МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра автоматизированных систем управления



ОТЧЁТ

по КУРСОВОЙ РАБОТЕ

**«***Объявление комплексного числа с инициализацией на языке Python***»**

по дисциплине: **«***Теория формальных языков и компиляторов***»**

Выполнил:Проверил:

Киреенко М.А. *д.т.н., профессор*

Студент гр. «*АВТ-113*» *Шорников Юрий Владимирович*

«\_\_\_» \_\_\_\_\_\_ 20\_\_г.«\_\_\_» \_\_\_\_\_\_ 20\_\_ г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись) (подпись)

Новосибирск 2024

**РЕФЕРАТ**

Отчет 34 с., 16 рис., 8 источн., 3 прилож.

ЯЗЫКОВОЙ ПРОЦЕССОР, КОМПИЛЯТОР, ЛЕКСИЧЕСКИЙ АНАЛИЗ, СИНТАКСИЧЕСКИЙ АНАЛИЗ, АВТОМАТНАЯ ГРАММАТИКА, ГРАФ АТВОМАТНОЙ ГРАММАТИКИ, ДИАГНОСТИКА И НЕЙТРАЛИЗАЦИЯ ОШИБОК, МЕТОД АЙРОНСА

Цель работы – выполнить программную реализацию алгоритма синтаксического анализа комплексного числа с инициализацией на языке Python.

В результате проектирования был написан синтаксический анализатор (парсер) для комплексного числа с инициализацией на языке Python.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc69940111)

[1 Постановка задачи 5](#_Toc69940112)

[2 Разработка грамматики 6](#_Toc69940113)

[3 Классификация грамматики 7](#_Toc69940114)

[4 Метод анализа 8](#_Toc69940115)

[5 Диагностика и нейтрализация синтаксических ошибок 9](#_Toc69940116)

[5.1 Метод Айронса 9](#_Toc69940117)

[5.2 Метод Айронса для автоматной грамматики 9](#_Toc69940118)

[6 Тестовые примеры 11](#_Toc69940119)

[7 Листинг программы 14](#_Toc69940120)

[ЗАКЛЮЧЕНИЕ 15](#_Toc69940121)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 16](#_Toc69940122)

[Приложение А – Справка (руководство пользователя) 17](#_Toc69940123)

[Меню текстового редактора 17](#_Toc69940125)

[Пункт "Файл" меню текстового редактора 17](#_Toc69940126)

[Пункт "Правка" меню текстового редактора 17](#_Toc69940127)

[Пункт "Текст" меню текстового редактора 18](#_Toc69940128)

[Пункт "Пуск" меню текстового редактора 19](#_Toc69940129)

[Пункт "Справка" меню текстового редактора 19](#_Toc69940130)

[Пункт "Локализация" меню текстового редактора 20](#_Toc69940131)

[Пункт "Вид" меню текстового редактора 20](#_Toc69940132)

[Панель инструментов текстового редактора 20](#_Toc69940133)

[Дополнительные возможности текстового редактора 21](#_Toc69940134)

[Приложение Б – Информация о программе 23](#_Toc69940135)

[Приложение В – Листинг программы 25](#_Toc69940137)

# ВВЕДЕНИЕ

Цель курсовой работы – выполнить программную реализацию алгоритма объявления комплексного числа с инициализацией на языке Python.

Курсовая работа содержит следующие разделы:

* Постановка задачи;
* Грамматика;
* Классификация грамматики;
* Метод анализа;
* Диагностика и нейтрализация ошибок;
* Тестовый пример;
* Список литературы;
* Исходный код программы.

## 1 Постановка задачи

Комплексные числа представляют собой пару значений типа int или float, и имеют вид: <реальная часть>+<мнимая часть>

Для описания комплекса в языке Python используется служебное слово "complex".

Формат записи:

"имя\_идентификатора = complex(значение1,значение2)".

Примеры:

1. Целая десятичная константа – любое десятичное число без десятичной точки со знаком или без него: "z2 = complex(6,8.3)" В этом случае – это число 6.
2. Вещественная константа с фиксированной точкой – действительное число представленное в виде целой и дробной частей разделенных точкой: "z2 = complex(6,8.3)". В этом случае – это число 8.3.

В связи с разработанной автоматной грамматикой G[<КЧ>]  синтаксический анализатор (парсер) комплексных чисел будет считать верными следующие записи констант:

1. " z2 = complex(6,8.3) "
2. "z3 = complex(-1,+2.5) "
3. "z4 = complex(+5,-2.8) "

Справка (руководство пользователя) представлена в Приложении А. Информация о программе представлена в Приложении Б.

## 2 Разработка грамматики

Определим грамматику комплексного числа языка Python G[<КЧ>] в нотации Хомского с продукциями P:

1. <КЧ> -> letter COMPLEXREM
2. COMPLEXREM -> (letter | \_ | digit) COMPLEXREM | = ASSIGNTMENT
3. ASSIGNTMENT -> complex OPEN
4. OPEN -> ( REALPART
5. REALPART -> [+ | -] INTREALPART
6. INTREALPART -> digit0 INTREALPARTREM
7. INTREALPARTREM -> digit INTREALPARTREM | . DECIMALREALPART | , IMAGINARYPART
8. DECIMALREALPART -> digit DECIMALREALPARTREM
9. DECIMALREALPARTREM -> digit DECIMALREALPARTREM | , IMAGINARYPART
10. IMAGINARYPART -> [+ | -] INTIMAGINARYPART
11. INTIMAGINARYPART -> digit0 INTIMAGINARYPARTREM
12. INTIMAGINARYPARTREM -> digit INTIMAGINARYPARTREM | . DECIMALIMAGINARYPART
13. DECIMALIMAGINARYPART -> digit DECIMALIMAGINARYPARTREM
14. DECIMALIMAGINARYPARTREM -> digit DECIMALIMAGINARYPARTREM |  CLOSE
15. CLOSE -> )

* letter → ‘a’ | ‘b’ | … | ‘z’ | ‘A’ | ‘B’ | … | ‘Z’
* digit → ‘0’ | ‘1’ | … | ‘9’
* digit0 →  ‘1’ | … | ‘9’

Следуя введенному формальному определению грамматики, представим G[<КЧ>] ее составляющими:

* Z = <КЧ>;
* VT = { ‘a’…’z’, ‘A’…’Z’, '=', 'complex', '(', ')', '+',, '-', '0'...'9', '.', ‘\_’, ‘'‘ }
* VN = { <КЧ>, COMPLEXREM, ASSIGNTMENT, OPEN, REALPART, INTREALPART, INTREALPARTREM, DECIMALREALPART, IMAGINARYPART, DECIMALREALPARTREM, INTIMAGINARYPART, INTIMAGINARYPARTREM, DECIMALIMAGINARYPART, DECIMALIMAGINARYPARTREM, CLOSE}

## 3 Классификация грамматики

Согласно классификации Хомского, грамматика G[<КЧ>]  является автоматной.

