МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора GIA-2021»

Выполнил студент Горбач Илья Алексеевич

(Ф.И.О.)

Руководитель проекта ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой к.т.н., доц. Пацей Н.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтролер ст.пр. Наркевич Аделина Сергеевна

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

**Введение**

Задачей данного курсового проекта была поставлена разработка транслятора своего языка программирования GIA-2021. Он предназначен для выполнения простейших операций и арифметических действий над строками и числами.

Главной задачей транслятора заключается в том, чтобы сделать исходный код на данном языке программирования понятной компьютеру. В данном курсовом проекте мой исходный код транслируется на язык ассемблера. Язык ассемблера – тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком.

Исходя из цели курсового проекта, были определены следующие задачи:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений с помощью польской инверсии;

– генерация кода на язык ассемблера;

– тестирование транслятора;

Способы решения каждой задачи будут описаны в соответствующих главах курсового проекта.

В первой главе работы определена спецификация языка программирования.

Во второй главе представлена структура транслятора. В ней перечислены компоненты транслятора, их назначения и принципы взаимодействия.

В третьей главе описана разработка лексического анализатора, который создаёт таблицы лексем и идентификаторов.

В четвертой главе описана разработка синтаксического анализатора, который выполняет разбор исходного кода в соответствии с правилами языка программирования.

В пятой главе описана разработка семантического анализатора, …

**Глава 1. Спецификация языка программирования**

**1.1. Характеристика языка программирования**

Язык программирования GIA-2021 – это процедурный язык высокого уровня, который транслируется в язык ассемблера. Он строго типизируемый и не является объектно-ориентированным.

**1.2. Алфавит языка**

Алфавит языка программирования — набор символов, которые могут использоваться при написании исходного кода.

Он состоит из различных символов, представленных ниже:

— кириллические символы верхнего и нижнего регистров;

— латинские символы верхнего и нижнего регистров;

— арабские цифры;

— знаки препинания;

— знаки арифметических и логических операций.

**1.3. Символы сепараторы**

Сепараторы необходимы для разделения операция языка. Сепараторы, используемые в языке программирования GIA-2021, приведены в таблице 1.1.

Таблица 1.1 – Сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| ;  :  ‘ ’ (пробел)  , | Разделение конструкций |
| =  +  -  \*  /  >>  << | Арифметические операции |
| {} | Программный блок инструкций |
| () | Параметры операций (в выражениях) |
| [] | Параметры функций |

**1.4. Применяемые кодировки**

Для написания исходного кода на языке программирования GIA-2021 используется кодировка Windows-1251. Windows-1251 – набор символов и кодировка, являющаяся стандартной 8-битной кодировкой для русских версий Microsoft Windows до 10-й версии.

**1.5. Типы данных**

В языке программирования GIA-2021 используются три основных типа данных, которые описываются в таблице 1.2. Пользовательские типы данных не поддерживаются.

Таблица 1.2 – Типы данных

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| ubyte | Фундаментальный тип данных, используемый для объявления беззнаковых целочисленных данных(1 байт). Без явно указанной инициализации переменной, присваивается нулевое значение. Максимальное значение 255 |
| boolean | Фундаментальный тип данных, используемый для объявления логической переменной, которая принимает одно из двух значений: true или false. Без явно указанной инициализации переменной, присваивается нулевое значение (false). |
| str | Фундаментальный тип данных, используемый для объявления строк. Без явно указанной инициализации переменной, присваивается нулевое значение (пустая строка). |

**1.6. Преобразование типов данных**

В языке программирования GIA-2021 преобразование типов данных не поддерживается.

**1.7. Идентификаторы**

Идентификаторы применяются для наименования переменных, функция и параметров. Идентификаторы, объявленные внутри функционального блока, получают префикс, который отображается в таблице идентификаторов. Зарезервированные идентификаторы не поддерживаются. Предусмотрены несколько правил составления идентификатора:

– состоит из символов латинского алфавита и цифр;

– могут начинаться с символа нижнего подчеркивания «\_»;

– максимальная длина идентификатора равна 5;

– идентификатор не может совпадать с ключевыми словами языка программирования.

**1.8. Литералы**

В языке программирования GIA-2021 существует только 2 типа литералов: целые и символьные. Их краткое описание представлено в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Тип литерала | Описание |
| Литералы целого типа | Целочисленные неотрицательные литералы, инициализируются 0. Литералы только rvalue. |
| Строковые литералы | Символы, заключённые в ‘’ (одинарные кавычки), инициализируются пустой строкой, строковые переменные. Только rvalue. Максимальное число символов в литерале . |
| Логические литералы | Логические литералы по умолчанию инициализируются значением false; |

**1.9. Область видимости идентификаторов**

В языке программирования GIA-2021 перед использованием переменной необходимо её объявление. Допускается использование переменной только внутри её области видимости. Допускается объявление переменных с одинаковыми именами в разных программных блоках.

**1.10. Инициализация данных**

Способы инициализации переменных языка программирования GIA-2021 представлены в таблице 1.4.

Таблица 1.4 – Способы инициализации переменных

|  |  |
| --- | --- |
| Вид инициализации | Примечание |
| describe <тип данных> <идентификатор>; | Автоматическая инициализация переменной. ubyte – инициализируется нулем, boolean – false, str – пустой строкой. |
| describe <тип данных> <идентификатор> = <значение>; | Инициализация переменной с присваиванием значения. |
| <идентификатор> = <значение>; | Присваивание значения переменной. |

**1.11. Инструкции языка**

Все возможные инструкции языка программирования GIA-2021 представлены в общем виде в таблице 1.5.

