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 LPA-2024»

Выполнил студент Ленкевич Павел Андреевич

(Ф.И.О.)

Руководитель проекта Волчек Дарья Ивановна

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

Консультанты Волчек Дарья Ивановна

Нормоконтролер Волчек Дарья Ивановна

Курсовой проект защищен с оценкой

Минск 2024

**Содержание**

|  |  |
| --- | --- |
| Введение | 3 |
| 1. Спецификация языка программирования | 4 |
| 1.1. Характеристика языка программирования | 4 |
| 1.2. Определение алфавита языка программирования | 4 |
| 1.3. Применяемые сепараторы | 5 |
| 1.4. Применяемые кодировки | 5 |
| 1.5. Типы данных | 5 |
| 1.6. Преобразование типов данных | 6 |
| 1.7. Идентификаторы | 6 |
| 1.8. Литералы | 6 |
| 1.9. Объявление данных | 7 |
| 1.10. Инициализация данных | 8 |
| 1.11. Инструкции языка | 8 |
| 1.12. Операции языка | 9 |
| 1.13. Выражения и их вычисление | 10 |
| 1.14. Конструкции языка | 10 |
| 1.15. Область видимости идентификаторов | 11 |
| 1.16. Семантические проверки | 11 |
| 1.17. Распределение оперативной памяти на этапе выполнения | 12 |
| 1.18. Стандартная библиотека и её состав | 12 |
| 1.19. Ввод и вывод данных | 12 |
| 1.20. Точка входа | 13 |
| 1.21. Препроцессор | 13 |
| 1.22. Соглашения о вызовах | 13 |
| 1.23. Объектный код | 13 |
| 1.24. Классификация сообщений транслятора | 13 |
| 1.25. Контрольный пример | 14 |
| 2. Структура транслятора | 15 |
| Приложение А | 16 |

# **Введение**

Целью курсового проекта по дисциплине «Конструирование программного обеспечения» является написание спецификации и разработка компилятора для собственного языка программирования LPA-2024, ориентированного на выполнение простейших арифметических операций в консольном режиме. Компиляция кода языка LPA-2024 будет производиться в язык ассемблер.

Этапы разработки компилятора для языка LPA-2024:

– разработка спецификации языка программирования;

– разработка структуры транслятора;

– разработка лексического анализатора;

– разработка синтаксического анализатора;

– разработка семантического анализатора;

– обработка выражений;

– генерация кода на язык Assembler;

– тестирование транслятора.

Информация о каждом этапе разработки компилятора приведена в соответствующих разделах пояснительной записки.

В первом разделе приведена спецификация языка программирования.

Во втором разделе описана структура транслятора.

В третьем разделе описаны принцип работ и этапы разработки лексического анализатора, определены разрешенные символы и ключевые слова языка программирования.

В четвертом разделе описан принцип работы синтаксического анализатора.

В пятом разделе описаны принцип работы и основные функции семантического анализатора.

В шестом разделе описаны выражения, допускаемые языком, форма, принципы построения и вычисления выражений.

В седьмом разделе описан процесс генерации кода.

В восьмом разделе приведены примеры тестирования транслятора.

Язык программирования LPA-2024 предназначен для работы с консолью, выполнения простейших арифметических операций.

# **Спецификация языка программирования**

## **Характеристика языка программирования**

LPA-2024 – это компилируемый, строго типизированный, универсальный язык программирования, ориентированный на выполнение простейших арифметических операций в консольном режиме.

## **Определение алфавита языка программирования**

Алфавит языка LPA-2024 основан на кодировке Windows-1251, изображённой на рисунке 1.1.



Рисунок 1.1 – Кодировка Windows-1251

## **Применяемые сепараторы**

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены.

Сепараторы, применяемые в языке LPA-2024, приведены в таблице 1.1.

Таблица 1.1 Символы-сепараторы

|  |  |
| --- | --- |
| Разделители | Назначение |
| «пробел», «табуляция», «переход на новую строку» | Разделяют входные лексемы |
| +, -, \*, /, % | Арифметические операторы. Используются в арифметических операторах |
| = | Оператор присваивания. Используется для присваивания значения переменной |
| <, >, <=, >=, !=, == | Условные операторы. Используются в условии цикла |
| ( ) | Блок параметров функции, так же указывает на приоритет в арифметических операциях |
| , | Разделяет параметры функции |
| { } | Ограничивает программные конструкции |
| ; | Признак конца инструкции языка |

## **Применяемые кодировки**

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

## **1.5 Типы данных**

В языке LPA-2024 поддерживается 4 типа данных: целочисленный (2 байта), символьный, строковый и логический. Пользовательские типы данных не поддерживается. Описание типов данных представлено в таблице 1.2.