Правила (1)-(14) относятся к классу праворекурсивных продукций (A → aB | a | ε):

1. <КЧ> -> letter COMPLEXREM
2. COMPLEXREM -> (letter | \_ | digit) COMPLEXREM | = ASSIGNTMENT
3. ASSIGNTMENT -> complex OPEN
4. OPEN -> ( REALPART
5. REALPART -> [+ | -] INTREALPART
6. INTREALPART -> digit0 INTREALPARTREM
7. INTREALPARTREM -> digit INTREALPARTREM | . DECIMALREALPART | , IMAGINARYPART
8. DECIMALREALPART -> digit DECIMALREALPARTREM
9. DECIMALREALPARTREM -> digit DECIMALREALPARTREM | , IMAGINARYPART
10. IMAGINARYPART -> [+ | -] INTIMAGINARYPART
11. INTIMAGINARYPART -> digit0 INTIMAGINARYPARTREM
12. INTIMAGINARYPARTREM -> digit INTIMAGINARYPARTREM | . DECIMALIMAGINARYPART
13. DECIMALIMAGINARYPART -> digit DECIMALIMAGINARYPARTREM
14. DECIMALIMAGINARYPARTREM -> digit DECIMALIMAGINARYPARTREM |  CLOSE

Правило (15) является простым: 15. CLOSE → )

Данная грамматика содержит только праворекурсивные продукции, и, следовательно, является полностью автоматной.

## 4 Метод анализа

Грамматика G[<КЧ>] является автоматной.

Правила (1) – (15) для G[<КЧ>] реализованы на графе (см. рисунок 1).

Сплошные стрелки на графе характеризуют синтаксически верный разбор; пунктирные символизируют состояние ошибки (ERROR); дуга λ и непомеченные дуги предполагают любой терминальный символ, отличный от указанного из соответствующего узла.

Состояние К символизирует успешное завершение разбора.

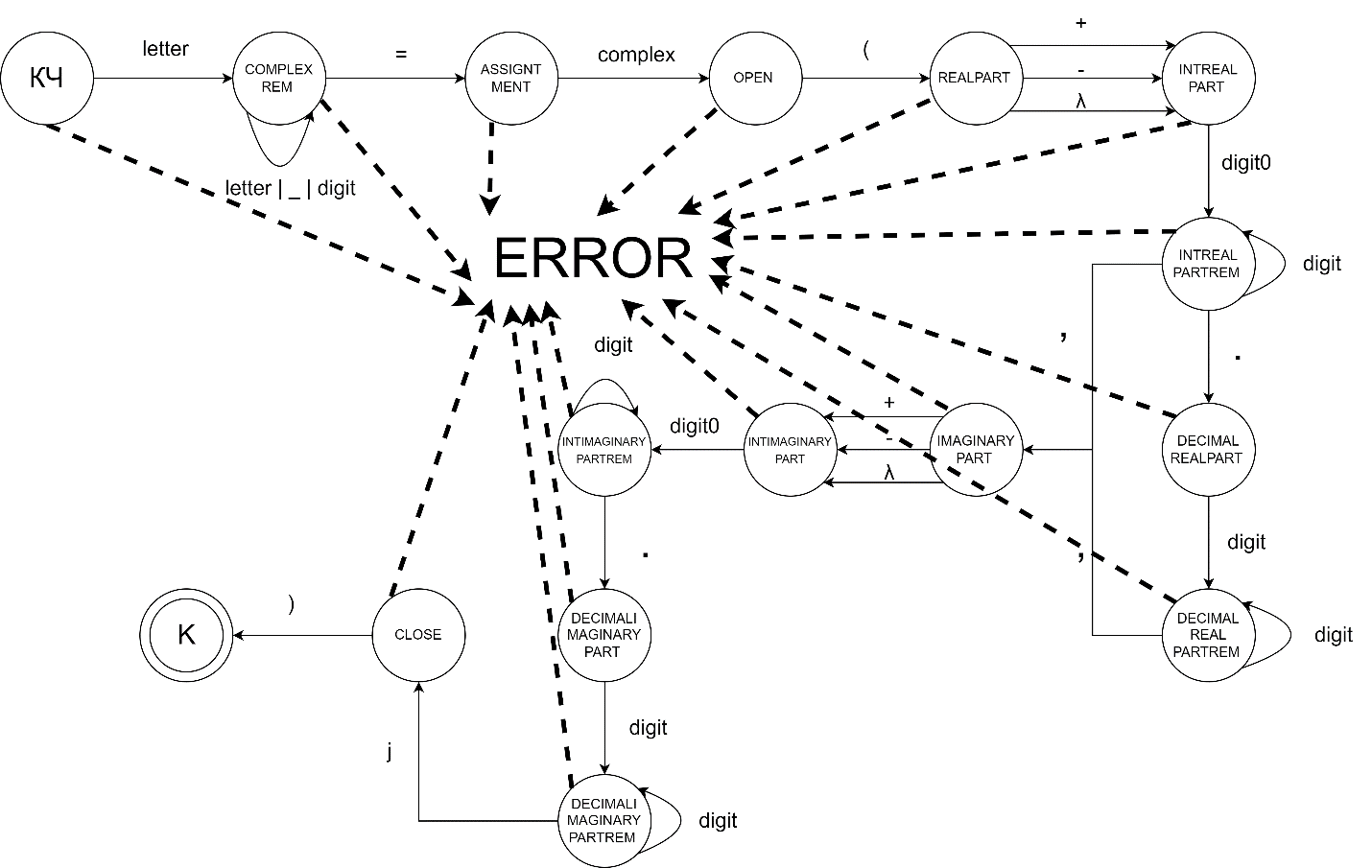


Рисунок 1 – Граф G[<КЧ>]

## 5 Диагностика и нейтрализация синтаксических ошибок

Согласно заданию на курсовую работу, необходимо реализовать нейтрализацию синтаксических ошибок, используя метод Айронса.

