МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной Инженерии

Специальность 6-05-0612-01 Программная инженерия

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора BAI – 2024»

Выполнил студент Барило Александр Иванович

(Ф.И.О.)

Руководитель проекта преп.-стажер Гончар Е.А.

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

Заведующий кафедрой к.т.н., доц. Смелов В.В.

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

Консультанты преп.-стажер Гончар Е.А.

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

Нормоконтролер преп.-стажер Гончар Е.А.

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

Курсовой проект защищён с оценкой

Минск 2024

**Содержание**

[Введение 5](#_Toc122449900)

[1. Спецификация языка программирования 6](#_Toc122449901)

[1.1 Характеристика языка программирования 6](#_Toc122449902)

[1.2 Определение алфавит языка программирования 6](#_Toc122449903)

[1.3 Применяемые сепараторы 6](#_Toc122449904)

[1.4 Применяемые кодировки 7](#_Toc122449905)

[1.5 Типы данных 7](#_Toc122449906)

[1.6 Преобразование типов данных 8](#_Toc122449907)

[1.7 Идентификаторы 8](#_Toc122449908)

[1.8 Литералы 8](#_Toc122449909)

[1.9 Область видимости идентификаторов 9](#_Toc122449910)

[1.10 Инициализация данных 9](#_Toc122449911)

[1.11 Инструкции языка 9](#_Toc122449912)

[1.13 Выражения и их вычисления 11](#_Toc122449913)

[1.14 Программные конструкции языка 11](#_Toc122449914)

[1.15 Область видимости идентификаторов 11](#_Toc122449915)

[1.16 Семантические проверки 11](#_Toc122449916)

[1.17 Распределение оперативной памяти на этапе выполнения 12](#_Toc122449917)

[1.18 Стандартная библиотека и её состав 12](#_Toc122449918)

[1.19 Ввод и вывод данных 13](#_Toc122449919)

[1.20 Точка входа 13](#_Toc122449920)

[1.21 Препроцессор 13](#_Toc122449921)

[1.22 Соглашения о вызовах 14](#_Toc122449922)

[1.23 Объектный код 14](#_Toc122449923)

[1.24 Классификация сообщений транслятора 14](#_Toc122449924)

[1.25 Контрольный пример 14](#_Toc122449925)

[2 Структура транслятора 15](#_Toc122449926)

[2.1 Компоненты транслятора их назначение и принципы взаимодействия 15](#_Toc122449927)

[2.2 Перечень входных параметров транслятора 16](#_Toc122449928)

[2.3 Перечень протоколов, формируемые транслятором 16](#_Toc122449929)

[3 Разработка лексического анализатора 17](#_Toc122449930)

[3.1 Структура лексического анализатора 17](#_Toc122449931)

[3.2 Контроль входных символов 17](#_Toc122449932)

[3.3 Удаление избыточных символов 18](#_Toc122449933)

[3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов 18](#_Toc122449934)

[3.5 Основные структуры данных 19](#_Toc122449935)

[3.6 Структура и перечень сообщений лексического анализатора 20](#_Toc122449936)

[3.7 Принцип обработки ошибок 20](#_Toc122449937)

[3.8 Параметры лексического анализатора 20](#_Toc122449938)

[3.9 Алгоритм лексического анализа 20](#_Toc122449939)

[3.10 Контрольный пример 21](#_Toc122449940)

[4 Разработка синтаксического анализатора 22](#_Toc122449941)

[4.1 Структура синтаксического анализатора 22](#_Toc122449942)

[4.2 Контекстно свободная грамматика, описывающая синтаксис языка 22](#_Toc122449943)

[4.3 Построение конечного магазинного автомата 24](#_Toc122449944)

[4.4 Основные структуры данных 25](#_Toc122449945)

[4.5 Описание алгоритма синтаксического разбора 25](#_Toc122449946)

[4.6 Структура и перечень сообщений синтаксического анализатора 26](#_Toc122449947)

[4.7 Параметры синтаксического анализатора и режимы его работы 26](#_Toc122449948)

[4.8 Принцип обработки ошибок 26](#_Toc122449949)

[4.9 Контрольный пример 26](#_Toc122449950)

[5 Разработка семантического анализатора 27](#_Toc122449951)

[5.1 Структура семантического анализатора 27](#_Toc122449952)

[5.2 Функции семантического анализатора 27](#_Toc122449953)

[5.3 Структура и перечень сообщений семантического анализатора 27](#_Toc122449954)

[5.4 Принцип обработки ошибок 28](#_Toc122449955)

[5.5 Контрольный пример 28](#_Toc122449956)

[5.6 Выражения, допускаемые языком 29](#_Toc122449957)

[5.7 Польская запись и принцип ее построения 29](#_Toc122449958)

[5.8 Программная реализация обработки выражений 30](#_Toc122449959)

[5.9 Контрольный пример 30](#_Toc122449960)

[6 Генерация кода 31](#_Toc122449961)

[6.1 Структура генератора кода 31](#_Toc122449962)

[6.2 Представление типов данных в оперативной памяти 31](#_Toc122449963)

[6.3 Статическая библиотека 32](#_Toc122449964)

[6.4 Алгоритма генерации кода 33](#_Toc122449965)

[6.5 Контрольный пример 35](#_Toc122449966)

[7 Тестирование транслятора 36](#_Toc122449967)

[7.1 Тестирование лексического анализатора 36](#_Toc122449968)

[7.2 Тестирование синтаксического анализатора 36](#_Toc122449969)

[7.3 Тестирование семантического анализатора 36](#_Toc122449968)

[Заключение 39](#_Toc122449972)

[Список использованных источников 40](#_Toc122449973)

[ПРИЛОЖЕНИЕ А 41](#_Toc122449974)

[ПРИЛОЖЕНИЕ Б 42](#_Toc122449975)

[ПРИЛОЖЕНИЕ В 43](#_Toc122449976)

[ПРИЛОЖЕНИЕ Г 54](#_Toc122449977)

[ПРИЛОЖЕНИЕ Д 57](#_Toc122449978)

[ПРИЛОЖЕНИЕ Е 59](#_Toc122449979)

[ПРИЛОЖЕНИЕ Ж 63](#_Toc122449980)

# **Введение**

Целью выполнения курсового проекта по дисциплине «Конструирование программного обеспечения» является разработка транслятора собственного языка программирования BAI-2024 на языке C++, который будет транслироваться в язык ассемблера.

В процессе выполнения курсового проекта будут изучены и приобретены:

– навыки проектирования систем программирования;

– основы теории формальных грамматик и общей теории компиляторов;

– навыки разработки ПО для систем программирования.

Для успешного завершения курсового проекта следует разработать:

– спецификацию языка программирования;

– программную реализацию лексического анализатора;

– программную реализацию синтаксического анализатора;

– программную реализацию семантического анализатора;

– программную реализацию генератора кода.

Также после разработки проекта следует провести его тестирование. Результаты тестирования необходимо внести в пояснительную записку.