Таблица 1.5 – Инструкции языка программирования GIA-2021

|  |  |
| --- | --- |
| Инструкция | Запись |
| Объявление переменной | describe <тип данных> <идентификатор>;  describe <тип данных> <идентификатор> = <значение>; |
| Присваивание | <идентификатор> = [<значение>|<идентификатор>]; |
| Объявление функции | function <тип данных> <идентификатор> (<тип данных> <идентификатор>, …) {…} |
| Блок инструкций | {  …  } |
| Возврат из подпрограммы | give <выражение>; |
| Вывод данных | speak <идентификатор> / <литерал>; |
| Оператор цикла | circuit [<условие>]  {…} |

**1.12. Операции языка**

Язык программирования GIA-2021 может выполнять арифметические операции, представленные в таблице 1.6.

Таблица 1.6 – Операции и их приоритеты

|  |  |
| --- | --- |
| Операция | Приоритет операции |
| (  ) | 0 |
| , | 1 |
| >>  << | 2 |
| +  - | 3 |
| \*  / | 4 |

**1.13. Выражения и их вычисления**

Круглые скобки в выражении используются для изменения приоритета операций. Также не допускается запись двух подряд идущих арифметических операций. Выражение может содержать вызов функции. Фигурные скобки используются для составления блоков кода функций, циклов. Квадратные – для передачи параметров функций.

**1.14. Программные конструкции языка**

Ключевые программные конструкции языка программирования GIA-2021 представлены в таблице 1.7

Таблица 1.7 – программные конструкции

|  |  |
| --- | --- |
| Главная функция (точка входа в приложение) | main  {…} |
| Функция | function <тип данных> <идентификатор> (<тип данных> <идентификатор>, …)  {  …  give <выражение>;  } |

**1.15. Область видимости**

В языке программирования GIA-2021 переменные должны находиться внутри программного блока функций. Внутри разных областей видимости разрешено объявление переменных с одинаковыми именами. Все переменные, параметры или функции внутри области видимости получают префикс, который отображается в таблице идентификаторов. Объявление глобальных переменных не предусмотрено. Объявление пользовательских областей видимости не предусмотрено.

**1.16. Семантические проверки**

**…**

**1.17. Распределение оперативной памяти на этапе выполнения**

Все переменные размещаются в стеке. Таблицы лексем и идентификаторов размещены в структуры с выделенной под них оперативной памятью, которая очищается по окончанию работы транслятора.

**1.18. Стандартная библиотека и её состав**

**…**

**1.19. Ввод и вывод данных**

В языке программирования GIA-2021 ввод данных не поддерживается. Вывод данных происходит с помощью функции speak [<идентификатор> / <литерал>];

**1.20. Точка входа**

Точкой входа в языке программирования GIA-2021 является функция main.

**1.21. Препроцессор**

В языке программирования GIA-2021 имеется препроцессор. Подключение библиотек языка происходит с помощью директивы #using\_<название библиотеки>

**1.22. Соглашения о вызовах**

**…**

**1.23. Объектный код**

Язык программирования GIA-2021 транслируется в язык ассемблера.

**1.24. Классификация сообщений транслятора**

В случае возникновения ошибки в исходном коде программы на языке программирования GIA-2021 и выявлении её транслятором в файл протокола выводится сообщение. Классификация обрабатываемых ошибок приведена в таблице 1.10.

Таблица 1.?? – Классификация сообщений транслятора

|  |  |
| --- | --- |
| Интервал | Описание ошибок |
| 0-110 | Системные ошибки |
| 200-220 | Ошибки лексического анализа |
| 600-620 | Ошибки синтаксического анализа |
| 700-720 | Ошибки семантического анализа |

**1.25. Контрольный пример**

**2 Структура транслятора**

**2.1 Компоненты транслятора, их назначение и принципы взаимодействия**

В языке GIA-2021 исходный код транслируется в язык Assembler. Транслятор языка разделён на отдельные части, которые взаимодействуют между собой и выполняют отведённые им функции, которые представлены в пункте 2.1. Для того чтобы получить ассемблерный код, используется выходные данные работы лексического анализатора, а именно таблица лексем и таблица идентификаторов. Для указания выходных файлов используются входные параметры транслятора, которые описаны в таблице 2.1. Структура транслятора языка GIA-2021 приведена на рисунке 1.



Рисунок 2.1 Структура транслятора языка программирования GIA-2021

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся последовательность символов входного языка. Он производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив отдельных слов (в теории компиляции вместо термина «слово» часто используют термин «токен»). Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация. Таблица лексем (ТЛ) и таблица идентификаторов (ТИ) являются входом для следующей фазы компилятора – синтаксического анализа (разбора, парсера).

Цели лексического анализатора:

− убрать все лишние пробелы;

− выполнить распознавание лексем;

− построить таблицу лексем и таблицу идентификаторов;

− при неуспешном распознавании или обнаружении некоторых ошибок во входном тексте выдать сообщение об ошибке.

Синтаксический анализатор – часть компилятора, выполняющая синтаксический анализ, то есть проверку исходного кода на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией является дерево разбора

Семантический анализатор – часть транслятора, выполняющая семантический анализ, то есть проверку исходного кода на наличие ошибок, которые невозможно отследить при помощи регулярной и контекстно-свободной грамматики. Входными данными являются таблица лексем и идентификаторов.

Генератор кода – часть транслятора, выполняющая генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. На вход генератора подаются таблица лексем и таблица идентификаторов, на основе которых генерируется файл с ассемблерным кодом.

**2.2 Перечень входных параметров транслятора**

Для формирования файлов с результатами работы лексического, синтаксического и семантического анализаторов используются входные параметры транслятора, которые приведены в таблице 2.1.

Таблица 2.1 Входные параметры транслятора языка GIA-2021

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к in-файлу> | Файл с исходным кодом на языке GIA-2021 , имеющий расширение .txt | Не предусмотрено |
| -log:<путь к log-файлу> | Файл журнала для вывода протоколов работы программы. | Значение по умолчанию:  <имя in-файла>.log |

**2.3 Перечень протоколов, формируемых транслятором и их содержимое**

В ходе работы программы формируются протоколы работы лексического, синтаксического и семантического анализаторов, которые содержат в себе перечень протоколов работы. В таблице 2.2 приведены протоколы, формируемые транслятором и их содержимое.