### 5.1 Метод Айронса

Суть метода Айронса заключается в следующем:

При обнаружении ошибки (во входной цепочке в процессе разбора встречается символ, который не соответствует ни одному из ожидаемых символов), входная цепочка символов выглядит следующим образом: Tt, где T – следующий символ во входном потоке (ошибочный символ), t – оставшаяся во входном потоке цепочка символов после T. Алгоритм нейтрализации состоит из следующих шагов:

1. Определяются недостроенные кусты дерева разбора;

2. Формируется множество L – множество остаточных символов недостроенных кустов дерева разбора;

3. Из входной цепочки удаляется следующий символ до тех пор, пока цепочка не примет вид Tt, такой, что U => T, где U ∈ L, то есть до тех пор, пока следующий в цепочке символ T не сможет быть выведен из какого-нибудь из остаточных символов недостроенных кустов.

4. Определяется, какой из недостроенных кустов стал причиной появления символа U в множестве L (иначе говоря, частью какого из недостроенных кустов является символ U).

Таким образом, определяется, к какому кусту в дереве разбора можно «привязать» оставшуюся входную цепочку символов после удаления из текста ошибочного фрагмента.

### 5.2 Метод Айронса для автоматной грамматики

Разрабатываемый синтаксический анализатор построен на базе автоматной грамматики. Реализация алгоритма Айронса для автоматной грамматики имеет следующую особенность.

Дерево разбора с использованием автоматной грамматики представлено на рисунке 2.

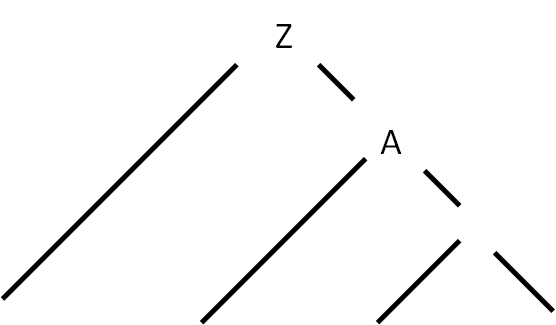


Рисунок 2 – Структура дерева разбора для автоматной грамматики

Таким образом, при возникновении синтаксической ошибки в процессе разбора с использованием автоматной грамматики, в дереве разбора всегда будет только один недостроенный куст (см. рисунок 3).

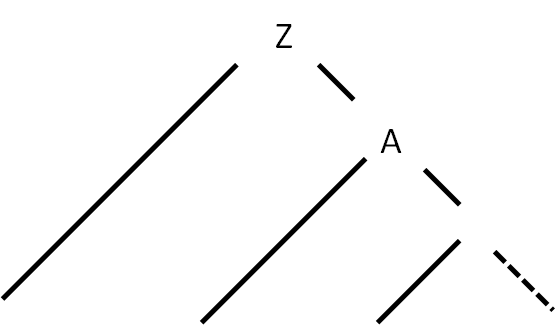


Рисунок 3 – Недостроенный куст при возникновении синтаксической ошибки (выделен пунктиром)

Поскольку единственный недостроенный куст – это тот, во время построения которого возникла синтаксическая ошибка, то это единственный куст, к которому можно привязать оставшуюся входную цепочку символов.

Предлагается свести алгоритм нейтрализации к последовательному удалению следующего символа во входной цепочке до тех пор, пока следующий символ не окажется одним из допустимых в данный момент разбора.

## 6 Тестовые примеры

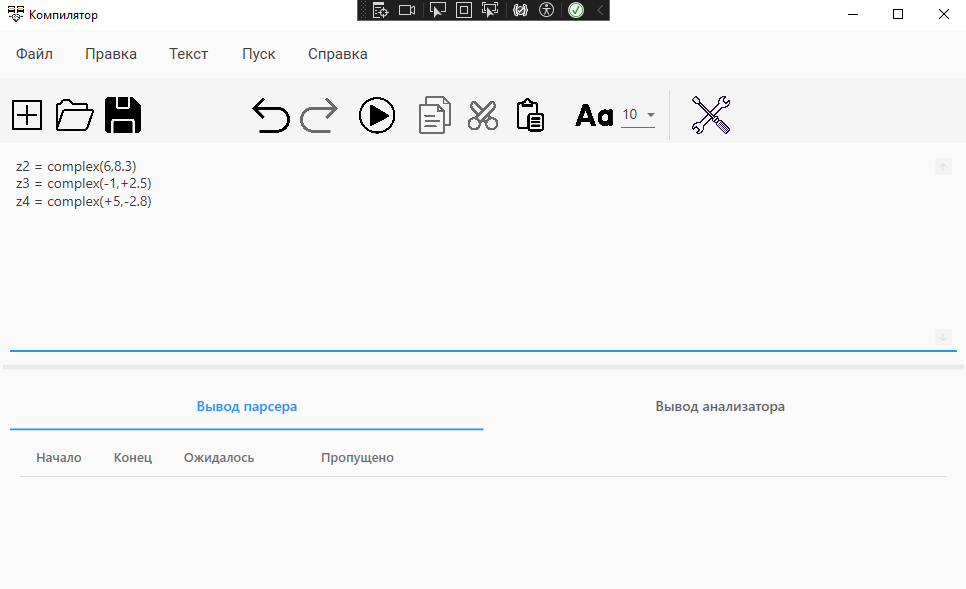
На рисунках 4-9 представлены тестовые примеры запуска разработанного синтаксического анализатора комплексного числа с инициализацией на языке Python.

Рисунок 4 – Тестовый пример 1 (Ввод корректен)