Таблица 1.2 – Типы данных языка программирования LPA-2024

|  |  |
| --- | --- |
| Тип данных | Описание типа данных |
| Целочисленный  (int) | В памяти занимает 2 байта.  Максимальное значение: 32767.  Минимальное значение: -32768.  Значение по умолчанию: 0.  Операции: сложение (+), вычитание (-), умножение (\*), деление нацело (/), остаток от деления (%), присваивание (=), а также условия для операторов цикла: больше (>), меньше (<), проверка на неравенство (!). |
| Символьный (char) | В памяти занимает 1 байт.  Значение по умолчанию: нулевое значение. |
| Строковый  (str) | В памяти занимает n + 1 байт, где n – количество символов + символ конца строки.  Максимальное количество символов в строке: 256.  Значение по умолчанию: строка нулевой длины «».  Операции: присваивание идентификатору значения другого строкового идентификатора, строкового литерала или значение строковой функции, использование библиотечных функций копирования строк и вычисления длины строки. |
| Логический (bool) | В памяти занимает 1 байт.  Принимает одно из двух значений: истинна (true) или ложь (false).  В зависимости от значения 1 бит числа установлен true: 1 или false: 0.  Логический тип необходим для работы условных операций в языке LPA-2024 |

## **Преобразование типов данных**

Преобразования типов данных в языке LPA-2024 не предусмотрено.

## **Идентификаторы**

Идентификаторы используются для именования переменных, функций. Идентификатор должен начинаться с строчной буквы латинского алфавита. После первого символа допускается использование строчных букв латинского алфавита и цифр. Идентификаторы не могут совпадать с ключевыми словами языка.

Пример корректных идентификаторов: yasha1, y002asha, yasha2.

Пример некорректных идентификаторов: 1yasha (начинается с цифры), Y2asha (содержит прописную букву), #yaSha2 (содержит недопустимый символ и прописную букву).

## **Литералы**

Литерал представляет собой запись в исходном коде программы, которая обозначает фиксированное значение. В языке программирования LPA-2024 поддерживается три типа литералов: целочисленные, строковые и символьные. Целочисленные литералы могут быть представлены в двух системах счисления: восьмеричной и десятеричной.

Описание литералов приведено в таблице 1.3.

Таблица 1.3 – Литералы языка LPA-2024

|  |  |
| --- | --- |
| Тип литерала | Характеристика |
| Целочисленный | Восьмеричный:  Последовательность восьмеричных цифр 0..7 с предшествующим знаком минус или без него, начинающаяся с символа «0» (признак восьмеричного целочисленного литерала).  Десятичный: Последовательность десятичных цифр 0..9 с предшествующим знаком минус или без него.  Допустимый диапазон значений:  От -32768 до 32767 в десятичной системе исчисления. |
| Строковый | Набор, состоящий из символов русского и латинского алфавитов, десятичных цифр и специальных символов, заключённый в двойные кавычки. Допустимый диапазон значений:  От 0 до 255 символов. |
| Символьный | Один символ из списка русского и латинского алфавитов, специальных символов, цифр, заключённый в одинарные кавычки.  Может содержать только 1 символ. |

Примеры корректных литералов: “str154str”, ‘g’, -056, 56, -56, 07.

Примеры некорректных литералов: str154str (не заключён в кавычки), o56 (недопустимый символ), 09 и -08 (восьмеричный литерал может принимать только цифры от 0 до 7), s (не заключён в одинарные кавычки).

## **Объявление данных**

В языке LPA-2024 для объявления переменных используются ключевые слова int, str, char и bool. Ключевое слово int обозначает целочисленные переменные, str – строковые, char – символьные и bool – логические. После ключевого слова указывается имя идентификатора. Инициализация переменной допустима прямо в объявлении.

int <имя идентификатора>;

int <имя идентификатора> = <литерал>;

str <имя идентификатора>;

str <имя идентификатора> = <литерал>;

char <имя идентификатора>;

char <имя идентификатора> = <литерал>;

bool <имя идентификатора>;

bool <имя идентификатора> = <литерал>;

Переменные могут быть объявлены в блоке main, в блоке функции или внутри цикла. Область видимости идентификаторов определяется границами блока кода, заключённого в фигурные скобки { }.

## **1.10 Инициализация данных**

В языке программирования LPA-2024 поддерживаются два способа инициализации переменных:

– Инициализация в месте объявления – задаётся непосредственно при объявлении переменной:

int <имя идентификатора> = <литерал>;

str <имя идентификатора> = <литерал>;

char <имя идентификатора> = <литерал>;

bool <имя идентификатора> = <литерал>;

– Инициализация после объявления – выполняется присваиванием значения уже объявленной переменной:

<имя идентификатора> = <литерал>;

При объявлении переменных без явного указания значения инициализация выполняется автоматически значениями по умолчанию:

* для типа int – 0
* для типа str – пустая строка «».
* для типа char – пустой символ «».

Примеры инициализации в месте объявления:

* int count = 11;
* str name = “LPA-2024”;
* char simvol = ‘a’;
* bool logika = true;

Примеры инициализации после объявления:

* int total;

total = 33;

* str username;

username = “Kapetto”;

* char idclass;

idclass = ‘a’;

* bool pravda;

pravda = true;

## **1.11 Инструкции языка**

Инструкции языка LPA-2024 приведены в таблице 1.4.