Таблица 2.2 Протоколы, формируемые транслятором языка GIA-2021

|  |  |
| --- | --- |
| Формируемый протокол | Описание выходного протокола |
| Файл журнала, заданный параметром "-log:" | Файл с протоколом работы транслятора языка программирования GIA-2021 . Содержит таблицу лексем и таблицу идентификаторов, протокол работы синтаксического анализатора и дерево разбора, полученные на этапе лексического и синтаксического анализа, а также результат работы алгоритма преобразования выражений к польской записи. |
| Выходной файл, c расширением ".asm" | Результат работы программы – файл, содержащий исходный код на языке ассемблера. |

**3 Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Первая стадия работы компилятора называется лексическим анализом, а программа, её реализующая, – лексическим анализатором (сканером). На вход лексического анализатора подаётся исходный код входного языка. Лексический анализатор выделяет в этой последовательности простейшие конструкции языка,. Лексический анализатор производит предварительный разбор текста, преобразующий единый массив текстовых символов в массив токенов.

Примеры лексических единиц: идентификаторы, числа, символы операций, служебные слова и т.д. Лексический анализатор преобразует исходный текст, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Каждой лексеме сопоставляется ее тип и запись в таблице идентификаторов, в которой хранится дополнительная информация.

Функции лексического анализатора:

− удаление «пустых» символов и комментариев. Если «пустые» символы (пробелы, знаки табуляции и перехода на новую строку) и комментарии будут удалены лексическим анализатором, синтаксический анализатор никогда не столкнется с ними (альтернативный способ, состоящий в модификации грамматики для включения «пустых» символов и комментариев в синтаксис, достаточно сложен для реализации);

− распознавание идентификаторов и ключевых слов;

− распознавание констант;

− распознавание разделителей и знаков операций.

Исходный код программы представлен в приложении А, структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 Структура лексического анализатора

**3.2. Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица входных символов представлена на рисунке 3.2, категории входных символов представлены в таблице 3.1.



Рисунок 3.2. Таблица контроля входных символов

Таблица 3.1 Соответствие символов и их значений в таблице

|  |  |
| --- | --- |
| Значение в таблице входных символов | Символы |
| Разрешенный | T |
| Запрещенный | F |
| Игнорируемый | I |
| Литерал | Q |
| Сепаратор | S |
| Перевод строки | N |
| Пробел, табуляция | P |

**3.3 Удаление избыточных символов**

Избыточными символами являются символы табуляции и пробелы. Избыточные символы удаляются на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы;

2. Встреча пробела или знака табуляции является своего рода встречей символа-сепаратора;

3. В отличие от других символов-сепараторов не записываем в очередь лексем эти символы, т.е. игнорируем.

## **3.4 Перечень ключевых слов**

Лексический анализатор преобразует исходный текст, заменяя лексические единицы лексемами для создания промежуточного представления исходной программы. Соответствие токенов и лексем приведено в таблице 3.2.

Таблица 3.2 Соответствие токенов и сепараторов с лексемами

|  |  |  |
| --- | --- | --- |
| Токен | Лексема | Пояснение |
| ubyte, str,boolean | t | Названия типов данных языка. |
| Идентификатор | i | Длина идентификатора – 5 символов. |
| Литерал | l | Литерал любого доступного типа. |
| Шестнадцатеричный литерал | h | Литерал в восьмиричном представлении. |
| function | f | Объявление функции. |
| give | g | Выход из функции/процедуры. |
| main | m | Главная функция. |
| describe | d | Объявление переменной. |
| speak | s | Вывод данных в консоль. |
| circuit | c | Указывает на начало тела цикла. |
| # | # | Разделение конструкций в цикле/условном операторе. |
| ; | ; | Разделение выражений. |
| , | , | Разделение параметров функций. |
| { | { | Начало блока/тела функции. |
| } | } | Закрытие блока/тела функции. |
| [ | [ | Передача параметров в функцию |
| ] | ] | Закрытие блока для передачи параметров |
| = | = | Знак присваивания. |
| +  -  \*  /  >>  << | v | Знаки операций. |

Продолжение таблицы 3.2

|  |  |  |
| --- | --- | --- |
| >  < | >  < | Знаки логических операторов |

Пример реализации таблицы лексем представлен в приложении Б.

Каждому выражению соответствует детерминированный конечный автомат, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся токен и с помощью регулярного выражения, соответствующего данному графу переходов, происходит разбор. В случае успешного разбора выражения оно записывается в таблицу лексем. Если выражение является идентификатором или литералом, информация также заносится в таблицу идентификаторов. Структура конечного автомата и пример графа перехода конечного автомата изображены на рисунках 3.3 и 3.4 соответственно.

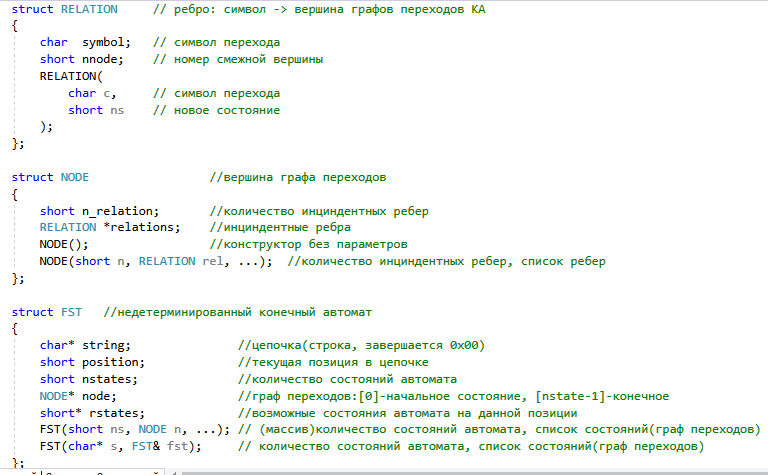


Рисунок 3.3 Структура конечного автомата

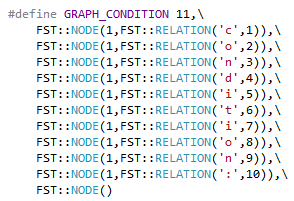


Рисунок 3.4 Пример реализации графа конечного автомата для токена condition

**3.5 Основные структуры данных**

Основными структурами данных лексического анализатора являются таблица лексем и таблица идентификаторов. Таблица лексем содержит номер лексемы, лексему (lexema), полученную при разборе, номер строки в исходном коде (sn), и номер в таблице идентификаторов, если лексема является идентификаторов (idxTI). Таблица идентификаторов содержит имя идентификатора (id), номер в таблице лексем (idxfirstLE), тип данных (iddatatype), тип идентификатора (idtype) и значение (или параметры функций) (value). Код C++ со структурой таблицы лексем представлен на рисунке 3.3. Код C++ со структурой таблицы идентификаторов представлен на рисунке 3.4.