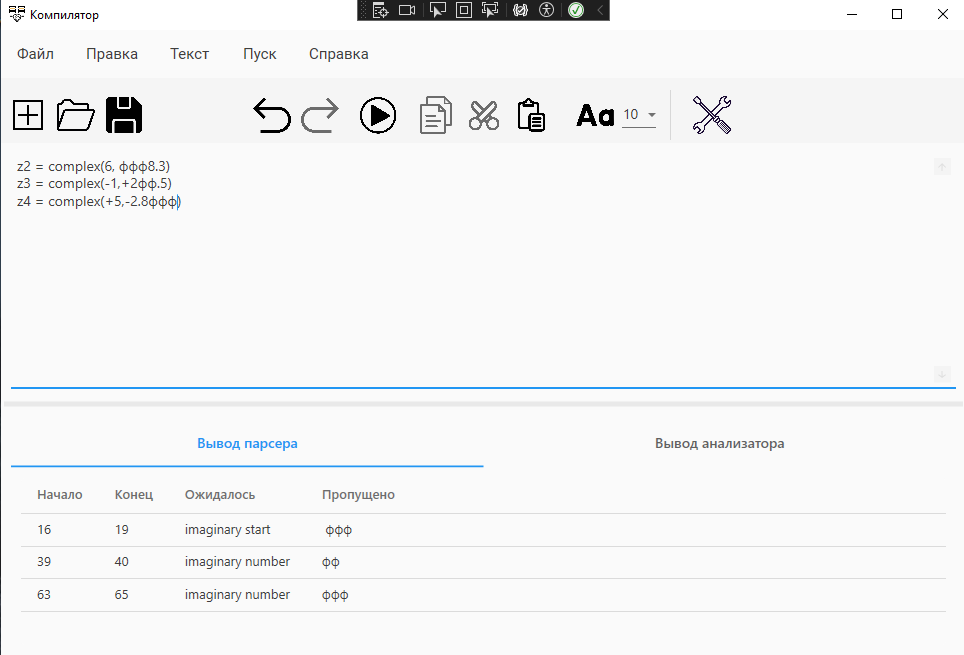


Рисунок 5 – Тестовый пример 2

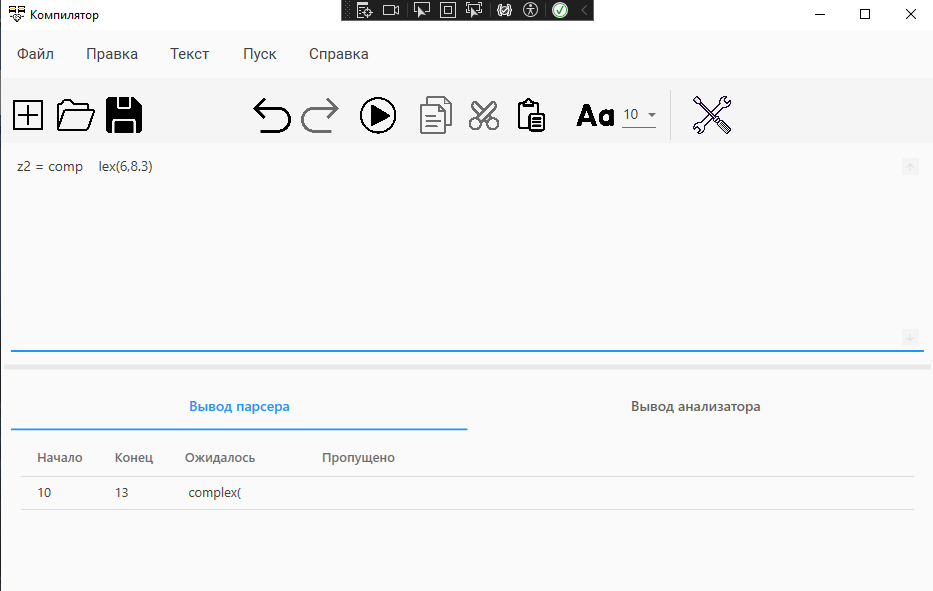


Рисунок 6 – Тестовый пример 3

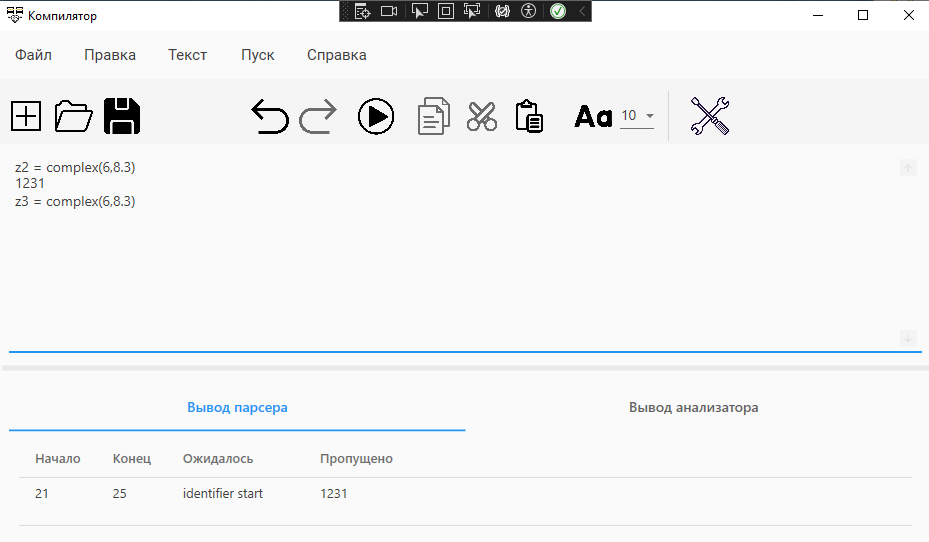


Рисунок 7 – Тестовый пример 4

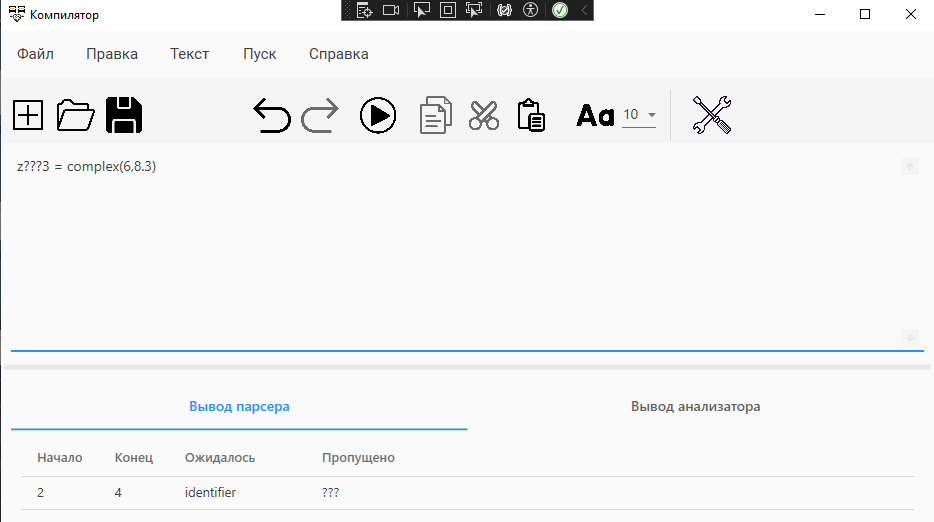


Рисунок 8 – Тестовый пример 5

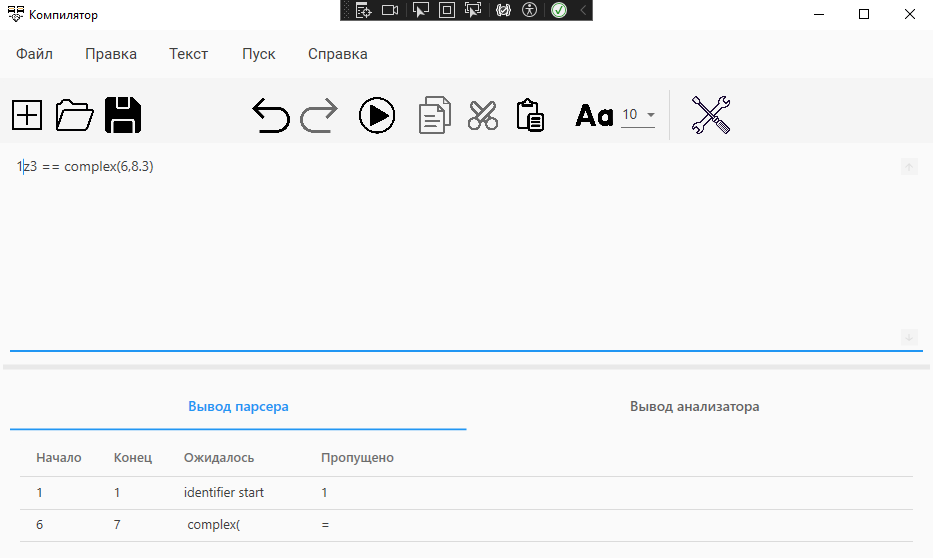


Рисунок 9 – Тестовый пример 6

## 7 Листинг программы

Листинг программной части разработанного синтаксического анализатора комплексного числа с инициализацией на языке Python представлен в приложении В.

# ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы был написан синтаксический анализатор (парсер) для комплексного числа с инициализацией на языке Python. Была определена грамматика комплексного числа с инициализацией на языке Python G[<КЧ>] в нотации Хомского. Согласно классификации Хомского, грамматика G[<КЧ>] является автоматной. Продукции P разработанной грамматики G[<КЧ>] были реализованы на графе. Была реализована нейтрализация синтаксических ошибок методом Айронса. Алгоритм был сведен к последовательному удалению следующего символа во входной цепочке до тех пор, пока следующий символ не окажется одним из допустимых в данный момент разбора.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шорников Ю.В. Теория и практика языковых процессоров : учеб. пособие / Ю.В. Шорников. – Новосибирск: Изд-во НГТУ, 2004.
2. Gries D. Designing Compilers for Digital Computers. New York, Jhon Wiley, 1971. 493 p.
