МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность ПОИТ

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Разработка компилятора KVV-2023»

Выполнил студент Корнелюк Валентин Владимирович

(Ф.И.О.)

Руководитель проекта доц. Наркевич Аделина Сергеевна

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

Заведующий кафедрой доц. Смелов Владимир Владиславович

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

Консультанты доц. Наркевич Аделина Сергеевна

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

Курсовой проект защищен с оценкой

Минск 2023

Оглавление

[Введение 3](#_Toc148170499)

[1 Спецификация языка программирования 4](#_Toc148170500)

[1.1 Характеристика языка программирования 4](#_Toc148170501)

[1.2 Алфавит языка 4](#_Toc148170502)

[1.3 Применяемые сепараторы 4](#_Toc148170503)

[1.4 Применяемые кодировки 5](#_Toc148170504)

[1.5 Типы данных 5](#_Toc148170505)

[1.6 Преобразование типов данных 6](#_Toc148170506)

[1.7 Идентификаторы 6](#_Toc148170507)

[1.8 Литералы 6](#_Toc148170508)

[1.9 Объявление данных 7](#_Toc148170509)

[1.10 Инициализация данных 7](#_Toc148170510)

[1.11 Инструкции языка 7](#_Toc148170511)

[1.12 Операции языка 8](#_Toc148170512)

[1.13 Выражения и их вычисление 9](#_Toc148170513)

[1.14 Конструкции языка 9](#_Toc148170514)

[1.15 Области видимости идентификаторов 10](#_Toc148170515)

[1.16 Семантические проверки 10](#_Toc148170516)

[1.17 Распределение оперативной памяти на этапе выполнения 11](#_Toc148170517)

[1.18 Стандартная библиотека и её состав 11](#_Toc148170518)

[1.19 Ввод и вывод данных 11](#_Toc148170519)

[1.20 Точка входа 12](#_Toc148170520)

[1.21 Препроцессор 12](#_Toc148170521)

[1.22 Соглашения о вызовах 12](#_Toc148170522)

[1.23 Объектный код 12](#_Toc148170523)

[1.24 Классификация сообщений транслятора 12](#_Toc148170524)

[1.25 Контрольный пример 13](#_Toc148170525)

# Введение

В данном курсовом проекте поставлена задача разработки собственного языка программирования и транслятора для него. Название языка – KVV-2023. Написание транслятора будет осуществляться на языке C++.

Задание на курсовой проект можно разделить на следующие задачи:

1. Разработка спецификации языка KVV-2023;
2. Разработка лексического анализатора;
3. Разработка синтаксического анализатора;
4. Разработка семантического анализатора;
5. Разбор арифметических выражений;
6. Разработка генератора кода;
7. Тестирование транслятора.

# 1 Спецификация языка программирования

## 1.1 Характеристика языка программирования

Язык программирования KVV-2023 является процедурным, универсальным строго типизированным, не объектно-ориентированным, компилируемым.

## 1.2 Алфавит языка

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

При написании программы на языке KVV-2023 используется таблица символов ASCII, представленная в рис.1.1.



Рисунок 1.1 Алфавит входных символов

Символы, используемые на этапе выполнения: [a…z], [A…Z], [0…9], символы пробела, табуляции и перевода строки, спецсимволы: [ ] ( ) , ; : # + - / \* > < & !.

## 1.3 Применяемые сепараторы

Символы-сепараторы служат в качестве разделителей цепочек языка во время обработки исходного текста программы с целью разделения на токены. Они представлены в таблице 1.1.

Таблица 1.1 – Символы-сепараторы

|  |  |
| --- | --- |
| Сепаратор | Назначение |
| **=** | Оператор присваивания |

Продолжение таблицы 1.1

|  |  |
| --- | --- |
| ‘пробел’ | Разделитель цепочек. Допускается везде кроме названий идентификаторов и ключевых слов |
| **[** … **]** | Блок функции или условной конструкции/цикла |
| **(** … **)** | Блок фактических или формальных параметров функции, а также приоритет арифметических операций |
| **,** | Разделитель параметров функций |
| **#** | Символ, отделяющий условные конструкции/циклы |
| **+ - \*/** | Арифметические операции |
| **> < & !** | Логические операции (операции сравнения: больше, меньше, проверка на равенство, на неравенство), используемые в условии цикла/условной конструкции. |
| **;** | Разделитель программных конструкций |
| }{ | Операторы сдвигов |

## 1.4 Применяемые кодировки

Для написания программ язык KVV-2023 использует кодировку ASCII, содержащую английский алфавит, а также некоторые специальные символы, такие как [ ] ( ) , ; : # + - / \* > < & !{}.

## 1.5 Типы данных

В языке KVV-2023 реализованы два фундаментальных типа данных: целочисленный и строковый. Описание типов приведено в таблице 1.2.