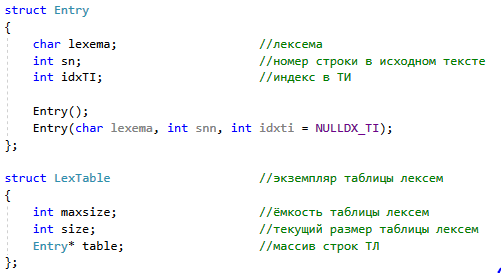


Рисунок 3.3 Структура таблицы лексем

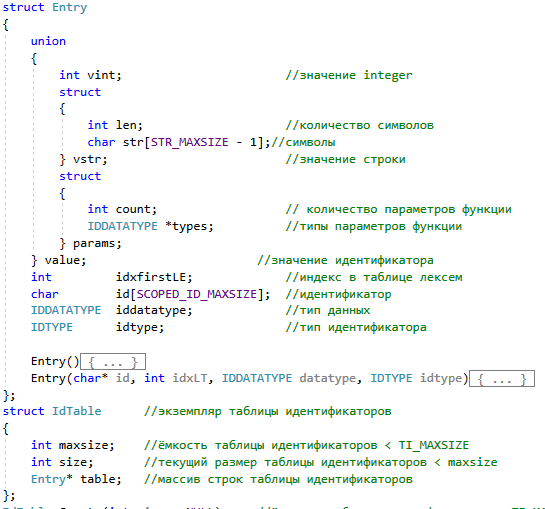


Рисунок 3.4 Структура таблицы идентификаторов

## **3.6 Принцип обработки ошибок**

Для обработки ошибок лексический анализатор использует таблицу с сообщениями. Структура сообщений содержит информацию о номере сообщения, номер строки и позицию, где было вызвано сообщение в исходном коде, информацию об ошибке. При возникновении сообщения, лексический анализатор игнорирует найденную ошибку и продолжает работу с исходным кодом. Перечень сообщений представлен на рисунке 3.5.



Рисунок 3.5 - Сообщения лексического анализатора

## **3.7 Структура и перечень сообщений лексического анализатора**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

**3.8 Параметры лексического анализатора**

Результаты работы лексического анализатора, а именно таблицы лексем и идентификаторов выводятся как в файл журнала.

**3.9 Алгоритм лексического анализа**

* проверяет входной поток символов программы на исходном языке на допустимость, удаляет лишние пробелы и добавляет сепаратор для вычисления номера строки для каждой лексемы;
* для выделенной части входного потока выполняется функция распознавания лексемы;
* при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем и таблицу идентификаторов, и алгоритм возвращается к первому этапу;
* формирует протокол работы;
* при неуспешном распознавании выдается сообщение об ошибке.

Распознавание цепочек основывается на работе конечных автоматов. Работу конечного автомата можно проиллюстрировать с помощью графа переходов. Пример графа для цепочки «**describe**» представлен на рисунке 3.2, где S0 – начальное, а S6 – конечное состояние автомата.

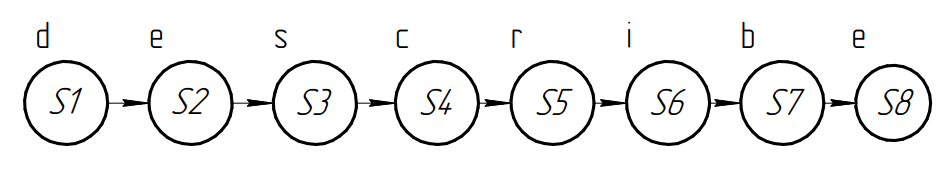


Рисунок 3.2 Пример графа переходов для цепочки describe

**3.10 Контрольный пример**

Результат работы лексического анализатора в виде таблиц лексем и идентификаторов, соответствующих контрольному примеру, представлен в приложении Б.

**4. Разработка синтаксического анализатора**

**4.1 Структура синтаксического анализатора**

Синтаксический анализатор: часть компилятора, выполняющая синтаксический анализ, то есть исходный код проверяется на соответствие правилам грамматики. Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией– дерево разбора

Описание структуры синтаксического анализатора языка представлено на рисунке 4.1.

Рисунок 4.1 Структура синтаксического анализатора.

## **4.2 Контекстно-свободная грамматика, описывающая синтаксис языка**

В синтаксическом анализаторе транслятора языка GIA-2021 используется контекстно-свободная грамматика , где T – множество терминальных символов (было описано в разделе 1.2 данной пояснительной записки),

N – множество нетерминальных символов (первый столбец таблицы 4.1),

P – множество правил языка (второй столбец таблицы 4.1),

S – начальный символ грамматики, являющийся нетерминалом.

Эта грамматика имеет нормальную форму Грейбах, т.к. она не леворекурсивная (не содержит леворекурсивных правил) и правила  имеют вид:

1. , где ; (или , или );
2. , где — начальный символ, при этом если такое правило существует, то нетерминал  не встречается в правой части правил.

Описание нетерминальных символов содержится в таблице 4.1.

