**Автономная некоммерческая общеобразовательная организация "Физтех-лицей"
(АНОО «Физтех-лицей» им. П.Л. Капицы)**

XX научно-практическая конференция

«Старт в инновации»

Разработка языка программирования для образовательных целей

Выполнили:
Кобзев Павел
Ермолаев Дмитрий
Лашин Виктор

Руководитель:
Зыонг К.К. – учитель информатики

Московская область, г. Долгопрудный

2021

Содержание

1. Введение

В большинстве школ для обучения основам программирования чаще всего используют языки программирования Pascal и Python. По ряду причин, они не в полной мере подходят на начальной стадии обучения. Например, язык Pascal является устаревшим и уже не используется, язык Python не отражает некоторых основных принципов программирования, таких как строгая типизация и т.д. В связи с этим возникает необходимость создать новый современный язык программирования, наиболее оптимальный для начала обучения программированию.

Работа состоит из двух глав. В первой главе рассматриваются теоретические вопросы, связанные с интерпретаторами, языками программирования, используемыми понятиями и определениями; проведен анализ достоинств и недостатков языков программирования, используемых для обучения в настоящее время. Во второй главе представлены вопросы, связанные непосредственно с применением на практике знаний, полученных в результате исследования, для реализации собственного языка программирования. В заключении сделаны выводы, связанные с итогами исследования и проделанной работой, оценкой готового продукта.

Цель работы:

Провести исследование и на основе анализа собранной информации создать новый язык программирования, наиболее полно подходящий для обучения основам программирования.

Область исследования – информатика.

Задачи исследования:

1. На основании научной литературы провести исследование и изучить основные положения теории реализации языков программирования и теорию, связанную с построением интерпретаторов.
2. Выявить достоинства и недостатки языков программирования, используемых для обучения в настоящее время.
3. Спроектировать на основе собранных данных новый язык программирования, наиболее полно подходящий для начала обучения.
4. Реализовать данный язык программирования.
5. Сделать выводы по итогу проделанной работы.

Методы исследования:

1. Анализ литературы о разработке языков программирования и интерпретаторов и ее обобщение.
2. Анализ информации из сети Интернет

Актуальность:

Информатика все больше внедряется в быт современного человека, и знакомство с этой наукой начинается еще в школе. Для того чтобы заинтересовать обучающегося необходима отработанная программа, интересные задания, а главное, правильные методы обучения и необходимые средства, ориентированные на образование. Это особенно актуально для обучающихся профильных учебных заведений, таких как АНОО «Физтех-лицей имени П.Л.Капицы».

К сожалению, в настоящее время, языки программирования, используемые для обучения основам программирования – Python, Pascal, C++, – не являются оптимальными для процесса обучения по ряду причин. Так Python не раскрывает некоторых базовых концепций программирования и не подходит для начала обучения, Pascal является морально устаревшим и почти не используемым в настоящее время языком, а C++ является достаточно сложным для начала обучения языком. Именно поэтому появляется необходимость разработать новый современный язык программирования, ориентированный на обучение основам программирования.

2. Теоретическая часть

2.1 Основные понятия и определения:

Язык программирования – формальный язык, предназначенный для записи компьютерных программ[1]. Язык программирования определяет набор лексических, синтаксических и семантических правил, определяющих внешний вид программы и действия, которые выполнит исполнитель (обычно — ЭВМ) под её управлением.

Формальный язык – множество конечных слов (строк, цепочек) над конечным алфавитом. Понятие языка чаще всего используется в теории автоматов, теории вычислимости и теории алгоритмов[2].

Интерпретатор – программа, являющаяся разновидностью транслятора и выполняющая интерпретацию кода – построчный анализ и выполнение кода на языке программирования.

Обычно интерпретация состоит из следующих шагов:

1. Лексический анализ
2. Синтаксический анализ
3. Семантический анализ
4. Исполнение кода

Интерпретатор состоит из нескольких частей, каждая из которых отвечает за определенный шаг.