Таблица 1.2 – Типы данных языка KVV-2023

|  |  |
| --- | --- |
| Тип данных | Характеристика |
| Целочисленный тип данных **number** | Фундаментальный тип данных. Используется для работы с числовыми значениями. В памяти занимает 1 байт. Максимальное значение: 127.  Минимальное значение: -128.  Инициализация по умолчанию: значение 0.  Поддерживаемые операции:  **+** (бинарный) – оператор сложения;  - (бинарный) – оператор вычитания;  **\*** (бинарный) – оператор умножения;  **/** (бинарный) – оператор деления;  **=** (бинарный) – оператор присваивания.  В качестве условия цикла/условного оператора поддерживаются следующие логические операции:  **>** (бинарный) – оператор «больше»; |

Продолжение таблицы 1.2

|  |  |
| --- | --- |
|  | **<** (бинарный) – оператор «меньше»;  **!** (бинарный) – оператор проверки на неравенство. |
| Строковый тип данных **string** | Фундаментальный тип данных. Используется для работы с символами, каждый из которых занимает 1 байт. Максимальное количество символов – 255.  Инициализация по умолчанию: строка нулевой длины “”. Операции над данными строкового типа: присваивание строковому идентификатору значения другого строкового идентификатора, строкового литерала или значения строковой функции, а также использование библиотечных функций. |

## 1.6 Преобразование типов данных

В языке программирования KVV-2023 присутствует преобразование строки в число с помощью функции стандартной библиотеки atoii(string), которая возвращает значение number.

## 1.7 Идентификаторы

Общее количество идентификаторов ограничено максимальным размером таблицы идентификаторов. Идентификаторы должны содержать только символы нижнего регистра латинского алфавита. Максимальная длина идентификатора равна восьми символам. Идентификаторы, объявленные внутри функционального блока, получают префикс, идентичный имени функции, внутри которой они объявлены. Префикс занимает 8 дополнительных символов. В случае превышения заданной длины, идентификаторы усекаются до длины, равной 16 символов (8 символов на имя идентификатора, 8 символов на префикс). Данные правила действуют для всех типов идентификаторов. Зарезервированные идентификаторы не предусмотрены. Идентификаторы не должны совпадать с ключевыми словами. Типы идентификаторов: имя переменной, имя функции, параметр функции. Имена идентификаторов-функций не должны совпадать с именами команд ассемблера (это не касается имён идентификаторов-переменных).

## 1.8 Литералы

С помощью литералов осуществляется инициализация переменных. Все литералы являются **rvalue**. Типы литералов языка KVV-2023 представлены в таблице 1.3.

Таблица 1.3 – Литералы

|  |  |
| --- | --- |
| Литералы | Пояснение |
| Целочисленные литералы в десятичном представлении | Последовательность цифр 0…9 с предшествующим знаком минус или без него (знак минус не должен отделяться пробелом) |

Продолжение таблицы 1.3

|  |  |
| --- | --- |
| Строковые литералы | Набор символов (от 1 до 255), заключённых в двойные кавычки |
| Целочисленные литералы в восьмеричном представлении | Последовательность цифр 0…7 с предшествующим символом ‘q’ |

Ограничения на строковые литералы языка KVV-2023: внутри литерала не допускается использование символов кириллицы, а также одинарных и двойных кавычек. Ограничения на целочисленные литералы: не могут начинаться с нуля, если их значение не ноль; если литерал отрицательный, после знака “-” не может быть нуль.

## 1.9 Объявление данных

Для объявления переменной используется ключевое слово **new**, после которого указывается тип данных и имя идентификатора. Допускается инициализация при объявлении.

Пример объявления числового типа с инициализацией:

**new** **number** num1 = -1

**new** **number** num2 = q80

Пример объявления переменной символьного типа с инициализацией:

**new** **string** str1= “hello world”

Для объявления функций используется ключевое слово **function**, перед которым указывается тип функции (если функция возвращает значение), или ключевое слово **procedure**, если функция ничего не возвращает, а после – имя функции либо процедуры. Далее обязателен список параметров и тело функции.

## 1.10 Инициализация данных

При объявлении переменной допускается инициализация данных. При этом переменной будет присвоено значение литерала или идентификатора, стоящего справа от знака равенства. Объектами-инициализаторами могут быть только идентификаторы или литералы. При объявлении без инициализации предусмотрены значения по умолчанию: значение 0 для типа **number** и строка длины 0 (“”) для типа **string**.

## 1.11 Инструкции языка

Инструкции языка KVV-2023 представлены в таблице 1.4.