Таблица 4.1 Таблица правил переходов нетерминальных символов

|  |  |  |
| --- | --- | --- |
| Символ | Правила | Какие правила порождает |
| S | S->m{NgE;};  S->tfi[F]{NgE;};S  S->tfi[F]{NgE;};  S->m{NgE;}; | Стартовые правила, описывающее общую структуру программы |
| N | N->dti=E;  N->dti=E;N  N->dti;N  N->i=E;  N->i=E;N  N->sE;N  N->sE;  N->nE;N  N->nE;  N->dti;  N->c[W]{C}e;  N->c[W]{C}e;N  N->(R){U};  N->(R){U};N  N->(R){U}o{U};  N->(R){U}o{U};N | Правила для параметров объявляемых функций |
| E | E->i  E->l  E->(E)  E->i[W]  E->iM  E->lM  E->(E)M  E->i[W]M | Правила для тела функций |
| F | G->ti  G->ti,F | Правила для тела процедур |
| W | W->i  E->l  W->i,W  W->l,W | Правила для списка параметров функции |
| M | M->vE  M->vEM | Правила для вывозов функций(в т.ч. и в выражениях) |
| K | K->[W]  K->[] | Правила для параметров вызываемых функций |
| C | C->dti=E;  C->dti=E;C  C->dti;C  C->i=E;  C->i=E;C  C->sE;C  C->sE;  C->nE;C  C->nE;  C->dti;  C->c[W]{C}e;  C->c[W]{C}e;C  C->(R){U};  C->(R){U};C  C->(R){U}o{U};  C->(R){U}o{U};C | Правила для тела цикла |
| U | U->dti=E;  U->dti=E;U  U->dti;U  U->i=E;  U->i=E;U  U->sE;U  U->sE;  U->nE;U  U->nE;  U->dti;  U->c[W]{C}e;  U->c[W]{C}e;U  U->(R){U};  U->(R){U};U  U->(R){U}o{U};  U->(R){U}o{U};U | Правила для тела условной конструкции |

**4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку, описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Д.

Таблица 4.2 – Описание компонентов магазинного автомата

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | Множество состояний автомата | Состояние автомата представляет из себя структуру, содержащую позицию на входной ленте, номера текущего правила и цепочки и стек автомата |
|  | Алфавит входных символов | Алфавит является множеством терминальных и нетерминальных символов, описание которых содержится в разделе 1.2 и в таблице 4.1. |
|  | Алфавит специальных магазинных символов | Алфавит магазинных символов содержит стартовый символ и маркер дна стека |
|  | Функция переходов автомата | Функция представляет из себя множество правил грамматики, описанных в таблице 4.1. |
|  | Начальное состояние автомата | Состояние, которое приобретает автомат в начале своей работы. Представляется в виде стартового правила грамматики (нетерминальный символ А) |
|  | Начальное состояние магазина автомата | Символ маркера дна стека ($) |
|  | Множество конечных состояний | Конечные состояние заставляют автомат прекратить свою работу. Конечным состоянием является пустой магазин автомата и совпадение позиции на входной ленте автомата с размером ленты |

**4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора представляются в виде структуры магазинного конечного автомата, выполняющего разбор исходной ленты, и структуры грамматики Грейбах, описывающей синтаксические правила языка GIA-2021 . Данные структуры в приложении В.

**4.5 Описание алгоритма синтаксического разбора**

Принцип работы автомата следующий:

1. В магазин записывается стартовый символ;
2. На основе полученных ранее таблиц формируется входная лента;
3. Запускается автомат;
4. Выбирается цепочка, соответствующая нетерминальному символу, записывается в магазин в обратном порядке;
5. Если терминалы в стеке и в ленте совпадают, то данный терминал удаляется из ленты и стека. Иначе возвращаемся в предыдущее сохраненное состояние и выбираем другую цепочку нетерминала;
6. Если в магазине встретился нетерминал, переходим к пункту 4;
7. Если наш символ достиг дна стека, и лента в этот момент пуста, то синтаксический анализ выполнен успешно. Иначе генерируется исключение.

**4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен на рисунке 4.3.

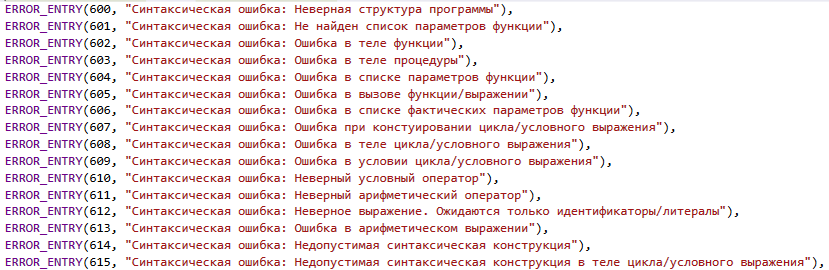


Рисунок 4.3 - Сообщения синтаксического анализатора

**4.7. Параметры синтаксического анализатора и режимы его работы**

Входной информацией для синтаксического анализатора является таблица лексем и идентификаторов. Кроме того используется описание грамматики в форме Грейбах. Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью выводятся в журнал работы программы.

**4.8. Принцип обработки ошибок**

Синтаксический анализатор выполняет разбор исходной последовательности лексем до тех пор, пока не дойдёт до конца цепочки лексем или не найдёт ошибку. Тогда анализ останавливается и выводится сообщение об ошибке (если она найдена). Если в процессе анализа находятся более трёх ошибок, то анализ останавливается.

**4.9. Контрольный пример**

Результаты работы лексического разбора, а именно дерево разбора и протокол работы автомата с магазинной памятью приведены в приложении В.

**5 Разработка семантического анализатора**

**5.1 Структура семантического анализатора**

Семантический анализатор принимает на свой вход результаты работ лексического и синтаксического анализаторов, то есть таблицы лексем, идентификаторов и результат работы синтаксического анализатора, то есть дерево разбора, и последовательно ищет необходимые ошибки. Некоторые проверки (такие как проверка на единственность точки входа, проверка на предварительное объявление переменной) осуществляются в процессе лексического анализа. Общая структура обособленно работающего (не параллельно с лексическим анализом) семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1. Структура семантического анализатора

**5.2 Функции семантического анализатора**

Семантический анализатор выполняет проверку на основные правила языка (семантики языка), которые описаны в разделе 1.16.

