 1)

        # Второй проход: создаем информацию о словах
        for index, word in enumerate(words):
            if progress.wasCanceled():
                break

            parsed_word = parsed_words[word]  # Извлекаем результаты из кэша
```

```
            lemma = parsed_word.normal_form

            # Получаем частоту для слова и леммы
            wordform_freq = f"{word_freq[word]}/{len(words)}"
            lemma_freq_value = f"{lemma_freq[lemma]}/{len(words)}"

            if word in word_info_dict:
                word_info_dict[word]["frequency"]["wordform_frequency"] = wordform_freq
                word_info_dict[word]["frequency"]["lemma_frequency"] = lemma_freq_value
                progress.setValue(progress.value() + 1)
                continue

            concordance = self.generate_concordance(word, corpus)

            stem = parsed_word.normalized.word  # Основа (лемма)
            ending = word[len(stem):]

            word_info_dict[word] = {
                "word": word,
                "lemma": lemma,
                "part_of_speech": self.determine_pos(parsed_word.tag.POS),
                "grammatical_categories": {
                    "gender": self.determine_gender(parsed_word.tag.gender),
                    "number": self.determine_number(parsed_word.tag.number),
                    "case": self.determine_case(parsed_word.tag.case),
                    "tense": self.determine_tense(parsed_word.tag.tense),
                    "role": self.determine_role(parsed_word.tag.POS, parsed_word.tag.case,
parsed_word.tag.tense)
                },
                "morphological_characteristics": {
                    "stem": stem,
                    "ending": ending
                },
                "frequency": {
                    "wordform_frequency": wordform_freq,
                    "lemma_frequency": lemma_freq_value
                },
                "concordance": concordance
            }

            # Обновление прогресса
            progress.setValue(progress.value() + 1)

        progress.setValue(len(words)*2)
        return sorted(word_info_dict.values(), key=lambda x: x["word"])

    @staticmethod
    def determine_role(pos, case, tense):
        if pos == "NOUN":  # Существительное
            if case == "nomn":  # Именительный падеж
                return "Подлежащее"
            if case in [
                "gent",
                "datv",
                "accs",
                "ablt",
                "loct",
            ]:  # Родительный, дательный, винительный, творительный, предложный
                return "Дополнение"
        elif pos == "ADJF":  # Полное прилагательное
            return "Определение"
        elif pos == "ADJS":  # Краткое прилагательное
            return "Именная часть составного сказуемого"
        elif pos == "COMP":  # Компаратив
            return "Определение (сравнительная степень)"
        elif pos == "VERB":  # Глагол (личная форма)
            if tense == "pres" or tense == "futr":  # Настоящее или будущее время
                return "Сказуемое"
            if tense == "past":  # Прошедшее время
                return "Сказуемое (прошедшее время)"
        elif pos == "INFN":  # Инфинитив
            return "Сказуемое (инфинитив)"
        elif pos == "PRTF":  # Полное причастие
            return "Определение (причастный оборот)"
        elif pos == "PRTS":  # Краткое причастие
            return "Именная часть составного сказуемого"
        elif pos == "GRND":  # Деепричастие
            return "Обстоятельство (деепричастный оборот)"
        elif pos == "NUMR":  # Числительное
            if case == "nomn":  # Именительный падеж
                return "Подлежащее (числительное)"
            return "Дополнение (числительное)"
        elif pos == "ADVB":  # Наречие
```

```
            return "Обстоятельство"
        elif pos == "NPRO":  # Местоимение-существительное
            if case == "nomn":  # Именительный падеж
                return "Подлежащее (местоимение)"
            return "Дополнение (местоимение)"
        elif pos == "PRED":  # Предикатив
            return "Сказуемое (предикатив)"
        elif pos == "PREP":  # Предлог
            return "Предлог"
        elif pos == "CONJ":  # Союз
            return "Союз"
        elif pos == "PRCL":  # Частица
            return "Частица"
        elif pos == "INTJ":  # Междометие
            return "Междометие"
        else:
            return "-"

    @staticmethod
    def determine_pos(pos):
        """Переводит часть речи на русский"""
        pos_map = {
            "NOUN": "имя существительное",
            "ADJF": "имя прилагательное (полное)",
            "ADJS": "имя прилагательное (краткое)",
            "COMP": "компаратив",
            "VERB": "глагол (личная форма)",
            "INFN": "глагол (инфинитив)",
            "PRTF": "причастие (полное)",
            "PRTS": "причастие (краткое)",
            "GRND": "деепричастие",
            "NUMR": "числительное",
            "ADVB": "наречие",
            "NPRO": "местоимение-существительное",
            "PRED": "предикатив",
            "PREP": "предлог",
            "CONJ": "союз",
            "PRCL": "частица",
            "INTJ": "междометие",
        }
        return pos_map.get(pos, "-")

    @staticmethod
    def determine_case(case):
        """Переводит падеж на русский"""
        case_map = {
            "nomn": "именительный",
            "gent": "родительный",
            "datv": "дательный",
            "accs": "винительный",
            "ablt": "творительный",
            "loct": "предложный",
            "voct": "звательный",
            "gen2": "второй родительный",
            "acc2": "второй винительный",
            "loc2": "второй предложный",
        }
        return case_map.get(case, "-")

    @staticmethod
    def determine_number(number):
        """Переводит число на русский"""
        number_map = {"sing": "единственное", "plur": "множественное"}
        return number_map.get(number, "-")

    @staticmethod
    def determine_tense(tense):
        """Переводит время на русский"""
        tense_map = {"pres": "настоящее", "past": "прошедшее", "futr": "будущее"}
        return tense_map.get(tense, "-")

    @staticmethod
    def determine_gender(gender):
        """Определяет род на русском"""
        gender_map = {
            "masc": "мужской",
            "femn": "женский",
            "neut": "средний"
        }
        return gender_map.get(gender, "неизвестно")
```



Вывод: в ходе выполнения лабораторной работы освоил принципы построения корпусов текстов, виды разметки и способы аннотирования, инструменты работы с корпусами текстов.