Транслятор BAI-2024 будет состоять из следующих частей:

– лексический анализатор;

– синтаксический анализатор;

– семантический анализатор;

– генератор исходного кода на языке ассемблера.

# **1 Спецификация языка программирования**

## **Характеристика языка программирования**

Язык программирования BAI-2024 представляет собой процедурный, компилируемый, не объектно-ориентированный и высокого уровня язык. Он обладает строгой типизацией, и относится к категории транслируемых языков программирования. Преобразование типов осуществляется с помощью стандартной библиотеки.

## **Определение алфавит языка программирования**

В языке программирования BAI-2024 можно использовать прописные и строчные буквы латиницы, цифры, спецсимволы. Разрешённые спецсимволы в языке: !, @, %,\*, (, ), -, +, =, \, ‘, <, > , ; , :, {, } и запятая. В качестве значения строки можно использовать также буквы кириллического алфавита, символы начиная с символа «\t» в Windows-1251 и, заключив в два символа «косая черта», можно вписать такие спецсимволы, как «двойная кавычка» и «одинарная кавычка», «косая черта» и «обратная косая черта» , табуляцию и перенос на новую строку, спецсимвол «at», который предназначен для написания комментариев в исходном коде.

## **Применяемые сепараторы**

Сепараторы языка программирования BAI-2024 приведены в таблице 1.1 спецсимвол at, внутри которых можно закомментировать строку всех возможных букв и символов.

Таблица 1.1 – Сепараторы

|  |  |  |
| --- | --- | --- |
| Сепаратор | Название | Область применения |
| ‘ ’ | Пробел | Используется для разделения компонентов кода |
| {…} | Фигурные скобки | Используются для заключения тело функции |
| ; | Точка с запятой | Используется для разделение конструкций |
| (…) | Круглые скобки | Используются в арифметических выражениях для изменения приоритета операций, при определении параметров функции, при вызове функции, в условном операторе при определении условия и при заключении выражения |
| , | Запятая | Используется при разделении параметров и аргументов |
| “…” | Двойные кавычки | Используется при определении строкового литерала |
| ‘…’ | Одинарные кавычки | Используется при определении символьного литерала |
| = | Знак «равно» | Используется при присваивании значения |
| >  >=  <  <=  :  :! | Знаки «больше», «больше или равно», «меньше», «меньше или равно», «равенство», «не равенство» | Используются в условных выражениях |
| /  <<  >>  +  -  \*  % | Знаки «косая черта», «оператор сдвига влево»,  «оператор сдвига вправо»,  «плюс», «минус», «знак умножения», «оператор остатка от деления» | Используются в выражениях |

## **Применяемые кодировки**

Для написания исходного кода на языке программирования BAI-2024 используется кодировка Windows-1251.

## **Типы данных**

В языке BAI-2024 реализованы семь типов данных: целочисленный(int), целочисленный беззнаковый(uint), число с плавающей точкой, символьный(char), строковый(string), логический(bool), неявно типизированный(let). Пользовательские типы данных не поддерживаются.

Таблица 1.2 – Типы данных языка BAI-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный тип данных(int) | Хранит в себе целочисленное значение. Максимальноезначение – 2147483647, минимальное значение – -2147483648. Размер: 4 байта. Применяемые операции: сложение, вычитание, умножение, деление, получение остатка от деления, битовый сдвиг вправо и влево. Также целочисленные переменные могут участвовать в условных операциях, однако только со своим типом данным. |
| Целочисленный беззнаковый(uint) | Хранит в себе целочисленное значение. Максимальное значение – 4 294 967 295, минимальное – 0. Размер: 4 байта. Применяемые операции: сложение, вычитание, умножение, деление, получение остатка от деления, битовый сдвиг вправо и влево. Также целочисленные беззнаковые переменные могут участвовать в условных операциях, однако только со своим типом данным. |
| Число с плавающей точкой(float) | Содержит числа с плавающей точкой. Диапазон значений для типа данных float определяется стандартом IEEE 754. Он составляет примерно от 1.2E-38 до 3.4E+38. Это означает, что float может представлять числа в этом диапазоне, включая как положительные, так и отрицательные значения. Размер: 4 байта. В языке программирования BAI-2024 с числами с плавающей точкой можно совершать сложение, вычитание, деление, деление с получением остатка |
| Символьный(char) | Значение может состоять только из одного символа, буквы, цифры или специального символа, заключённого в одинарные кавычки. Размер: 4 байт. |
| Строковой(string) | Значение строки может включать любые символы, но должно быть заключено в двойные кавычки. Максимальная допустимая длина строки – 256. В языке BAI-2024 строки можно использовать в такой условной операции,как сравнение, однако только с переменными типа string. Также с помощью стандартной библиотеки можно совершать конкатенацию строк и преобразовании строки в число. |
| Логический(bool) | Значение может быть только true или false. Размер: 4 байт. Применяемые операции: условные операции. |
| Неявно типизированный  (let) | Переменной, которая была объявлена с помощью let, тип данных определяется по литералу, который был присвоен при её объявлении. |

Пользовательские типы данных не поддерживаются.

## **Преобразование типов данных**

Преобразование типов данных не поддерживается, что делает язык строго типизированным. Однако в стандартной библиотеке имеются функции для преобразования строки в число, при условии, что строка содержит только числовые значения, число любого типа языка программирования BAI в строку,логический тип в строку, символ в строку .

## **Идентификаторы**

В идентификаторе могут использоваться буквы латинского алфавита и цифры, однако он не может начинаться с цифры. Максимальная длина идентификатора составляет 12 символов. Если вы введёте идентификатор, превышающий это количество символов, возникнет ошибка от лексического анализатора. Также имя идентификатора не должно совпадать с именем уже существующей функции из стандартной библиотеки и с ключевым словом языка программирования. БНФ идентификатора:

<идентификатор> ::= <буква> { (<буква> | <цифра>) } ;

<буква>::=a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | q | r | s | t | u | v | w | x | y | z | p | Q | W | E | R | T | Y | U | I | O | P | A | S | D | F | G | H | J | K | L | Z | X | C | V | B | N | M ;

<цифра>::=0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 ;

Примеры правильных идентификаторов:

– time ;

– counterOfProcess;

– tree1;

Примеры неправильных идентификаторов:

– 12time ;

– first\_node;

– counter-of-process;

## **Литералы**

С помощью литералов осуществляется инициализация переменных. В языке существует 5 типов литералов. Они представлены в таблице 1.3