2.2 Лексический анализ

Лексический анализ – разбор входной последовательности на токены, определенные конкретным языком программирования, без проверки на соответствие грамматике языка. Лексический анализ осуществляет часть интерпретатора, называемая **лексическим анализатором** или **сканером**. На вход лексическому анализатору поступает последовательность символов – текст исходной программы на языке программирования. Лексический анализатор выделяет в тексте исходной программы лексемы данного языка программирования и передает их далее в синтаксический анализатор.

В контексте лексического анализа выделяются следующие термины:

- **Токен** – структура, состоящая из имени токена и необязательного атрибута. Имя токена – это символ, которые обозначает тип лексемы. Имена токенов далее обрабатываются синтаксическим анализатором.

- **Шаблон** – описание того, какой вид может принимать определенная лексема. Например, для идентификатора это последовательность символов, первым из которых обязательно должна являться буква, а остальные могут быть произвольно выбраны из алфавита для данного языка программирования.

- **Лексема** (также – лексическая единица языка) – это структурная единица языка, которая является последовательностью элементарных символов языка, соответствует шаблону токена и идентифицируется как экземпляр токена. Лексема не содержит в своем составе других структурных единиц языка. Лексемами являются ключевые слова, идентификаторы, константы, знаки операций, разделители. Состав лексем определяется синтаксисом языка программирования и для разных языков может различаться.

- **Таблица лексем** – представление перечня лексем, найденных в тексте исходной программы, в виде таблицы. Эта таблица является результатом работы лексического анализатора. Каждой лексеме в таблице соответствует свой условный код, являющийся уникальным для каждой лексемы и зависящий от типа лексемы, а также дополнительная служебная информация. Лексемы в таблице находятся в том порядке, в котором они встречаются в программе.

- **Таблица символов** – это таблица, в которой хранится информация обо всех идентификаторах и константах, встреченных в программе. Таблица необязательно организована таким образом, что идентификаторы в ней находятся в том же порядке, что и во входной программе, что необходимо для удобства поиска в таблице. В таблице для каждого идентификатора хранится его описание. Для блочных программ может так же храниться информация об области видимости имен. Это необходимо для реализации принципа локализации имен.

Принципы работы лексического анализатора.

Лексическому анализатору на вход подается последовательность символов, являющаяся кодом исходной программы. Анализатор должен выделить в последовательности токены и заполнить таблицу символов, а также передать последовательности имен токенов синтаксическому анализатору. В простейшей обобщенной реализации для выполнения своей задачи он производит следующие действия:

1. Считывает символ из входного потока.
2. Игнорирует символ, если это очередной пробел, или игнорирует последовательность символов, если встретил признак начала комментария, и переходит к считыванию следующего символа.
3. Собирает считанные символы в строку.
4. Если встречает пробельный символ после очередного символа, проверяет, является ли слово ключевым, и возвращает соответствующий токен.
5. Если слово не ключевое, проверяет, является ли лексема константой (числовой или другой), и возвращает соответствующий токен.

6. Если слово не является константой, значит, слово является идентификатором, и анализатор возвращает соответствующий токен.

В более сложной и продуктивной реализации рассматривается буферизация входного потока, но для реализации простейшего лексического анализатора используется опережающее чтение на один символ, чего вполне достаточно для нормальной работы. Также в лексическом анализаторе может быть реализована простейшая функция поиска ошибок, но, так как это не является основной целью лексического анализа, данная функция необязательна.

2.3 Синтаксический анализатор

Синтаксический анализатор – часть программы, которая принимает от лексического анализатора строку имен токенов и проверяет, может ли данная строка быть построена, исходя из формальной грамматики данного языка. Также синтаксический анализатор подробно сообщает обо всех найденных ошибках

На выходе из синтаксического анализатора необходимо получить дерево разбора и необходимые таблицы, например таблицу типов или таблицу идентификаторов. Возможны также реализации без деревьев разбора, но, скорее всего, они не будут такими же универсальными. Дерево разбора и таблицы необходимо передать на следующий этап интерпретации – в семантический анализатор, либо возможно отслеживание ошибок уже напрямую во время исполнения исходного кода программы.