**5.3 Структура и перечень сообщений семантического анализатора**

Сообщения, формируемые семантическим анализатором, представлены на рисунке 5.1.

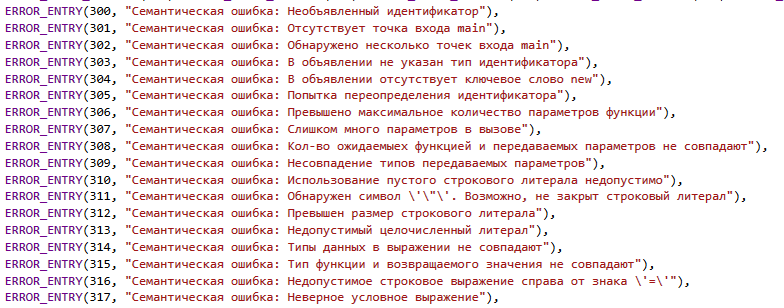


Рисунок 5.2 – Перечень сообщений семантического анализатора

**5.4 Принцип обработки ошибок**

Ошибки, возникающие в процессе трансляции программы, фиксируются в протокол, заданный входным параметрами. В случае возникновения ошибок происходит их протоколирование с номером ошибки и диагностическим сообщением. Анализ останавливается после того, как будут найдены все ошибки.

**5.5 Контрольный пример**

Соответствие примеров некоторых ошибок в исходном коде и диагностических сообщений об ошибках приведено в таблице 5.1.

Таблица 5.1. Примеры диагностики ошибок

|  |  |
| --- | --- |
| Исходный код | Текст сообщения |
| main[  number x = 9;  Write x;  ] | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 2 |
| main[  new number x = 9;  new string y =x;  ] | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 3 |
| main[  new number x = 9;  ]  main[  new string y = "qwerty";  ] | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0 |

**6. Вычисление выражений**

**6.1 Выражения, допускаемые языком**

В языке GIA-2021 допускаются вычисления выражений целочисленного типа данных с поддержкой вызова функций внутри выражений. Приоритет операций представлен на таблице 6.1.

Таблица 6.1. Приоритеты операций

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( ) | 3 |
| \* | 2 |
| / | 2 |
| + | 1 |
| - | 1 |
| } | 0 |
| { | 0 |

**6.2 Польская запись и принцип её построения**

Все выражения языка GIA-2021 преобразовываются к обратной польской записи.

Польская запись - это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок. Существует два типа польской записи: прямая и обратная, также известные как префиксная и постфиксная. Отличие их от классического, инфиксного способа заключается в том, что знаки операций пишутся не между, а, соответственно, до или после аргументов. Алгоритм построения польской записи:

* исходная строка: выражение;
* результирующая строка: польская запись;
* стек: пустой;
* исходная строка просматривается слева направо;
* операнды переносятся в результирующую строку;
* операция записывается в стек, если стек пуст;
* операция выталкивает все операции с большим или равным приоритетом в результирующую строку;
* отрывающая скобка помещается в стек;
* закрывающая скобка выталкивает все операции до открывающей скобки, после чего обе скобки уничтожаются.

**6.3 Программная реализация обработки выражений**

Программная реализация алгоритма преобразования выражений к польской записи представлена в приложении Г.

**6.4 Контрольный пример**

Пример преобразования выражений из контрольных примеров к обратной польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления и преобразования к ассемблерному коду. В приложении Г приведены изменённые таблицы лексем и идентификаторов, отображающие результаты преобразования выражений в польский формат.

Таблица 6.2. Преобразование выражений к ПОЛИЗ

|  |  |
| --- | --- |
| Выражение | Обратная польская запись для выражения |
| i[2]=(((l[3]+l[4])-i[0])\*l[5])/l[6]; | i[2]=l[3]l[4]+i[0]-l[5]\*l[6]/ |
| i[23]=(i[23]+l[26])\*l[26] | i[23]=i[23]l[26]+l[26]\* |
| i[3]=(((l[4]+l[5])-i[0])\*l[6]) | i[3]=l[4]l[5]+i[0]-l[6]\* |

**7. Генерация кода**

**7.1 Структура генератора кода**

В языке GIA-2021 генерация кода является заключительным этапом трансляции. Генератор принимает на вход таблицы лексем и идентификаторов, полученные в результате лексического анализа. В соответствии с таблицей лексем строится выходной файл на языке ассемблера, который будет являться результатом работы транслятора. В случае возникновения ошибок генерация кода не будет осуществляться. Структура генератора кода GIA-2021 представлена на рисунке 7.1.



Рисунок 7.1 – Структура генератора кода

**7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены сегментах .data и .const языка ассемблера. Соответствия между типами данных идентификаторов на языке GIA-2021 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка GIA-2021 и языка ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке GIA-2021 | Тип идентификатора на языке ассемблера | Пояснение |
| ubyte | dword | Хранит беззнаковый целочисленный тип данных. |
| str | byte | Хранит указатель на начало строки. |
| boolean | dword | Хранит логический тип данных |

**7.3 Статическая библиотека**

В языке GIA-2021 предусмотрена статическая библиотека. Статическая библиотека содержит функции, написанные на языке C++. Объявление функций статической библиотеки генерируется автоматически в коде ассемблера. Объявление функций статической библиотеки генерируется автоматически.