3. Теория формальных языков и компиляторов [Электронный ресурс] / Электрон. дан. URL: https://dispace.edu.nstu.ru/didesk/course/show/8594, свободный. Яз.рус. (дата обращения 10.03.2024).
4. Грис Д. Конструирование компиляторов для цифровых вычислительных машин / Д. Грис ; пер. с. англ. Е. Б. Докшицкой, Л. А. Зелениной, Л. Б. Морозовой, В. С. Штаркмана, под ред. Ю. М. Баяковского, Вс. С. Штаркмана. - М., 1975. - 544 с. : табл., схемы
5. Ахо А. В. Компиляторы : Принципы, технологии, инструменты / А. Ахо, Р. Сети, Д. Ульман. - М., 2003. - 768 с.
6. Малявко, А. А. Формальные языки и компиляторы : учебник / Малявко А. А. - Новосибирск : Изд-во НГТУ, 2014. - 431 с. (Серия "Учебники НГТУ")
7. Свердлов С. З. Языки программирования и методы трансляции: Учебное пособие. — 2е изд., испр. — СПб.: Издательство «Лань», 2019. — 564 с.: ил. — (Учебники для вузов. Специальная литература).
8. Мозговой М. В. Классика программирования : алгоритмы, языки, автоматы, компиляторы : практический подход / Мозговой М. В. - СПб., 2006. - 320 с. : ил.

# Приложение А

# Справка (руководство пользователя)

## Меню текстового редактора

### Пункт "Файл" меню текстового редактора

В пункте "Файл" меню текстового редактора реализован следующий функционал (см. рисунок А.1):

* Создание документа
* Открытие документа
* Сохранение текущих изменений в документе
* Сохранение документа в новый файл
* Выход из текстового редактора

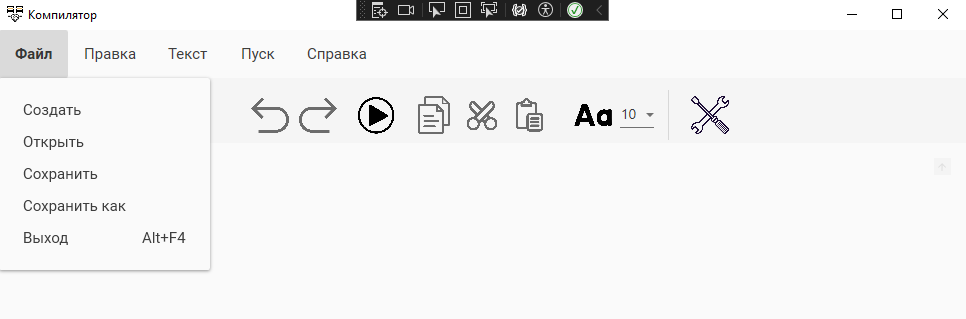
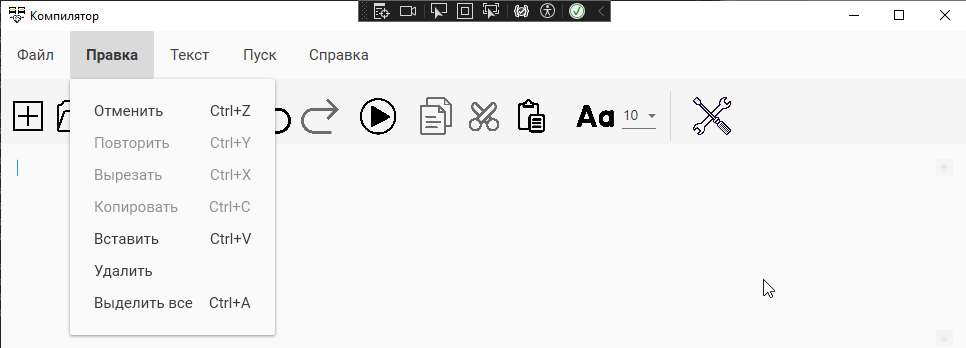