Таблица 1.4 – Инструкции языка LPA-2024

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Объявление переменной | <тип данных> <имя идентификатора>; |
| Объявление переменной с явной инициализацией | <тип данных> <имя идентификатора> = <значение>; |
| Присваивание | <имя идентификатора> = <значение>; |
| Объявление функции | <тип данных> <имя идентификатора> (<тип данных> <имя идентификатора>, …) {  /тело функции/  return <имя идентификатора/литерал>;  }; |
| Возврат из функции | Для функций, возвращающих значение:  return <имя идентификатора/литерал>; |
| Вызов функции | <имя идентификатора> (<имя идентификатора>, …); |
| Вывод данных | write <литерал/имя идентификатора>;  writeline <литерал/имя идентификатора>; |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция | Форма записи |
| Оператор цикла | cycle (<имя идентификатора переменной/литерал> <условный оператор> <имя идентификатора переменной/литерал>)  {  /тело цикла/ } |

## **1.12 Операции языка**

Язык LPA-2024 поддерживает арифметические и логические операции. Приоритет: умножение, деление, деление с остатком, сложение и вычитание. Логические операции имеют равный приоритет.

Описание операций языка LPA-2024 приведено в таблице 1.5

Таблица 1.5 – Операции языка LPA-2024

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические операции | \* – умножение  / – деление  % – остаток от деления  + – сложение  - – вычитание  = – присваивание |
| Логические операции | > – больше  < – меньше  >= – больше или равно  <= – меньше или равно  == – проверка на равенство  != – проверка на неравенство |

## **1.13 Выражения и их вычисления**

Выражение языка программирования LPA-2024 представляет собой совокупность переменных, литералов, вызовов функций, знаков операций, скобок, которая может быть вычислена в соответствии с синтаксисом языка.

Правила составления выражений:

* Выражение записывается в строку без переносов;
* В выражении могут присутствовать только операнды одинакового типа;
* В выражении могут использоваться функции. Как стандартные, так и пользовательские;
* В выражении не могут идти подряд два оператора;
* Допускается использование круглых скобок для смены приоритета операций.

В арифметических выражениях допускаются только операнды целочисленного типа.

Перед генерацией кода выражения приводятся к ПОЛИЗ для более удобного вычисления на языке ассемблера.

## **Конструкции языка**

Программа на языке LPA-2024 состоит из функций пользователя и главной функции main. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для улучшения читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка LPA-2024

|  |  |
| --- | --- |
| Конструкция | Описание |
| Главная функция | main  {  …  }; |
| Пользовательская функция | <тип данных возвращаемого значения> <имя идентификатора> (<тип данных параметра> <имя идентификатора параметра>, …)  {  …  return <имя идентификатора переменной/литерал>;  };  Максимальное количество параметров: 8. |
| Цикл | cycle (<имя идентификатора переменной/литерал> <условный оператор> <имя идентификатора переменной/литерал>)  {  …  } |

## **Область видимости идентификаторов**

Область видимости идентификаторов ограничивается блоками, в которых они объявлены. Переменные внутри функции недоступны вне её. Идентификаторы уникальны и локальны в пределах функции. Идентификатор нельзя использовать до его объявления.

## **Семантические проверки**

Семантическим анализатором языка LPA-2024 предусмотрены следующие проверки:

* Наличие блока main – точки входа в программу;
* Единственность точки входа;
* Использование идентификаторов до их объявления;
* Переопределение идентификаторов;

– Соответствие параметров, передаваемых в функцию, с параметрами в объявлении функции;

* Соответствие типа возвращаемого значения с типом функции;
* Соответствие типов в выражениях;
* Превышение размера целочисленных и строковых литералов;
* Превышение длины лексемы;

– Соответствие операторов типам данных, для работы с которыми они предназначены.

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

Память делится на сегмент констант для хранения литералов и сегмент данных для переменных и параметров. Локальность обеспечивается префиксами в именах идентификаторов. Префикс определяется именем функции, в которой находится переменная.

## **Стандартная библиотека и её состав**

В языке LPA-2024 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержа https://github.com/viyviyleta/KPO\_3term/tree/main/lab17 ние библиотеки и описание функций представлено в таблице 1.7.

Таблица 1.7 – Стандартная библиотека языка LPA-2024

|  |  |
| --- | --- |
| Функция | Описание |
| str strduplicate (str parm1, str parm2); | Строковая функция. Выполняет копирование строки parm2 в parm1. Возвращает parm1. |
| int strlength (str parm1); | Целочисленная функция. Вычисляет и возвращает длину строки parm1. |
| int strtransint (str parm1); | Целочисленная функция. Преобразует строку parm1 в число. Строка не должна состоять из буквенных или специальных символов. Необходима для работы функции strlength. Возвращает число. |

Стандартная библиотека написана на языке C++. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций.

## **Ввод и вывод данных**

Вывод данных осуществляется с помощью операторов write и writeline. Допускается использование операторов с литералами и идентификаторами.

Функции управляющие выводом данных, реализованы на языке C++ и вызываются из транслированного кода, конечному пользователю недоступны. На этапе генерации кода операторы вывода языка LPA-2024 заменяются на встроенные функции, находящиеся в стандартной библиотеке.