В контексте синтаксического анализа выделяются следующие термины:

Формальная грамматика языка – это способ описания формального языка. Формальная грамматика, по сути, является набором правил, которые определяют, какой набор лексем является корректным для данного языка.

Терминальные символы (терминалы) – базовые символы, имеющие конкретные неизменяемые значения. Из этих символов формируются строки языка.

Нетерминал – синтаксическая переменная, которая обозначает множество строк (какую-либо сущность языка) и не имеет какого-либо конкретного значения.

Продукция – получение новой строки из предыдущей путем ее полной замены или замены ее части по одному из правил.

Контекстно-свободная грамматика – формальная грамматика, у всех продукций которой левая часть является одиночным нетерминалом.

Для описания грамматики языков программирования чаще всего используются **формы Бэкуса-Наура (РБНФ)**. Форма Бэкуса-Наура, по сути, является формой записи контекстно-свободной грамматики. РБНФ – расширенная форма Бэкуса-Наура – является усовершенствованной БНФ с измененным синтаксисом.

Основным в форме Бэкуса-Наура является то, что, как для контекстно-свободной грамматики, для нее последовательно, одна через другую, описываются синтаксические категории – нетерминалы. БНФ определяет, каким образом символы могут, в конечном итоге, быть заменены на цепочки символов.

Алгоритм построения РБНФ:

1. Определяемое понятие записывается в левой части.
2. После определяемого понятия ставится знак =, который означает «это есть по определению».
3. Далее, справа от знака = идет последовательность терминалов и/или нетерминалов, причем используются следующие условные обозначения:

    - [*smth*] – условное вхождение *smth* в предложение (0 или 1 раз).
    - { *smth* } – вхождение *smth* в предложение 0 или более раз.
    - *Smth* | *smth* – выбор либо *Smth*, либо *smth.*
    - *(smth)* – круглые скобки обозначают группировку
    - ‘abc’ – терминалы (будь то один символ или последовательность символов) берутся в кавычки.
    - *Smth, smth* – конкатенация (за *Smth* обязательно следует *smth*)

Архитектура синтаксического анализатора строится так, чтобы отследить, можно ли получить полученный набор имен токенов по исходной, заранее построенной БНФ. При этом он как бы «раскручивает» цепочки токенов так, чтобы свести к полученный набор к правилам грамматики.

После всех проверок грамматики, при корректном коде программы, начинается следующий шаг.

2.4 Семантический анализатор. Исполнение кода

Следующим шагом будет являться анализ кода на соблюдение **семантики** языка.

Семантика языка программирования – система правил для определения поведения отдельных языковых конструкций, представленная в виде математической модели.

Анализ семантики языка предполагает, что на предыдущем шаге выражение было проверено на соблюдение правил грамматики. Основной задачей на этапе семантического анализа будет являться проверка типов. Данная проверка анализирует конструкции, полученные синтаксическим анализатором, по типу операндов определяет, какие с ними можно провести операции. Также здесь отслеживается использование необъявленных и неинициализированных переменных. Семантический анализатор сообщаем обо всех ошибках, найденных при анализе.

За семантическим анализом идет непосредственно **исполнение кода**. На этом этапе уже обработанные и проверенные на ошибки выражения будут исполняться интерпретатором как исходный код программы, написанной на данном языке программирования. На этом этапе все еще могут встречаться ошибки, например, деление на ноль или открытие несуществующего файла. Такие ошибки обычно называются ошибками времени исполнения (runtime error). Данный факт связан с тем, что некоторые конструкции являются верными и с точки зрения грамматики языка, и с точки зрения

семантики, однако физически не имеют смысла. Если же программа является полностью корректной, после исполнения кода будет получен результат работы программы.