Рисунок А.1 – Пункт "Файл" меню

### Пункт "Правка" меню текстового редактора

В пункте "Правка" меню текстового редактора реализован следующий функционал (см. рисунок А.2):

* Отмена изменений
* Повтор последнего изменения
* Вырезать текстовый фрагмент
* Копировать текстовый фрагмент
* Вставить текстовый фрагмент
* Удалить текстовый фрагмент
* Выделить все содержимое документа

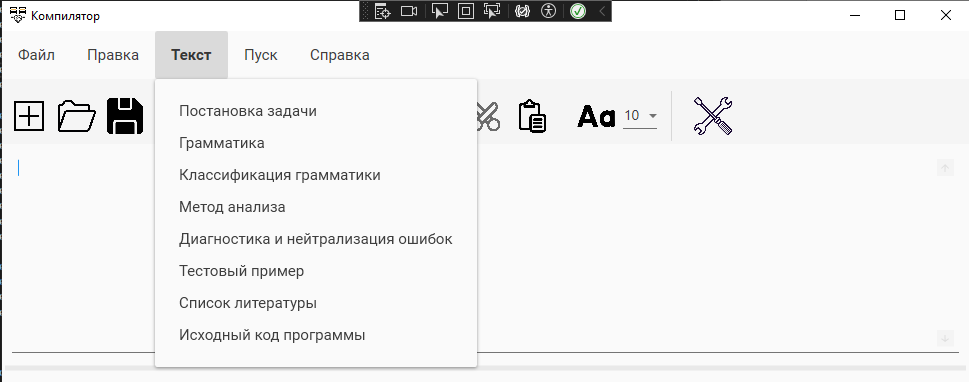
  
Рисунок А.2 – Пункт "Правка" меню

### Пункт "Текст" меню текстового редактора

При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе "Объявление комплексного числа с инициализацией на языке Python".

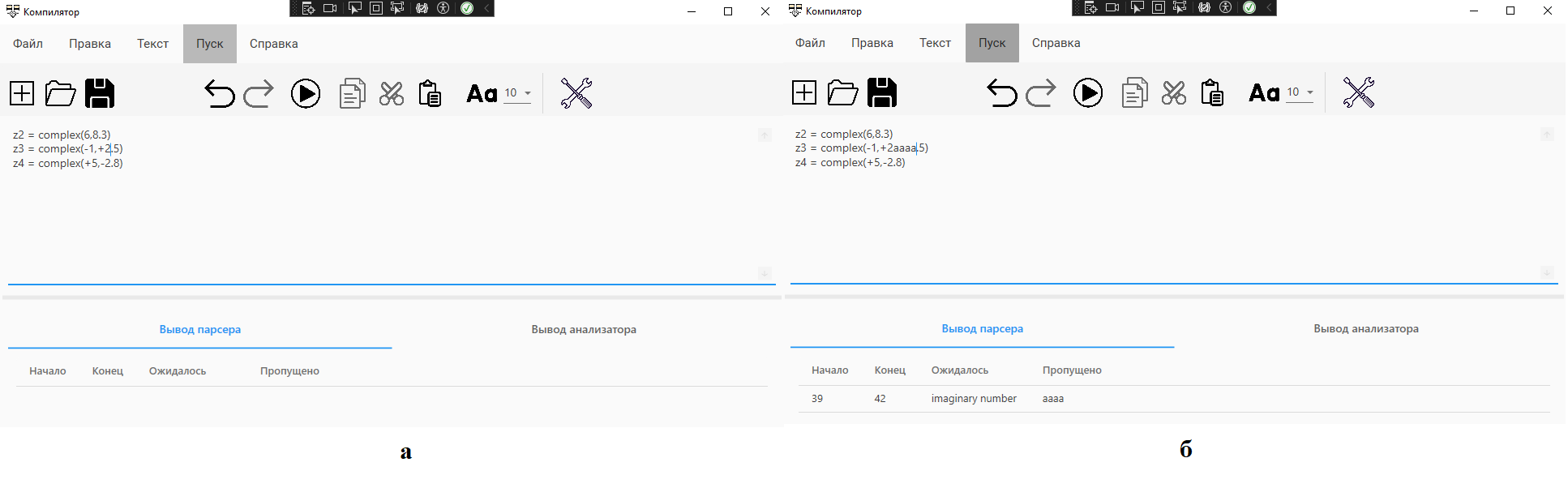
Пункт меню "Текст" содержит следующую информацию (см. рисунок А.3):

* Постановка задачи
* Грамматика
* Классификация грамматики
* Метод анализа
* Диагностика и нейтрализация ошибок
* Тестовый пример
* Список литературы
* Исходный код программы

  
Рисунок А.3 – Пункт "Текст" меню

### Пункт "Пуск" меню текстового редактора

При нажатии на пункт "Пуск" происходит запуск синтаксического анализатора текста (см. рисунок А.4).

  
Рисунок А.4 – Пункт "Пуск" меню

### Пункт "Справка" меню текстового редактора

Приложение имеет справочную систему, запускаемую командой «Вызов справки» (см. рисунок А.5).

Справка содержит описание всех реализованных функций меню.(см. рисунок А.6)

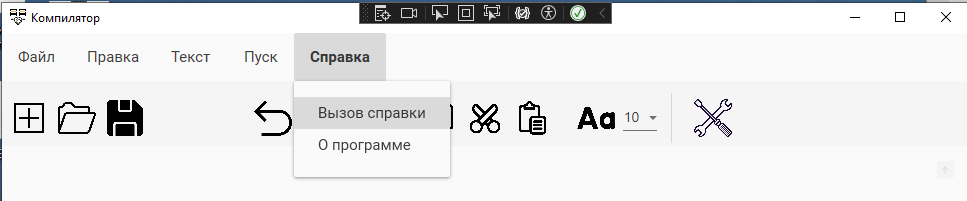
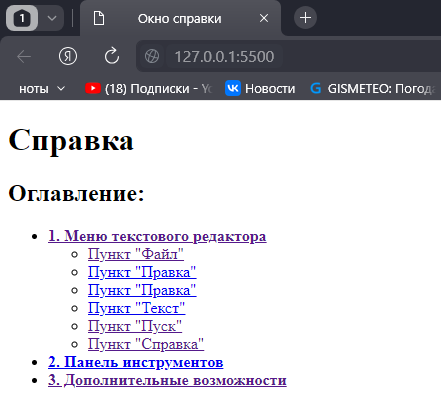
  