## **Точка входа**

В языке LPA-2024 точкой входа в программу является главная функция main.

## **Препроцессор**

Препроцессор в языке программирования LPA-2024 не предусмотрен.

## **Соглашение о вызовах**

В языке LPA-2024 вызов функций происходит по стандартному соглашению о вызовах stdcall. Данное соглашение имеет особенности:

* все параметры функции передаются через стек;
* память высвобождает вызываемый код;
* занесение в стек параметров идёт справа налево.

## **1.23 Объектный код**

Язык LPA-2024 транслируется в язык ассемблера, а затем в объектный код.

## **1.24 Классификация сообщений транслятора**

Описание и классификация сообщений компилятора об ошибках приведено в таблице 1.8.

Таблица 1.8 – Описание ошибок транслятора языка LPA-2024

|  |  |
| --- | --- |
| Интервал кодов | Описание |
| 0-9 | Системные ошибки. |
| 10-19 | Ошибки параметров. |
| 20-29 | Ошибки файлов. |
| 110-129 | Ошибки лексического анализатора. |
| 130-149 | Ошибки семантического анализатора. |
| 600-609 | Ошибки синтаксического анализатора. |
| 610-999 | Зарезервированные коды ошибок. |

## **1.25 Контрольный пример**

Контрольный пример демонстрирует главные особенности языка LPA-2024: его фундаментальные типы, основные структуры, функции, скрипты, использование функций статической библиотеки. Исходный код контрольного примера представлен в приложении А.

1. **Спецификация языка программирования**

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

Транслятор – программа, преобразующая исходный код на одном языке программирования в исходный код на другом языке.

Схема принципа работы транслятора изображена на рисунке 2.1.

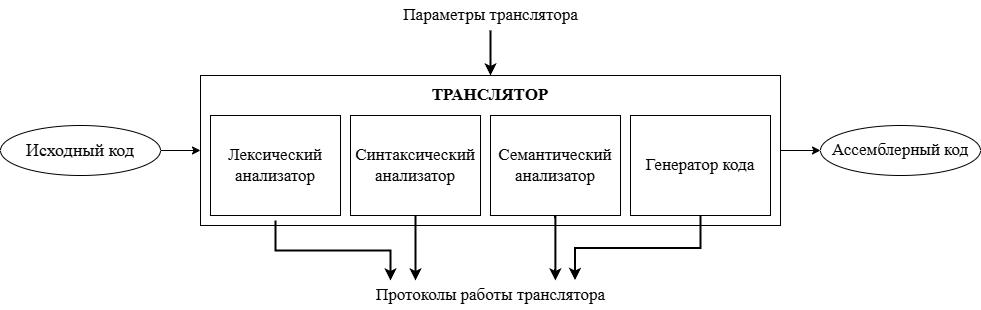


Рисунок 2.1 – принцип работы транслятора

В языке LPA-2024 исходный код транслируется в язык ассемблера. Транслятор разделён на 4 основные части: лексический анализатор, синтаксический анализатор, семантический анализатор и генератор кода. Каждый этап имеет входные и выходные данные, которые передаются следующему компоненту транслятора.

Первый этап работы компилятора – лексический анализатор. На вход подаётся исходный код программы. Лексический анализатор делит код на токены, которые затем идентифицируются и заменяются на лексемы. На выходе мы получаем две таблицы: таблицу лексем и таблицу идентификаторов.

Второй этап работы компилятора – синтаксический анализатор. На вход подаётся таблица лексем и таблица идентификаторов. На выходе мы получаем дерево разбора.

Третий этап работы компилятора – семантический анализатор. На этом этапе проверяется соблюдения в исходной программе семантических правил входного языка. На вход подаётся таблица идентификаторов, таблица лексем и дерево разбора.

Последний этап работы компилятора – генератор кода. На вход генератора подаются таблица лексем и таблица идентификаторов. На основе таблиц 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 – Стандартная библиотека языка LPA-2024

|  |  |  |
| --- | --- | --- |
| Входной параметр | Описание параметра | Значение по умолчанию |
| -in:<путь к файлу> | Файл с исходным кодом на языке LPA-2024 в формате .txt. | Не предусмотрено |
| -out:<путь к файлу> | Выходной файл, содержащий код на языке ассемблера | Не предусмотрено |
| -log:<путь к файлу> | Файл с протоколом работы транслятора. | <имя файла in>.log |
| -data:<путь к файлу> | Файл, содержащий таблицы лексем, таблицу идентификаторов и дерево разбора. | <имя файла>.data.txt |

**2.3 Протоколы, формируемые транслятором**

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.

Таблица 2.2 – Стандартная библиотека языка LPA-2024

|  |  |
| --- | --- |
| Входной параметр | Описание |
| Файл, заданный параметром «-log:» | Содержит информацию о ходе выполнения трансляции: перечисление входных параметров, количество символов и строк, успех или ошибку по каждому этапу трансляции. Если обнаружена ошибка, в файл будет выведена вся информация об ошибке. |
| Файл, заданный параметром «-data:» | Содержит таблицы лексем и идентификаторов и дерево разбора. |

**3. Разработка лексического анализатора**

**3.1 Структура лексического анализатора**

Лексический анализатор – это программа, которая преобразует исходный текст программы, заменяя лексические единицы их внутренним представлением – лексемами, для создания промежуточного представления исходной программы. Структура лексического анализатора изображена на рисунке 3.1.