Таблица 7.3 – Функции статической библиотеки

|  |  |
| --- | --- |
| Функция | Назначение |
| void IntOut(int a) | Вывод на консоль целочисленной переменной а |
| void IntOutLn(int a) | Вывод на консоль целочисленной переменной a с переводом курсора на новую строку |
| void StrOut(char\* ptr) | Вывод на консоль строки str |
| void StrOutLn(char\* ptr) | Вывод на консоль строки str с переводом курсора на новую строку |
| int \_strlen(char\* ptr) | Вычисление длины строки ptr |

Продолжение таблицы 7.3

|  |  |
| --- | --- |
| char\* \_strcopy(char\* s1, char\* s2) | Копирование строки s2 в s1 |
| int \_exp(short num, short exp) | Возведение числа num в степень exp |
| int \_random(short min, short max) | Генерация случайного числа из диапазона [min, max] |

**7.4 Особенности алгоритма генерации кода**

В языке GIA-2021 генерация кода строится на основе таблиц лексем и идентификаторов. Общая схема работы генератора кода представлена на рисунке

Рисунок 7.2 – Структура генератора кода

**7.5 Входные параметры генератора кода**

На вход генератору кода поступают таблицы лексем и идентификаторов исходного код программы на языке GIA-2021. Результаты работы генератора кода выводятся в файл с расширением .asm.

**7.6 Контрольный пример**

Результат генерации ассемблерного кода на основе контрольного примера из приложения А приведен в приложении Д. Результат работы контрольного примера приведён на рисунке 7.2.

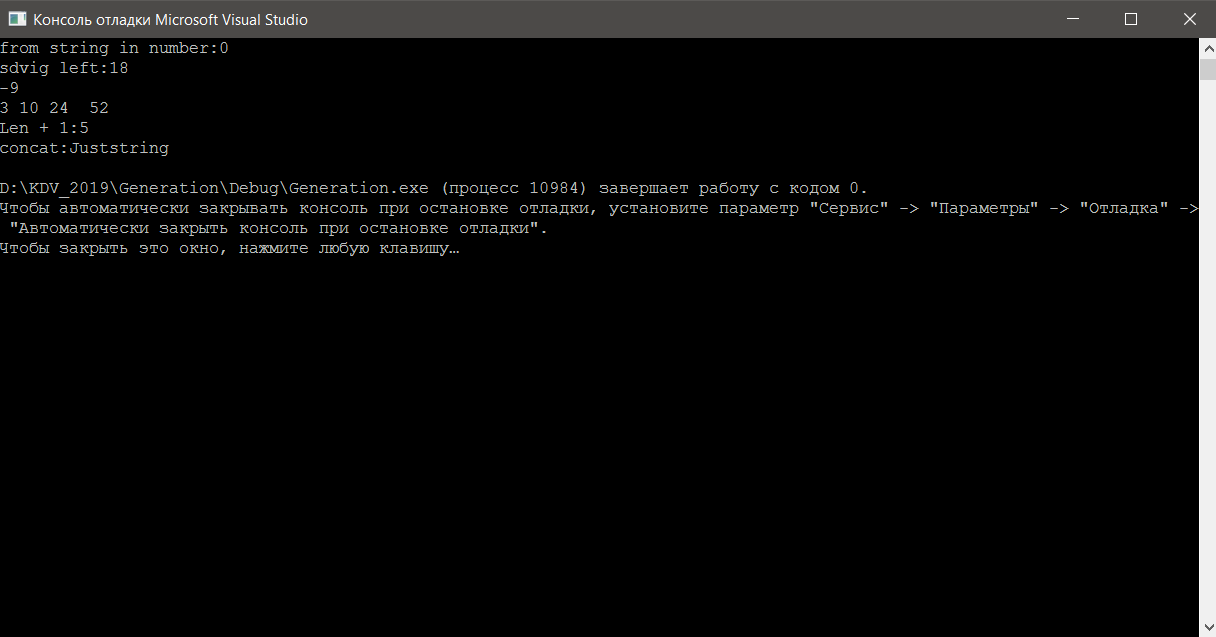


Рисунок 7.2 Результат работы программы на языке GIA-2021

**8. Тестирование транслятора**

**8.1 Тестирование проверки на допустимость символов**

В языке GIA-2021 не разрешается использовать запрещённые входным алфавитом символы. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 - Тестирование проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main [ё] | Ошибка N200: Лексическая ошибка: Недопустимый символ в исходном файле(-in) Строка: 2 Позиция в строке: 2 |

**8.2 Тестирование лексического анализатора**

На этапе лексического анализа в языке GIA-2021 могут возникнуть ошибки, описанные в пункте 3.7. Результаты тестирования лексического анализатора показаны в таблице 8.2.

Таблица 8.2 - Тестирование лексического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main  [new number x11;] | Ошибка N201: Лексическая ошибка: Неизвестная последовательность символов Строка: 3 |

**8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа в языке GIA-2021 могут возникнуть ошибки, описанные в пункте 4.6. Результаты тестирования синтаксического анализатора показаны в таблице 8.3.

Таблица 8.3 - Тестирование синтаксического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main new number x; ] | Ошибка 600: строка 1, Синтаксическая ошибка: Неверная структура программы |
| string function fi([]  main[] | Ошибка 601: строка 1, Синтаксическая ошибка: Не найден список параметров функции |
| string function fi()  [newline; write]  main[] | Ошибка 602: строка 1, Синтаксическая ошибка: Ошибка в теле функции |
| procedure function fi()  [newline write;]  main[] | Ошибка 603: строка 1, Синтаксическая ошибка: Ошибка в теле процедуры |