Рисунок А.5 – Пункт "Справка" меню  


Рисунок А.6 – Справочная система приложения

## Панель инструментов текстового редактора

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

* Создание документа (см. рисунок А.7, а)
* Открытие документа (см. рисунок А.7, б)
* Сохранение текущих изменений в документе (см. рисунок А.7, в)
* Отмена изменений (см. рисунок А.7,г)
* Повтор последнего изменения (см. рисунок А.7, д)
* Запуск синтаксического анализатора (см. рисунок А.7, е)
* Копировать текстовый фрагмент (см. рисунок А. 7, ж)
* Вырезать текстовый фрагмент (см. рисунок А.7, з)
* Вставить текстовый фрагмент (см. рисунок А.7, и)
* Изменение размера шрифта (см. рисунок А.7, к)
* Исправление ошибок (см. рисунок А.7, л)

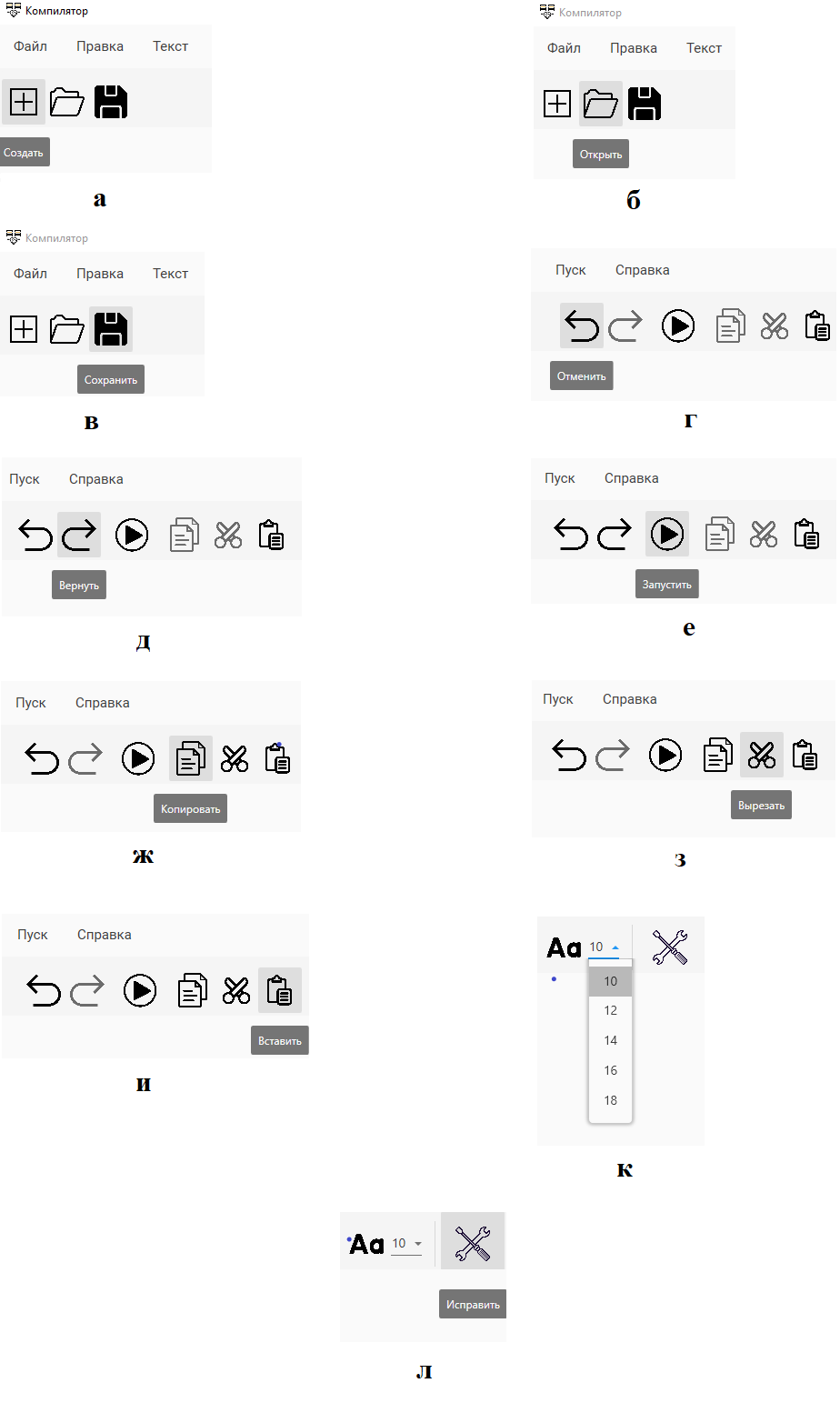


Рисунок А.7 – Панель инструментов

## Дополнительные возможности текстового редактора

1. Интерфейс позволяет открывать файл при перетаскивании его иконки в область редактирования.
2. Изменение размеров текста в окне редактирования и окне вывода результатов.

# Приложение Б

# Информация о программе

Программа написана в рамках лабораторных работы по дисциплине "Теория формальных языков и компиляторов".

Программа доработана в рамках курсовой работы.

Техническое задание:

Разработать приложение – текстовый редактор, дополненный функциями языкового процессора.

Приложение имеет графический интерфейс пользователя.

Язык реализации: C#.

Текстовый редактор имеет следующие элементы:

1. Основное меню программы;
   * Пункт меню "Текст";

При вызове команд этого меню должны открываться окна с соответствующей информацией по курсовой работе «Объявление комплексного числа с инициализацией на языке Python»

1. Панель инструментов;

Панель инструментов содержит кнопки вызова часто используемых пунктов меню:

* + Файл – Создать, Открыть, Сохранить;
  + Правка – Отменить, Повторить, Вырезать, Копировать, Вставить;
  + Пуск;

Команда «Пуск» предназначена для запуска анализатора текста.

Приложение имеет справочную систему, запускаемую командой «Вызов справки».

Cправка содержит описание всех реализованных функций меню.

1. Окно/область ввода/редактирования текста;

Область редактирования представляет текстовый редактор.