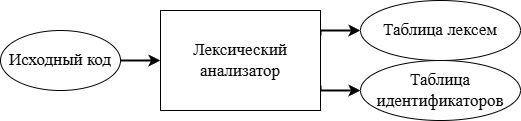


Рисунок 3.1 – принцип работы транслятора

Лексический анализатор работает в два этапа:

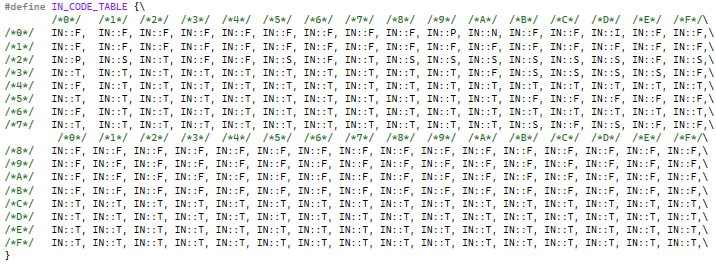
* Разбиение исходного кода программы на токены.
* Идентификация токенов и замена их на лексемы. Заполнение таблицы лексем и таблицы идентификаторов.

Входные данные: исходный код программы.

Результат работы: таблица лексем и таблица идентификаторов.

**3.2 Контроль входных символов**

Для удобной работы с исходным кодом, при передаче его в лексический анализатор, все символы разделяются по категориям. Таблица контроля входных символов языка LPA-2024 представлена на рисунке 3.2.

 Рисунок 3.2 – Таблица контроля входных символов

Каждому элементу соответствует значение в шестнадцатеричной системе счисления из кодировки Windows-1251, изображённой на рисунке 1.1.

Каждый считанный символ из исходного кода проверяется на допустимость. Если символ в таблице обозначен как T, то данный символ считается разрешённым и записывается в строку, если символ обозначен как F, то его позиция в исходном коде записывается в структуру ошибок. В случае обнаружения недопустимого символа будет сгенерирована ошибка и она будет выведена в консоль вместе с позицией в исходном коде. Символ, обозначенный буквой I – игнорируемый, S – символы-сепараторы, P – пробел и табуляция, N – переход на новую строку.

**3.3 Удаление избыточных символов**

Избыточный символ – это символ, отсутствие которого не влияет на исходный текст программы. В языке LPA-2024 символ пробела и табуляции являются избыточными символами. Их удаление предусмотрено на этапе разбиения исходного кода на токены.

Описание алгоритма удаления избыточных символов:

1. Посимвольно считываем файл с исходным кодом программы;
2. Пробел и табуляция являются символами-сепараторами, которые сигнализируют о начале или конце записи токена;
3. В отличии от других символов-сепараторов пробел и табуляцию не записываем в очередь лексем. Таким образом они будут проигнорированы.

**3.4 Перечень ключевых слов**

Ключевые слова, сепараторы, символы операций, соответствующие им лексемы приведены в таблице 3.1.

Таблица 3.1 – Соответствие токенов и сепараторов с лексемами.

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| int | e | Целочисленный тип данных. |
| char | n | Символьный тип данных. |
| str | s | Строковый тип данных. |
| bool | b | Логический тип данных. |
| идентификатор | i | Идентификатор типов данных. |
| литерал | l | Литерал любого типа данных. |
| main | m | Главная функция – точка входа в программу. |
| return | r | Выход из функции и возврат значения. |
| strduplicate | d | Стандартная строковая функция, выполняющая копирование строк. |
| strlength | f | Стандартная целочисленная функция. Вычисляет и возвращает длину строки. |

Продолжение таблицы 3.1

|  |  |  |
| --- | --- | --- |
| Слово (токен) | Лексема | Описание |
| strtransint | t | Стандартная целочисленная функция. Преобразует строку в число. |
| writeline | w | Оператор вывода в консоль с добавлением новой строки в конце. |
| write | v | Оператор вывода в консоль без добавления новой строки в конце. |
| cycle | c | Оператор цикла. |
| { | { | Начало блока кода. |
| } | } | Конец блока кода. |
| ( | ( | Начало перечислений параметров у функции. Начало приоритета операций в выражениях. |
| ) | ) | Конец перечислений параметров у функции. Конец приоритета операций в выражениях. |
| + | + | Арифметический оператор (сложение). |
| - | - | Арифметический оператор (вычитание). |
| \* | \* | Арифметический оператор (умножение). |
| / | / | Арифметический оператор (деление). |
| % | % | Арифметический оператор (остаток от деления). |
| = | = | Арифметический оператор (присваивание). |
| > | > | Оператор сравнения (больше). |
| < | < | Оператор сравнения (меньше). |
| >= | ] | Оператор сравнения (больше либо равно). |
| <= | [ | Оператор сравнения (меньше либо равно). |
| == | # | Оператор сравнения (равенство). |
| != | ! | Оператор сравнения (неравенство). |
| , | , | Разделитель параметров функции. |
| ; | ; | Конец инструкции. |

Каждому токену соответствует детерминированный конечный автомат, по которому происходит разбор выражения. Графы переходов конечных автоматов для ключевых слов языка LPA-2024 представлены в приложении Б.