Таблица 1.3 – Описание литералов

|  |  |  |  |
| --- | --- | --- | --- |
| Тип литерала | Регулярное выражение | Описание | Пример |
| Целочисленный литерал | [-+]?\d+ | Целочисленный литерал. Можно присвоить к переменной типа данных int | int sum = 10;  10 – целочисленный литерал. |
| Целочисленный беззнаковый литерал | \d+ | Целочисленный литерал без знака.Можно присвоить к переменной типа данных uint | uint sum =15;  15 – целочисленный литерал. |
| Численный литерал с плавающей точкой | [-+]?\d\*\.\d+([-+]?\d+)? | Численный литерал с плавающей точкой.Можно присвоить к переменной типа данных float | float fl =4.5;  4.5 – Численный литерал с плавающей |
| Символьный литерал | '[^']'  @, ‘, \n \t должны в символ записаны таким образом: ’/@/’ | Символ, заключённый в ‘’ (одинарные кавычки). | char symb = ‘K’;  K – символьный литерал. |
| Строковый литерал | "[^"\n]\*"  @, “, \n \t должны в символ записаны таким образом: ”/@/” | Строковые литералы, максимальная длина строки 255 символов( 256 символом всегда должен выступать нулевой бит) | string message= “Hello World!!!/\n/”  Hello World!!!/\n/ –  Строковой литерал |
| Логический литерал | true|false | Логический литерал может иметь только два значения.Может присваиваться к переменным типа данных bool | bool open = false;  false – логический литерал |

## **Область видимости идентификаторов**

В языке BAI-2024 область видимости переменных основывается на принципе вложенности. Переменные можно объявлять как внутри функций, так и за их пределами. Переменные могут быть созданы как внутри функций, так и вне их. Глобальные переменные необходимо объявить перед всеми функциями. Можно создавать переменные с одинаковыми именами, если они объявлены в разных областях видимости. Каждая переменная имеет свою область видимости, которая записывается в таблицу идентификаторов. Функции стандартной библиотеки можно вызывать в любой части кода.

## **Инициализация данных**

При объявлении переменной в языке BAI-2024 инициализация обязательна. Но в коде после объявлении переменной можно поменять её значение.

## **Инструкции языка**

Все возможные инструкции языка программирования BAI-2024 представлены в таблице 1.4

Таблица 1.4 – Инструкции языка программирования BAI-2024

|  |  |
| --- | --- |
| Инструкция | Запись на языке BAI-2024 |
| Объявление глобальной переменной | <тип данных> <идентификатор> = <литерал>; |
| Объявление локальной переменной | <тип данных> <идентификатор> = <литерал>;  <тип данных> <идентификатор> = <вызов функции>;  <тип данных> <идентификатор> = <выражение>;  <тип данных> <идентификатор> = <идентификатор>; |
| Объявление функции | fn <тип данных возвращаемого значения> <идентификатор> (<тип данных параметра> <идентификатор>, …) {<тело функции>return<возвращаемое значение>;} |
| Возвращение значение из функции | return <идентификатор>;  return <литерал>;  return <выражение>;  return <вызов функции> |
| Вывод данных | print (<идентификатор>|<литерал>);  println(<литерал>|<идентификатор>); |
| Комментарий | @<комментарий>@ |
| Объявление главной функции(точки входа) | fn <тип данных возвращаемого значения>main(){<тело функции>return<литерал>;} |
| Инструкция цикла | while(<условие>){<тело условия>}; |
| Условная инструкция | if(<условие>){<блок кода>}else{<блок кода>} |
| Вызов функции | <идентификатор функции>(<аргументы>) |

Инструкции (кроме функций,инструкции цикла и условной инструкции) требуют закрывающую «;».

* 1. **Операции языка**

Язык программирования BAI-2024 может выполнять операции, представленные в таблице 1.5.

Таблица 1.5 – Операции языка программирования BAI-2024

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| ( | Приоритет операций | - | int v = (a + b) \* c; |
| ) |
| + | Суммирование | (uint, uint)  (int, int)  (float,float) | int sum = a + b; |
| - | Вычитание | (uint, uint)  (int, int)  (float,float) | int diff = a – b; |

Продолжение таблицы 1.5

|  |  |  |  |
| --- | --- | --- | --- |
| Операция | Примечание | Типы данных | Пример |
| \* | Умножение | (uint, uint)  (int, int)  (float,float) | int mul = a\*b; |
| / | Деление | (uint, uint)  (int, int)  (float,float) | int div = a/b; |
| % | Остаток от деления | (uint, uint)  (int, int)  (float,float) | int mod = a%b; |
| = | Присваивание | (uint, uint)  (int, int)  (float,float)  (char, char)  (bool, bool)  (string, string)  (let, все литералы языка BAI) | let sum = 15;  let c = ‘T’; |
| <,>  <=, >= | Знаки «больше», «меньше»,«больше или равно», «меньше или равно» для условной инструкции | (uint, uint)  (int, int)  (float,float) | if(sum < diff) {  …  } |
| :, :! | Знаки «равенство», «неравенство» для условной операции | (uint, uint)  (int, int)  (float,float)  (char, char)  (bool, bool)  (string, string) | while(c:!’k’) {  …  } |
| <<,>> | Операции битового сдвига | (uint, uint)  (int, int) | int result = 10<<4 |

В языке программирования BAI-2024 с помощью функции стандартной библиотеки можно совершить также конкатенацию строк

## **Выражения и их вычисления**

Арифметические выражения используют круглые скобки для изменения приоритета операций. Две подряд идущие арифметические операции не допускаются. Выражения вычисляются только после оператора присваивания и в порядке приоритета арифметических операций, если их несколько.В выражении могут участвовать переменные и литералы одного типа данных. Также выражение может содержать вызов функции, если тип данных возвращаемого значения совпадает с типом данных других переменных или литералов выражения.

В условных выражениях, как и в арифметических, могут использоваться переменные, литералы и возвращаемые значения функций одного типа данных. В условных выражениях допускается только один условный оператор. Если в условном выражении присутствует арифметическое выражение, его необходимо заключить в круглые скобки.

## **Программные конструкции языка**

Ключевые программные конструкции языка программирования BAI-2024 представлены в таблице 1.6.

Таблица 1.7 – Программные конструкции языка BAI-2024

|  |  |
| --- | --- |
| Конструкция | Запись на языке BAI-2024 |
| Главная функция (точка входа) | fn int main(){…return 0;} |
| Функция | fn <тип возвращаемого значения> <идентификатор> (<тип> <идентификатор>, …)  {  …  return <идентификатор> | <литерал> | <выражение> |<вызов функции>;  } |
| Цикл | while(count > 0){ …}; |
| Условный оператор | if(check |! true){…}else {…}  if(check |! true){…} |

Программные конструкции языка программирования BAI-2024 обеспечивают основной функционал для выполнения разнообразных операций, что позволяет решать задачи различной сложности.

## **Область видимости**

В языке BAI-2024 каждая переменная имеет свою область видимости, которая определяется такими конструкциями, как функции, главная функция, циклы и условные операторы. Функции, главная функция и глобальные переменные находятся в глобальной области видимости. Области видимости идентификаторов указаны в таблице идентификаторов.

## **Семантические проверки**

Семантический анализ предназначен для проверки смысловой корректности конструкций языка программирования. Перечень семантических проверок, предусмотренных языком, можно найти в таблице 1.8.