Продолжение таблицы 8.3

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| procedure function fi(number number)[]  main[] | Ошибка 604: строка 1, Синтаксическая ошибка: Ошибка в списке параметров функции |
| string function fi(number x)[return 3;]  main [newline;fi(5,5; newline;] | Ошибка 605: строка 2, Синтаксическая ошибка: Ошибка в вызове функции/выражении |
| string function fi(number x)[return 3;]  main [newline;fi(5,5,5 5);] | Ошибка 606: строка 2, Синтаксическая ошибка: Ошибка в списке фактических параметров функции |
| main [new number x; condition: x > 2 # cycle #] | Ошибка 607: строка 1, Синтаксическая ошибка: Ошибка при конструировании цикла/условного выражения |
| main [new number x; condition: x > 2 # cycle #] | Ошибка 608: строка 1, Синтаксическая ошибка: Ошибка в теле цикла/условного выражения |
| main [condition: 1 = 2 #] | Ошибка 609: строка 1, Синтаксическая ошибка: Ошибка в условии цикла/условного выражения |
| main [condition: 1 = 2 #] | Ошибка 610: строка 1, Синтаксическая ошибка: Неверный условный оператор |
| main [new number x; x = x ! x;] | Ошибка 611: строка 1, Синтаксическая ошибка: Неверный арифметический оператор |
| main [new number x; write new;] | Ошибка 612: строка 1, Синтаксическая ошибка: Неверное выражение. Ожидаются только идентификаторы/литералы |
| main [new number x; x = 1 ++;] | Ошибка 613: строка 1, Синтаксическая ошибка: Ошибка в арифметическом выражении |
| main [newline; 4;] | Ошибка 614: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция |
| main [new number a; condition: a < 3 # istrue [newline; 3;] #] | Ошибка 615: строка 1, Синтаксическая ошибка: Недопустимая синтаксическая конструкция в теле цикла/условного выражения |

**8.4 Тестирование семантического анализатора**

Семантический анализ в языке GIA-2021 содержит множество проверок по семантическим правилам, описанным в пункте 1.16. Итоги тестирования семантического анализатора на корректное обнаружение семантических ошибок приведены в таблице 8.4.

Таблица 8.4 - Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main [ a = 1 ] | Ошибка N300: Семантическая ошибка: Необъявленный идентификатор Строка: 1 |
| string function fi()[] | Ошибка N301: Семантическая ошибка: Отсутствует точка входа main Строка: 0 |
| main[]  main[] | Ошибка N302: Семантическая ошибка: Обнаружено несколько точек входа main Строка: 0 |
| main[a = 1;] | Ошибка N304: Семантическая ошибка: В объявлении отсутствует ключевое слово new Строка: 1 |
| main[new number t;  new string t;] | Ошибка N305: Семантическая ошибка: Попытка переопределения идентификатора Строка: 3 |
| procedure function fi()[]  main[fi(“a”,”b”,”c”,”d”] | Ошибка N307: Семантическая ошибка: Слишком много параметров в вызове Строка: 1 |
| string function fi(string x, string y, string z, string s)  main[] | Ошибка N306: Семантическая ошибка: Превышено максимальное количество параметров функции Строка: 1 |
| string function fi(string x)[return "a";]  main[fi("a", "b");] | Ошибка N308: Семантическая ошибка: Кол-во ожидаемых функцией и передаваемых параметров не совпадают Строка: 2 |
| string function fi(string x)[return "a";]  main[fi("a", "b");] | Ошибка N309: Семантическая ошибка: Несовпадение типов передаваемых параметров Строка: 2 |
| main[new string x="";] | Ошибка N310: Семантическая ошибка: Использование пустого строкового литерала недопустимо Строка: 1 |
| main[new string x=";] | Ошибка N311: Семантическая ошибка: Обнаружен символ '"'. Возможно, не закрыт строковый литерал Строка: 1 |
| main[new number x=99999999999999999;] | Ошибка N313: Семантическая ошибка: Недопустимый целочисленный литерал Строка: 1 |
| main[new number x; x = 5 + "abc";] | Ошибка N314: Семантическая ошибка: Типы данных в выражении не совпадают Строка: 1 |
| string function fi()[return 5;]  main[newline;] | Ошибка N315: Семантическая ошибка: Тип функции и возвращаемого значения не совпадают Строка: 1 |
| main[new string x; x = "abc" + "d";] | Ошибка N316: Семантическая ошибка: Недопустимое строковое выражение справа от знака '=' Строка: 1 |
| main  [condition: "string"& 6#  istrue[write "string";]] | Ошибка N317: Семантическая ошибка: Неверное условное выражение Строка: 1 |
| main[new number a =5;  a = a/0; write a;] | Ошибка N318: Семантическая ошибка: Деление на ноль Строка: 4 |

# Заключение

В ходе выполнения курсовой работы был разработан транслятор и генератор кода для языка программирования GIA-2021 со всеми необходимыми компонентами. Таким образом, были выполнены основные задачи данной курсовой работы:

1. Сформулирована спецификация языка GIA-2021;
2. Разработаны конечные автоматы и важные алгоритмы на их основе для эффективной работы лексического анализатора;
3. Осуществлена программная реализация лексического анализатора, распознающего допустимые цепочки спроектированного языка;
4. Разработана контекстно-свободная, приведённая к нормальной форме Грейбах, грамматика для описания синтаксически верных конструкций языка;
5. Осуществлена программная реализация синтаксического анализатора;
6. Разработан семантический анализатор, осуществляющий проверку используемых инструкций на соответствие логическим правилам;
7. Разработан транслятор кода на язык ассемблера;
8. Проведено тестирование всех вышеперечисленных компонентов.

Окончательная версия языка GIA-2021 включает:

1. 3 типа данных;
2. Поддержка операторов вывода и перевода строки;
3. Возможность вызова функций стандартной библиотеки;
4. Наличие 6 арифметических операторов для вычисления выражений;
5. Поддержка функций, процедур, операторов цикла;
6. Структурированная и классифицированная система для обработки ошибок пользователя.

Проделанная работа позволила получить необходимое представление о структурах и процессах, использующихся при построении трансляторов, а также основные различия и преимущества тех или иных средств трансляции.

# Список использованных источников

1. Курс лекций по ЯП Наркевич А.С.

2. Ахо, А. Компиляторы: принципы, технологии и инструменты / А. Ахо, Р. Сети, Дж. Ульман. – M.: Вильямс, 2003. – 768с.

3. Герберт, Ш. Справочник программиста по C/C++ / Шилдт Герберт. - 3-е изд. – Москва : Вильямс, 2003. - 429 с.

4. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М., 2006 — 1104 c.

5. Страуструп, Б. Принципы и практика использования C++ / Б. Страуструп – 2009 – 1238 с