**3.5 Основные структуры данных**

Основные структуры данных лексического анализатора: таблица токенов, таблица лексем и таблица идентификаторов.

В листинге 3.2 представлена реализация структуры токена и таблицы токенов в языке LPA-2024.

|  |
| --- |
| // Структура токена  struct Token  {  char token[258];  int length;  int line;  int linePosition;  };  // Структура таблицы токенов  struct TokenTable  {  int maxsize;  int size;  Token\* table;  }; |

Листинг 3.2 – Структура токена и таблицы токенов

Структура Token представляет собой строку таблицы TokenTable, где в массив token записывается слово, length – длина слова, line – номер строки в исходном тексте программы, linePosition – позиция в строке.

Структура TokenTable представляет таблицу токенов, где maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

В листинге 3.3 представлена реализация структуры таблицы лексем.

|  |
| --- |
| struct Entry  {  char lexema; // символ лексемы  int sn; // номер строки в коде  int idxTI; // номер в IT  };  struct LexTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.3 – Структура таблицы лексем

Структура Entry представляет собой строку таблицы LexTable, где lexeme – лексема, sn – номер строки в исходном тексте программы, idxTI – номер в таблице идентификаторов.

Структура LexTable представляет таблицу лексем, где maxsize – число, равное максимальному размеру таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

В листинге 3.4 представлена реализация структуры таблицы идентификаторов.

|  |
| --- |
| struct Entry  {  int idxfirstLE; // идекс первого вхождения в таблицу лексем  std::string id; // идентификатор  IDDATATYPE idDataType; // тип данных  IDTYPE idType; // тип идентиикатора  struct  {  int vshort; // целочисленное значение  struct  {  int len; // длин астроки  std::string str; // строка  } vstr;  } value;  };  struct IdTable  {  int maxsize;  int size;  Entry\* table;  }; |

Листинг 3.4 – Структура таблицы идентификаторов

Структура Entry представляет собой строку таблицы idTable, где переменная idxfirstLE – индекс первого вхождения в таблицу лексем, id – идентификатор, idDataType – тип данных, idType – тип идентификатора, vshort – целочисленное значение, len – длина строки, str – строка.

Структура IdTable представляет таблицу идентификаторов, где maxsize – максимальный размер таблицы, size – текущий размер таблицы, table – указатель на строку таблицы.

**3.6 Структура и перечень сообщений лексического анализатора**

При возникновении ошибки в лексическом анализаторе формируется ошибка в следующем формате: Номер ошибки, пояснительный текст, строка в исходном тексте, позиция в строке. Перечень сообщений лексического анализатора приведен в таблице 3.2.

Таблица 3.2 – Перечень сообщений лексического анализатора языка LPA-2024

|  |  |
| --- | --- |
| Код ошибки | Пояснительный текст |
| 110 | Ошибка лексического анализатора: Недопустимый символ в исходном файле (-in:) |
| 111 | Ошибка лексического анализатора: Превышена емкость таблицы лексем |
| 112 | Ошибка лексического анализатора: Превышено количество строк в таблице лексем |
| 113 | Ошибка лексического анализатора: В таблице лексем отсутствует строка с заданным номером |
| 114 | Ошибка лексического анализатора: Превышена емкость таблицы идентификаторов |
| 115 | Ошибка лексического анализатора: Превышено количество строк в таблице идентификаторов |
| 116 | Ошибка лексического анализатора: В таблице идентификаторов отсутствует строка с заданным номером |
| 117 | Ошибка лексического анализатора: Не удалось определить тип лексемы |
| 118 | Ошибка лексического анализатора: Превышена емкость таблицы токенов |
| 119 | Ошибка лексического анализатора: Превышено количество токенов в таблице токенов |
| 120 | Ошибка лексического анализатора: Ошибка с разбиением исходного текста на токены |
| 121 | Ошибка лексического анализатора: Ошибка с разбором строкового литерала |

* 1. **Принцип обработки ошибок**

При обнаружении ошибки в исходном коде программы лексический анализатор формирует сообщение об ошибке и выводит его в консоль и записывает в файл с протоколом работы, заданный параметром –log:. Максимальное количество кодов с ошибками – 1000. Максимальная длина поясняющего сообщения ошибки – 200 символов.

* 1. **Параметры лексического анализатора**

Входным параметром лексического анализатора является исходный текст программы после проверки на допустимость символов. Этот параметр используется для создания таблицы лексем и таблицы идентификаторов, необходимых для дальнейшей работы транслятора.