Таблица 1.8 – Семантические проверки

|  |  |
| --- | --- |
| Номер | Правило |
| 1 | Идентификаторы нельзя объявлять повторно в одной и той же области видимости. |
| 2 | Главная функция должна возвращать значение типа int. |
| 3 | Тип возвращаемого значения функции при её объявлении должен совпадать с типом, указанным в её определении. |
| 4 | Тип данных аргументов, передаваемых в функцию, должен соответствовать типу параметров, указанному при её объявлении. |
| 5 | Количество передаваемых в функцию параметров должно соответствовать ожидаемому числу. |
| 6 | Тип данных результата выражения должен соответствовать типу данных идентификатора, которому он присваивается. |
| 7 | Операнды выражения должны иметь одинаковые типы данных. |
| 8 | Функция main должна возвращать 0 |

Если семантическая проверка не проходит, то в лог журнал записывается соответствующая ошибка.

## **Распределение оперативной памяти на этапе выполнения**

Все переменные и литералы размещаются в сегменте данных.

## **Стандартная библиотека и её состав**

Стандартная библиотека представляет собой набор готовых функций, предоставляемых языком программирования для упрощения разработки программ. В BAI-2024 стандартная библиотека написана на C++ и подключена по умолчанию. Чтобы использовать функцию из стандартной библиотеки, её нужно вызвать по идентификатору и, при необходимости, передать аргументы в круглых скобках. Функции стандартной библиотеки с описанием представлены в таблице 1.9.

Таблица 1.9 – Состав стандартной библиотеки

|  |  |  |
| --- | --- | --- |
| Функция | Возвращаемое значение | Описание |
| string IToString(int) | string | Возвращает строку, содержащую переданный символ |
| string UIToString(uint) | string | Делает то же самое,что и IToString, но в качестве параметра принимает значение тип данных uint |
| string Concat(string, string) | string | Возвращает конкатенацию переданных двух строк |

Продолжение таблицы 1.9

|  |  |  |
| --- | --- | --- |
| Функция | Возвращаемое значение | Описание |
| int Print(string) | int | Выводит в консоль переданную строку. В случае успеха возвращает значение 0 |
| int Println(string) | int | Выводит в консоль переданную строку и переносит на следующую строку. В случае успеха возвращает значение 0 |
| string FToString(float) | string | Делает то же самое,что и IToString, но в качестве параметра принимает значение тип данных float |
| string BoolToString(bool) | string | Делает то же самое,что и IToString, но в качестве параметра принимает значение тип данных bool |

Функции стандартной библиотеке возвращают значение. Следовательно при вызове функции возвращаемое значение следует присвоить переменной

## **Ввод и вывод данных**

В языке BAI-2024 не реализованы средства ввода данных. Для вывода данных в стандартный поток вывода предусмотрена функции стандартной библиотеки Print, Println.

## **Точка входа**

В языке BAI-2024 каждая программа должна содержать главную функцию main, с которой начинается последовательное выполнение программы.

## **Препроцессор**

Препроцессор в языке программирования BAI-2024 не предусмотрен.

## **Соглашения о вызовах**

В языке функции вызываются по соглашению stdcall (Standard Call). Это соглашение определяет порядок передачи аргументов и очистки стека после вызова функции. В stdcall аргументы передаются справа налево, а стек очищается вызываемой функцией.

## **Объектный код**

BAI-2024 транслируется в язык ассемблера.

## **Классификация сообщений транслятора**

В случае возникновения ошибки в коде программы на языке BAI-2024 и выявления её транслятором в текущий файл протокола выводится сообщение. Классификация сообщений приведена в таблице 1.10.

## **Контрольный пример**

Код контрольного примера представлен в Приложении А.

# **2 Структура транслятора**

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

Транслятор преобразует программу, написанную на языке BAI-2024 в программу на языке ассемблера [1][2]. Для указания выходных файлов используются входные параметры транслятора, которые описаны в пункте 2.2. Компонентами транслятора являются лексический, синтаксический и семантический анализаторы, а также генератор кода на язык ассемблера. Принцип их взаимодействия представлен на рисунке 2.1.



Рисунок 2.1 – Структура транслятора

Лексический анализ – первая фаза трансляции. Назначением лексического анализатора является нахождение ошибок лексики языка BAI-2024 и формирование таблицы лексем и таблицы идентификаторов.

Семантический анализ в свою очередь является проверкой исходной программы BAI-2024 на семантическую согласованность с определением языка, т.е. проверяет правильность текста исходной программы с точки зрения семантики.

Синтаксический анализ – это основная часть транслятора, предназначенная для распознавания синтаксических конструкций и формирования промежуточного кода BAI-2024. Для этого используются таблица лексем и идентификаторов. Синтаксический анализатор распознаёт синтаксические конструкции, выявляет синтаксические ошибки при их наличии и формирует дерево разбора

Генератор кода – этап транслятора, выполняющий генерацию ассемблерного кода на основе полученных данных на предыдущих этапах трансляции. Генератор кода принимает на вход таблицы идентификаторов и лексем и транслирует код на языке BAI-2024, прошедший успешно все предыдущие этапы, в код на языке Ассемблера.

## **2.2 Перечень входных параметров транслятора**

Входные параметры представлены в таблице 2.1.

Таблица 2.1 – Входные параметры транслятора языка BAI-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание | Значение по умолчанию |
| -in:<имя\_файла> | Входной файл с любым расширением, в котором содержится исходный код на языке BAI-2024. Данный параметр должен быть указан обязательно. В случае если он не будет задан, то выполнение этапа трансляции не начнётся. | Не предусмотрено |
| -log:<имя\_файла> | Файл содержит в себе краткую информацию об исходном коде на языке BAI-2024. В этот файл могут быть выведены таблицы идентификаторов, лексем, а также дерево разбора. | <имя\_файла>.log |
| -out:<имя\_файла> | В этот файл будет записан результат трансляции кода на язык assembler | <имя\_файла>.asm |

Таблицы лексем и дерево разбора синтаксического анализатора выводятся в лог журнал.

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

Таблица с перечнем протоколов, формируемых транслятором языка BAI-2024 и их назначением представлена в таблице 2.2.

Таблица 2.2 – Протоколы, формируемые транслятором языка BAI-2024

|  |  |
| --- | --- |
| Формируемый протокол | Описание протокола |
| Файл журнала, “\*.log ” | Файл содержит в себе краткую информацию об исходном коде на языке BAI-2024. В этот файл выводится протокол работы анализаторов, а так же различные ошибки |
| “\*.asm” | Содержит сгенерированный код на языке Ассемблера. |

В log файл выводятся все ошибки, за исключением тех, что связаны с открытием файла log или считывания параметров.

# **3 Разработка лексического анализатора**

## **3.1 Структура лексического анализатора**

Лексический анализатор – часть транслятора, выполняющая лексический анализ. Лексический анализатор принимает обработанный и разбитый на отдельные компоненты исходный код на языке BAI-2024. Структура лексического анализатора представлена на рисунке 3.1.



Рисунок 3.1 – Структура лексического анализатора

Результатом работы лексического анализатора являются заполненные таблица лексем и таблица идентификаторов.

## **3.2 Контроль входных символов**

Исходный код на языке программирования BAI-2024 прежде чем транслироваться проверяется на допустимость символов. То есть изначально из входного файла считывается по одному символу и проверяется является ли он разрешённым.