Команды меню "Файл" и "Правка" работают с содержимым этой области.

1. Окно/область отображения результатов работы языкового процессора (в этой области ввод текста запрещен).

В область отображения результатов выводятся сообщения и результаты работы языкового процессора.

Интерфейс содержит дополнительные элементы и возможности:

1. Открытие файла при перетаскивании иконки в окно программы.
2. Изменение размеров текста в окне редактирования и окне вывода результатов.

# Приложение В

# Листинг программы

|  |
| --- |
| AssignmentState.cs |
| using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class AssignmentState : IState  {  public string Handle(Parser parser, string code, int position)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  StringBuilder errorBuffer = new StringBuilder();  if (code[position] != '=')  {  parser.AddError(new ParseError(position, position, "assignment operator (=)", code[position].ToString()));  }  position++;  parser.State = new ComplexState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| ComplexState.cs |
| using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class ComplexState : IState  {  public string Handle(Parser parser, string code, int position)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  string whitespaceCharacters = " \n\r\t";  while (whitespaceCharacters.Contains(code[position]))  {  code = code.Remove(position, 1);  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  }  code = code.Insert(position, " ");  string expected = " complex(";  StringBuilder errorBuffer = new StringBuilder();  for (int expectedPos = 0; expectedPos < expected.Length; expectedPos++)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  if (expected[expectedPos] != code[position])  {  errorBuffer.Append(code[position]);  code = code.Remove(position, 1);  expectedPos--;  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, expected, errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  parser.State = new RealPartState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| EndState.cs |
| using System.Runtime.Versioning;  using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class EndState : IState  {  public string Handle(Parser parser, string code, int position)  {  StringBuilder errorBuffer = new StringBuilder();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  char c = code[position];  if (c != '\n')  {  errorBuffer.Append(c);  code = code.Remove(position, 1);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "newline", errorBuffer.ToString()));  errorBuffer.Clear();  }  break;  }  }  position++;  if (position == code.Length)  {  return code;  }  parser.State = new IdentifierState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| IdentifierState.cs |
| using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class IdentifierState : IState  {  public string Handle(Parser parser, string code, int position)  {  char symbol;  StringBuilder errorBuffer = new StringBuilder();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  symbol = code[position];  if (!char.IsLetter(symbol) && symbol != '\_')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "identifier start", errorBuffer.ToString()));  errorBuffer.Clear();  }  break;  }  }  errorBuffer.Clear();  while (position < code.Length)  {  symbol = code[position];  if (symbol == '=')  {  position++;  break;  }  if (symbol == ' ')  {  position++;  break;  }  if (!char.IsLetter(symbol) && !char.IsDigit(symbol) && symbol != '\_')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "identifier", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  parser.State = new AssignmentState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| ImaginaryPartState.cs |
| using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class ImaginaryPartState : IState  {  public string Handle(Parser parser, string code, int position)  {  char symbol;  StringBuilder errorBuffer = new StringBuilder();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  char c = code[position];  if (!char.IsDigit(c) && c != '+' && c != '-')  {  errorBuffer.Append(c);  code = code.Remove(position, 1);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary start", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  }  errorBuffer.Clear();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  symbol = code[position];  if (!char.IsDigit(symbol) && symbol != '.' && symbol != ')')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else if (symbol == '.')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  else if (symbol == ')')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  parser.State = new EndState();  return parser.State.Handle(parser, code, position);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  errorBuffer.Clear();  while (position < code.Length)  {  symbol = code[position];  if (!char.IsDigit(symbol) && symbol != ')')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else if (symbol == ')')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "imaginary number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  parser.State = new EndState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| IState.cs |
| namespace CompilerDemo.Model.Parser.States  {  internal interface IState  {  string Handle(Parser parser, string code, int position);  }  } |

|  |
| --- |
| RealPartState.cs |
| using System.Text;  namespace CompilerDemo.Model.Parser.States  {  internal class RealPartState : IState  {  public string Handle(Parser parser, string code, int position)  {  char symbol;  StringBuilder errorBuffer = new StringBuilder();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  char c = code[position];  if (!char.IsDigit(c) && c != '+' && c != '-')  {  errorBuffer.Append(c);  code = code.Remove(position, 1);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real start", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  }  errorBuffer.Clear();  while (position < code.Length)  {  if (position >= code.Length)  {  parser.AddError(new ParseError(position, position, "incomplete line", ""));  return code;  }  symbol = code[position];  if (!char.IsDigit(symbol) && symbol != '.' && symbol != ',')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else if (symbol == '.')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  else if (symbol == ',')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  parser.State = new ImaginaryPartState();  return parser.State.Handle(parser, code, position);  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  errorBuffer.Clear();  while (position < code.Length)  {  symbol = code[position];  if (!char.IsDigit(symbol) && symbol != ',')  {  errorBuffer.Append(symbol);  code = code.Remove(position, 1);  }  else if (symbol == ',')  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  break;  }  else  {  if (errorBuffer.Length > 0)  {  parser.AddError(new ParseError(position + 1, position + errorBuffer.Length, "real number", errorBuffer.ToString()));  errorBuffer.Clear();  }  position++;  }  }  parser.State = new ImaginaryPartState();  return parser.State.Handle(parser, code, position);  }  }  } |

|  |
| --- |
| Parser.cs |
| using CompilerDemo.Model.Parser.States;  using Microsoft.Xaml.Behaviors.Core;  using System.Collections.Generic;  namespace CompilerDemo.Model.Parser  {  internal class Parser  {  public IState State { get; set; }  private List<ParseError> Errors { get; set; }  public Parser()  {  Errors = new List<ParseError>();  State = new ComplexState();  }  public (List<ParseError>,string) Parse(string code)  {  code = code.Replace("\r", string.Empty);  Errors.Clear();  State = new IdentifierState();  string cleanCode = State.Handle(this, code, 0);  return (Errors, cleanCode);  }  public void AddError(ParseError error)  {  Errors.Add(error);  }  }  } |

|  |
| --- |
| ParseError.cs |
| namespace CompilerDemo.Model.Parser  {  internal class ParseError  {  public string ExpectedTokenType { get; private set; }  public string DiscardedFragment { get; private set; }  public int StartPos { get; private set; }  public int EndPos { get; private set; }  public ParseError(int startPosition, int endPosition, string expected, string discardedFragment)  {  ExpectedTokenType = expected;  DiscardedFragment = discardedFragment;  StartPos = startPosition;  EndPos = endPosition;  }  }  } |