* 1. **Алгоритм лексического анализа**

Алгоритм лексического анализа языка LPA-2024 выполняется по следующему порядку:

1. считывается исходный текст программы и делится на токены, формируя структуру таблицы токенов;
2. слова из токенов проходят через графы конечного автомата, определяя тип лексем;
3. составляется таблица лексем и таблица идентификаторов

Исходный код программы, реализующий алгоритм лексического анализа представлен в листинге 3.5.

|  |
| --- |
| namespace LA  {  struct LEX  {  IT::IdTable idtable;  LT::LexTable lextable;  LEX();  LEX(int lexTableSize, int idTableSize);  };  // Определяем тип лексемы  char LexType(Tokens::Token token);  // Заполнение таблиц  LEX FillingInTables(Tokens::TokenTable tokenTable);  // Поиск id в таблице  int SearchID(stack<int> areaOfVisibility, string id, IT::IdTable& idTable);  // Поиск id функции в таблице  int SearchGlobalFunctionID(int globalAreaOfVisibility, string id, IT::IdTable& idTable);  } |

Листинг 3.5 – Реализация лексического анализатора

* 1. **Контрольный пример**

Результат работы лексического анализатора, полученный при выполнении контрольного примера представлен в приложении В.

# Приложение А

Контрольный пример

int testfunc1 (int a, int b)

{

int res = a + b;

return res;

};

str testfunc2 (str parm1)

{

str parm2 = "TestText20";

strduplicate(parm1, parm2); // Копирование строки parm2 в parm1

return parm1;

};

char testfunc3 (char simbol)

{

int i = 0;

cycle(i < 10) {

write simbol;

i = i + 1;

}

}

bool testfunc4(int chetnoe)

{

return chetnoe % 2 == 0;

}

main

{

writeline "Арифметические операции:";

int a;

int b = 15;

a = 10 \* 5; // Умножение

write a;

a = 10 / 5; // Деление

write a;

a = 15 % 10; // Деление с остатком

write a;

a = 10 + 5; // Сумма

write a;

a = 10 - 5; // Разница

write a;

writeline "Вызов функций:";

// Пользовательские функции:

int number = testfunc1 (0, 5);

write number;

str parm1 = "TestText1";

str string = testfunc2 (parm1);

writeline string;

char alpha = 'a';

testfunc3 (alpha);

int myint = 120;

bool pravda = testfunc4(myint);

writeline pravda;

// Библиотечные функции:

str stroka = "Тестовая строчка";

str strokachislo = "125";

strduplicate (copystr, stroka); // Копирование строки stroka в copystr

writeline copystr;

int dlina = strlength (stroka); // Вычисление длины строки stroka

writeline dlina;

int testint = strtransint (strokachislo);

int testint2 = 5 + testint;

writeline testint2;

writeline "Цикл:";

int i = 0;

cycle(i < 10)

{

write i;

i = i + 1;

}

}

Контрольный пример, содержащий 3 семантические ошибки

int testfunc1(int a, int b)

{

int res = a + b;

return res;

};

str testfunc1(str parm1) // Ошибка 130 (существующий идентификатор функции)

{

str parm2 = "TestText20";

strduplicate(parm1, parm2); // Копирование строки parm2 в parm1

return parm1;

};

char testfunc3(char simbol)

{

int i = 0;

cycle(i < 10) {

write simbol;

i = i + 1;

}

}

bool testfunc4(int chetnoe)

{

return chetnoe % 2 == 0;

}

main

{

writeline "Арифметические операции:";

int a;

int b = 15;

str c = "text";

int a = 10 \* 5; // Умножение. Ошибка 131 (повторное объявление идентификатора)

write a;

a = 10 / 5; // Деление

write a;

a = 15 % 10; // Деление с остатком

write a;

a = 10 + 5; // Сумма

write a;

a = 10 - 5; // Разница

write a;

d = c + 10; // Ошибка 132 (оператор не предназначен для работы со строками

writeline "Вызов функций:";

// Пользовательские функции:

int number = testfunc1(0, 5);

write number;

str parm1 = "TestText1";

str string = testfunc1(parm1);

writeline string;

char alpha = 'a';

testfunc3(alpha);

int myint = 120;

bool pravda = testfunc4(myint);

writeline pravda;

// Библиотечные функции:

str stroka = "Тестовая строчка";

str strokachislo = "125";

strduplicate(copystr, stroka); // Копирование строки stroka в copystr

writeline copystr;

int dlina = strlength(stroka); // Вычисление длины строки stroka

writeline dlina;

int testint = strtransint(strokachislo);

int testint2 = 5 + testint;

writeline testint2;

writeline "Цикл:";

int i = 0;

cycle(i < 10)

{

write i;

i = i + 1;

}

}

Допущенные семантические ошибки:

* Ошибка 130: Ошибка семантического анализа: Функция с таким идентификатором уже существует, строка 7, позиция 5
* Ошибка 131: Ошибка семантического анализа: Повторное объявление идентификатора, строка 35, позиция 6
* Ошибка 132: Ошибка семантического анализа: Оператор не предназначен для работы со строками, строка 46, позиция 7

# Приложение Б

Графы переходов конечных автоматов для ключевых слов

// Типы данных

#define GRAPH\_INT \

4, \

FST::NODE(1, FST::RELATION('i', 1)), \

FST::NODE(1, FST::RELATION('n', 2)), \

FST::NODE(1, FST::RELATION('t', 3)), \

FST::NODE()