2.5 Анализ достоинств и недостатков языков программирования, применяемых для обучения

На данный момент обязательным предметом во многих школах, в том числе в профильных классах, является информатика. В старших классах на уроках информатики начинают изучать непосредственно программирование. Но многие ученики сталкиваются с проблемами при изучении основ по множеству возможных причин. Среди них есть как и ошибки в преподавании дисциплины, так и неудачный выбор средств для обучения.

Чаще всего, для изучения основ программирования используются языки C++, Python, Pascal. Далее будет приведен анализ достоинств и недостатков каждого из языков.

C++:

Достоинства:

- Язык является очень быстрым по времени выполнения программ, в промышленном программировании его наиболее часто используют для написания программ, требующих особенно высокой скорости работы.
- Язык имеет статическую типизацию, что помогает лучше понять физическое представление данных и работу с типами.
- Язык имеет слабую типизацию, при этом сочетает свойства высокоуровневых и низкоуровневых языков программирования, имеется непосредственная работа с памятью.

Недостатки:

- Язык является компилируемым, на компиляцию в некоторых случаях уходит значительное количество времени.
- Язык является небезопасным из-за непосредственной работы с памятью. Работа с памятью вызывает сложности у новичков, и они часто совершают много неочевидных на первый взгляд ошибок.
- Язык является очень обширным, и для начала программирования на нем необходимы опыт в программировании и начальные знания об устройстве языка и стандартных библиотек.

Все вышеописанные недостатки делают язык сложным для начала изучения программирования и неподходящим для новичков, а достоинства не покрывают недостатки, так как время работы на начальном этапе обучения не представляет особой значимости.

Python:

Достоинства:

- Является достаточно простым для изучения языком, многие распространенные функции, которые на других языках необходимо реализовывать вручную, являются стандартными в данном языке.

- Язык является высокоуровневым, что хорошо подходит для начала обучения, так как обучение предполагает написание достаточно простых сценариев.

- Концепция языка предполагает написание каждой следующей команды с новой строки, что сразу помогает новичку писать красивый и понятный код.

Недостатки:

- Язык является слишком упрощенным. Если в языке базовые алгоритмы полностью реализованы за ученика, он не сможет ими в достаточной степени овладеть, к тому же не будет иметь желания, так как можно использовать уже готовые средства.
- В языке используется принцип динамической типизации, что не дает ученикам достаточного представления об устройстве типов, и в дальнейшем у них могут возникнуть трудности с переходом на язык программирования со статической типизацией.

Несмотря на все преимущества языка, его недостатки оказываются значительными для использования его при обучении основам программирования.

Pascal:

Достоинства:

- В языке отражены все базовые принципы программирования, что хорошо подходит для начала обучения.

- Язык имеет статическую типизацию, что помогает лучше понять физическое представление данных и работу с типами.

- Язык не является слишком обширным, и новички вполне могут изучить все основные средства языка.

Недостатки:

- Язык является морально и функционально устаревшим, сейчас является практически невостребованным и неиспользуемым, поскольку современные языки лучше по многим параметрам.

- В языке не слишком много «синтаксического сахара», который во многом облегчает работу программисту.

- Язык имеет сильную типизацию, что усложнит переход ученика на многие другие языки программирования.

Pascal является достаточно неплохим языком для начала обучения, но главный его минус в том, что он является устаревшим и требует себе современной альтернативы.

Проанализировав достоинства и недостатки вышеупомянутых языков программирования, мы решили выделить положительные стороны каждого. Совместив их, мы получим современный язык программирования, наиболее удачно подходящий для обучения основам программирования.