Таблица для контроля входных символов представлена в приложении Б.

Принцип работы таблицы заключается в соответствии значения каждому элементу в шестнадцатеричной системе счисления значению в таблице ASCII.

Описание значения символов: T – разрешённый символ, F – запрещённый символ, S – пробельный символ, С – символ одинарной кавычки, L – символ-разделитель, D – символ двойной кавычки, O – символ начала комментария, N – символ новой строки.

## **3.3 Удаление избыточных символов**

Удаление избыточных символов не предусмотрено, так как после проверки на допустимость символов исходный код на языке программирования BAI-2024 разбивается на токены, которые записываются в очередь.

## **3.4 Перечень ключевых слов, сепараторов, символов операций и соответствующих им лексем, регулярных выражений и конечных автоматов**

Лексемы – это символы, соответствующие ключевым словам, символам операций и сепараторам, необходимые для упрощения дальнейшей обработки исходного кода программы. Данное соответствие описано в таблице 3.1.

Таблица 3.1 – Соответствие ключевых слов, символов операций и сепараторов с лексемами

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
| Тип данных | Целочисленный тип данных(размер 4 байта) | int | t |
| Целочисленный тип данных(размер 1 байт) | byte | t |
| Число двойной точности с плавающей точкой | double | t |
| Логический тип данных | bool | t |
| Строковый тип данных | string | t |
| Символьный тип данных | char | t |
| Лексема | Объявление функций | fn | f |
| Возврат значения из функции | return | r |
| Инструкция цикла | while | w |
| Инструкция условного оператора if | if | q |
| Инструкция условного оператора else | else | s |
| Блок функции, цикла, тела условного оператора | { | { |
| } | } |
| Круглые скобки | ( | ( |
| ( | ( |
| Оператор присваивания | = | = |
| Сепараторы | ; | ; |
| , | , |
| Условные операторы | < | u |
| > | u |
| <= | u |

Продолжение таблицы 3.1

|  |  |  |  |
| --- | --- | --- | --- |
| Тип цепочки | Примечание | Цепочка | Лексема |
|  | Условные операторы | >= | u |
| : | u |
| :! | u |
| Оператор | Знаки арифметических операций | + | o |
| - | o |
| \* | o |
| / | o |
| \ | o |
| : | o |
| % | o |
| Идентификатор |  | [a-z|A-Z]+  [a-z|A-Z|0-9]\* | i |
| Литерал | Целочисленный литерал | [1-9]+[0-9]+ | l |
| Числовой двойной точности с плавающей точки литерал | [-+]?\d\*\.?\d+([-+]?\d+)? | l |
| Символьный литерал | '[^'|/’/]'  \t, \n, # - заключить в / | l |
| Логический литерал | true|false | l |
| Строковый литерал | “[a-z|A-Z|0-9|\w!#%()\*+=/'<>,;:{}| \n\t$&.:?@[\]^\_~]+” “, \t, \n, # - заключить в / | l |
| Точка входа |  | main | m |

Каждому выражению соответствует детерминированный конечный автомат, то есть автомат с конечным состоянием, по которому происходит разбор данного выражения. На каждый автомат в массиве подаётся фраза 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. Пример реализации таблицы лексем представлен в приложении В.

Также в приложении В находятся конечные автоматы, соответствующие лексемам языка BAI-2024.

## **3.5 Основные структуры данных**

В таблице лексем содержится лексема, её номер, полученный при разборе, номер строки в исходном коде, номер столбца в исходном коде, индекс таблицы идентификаторов (если нет соответствующего идентификатора, то индекс равен -1), а также специальное поле, в котором хранится значение лексемы. В таблице идентификаторов содержится имя идентификатора, номер в таблице лексем, тип данных, тип идентификатора, его значение, а также область видимости.

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

Строка и индекс позиции,в котором было обнаружена лексическая ошибка, выводится в файл log. А также выводится сообщение,заранее определённая. Все возможные сообщения об ошибках при лексическом анализе предоставлены в листинге 3.1

|  |
| --- |
| errorSymbol = Utils::Pop(stack);  std::string mes = std::format("Source code, line {}, simbol {}", std::get<0>(errorSymbol), std::get<1>(errorSymbol));  ERROR\_LOG(mes, std::format("Лишний символ {}", std::get<2>(errorSymbol)));  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter), "Нарушен уровень вложенности");  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter), std::format("Нераспознаная лексема {}", token));  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", entry.line, entry.pos), "Длина строки не должна превышать 255 символов");  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter), std::format("Необъявлееный идентификатор {}", \*token));  ERROR\_LOG("Sourse code","Отсутствует функция main");  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter),std::format("Выход за пределы допустимого диапазона типа {}", token));  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter), std::format("Пустая строка не допускается {}", token));  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter),std::format("Переполнение типа byte {}", token));  ERROR\_LOG(std::format("Sourse code: строка {}, лексема {}.", line, counter), "Множественная инициализация"); |

## Продолжение листинга 3.1

|  |
| --- |
| ERROR\_LOG(std::format("Sourse code: строка {}, лексема {}.", line, counter),  std::format("{} {}", "Неверно указан тип", \*currentToken));  ERROR\_LOG(std::format("Sourse code: строка {}, лексема {}.", line, counter),  std::format("{} {}", “Неверно задан идентификатор”, \*currentToken));  ERROR\_LOG(std::format("Sourse code: строка {}, лексема {}.", line, counter),  "Недопустимая вложенная функция");  ERROR\_LOG(std::format("Sourse code: строка {}, лексема {}.", line, counter),  std::format("Неверная форма записи параметров {}", \*currentToken));  ERROR\_LOG(std::format("Sourse code: строка {}, индекс лексемы {}", line, counter), std::format("Недопустимое именование идентификатора {}", \*currentToken)); |

## **3.7 Принцип обработки ошибок**

Все ошибки являются критическими и приводят к прекращению работы транслятора и выводу диагностического сообщения в log-файл.

## **3.8 Параметры лексического анализатора**

Входные параметры используются для вывода результата работы лексического анализатора. Они передаются аргументами через командную строку и рассмотрены в таблице 2.1

## **3.9 Алгоритм лексического анализа**

Алгоритм работы лексического анализа заключается в последовательном распознавании и разборе цепочек исходного кода и заполнение таблиц идентификаторов и лексем. Лексический анализатор производит распознаёт и разбирает цепочки исходного текста программы. Это основывается на работе конечных автоматов, которую можно представить в виде графов. В случае, если подходящий автомат не был обнаружен, запоминается номер строки, в которой находился этот токен и выводится сообщение об ошибке. Если токен разобран, то дальнейшие действия, которые будут с ним производиться, будут зависеть от того, чем он является. Регулярные выражения — аналитический или формульный способ задания регулярных языков. Они состоят из констант и операторов, которые определяют множества строк и множество операций над ними. Любое регулярное выражение можно представить в виде графа.

В случае, если токен является идентификатором, перед его именем записывается название функции, в которой он объявлен и после этого он заносится в таблицу идентификаторов.