#define GRAPH\_CHAR \

5, \

FST::NODE(1, FST::RELATION('c', 1)), \

FST::NODE(1, FST::RELATION('h', 2)), \

FST::NODE(1, FST::RELATION('a', 3)), \

FST::NODE(1, FST::RELATION('r', 4)), \

FST::NODE()

#define GRAPH\_STR \

4, \

FST::NODE(1, FST::RELATION('s', 1)), \

FST::NODE(1, FST::RELATION('t', 2)), \

FST::NODE(1, FST::RELATION('r', 3)), \

FST::NODE()

#define GRAPH\_BOOL \

5, \

FST::NODE(1, FST::RELATION('b', 1)), \

FST::NODE(1, FST::RELATION('o', 2)), \

FST::NODE(1, FST::RELATION('o', 3)), \

FST::NODE(1, FST::RELATION('l', 4)), \

FST::NODE()

#define GRAPH\_BOOL\_TRUE \

5, \

FST::NODE(1, FST::RELATION('t', 1)), \

FST::NODE(1, FST::RELATION('r', 2)), \

FST::NODE(1, FST::RELATION('u', 3)), \

FST::NODE(1, FST::RELATION('e', 4)), \

FST::NODE()

#define GRAPH\_BOOL\_FALSE \

6, \

FST::NODE(1, FST::RELATION('f', 1)), \

FST::NODE(1, FST::RELATION('a', 2)), \

FST::NODE(1, FST::RELATION('l', 3)), \

FST::NODE(1, FST::RELATION('s', 4)), \

FST::NODE(1, FST::RELATION('e', 5)), \

FST::NODE()

// Функции

#define GRAPH\_MAIN \

5, \

FST::NODE(1,FST::RELATION('m', 1)), \

FST::NODE(1,FST::RELATION('a', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('n', 4)), \

FST::NODE()

#define GRAPH\_RETURN \

7, \

FST::NODE(1,FST::RELATION('r', 1)), \

FST::NODE(1,FST::RELATION('e', 2)), \

FST::NODE(1,FST::RELATION('t', 3)), \

FST::NODE(1,FST::RELATION('u', 4)), \

FST::NODE(1,FST::RELATION('r', 5)), \

FST::NODE(1,FST::RELATION('n', 6)), \

FST::NODE()

#define GRAPH\_STRDUPLICATE \

13, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('d', 4)), \

FST::NODE(1,FST::RELATION('u', 5)), \

FST::NODE(1,FST::RELATION('p', 6)), \

FST::NODE(1,FST::RELATION('l', 7)), \

FST::NODE(1,FST::RELATION('i', 8)), \

FST::NODE(1,FST::RELATION('c', 9)), \

FST::NODE(1,FST::RELATION('a', 10)), \

FST::NODE(1,FST::RELATION('t', 11)), \

FST::NODE(1,FST::RELATION('e', 12)), \

FST::NODE()

#define GRAPH\_STRLENGTH \

10, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('l', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE(1,FST::RELATION('n', 6)), \

FST::NODE(1,FST::RELATION('g', 7)), \

FST::NODE(1,FST::RELATION('t', 8)), \

FST::NODE(1,FST::RELATION('h', 9)), \

FST::NODE()

#define GRAPH\_STRTRANSINT \

12, \

FST::NODE(1,FST::RELATION('s', 1)), \

FST::NODE(1,FST::RELATION('t', 2)), \

FST::NODE(1,FST::RELATION('r', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('r', 5)), \

FST::NODE(1,FST::RELATION('a', 6)), \

FST::NODE(1,FST::RELATION('n', 7)), \

FST::NODE(1,FST::RELATION('s', 8)), \

FST::NODE(1,FST::RELATION('i', 9)), \

FST::NODE(1,FST::RELATION('n', 10)), \

FST::NODE(1,FST::RELATION('t', 11)), \

FST::NODE()

// Операторы вывода и цикл

#define GRAPH\_WRITELINE \

10, \

FST::NODE(1,FST::RELATION('w', 1)), \

FST::NODE(1,FST::RELATION('r', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE(1,FST::RELATION('l', 6)), \

FST::NODE(1,FST::RELATION('i', 7)), \

FST::NODE(1,FST::RELATION('n', 8)), \

FST::NODE(1,FST::RELATION('e', 9)), \

FST::NODE()

#define GRAPH\_WRITE \

6, \

FST::NODE(1,FST::RELATION('w', 1)), \

FST::NODE(1,FST::RELATION('r', 2)), \

FST::NODE(1,FST::RELATION('i', 3)), \

FST::NODE(1,FST::RELATION('t', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE()

#define GRAPH\_CYCLE \

6, \

FST::NODE(1,FST::RELATION('c', 1)), \

FST::NODE(1,FST::RELATION('y', 2)), \

FST::NODE(1,FST::RELATION('c', 3)), \

FST::NODE(1,FST::RELATION('l', 4)), \

FST::NODE(1,FST::RELATION('e', 5)), \

FST::NODE()