3. Практическая часть

3.1 Начало работы

3.1.1 Выбор способа реализации языка

Для начала работы над нашим продуктом, новым языком программирования, нам нужно было определиться со способом реализации языка программирования. Было необходимо решить, какой язык мы будем писать – компилируемый или интерпретируемый. Преимуществами интерпретируемого языка являются простота отладки, скорость разработки и возможность изменения программы во время компиляции, достоинствами компилируемого являются скорость исполнения и эффективность. После анализа собранной в Интернете информации о компилируемых и интерпретируемых языках нами было принято решение разрабатывать интерпретируемый язык. Данный выбор был обусловлен рядом причин:

- Для начинающих программистов будет полезно указать на конкретную ошибку на этапе выполнения, что проще реализовать в интерпретируемом языке.
- В интерпретируемом языке намного легче реализовать выполнение программы по шагам, что наглядно покажет начинающему программисту, как именно работает его программа, и что происходит с переменными во время исполнения. Такой подход к обучению позволит формировать у ученика склад ума, необходимый для программирования, развивать у него логическое мышление, последовательность.
- Будут сокращены затраты времени, благодаря отсутствию компиляции, что позволит выделить сэкономленное время непосредственно на изучение нового и решение задач.
- Также, отредактировать программу во время исполнения без перекомпиляции может быть полезно в некоторых случаях, в частности, также для экономии времени.

У интерпретируемых языков также есть ряд недостатков:

- Невысокая скорость работы, по сравнению с компилируемыми языками. В нашем случае данная особенность не будет существенно сказываться на итоговом результате, так как язык разрабатывается для обучения основам программирования, где не важна высокая скорость работы.
- В больших проектах может возникнуть проблема, связанная с отображением ошибок на этапе выполнения. Для того чтобы отследить ошибку в коде, необходимо будет выполнить программу до определенной строки, которая может находиться в конце проекта. На это будет затрачено много времени, которое могло бы быть использовано продуктивно. Тем не менее, наш язык имеет другое назначение и не предполагает написания больших проектов, поэтому этот недостаток так же не является существенным.

Таким образом, описанные выше достоинства интерпретируемых языков оказываются более весомыми, чем недостатки, значит, данный выбор является наиболее разумным и оптимальным для решения поставленной задачи.

3.1.2 Изучение литературы

Для того чтобы получить начальные знания по теме, мы решили подобрать подходящую техническую литературу. Классическим пособием для написания своего языка программирования является книга «Компиляторы: принципы, технологии и инструментарий» Альфреда Ахо. В этой книге содержатся исчерпывающая информация по теории компиляторов – способы реализовать архитектуру компилятора, принципы, используемые при построении архитектуры, и т.д. Хотя мы решили разрабатывать интерпретируемый язык, информация из книги оказалась очень полезной и помогла понять, какие подходы лучше использовать для написания своего языка, а так же в книге мы нашли теоретическую информацию о правильном описании конструкций языка программирования, что помогло нам в дальнейшем проектировании.

3.2 Проектирование языка программирования

Самым первым и, возможно, самым главным шагом стал выбор названия и логотипа языка, так как использование запоминающихся образов является серьезным вкладом в успешность будущего продукта.

Для начала необходимо определить типизацию в нашем языке. Мы решили, что наиболее удачной для обучения является статическая типизация, так как она используется в большом количестве языков программирования, а значит, ученик сможет без труда перейти на новый язык программирования, после того как в достаточной степени овладеет основами и научится программировать на нашем языке. К тому же, если начинать обучение на языке с динамической типизаций, будет труднее перейти на язык со статической типизацией, чем наоборот, так как у ученика будет складываться несколько неверное представление о физической работе машины с памятью.

Следующим шагом стало определение базовых типов, структуры программы и грамматики языка. Для этого мы решили использовать форму Бэкуса-Наура, так как она является наиболее удобной для описания формальной грамматики и наиболее часто для этого используется, по сравнению с другими способами записи формальной грамматики. Также в ней будут отражены базовые типы нашего языка. Полная БНФ приведена в Приложении.

Для реализации архитектуры интерпретатора для нашего языка был выбран язык программирования C++, так как у него есть необходимый функционал для решения поставленной задачи, он обладает высокой скоростью работы, и каждый участник нашей группы в достаточной степени владеет данным языком программирования.