В случае, если токен является идентификатором функции, название функции в которой он объявлен не записывается.

В случае, если токен является литералом, то он заносится в таблицу идентификаторов в виде abi, где a – имя функции, где объявлен литерал, b – “$LEX”, c –количество определённых литералов+1.

Когда встречаем токен, являющийся ключевым словом, которое отвечает за тип данных или вид идентификатора, заносим лексему, соответствующую ему, в таблицу лексем и запоминаем тип данных или вид идентификатора, которому он соответствует.

В последствии, когда встречаем идентификатор, заносим его в таблицу идентификаторов с соответствующим ему типом данных и видом идентификатора, и именем вида “ab”, где a – имя функции, где объявлен идентификатор, b – имя идентификатора.

## **3.10 Контрольный пример**

Результат работы лексического анализатора – таблицы лексем и идентификаторов – представлен в приложении В.

# **4 Разработка синтаксического анализатора**

## **4.1 Структура синтаксического анализатора**

Синтаксический анализ – это фаза трансляции, выполняемая после лексического анализа и предназначенная для распознавания синтаксических конструкций BAI-2024. Входом для синтаксического анализа является таблица лексем и таблица идентификаторов, полученные после фазы лексического анализа. Выходом – дерево разбора. Структура синтаксического анализатора представлена на рисунке 4.1.



Рисунок 4.1 – Структура синтаксического анализатора BAI-2024

Входной информацией для синтаксического анализа является таблица лексем и таблица идентификаторов. Выходной информацией – дерево разбора.

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

В синтаксическом анализаторе транслятора языка BAI-2024 используется контекстно-свободная грамматика [2], 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.

Грамматика языка BAI-2024 представлена в приложении Г.

TS – терминальные символы, которыми являются сепараторы, знаки арифметических операций и некоторые строчные буквы.

NS – нетерминальные символы, представленные несколькими заглавными буквами латинского алфавита.

Таблица 4.1 – Перечень правил и описание нетерминальных символов BAI-2024

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| S | ftmr()Y  tiQ;G  tiQ;F  tiQ;M  fti(P)W | Порождает правила, описывающее общую структуру программы |
| G | tiQ;G  tiQ;F  tiQ;M  iQ;G  iQ;F  iQ;M | Порождает правила, описывающие конструкции глобальных переменных |
| Q | =l  =X | Порождает правила, описывающие литералы глобальных переменных |
| X | lD  (X)  (X)D | Порождает правила, описывающие правила для начало выражений глобальных переменных |
| D | olD  ol | Порождает правила, описывающие правила для конца выражений глобальных переменных |
| M | ftm()Y | Порождает правила, описывающие правила структуры функции main |
| F | fti(P)W | Порождает правила, описывающие правила структуры функции |
| Y | {Brl;}  {rl;} | Порождает правила, описывающие правила тело функции main |
| W | {BrR;Z  {BrV;Z  {rR;Z  {rV;Z | Порождает правила, описывающие начало правила тело функции |
| Z | }M  }F | Порождает правила, описывающие конец правила тело функции |
| J | ;B  ; | Порождает правила, описывающие правила конца тело функции |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| K | {B  {BrV;  {BrR;  {rV;  {rR; | Порождает правила, описывающие тело условного оператора и цикла |
| E | }sK}  }sK}B  }B  }E  } | Порождает правила, описывающие конец тела условного оператора |
| B | tiLJ  ti=i(A)J  iLJ  q(U)KE  w(U)K}  w(U)K}B | Порождает правила, описывающие тело функций, условных операторов, циклов |
| L | =R  =V | Порождает правила, описывающие операцию присвоения |
| P | ti,P  ti | Порождает правила, описывающие операцию объявления параметров |
| A | I  L  iT  lT  i(A)T  i(A)  (V)T  (V)  i,A  l,A  iT,A  lT,A  i(A),(A)  i(A)T,A  (V)T,A  (V),A | Порождает правила, описывающие операцию передачи аргументов в функцию |
| V | iT  lT  i(A)T  (V)  (V)T | Порождает правила, описывающие начало арифметического выражения |

Продолжение таблицы 4.1

|  |  |  |
| --- | --- | --- |
| Нетерминал | Цепочки правил | Описание |
| T | oRT  oR | Порождает правила, описывающие правила продолжения арифметического выражения |
| R | i  l  i(A)  (V) | Порождает правила, описывающие возвращаемое значение |
| U | iI  lI  (V)I  i(A)I  (U) | Порождает правила, описывающие начало условного выражения |
| I | uR  u(V) | Порождает правила, описывающие конец условного выражения |

Протокол и ошибки работы синтаксического анализатора выводятся в лог журнал.

## **4.3 Построение конечного магазинного автомата**

Конечный автомат с магазинной памятью представляет собой семерку [1], описание которой представлено в таблице 4.2. Структура данного автомата показана в приложении Г.

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 |

Продолжение таблицы 4.2

|  |  |  |
| --- | --- | --- |
| Компонента | Определение | Описание |
|  | 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 |

Для вывода результата работы синтаксического анализатора нужно использовать флаг m.

## **4.4 Основные структуры данных**

Основные структуры данных синтаксического анализатора включают в себя структуру магазинного конечного автомата и структуру грамматики Грейбах, описывающей правила языка BAI-2024. Данные структуры представлены в приложении Г.

## **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 и формируется дерево разбора. Иначе генерируется исключение.

## **4.6 Структура и перечень сообщений синтаксического анализатора**

Перечень сообщений синтаксического анализатора представлен в таблице 4.3.

|  |  |
| --- | --- |
| Номер | Описание |
| 1 | Неверное структура кода |
| 2 | Неверное объявление функций |
| 3 | Неправильное объявление параметров |
| 4 | Неправильно структурированы условный оператор или цикл |

В сообщении об ошибке сообщается номер строки и номер лексемы.

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

Для вывода результата работы синтаксического анализатора используются входные параметры, описанные в пункте 2.2Перечень входных параметров транслятора в таблице 2.1.

## **4.8 Принцип обработки ошибок**

Обработка ошибок происходит следующим образом:

1. Синтаксический анализатор перебирает все правила и цепочки правила грамматики для нахождения подходящего соответствия с конструкцией, представленной в таблице лексем.
2. Если невозможно подобрать подходящую цепочку, то генерируется соответствующая ошибка.
3. В случае ошибки выводится соответствующее сообщение в журнал лога и компилятор прекращает работу.

## **4.9 Контрольный пример**

Пример разбора синтаксическим анализатором исходного кода на языке BAI-2024 представлен в приложении Д. Дерево разбора исходного кода также представлено в приложении Д.

# **5 Разработка семантического анализатора**

## **5.1 Структура семантического анализатора**

Семантический анализ происходит при выполнении фазы лексического анализа и реализуется в виде отдельных проверок текущих ситуаций в конкретных случаях: установки флага или нахождении в особом месте программы (оператор выхода из функции, оператор ветвления, вызов функции стандартной библиотеки). Структура семантического анализатора представлена на рисунке 5.1.