Итого, имеем описание нашего языка: интерпретируемый язык программирования со статической типизацией, интерпретатор для которого реализован на языке C++ по разработанной грамматике, представленной в виде БНФ.

3.3 Реализация языка программирования

3.3.1 Лексический анализатор

Первым этапом реализации нашего языка является написание лексического анализатора. После получения необходимых знаний по теме было принято решение создать лексический анализатор, который разбивает считываемую строку на

последовательность токенов. Токены были определены в программе в соответствии с алфавитом и грамматикой нашего языка в классе перечисления и в векторе строк для удобства. Далее, после разбора строки на токены, их последовательность передается синтаксическому анализатору.

Сначала был определен абстрактный класс Token, от которого были унаследованы уже классы конкретных токенов. при этом каждый экземпляр класса имеет свой тег, который однозначно задает, какое перед нами ключевое слово или операция.

Далее был создан сам класс лексического анализатора, для которого определен следующий конструктор: при создании класса лексического анализатора ему передается название файла исходной программы, далее из этого файла производится считывание в буфер, с которым в дальнейшем будет работать лексический анализатор.

Лексический анализатор имеет метод scan, который выделяет в строке токен и возвращает его. Токен выделяется методом опережающего чтения на один символ, что позволяет однозначно определить встречаемые токены. При этом, сначала определяется, является ли лексема символом или знаком операции, затем отслеживаются числовые константы, после ключевые слова и идентификаторы.

Выбранная нами концепция позволила создать работающий лексический анализатор, что было подтверждено тестами. В процессе работы на данном шаге мы сталкивались с большим количеством проблем, в основном они были связаны с невнимательностью и недостаточным опытом в данной сфере.

3.3.2 Синтаксический анализатор. Семантический анализатор. Исполнение кода

Следующим этапом реализации стало написание синтаксического анализатора. На этом этапе мы уже имеем рабочий лексический анализатор, который разбирает строку на токены. Лексический анализатор было решено организовать в составе блока, в который также войдет семантический анализатор, и подобие виртуальной машины, которая будет исполнять код программы на нашем языке.

Здесь в классе Parser был реализован описанный функционал, а именно: определение встреченной конструкции языка, определение приоритета операций, разбор выражений и исполнение команд. На данном шаге не был реализован полностью весь функционал, задуманный ранее, но он будет добавлен в будущем.

4. Выводы

В ходе создания проекта мы проделали большую работу. Нами было проведено исследование, а именно:

- Были выявлены плюсы и минусы языков программирования, в настоящее время.
- Проведен анализ и разработана концепция языка программирования, совмещающего достоинства уже существующих языков, для применения в сфере образования.
- Была изучена литература по данной теме.
- Была разработана грамматика языка программирования.
- Была разработана архитектура реализации языка, соответствующая концепции.

Тем не менее, проект находится на стадии завершения, хотя и не все задачи решены полностью. Сейчас язык программирования активно дорабатывается: исправляются ошибки и вносятся улучшения, повышающие скорость работы, удобность и понятность для пользователя, добавляется «синтаксический сахар».

Также были определены пути дальнейшего развития проекта. Среди них присутствует доработка языка программирования, которая сделает его более удобным и понятным для пользователей. По окончании данной доработки возможно провести тестирование языка, предложив опробовать его всем желающим, особенно учащимся информационно-технологического профиля «Физтех-лицея». Это позволит выявить самые существенные недостатки языка и начать работу по улучшению нашего продукта. Новых пользователей, вероятнее всего, удастся привлечь во время научной конференции, где будет представлен проект. Также, для привлечения новых пользователей возможно использование рекламы, в частности, рекламы в социальных сетях.

5. Список литературы

Ахо, Альфред В., Лам, Моника С., Сети, Рави, Ульман, Джеффри Д. К63 Компиляторы: принципы, технологии и инструментарий, 2 е изд . : Пер . с англ. — М. : ООО “И.Д. Вильямс”, 2018. — 1184 с. : ил. — Парал. тит. англ.

Гавриков М. М., Иванченко А. Н., Гринченков Д. В. Теоретические основы разработки и реализации языков программирования. — КноРус, 2013. — 178 с. — ISBN 978-5-406-02430-0.

ISBN
978

5

8459

1932

8 (рус.)

[1] ISO/IEC/IEEE 24765:2010 Systems and software engineering — Vocabulary

[2] https://ru.wikipedia.org/wiki/Формальный_язык

Приложение 1. Описание грамматики языка с помощью BNF

```
// Pie++ syntax
// short description of syntax:
// | - or
// ( ) - grouping
// [ ] - option
// { } - repetition (0 or more)
// . - any symbol
// ' ' - symbol
// < > - ranges

ftl_program = {ftl_function_decl}, [ftl_global_vars], ftl_main;

ftl_global_vars = {ftl_variable_declaration, ';'};
ftl_main = 'int', 'main', '(', ')',ftl_compound_operator;
ftl_function_decl = (ftl_type | 'void'), ftl_name, '(',
  ftl_formal_parameter,  {',', ftl_formal_parameter}, {',', ftl_formal_parameter, '=', ftl_constant_exp},
  ')', ftl_compound_operator;

ftl_formal_parameter = ftl_type, ftl_name;

ftl_derived_operator = ftl_loop_operator | ftl_operator | ftl_conditional_operator;
ftl_loop_operator = ftl_if | ftl_while | ftl_for;
ftl_conditional_operator = ftl_if | ftl_switch;
ftl_compound_operator = '{', { ftl_operator,  ';' }, '}';
ftl_operator = ftl_assignment_operator | ftl_nothing | ftl_derived_operator | ftl_return;
ftl_return = 'return', {ftl_variable};
ftl_nothing = ;
ftl_assignment_operator = ftl_variable, ftl_op_assignment, ftl_expression;
ftl_op_assignment = '=' | "+=" | "-=" | "*=" | "/=" ;

ftl_digit = <0-9> ;
ftl_letter = <A-Z | a-z | _> ;
ftl_num = ftl_digit, { ftl_digit } ;

ftl_type = 'int' | 'float' | 'string' | 'bool' | 'char';
ftl_int = ftl_num ;
ftl_float = ftl_num, '.', ('0' | (['E', ['+' | '-'], ], ftl_num)) ;
ftl_bool = 'true' | 'false' ;
ftl_char = . ;
ftl_string = { ftl_char } ;
```

```
ftl_constant_expr = (['+' | '-'], (ftl_int | ftl_float)) | ([!], ftl_bool)
  | (''', ftl_char, ''') | ('"', ftl_string, '"');

ftl_name = ftl_letter, { ftl_letter | ftl_digit } ;
ftl_variable = ftl_name, { '[', ftl_index, ']' };
ftl_index = ftl_num;
ftl_variable_declaration = (ftl_type, ftl_name, [ '=', ftl_constant_expr ]) | ftl_array_declaration;

ftl_func_call = ftl_name, '(', [ftl_non-formal_parameter], {',', ftl_non-formal_parameter}, ')';
ftl_non-formal_parameter = ftl_constant_expr | ftl_variable | ftl_func_call;

ftl_array_declaration = ftl_common_array_declaration | ftl_dynamic_array_declaration;

ftl_common_array_declaration = "common" ftl_type ftl_name '[' ftl_num ']' ;
ftl_dynamic_array_declaration = "dynamic" ftl_type ftl_name '[' [ ftl_num ] ']' ;

ftl_if = 'if', '(', ftl_logical_expression, ')',
  ( (ftl_operator, ';') | ftl_compound_operator ),
  [ 'else', ( (ftl_operator, ';') | ftl_compound_operator ) ] ;

ftl_while = 'while', '(', flt_logical_expression, ')', ( (ftl_operator, ';') | ftl_compound_operator ) ;
ftl_for = 'for', '(', [ (ftl_variable_declaration | ftl_expression) ], ';',
  [ ftl_logical_expression ], ';' [ftl_expression], ')',
  ( (ftl_operator, ';') | ftl_compound_operator ) ;

ftl_switch = 'switch', '(', ftl_variable, ')', '{',
  { 'case', (ftl_constant_expression), ':', ( (ftl_operator, ';') | ftl_compound_operator ), ';' },
  [ 'default', ':', ( (ftl_operator, ';') | ftl_compound_operator ) ] '}' ;


 ftl_expression = ftl_logical_expression | ftl_arith_expression | ftl_conc;

 ftl_conc = ftl_string, { '+', ftl_string } ;


 ftl_arith_expression = ['+' | '-'], ftl_addend, { ['+' | '-'], ftl_addend } ;
 ftl_addend = ftl_multiplicand, { ftl_op_mul, ftl_multiplicand };
 ftl_multiplicand =  ftl_float | ftl_int | ftl_mul_var | ftl_func_call | ('(', ftl_arith_expression, ')') ;
 ftl_op_mul = '*' | '/' | '%';
 ftl_mul_var = ( {'++' | '--'}, ftl_variable ) | ( ftl_variable, ('++' | '--') ) ;


 ftl_logical_expression = ftl_simple_logical_expr = | ftl_comparison ;

 ftl_simple_logical_expr = ftl_logical_addend, { '||', ftl_logical_addend } ;
 ftl_logical_addend = ftl_logical_multiplicand, { '&&', ftl_logical_multiplicand } ;
 ftl_logical_multiplicand = ftl_variable | ftl_func_call
   | ftl_bool | ('!', ftl_logical_multiplicand) | ( '(', ftl_simple_logical_expr, ')');


 ftl_comparison = ftl_string_comp | ftl_scalar_comp;
 ftl_string_comp = (ftl_string | ftl_variable), ftl_op_comp, (ftl_string | ftl_variable);
 ftl_op_comp = '<' | '>' | "==" | ">=" | "<=" | "!=" ;

 ftl_scalar_comp = (ftl_arith_expression, ftl_op_comp, ftl_arith_expression)
   | (ftl_simple_logical_expr, ftl_op_comp, ftl_simple_logical_expr);
```