Рисунок 5.1 – Структура семантического анализатора

Функции семантического анализатора частично реализованы в лексическом анализаторе.

## **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**

Все ошибка семантического анализатора представлены в таблице 1.7. К сообщению ошибки также добавляется номер строки и номер лексемы и записывается в лог файл.

## **5.4 Принцип обработки ошибок**

При обнаружении хотя бы одной ошибки транслятор завершит свою работу c

Записью информации об ошибке в лог файл.

## **5.5 Контрольный пример**

Результат работы контрольного примера расположен в приложении А, где показан результат лексического анализатора, т.к. представленные таблицы лексем и идентификаторов проходят лексическую и семантическую проверки одновременно.

**6 Преобразование выражений**

## **6.1 Выражения, допускаемые языком**

В языке PEN-2024 допускаются выражения, применимые к целочисленным типам данным и числам двойной точности с плавающей точкой. В выражениях поддерживаются арифметические операции, такие как +, -, \*, /, % , () << и >>, и вызовы функций как операнды арифметических выражений. . Также в языке допускаемы условные выражения,которые применимы ко всем числовым, строковым и символьным типам данных.Приоритет операций представлен в таблице 6.1.

Таблица 6.1 – Приоритет операций в языке TDS-2022

|  |  |
| --- | --- |
| Операция | Значение приоритета |
| ( | 0 |
| ) | 0 |
| + | 2 |
| - | 2 |
| \* | 3 |
| : | 3 |
| % | 3 |
| << | 4 |
| >> | 4 |

Некоторые из операций таблицы 6.1 используются для типов, отличных от целочисленных.

## **6.2 Польская запись**

Выражения в языке BAI-2024 преобразовываются к обратной польской записи.

Польская запись – это альтернативный способ записи арифметических выражений, преимущество которого состоит в отсутствии скобок.

Обратная польская запись – это форма записи математических и логических выражений, в которой операнды расположены перед знаками операций. 

Алгоритм построения:

– читаем очередной символ;

– если он является идентификатором или литералом, то добавляем его к выходной строке;

– если символ является символом функции, то помещаем его в стек;

– если символ является открывающей скобкой, то она помещается в стек;

– исходная строка просматривается слева направо;

– если символ является закрывающей скобкой, то выталкиваем из стека в выходную строку все символы пока не встретим открывающую скобку. При этом обе скобки удаляются и не попадают в выходную строку;

– как только входная лента закончится все символы из стека выталкиваются в строку;

– в случае если встречаются операции, то выталкиваем из стека в выходную строку все операции, которые имеют выше приоритетность чем последняя операция;

Таблица 6.2 – Пример преобразования выражения в обратную польскую запись

|  |  |  |
| --- | --- | --- |
| Исходная строка | Результирующая строка | Стек |
| x+y\*5/(z-2) |  |  |
| +y\*5/(z-2) | x |  |
| y\*5/(z-2) | x | + |
| \*5/(z-2) | xy | + |
| 5/(z-2) | xy | +\* |
| /(z-2) | xy5 | +\* |
| (z-2) | xy5\* | +/ |
| z-2) | xy5\* | +/( |
| -2) | xy5\*z | +/( |
| 2) | xy5\*z | +/(- |
| ) | xy5\*z2 | +/(- |
|  | xy5\*z2- | +/ |
|  | xy5\*z2-/ | + |
|  | xy5\*z2-/+ |  |

Как результат успешного разбора, мы получаем пустой стек и заполненную результирующую строку.

## **6.3 Программная реализация обработки выражений**

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

## **6.4 Контрольный пример**

Пример преобразования выражения к польской записи представлен в таблице 6.2. Преобразование выражений в формат польской записи необходимо для построения более простых алгоритмов их вычисления.

# **7 Генерация кода**

## **7.1 Структура генератора кода**

Генерация объектного кода — это перевод компилятором внутреннего представления исходной программы BAI-2024 в цепочку символов выходного языка. На вход генератора подаются таблицы лексем и идентификаторов, на основе которых генерируется файл с ассемблерным кодом.



Рисунок 7.1 – Структура генератора кода

## **7.2 Представление типов данных в оперативной памяти**

Элементы таблицы идентификаторов расположены в разных сегментах языка ассемблера – .data и .const [4][5]. Соответствия между типами данных идентификаторов на языке BAI-2024 и на языке ассемблера приведены в таблице 7.1.

Таблица 7.1 – Соответствия типов идентификаторов языка BAI-2024 и языка Ассемблера

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BAI-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| char | DWORD | Хранит символьный тип данных. |
| string | BYTE | Хранит указатель на начало строки. |
| int | SDWORD | Хранит целочисленный тип данных. |
| uint | SDWORD | Хранит целочисленный тип данных. |
| float | REAL4 | Хранит число двойной точности с плавающей точкой. |
| bool | DWORD | Хранит или true или false |

Продолжение таблицы 7.1

|  |  |  |
| --- | --- | --- |
| Тип идентификатора на языке BAI-2024 | Тип идентификатора на языке ассемблера | Пояснение |
| Лексема | REAL8  DWORD  SDWORD | Литералы: символьные,  числовые, логические и строковые |

Идентификаторы и литералы языка PEN-2024 размещены в сегменте данных (.data).

## **7.3 Статическая библиотека**

Статическая библиотека реализована на языке программирования C++. Её код находится в проекте standartLib, в свойствах которого был выбран пункт «статическая библиотека .lib». Подключение библиотеки в языке ассемблера происходит с помощью директивы includelib на этапе генерации кода.

## **7.4 Алгоритм работы генератора кода**

Алгоритм генерации кода выглядит следующим образом:

1. Генерирует заголовочную информацию (Лист. 7.2): модель памяти, подключение библиотек, прототипы внешних функций, размер стека.

|  |
| --- |
| static const std::string BEGIN\_OF\_FILE = ".686p\n.xmm\n.MODEL FLAT, STDCALL\nincludelib kernel32.lib\nincludelib msvcrt.lib\nincludelib ucrt.lib\nincludelib vcruntime.lib\nincludelib msvcprt.lib\nincludelib standartLib.lib\nPrint PROTO arg:ptr byte\nPrintln PROTO arg:ptr byte\nIToString PROTO arg:sdword\nUIToString PROTO arg:dword\nFToString PROTO arg:real8\nBoolToString PROTO arg:sdword\nConcat PROTO arg1:ptr byte, arg2:ptr byte\nAbsb PROTO arg:sword\nCharToString PROTO arg:sdword\nExitProcess PROTO:DWORD\n.STACK 4096\n"; |

Листинг 7.2 –Заголовочная информация

1. Проходим таблицу идентификаторов и объявляем переменные в поле .data. (Лист. 7.4).

|  |
| --- |
| .DATA  real\_buff REAL8 0.0  null\_err BYTE "Ошибка, деление на ноль", 0  Literal\_New\_0\_\_none SDWORD 66  Literal\_1\_\_none REAL8 1.000000  Literal\_2\_\_none REAL8 0.000000  Literal\_3\_\_none REAL8 0.000000  Literal\_4\_\_none REAL8 0.000000  Literal\_New\_1\_\_none REAL8 0.000000  Literal\_7\_\_none BYTE "@|", 0  Literal\_8\_\_none DWORD '@'  Literal\_New\_2\_\_none SDWORD 0 |

Листинг 7.4 – Пример заполнения поля .data

1. Генерируем сегмент данных .code (Лист. 7.5). Сперва проходим по таблице идентификаторов и ищем функции. Объявляем их и генерируем код, содержащийся в функциях. Так же перед именем функции дописываем знак «$», чтобы исключить совпадение имени функции с ключевым словом ассемблера. При генерации кода, при встрече оператора присваивания, описываем вычисление выражения. Описание алгоритма преобразования выражений представлено в пункте 7.3.

|  |
| --- |
| .code  $FindFactor PROC uses ebx ecx edi esi , FindFactora: DWORD  ; Stringing #3 :ivl  push FindFactor$LEX1  pop FindFactoranswer  While17Start:  mov eax, FindFactora  mov ebx, FindFactor$LEX1  cmp eax, ebx  jl While17End  ; Stringing #6 :iviiv  push FindFactoranswer  push FindFactora  pop ebx  pop eax  mul ebx  push eax  pop FindFactoranswer  ; Stringing #7 :ivilv  push FindFactora  push FindFactor$LEX1  pop ebx  pop eax  sub eax, ebx  push eax  pop FindFactora  jmp While17Start  While17End:  mov eax, FindFactoranswer  ret  $FindFactor ENDP |

Листинг 7.5 – Пример заполнения поля .code

После генерации всех пользовательских функций, генерируется функция начала программы main в функции main по такому же принципу.

## **7.5 Контрольный пример**

Генерируемый код записывается в файл заданный параметром “-out”. Сгенерированный код можно посмотреть в приложении Ж.

# **8 Тестирование транслятора**

## **8.1 Тестирование фазы проверки на допустимость символов**

В языке BAI-2024 не разрешается использовать запрещённые входным алфавитом символы где-либо кроме строковых или символьных переменных. Результат использования запрещённого символа показан в таблице 8.1.

Таблица 8.1 – Тестирование фазы проверки на допустимость символов

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| main{в} | Ошибка: Недопустимый символ в исходном файле (-in), строка 1, столбец 4 |

Запрещённые символы можно посмотреть в приложении Б.

## **8.2 Тестирование лексического анализатора**

На этапе лексического анализа 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 |
| fn main(){  return 0;  } | Ошибка: Не указан тип, строка 2, столбец 11 |
| fn int main(){  s = 5;  } | Ошибка: Используется необъявленный идентификатор, строка 2, столбец 1 |
| fn int a(){  ui = 5;  } | Ошибка: Идентификатор не имеет типа, строка 2, столбец 6 |
| fn string func a(){  return 0;  } | Ошибка: Отсутствует точка входа |
| fn int main(){  return 0;  }  main(){  return 0;  } | Ошибка: Множественная инициализация |

Ошибка лексического анализатора приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.3 Тестирование синтаксического анализатора**

На этапе синтаксического анализа 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 |
| fn int main{  return 0;  }  fn int fun(int i) {  return 0; } | Ошибка: Неверная структура кода,строка 6, столбец 1 |

Ошибка синтаксического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

## **8.4 Тестирование семантического анализатора**

Итоги тестирования семантического анализатора приведены в таблице 8.4.

Таблица 8.4 – Тестирование семантического анализатора

|  |  |
| --- | --- |
| Исходный код | Диагностическое сообщение |
| fn int main(){ int result =5;  int y = result+45/6.4  } | Ошибка:Ожидался тип int, cтрока 3, лексема 8 |

Ошибка семантического анализатора также приводит к прекращению выполнения программы и записи соответствующей ошибки в лог журнал.

# **Заключение**

По окончании выполнения всех пунктов, изложенных ранее, получили рабочий транслятор языка программирования PEN-2024 на язык ассемблера.

Язык BAI-2024 поддерживает 6 типов данных:целочисленный(размер 4 байта, int), целочисленный (uint), тип данных числа и с плавающей точкой(float), символьный тип(char),строковой тип(string), логический(bool), тип данных let.

Для численных типов реализована обработка 5 арифметических действий, скобок, обозначающих приоритет операций.Также побитывые сдвиги.Для всех типов реализована обработка 6 условных действий.

На этапе семантического анализа производится проверка соответствия исходного кода спецификации по 8 правилам.

Реализованы 6 функций стандартной библиотеки.

# **Список использованных источников**

1. Карпов Ю. Теория и технология программирования. Основы построения трансляторов, 2005. – 272с.
2. Введение в теорию трансляторов [Электронный ресурс]. – Режим доступа: http://bourabai.ru/tpoi/compilers.htm. – Дата доступа: 15.11.2022.
3. Википедия: Обратная польская запись [Электронный ресурс]. – Режим доступа: https://en.wikipedia.org/wiki/Reverse\_Polish\_notation. – Дата доступа: 20.11.2022.
4. MASM для x86 [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/en-us/cpp/assembler/masm/masm-for-x64-ml64-exe?view=msvc-160>. – Дата доступа: 29.11.2022.
5. Ирвин К. Р. Язык ассемблера для процессоров Intel / К. Р. Ирвин. – M.: Вильямс, 2005. – 912с.

## ПРИЛОЖЕНИЕА

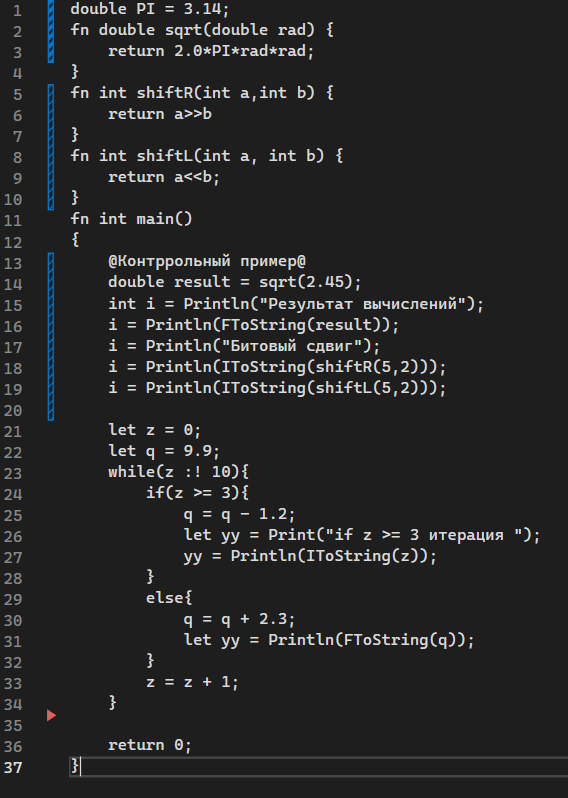


Рисунок 1 – Контрольный пример