Приложение 2. Скриншоты реализации языка

```
void Lexer::getChar()
{
	peek = program[0];
	program.erase(0, 1);
}

bool Lexer::getChar(char c)
{
	getChar();
	if (c == peek)
	{
		peek = ' ';
		return true;
	}
	else
		return false;
	}

Lexer::Lexer(string fileName)
{
	stream.open(fileName);
	while (!stream.eof())
	{
		string temp;
		getline(stream, temp);
		program += temp;
	}
	stream.close();
}


 class Token
 {
 public:
	Tag tag;
	virtual void Show() = 0;
 };

 class Symbol : public Token
 {
 public:
	Symbol(Tag t);
	void Show();
 };

 class Operator : public Token
 {
 public:
	Operator(Tag t);
	void Show();
 };

 class Word : public Token
 {
	void checkWord();
 public:
	string value;
	Word(string word);
	void Show();
 };

 class Number : public Token
 {
 public:
	long long value;
	Number(long long num);
	void Show();
 };
```

Приложение 3. Примеры программ на разработанном языке

```
int main()
{
    int a;
    int b;

    get(a, b);

    while (b != 0) {
        int t = b;
        b = a % b;
        a = t;
    }

    print(a);

    return 0;
}
```

Алгоритм Евклида

```
int main(){
    int F0=1;
    int F1=1;
    int N;
    get(N);
    for(int i=1;i<N;i++)
    {
        int tmp=F0;
        F0=F1;
        F1=F0+tmp;
    }
    print(F1);
    return 0;
}
```

Нахождение числа Фибоначчи по его порядковому номеру