Таблица 1.4 – ИнструкцииязыкаKVV-2023

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной | **new** <тип данных> <идентификатор>; |

Продолжение таблицы 1.4

|  |  |
| --- | --- |
| Инструкция | Реализация |
| Объявление переменной с явной инициализацией | **new** <тип данных> <идентификатор> = <значение>;  Значение – инициализатор конкретного типа. Может быть только литералом или идентификатором |
| Возврат из функции или процедуры | Для функций, возвращающих значение:  **return** <идентификатор/литерал>;  Для процедур:  **return;** |
| Вывод данных | **write** <идентификатор/литерал>; |
| Вызов функции или процедуры | <идентификатор функции> (<список параметров>);  Список параметров может быть пустым. |
| Перевод строки | **newline**; |
| Присваивание | <идентификатор> **=** <выражение>;  Выражением может быть идентификатор, литерал, или вызов функции соответствующего типа. Для целочисленного типа выражение может быть дополнено арифметическими операциями с любым количеством операндов с использованием скобок. Для строкового типа выражение может быть только идентификатором, литералом или вызовом функции, возвращающей значение строкового типа. |

## 1.12 Операции языка

В языке KVV-2023 предусмотрены следующие операции с данными. Приоритетность операции умножения выше приоритета операций сложения и вычитания. Для установки наивысшего приоритета используются круглые скобки. Операции языка представлены в таблице 1.5.

Таблица 1.5 – Операции языка KVV-2023

|  |  |
| --- | --- |
| Тип оператора | Оператор |
| Арифметические | 1. **+ –** сложение 2. - – вычитание 3. \* – умножение 4. **/** – деление 5. = – присваивание |
| Строковые | 1. **=** – присваивание |
| Логические | 1. **>** – больше  2. **<** – меньше  3. ! – проверка на неравенство |
| Сдвиговые | 1. } – cдвиг вправо  2. { – сдвиг влево |

## 1.13 Выражения и их вычисление

Вычисление выражений – одна из важнейших задач языков программирования. Всякое выражение составляется согласно следующим правилам:

1. Допускается использовать скобки для смены приоритета операций;
2. Выражение записывается в строку без переносов;
3. Использование двух подряд идущих операторов не допускается;
4. Допускается использовать в выражении вызов функции, вычисляющей и возвращающей целочисленное значение.

## 1.14 Конструкции языка

Программа на языке KVV-2023 оформляется в виде функций пользователя и главной функции. При составлении функций рекомендуется выделять блоки и фрагменты и применять отступы для лучшей читаемости кода.

Программные конструкции языка представлены в таблице 1.6.

Таблица 1.6 – Программные конструкции языка KVV-2023

|  |  |  |
| --- | --- | --- |
| Конструкция | | Реализация |
| Главная функция | | **main**  **[**  …  **]** |
| Внешняя функция | | <тип данных> **function** <идентификатор> **(**<тип> <идентификатор>, ...**)**  **[**…  **return** <идентификатор/литерал>;  **]** |
| Внешняя процедура | **procedure** **function** <идентификатор> **(** <тип> <идентификатор>, ...**)**  **[**…  **return**;  **]** | |
| Цикл | **condition**: <идентификатор1> <оператор> <идентификатор2> **#**  **cycle** [ … **]** **#**  Цикл (операторы внутри блока **cycle**) выполняется, пока истинно условие “<идентификатор1> <оператор> <идентификатор2>”, имеющее тот же смысл, что и в примере выше. | |

Продолжение таблицы 1.6

|  |  |
| --- | --- |
| Конструкция | Реализация |

|  |  |
| --- | --- |
| Условная конструкция | **condition:** <идентификатор1> <оператор> <идентификатор2> **#**  **istrue** **[** … **]**  **isfalse** **[** … **]** **#**  <идентификатор1>, <идентификатор2> - идентификаторы или литералы целочисленного типа (но не два литерала одновременно). <оператор> - один из операторов сравнения ( **> < & !** ), устанавливающий отношение между двумя операндами и организующий условие данной конструкции. При истинности условия выполняется код внутри блока **istrue**, иначе – код внутри блока **isfalse**. Любой из блоков **istrue**, **isfalse** может отсутствовать, но не оба блока одновременно. При отсутствии одного из блоков, в зависимости от истинности или ложности условия программа может как выполнить один из заявленных блоков, так и передать управление инструкции, следующей в коде за закрывающим условную конструкцию символом **‘#’**. |

## 1.15 Области видимости идентификаторов

Область видимости: сверху вниз (как и в С++). Переменные, объявленные в одной функции, недоступны в другой. Все объявления и операции с переменными происходят внутри какого-либо блока. Каждая переменная или параметр функции получают префикс – название функции, внутри которой они находятся.

Все идентификаторы являются локальными и обязаны быть объявленными внутри какой-либо функции. Глобальных переменных нет. Параметры видны только внутри функции, в которой объявлены.

## 1.16 Семантические проверки

В языке программирования KVV-2023 выполняются следующие семантические проверки:

1. Наличие функции **main** – точки входа в программу;
2. Единственность точки входа;
3. Переопределение идентификаторов;
4. Использование идентификаторов без их объявления;
5. Проверка соответствия типа функции и возвращаемого параметра;
6. Правильность передаваемых в функцию параметров: количество, типы;
7. Правильность строковых выражений;
8. Превышение размера строковых и числовых литералов;
9. Правильность составленного условия цикла/условного оператора.

## 1.17 Распределение оперативной памяти на этапе выполнения

Транслированный код использует две области памяти. В сегмент констант заносятся все литералы. В сегмент данных заносятся переменные и параметры функций. Локальная область видимости в исходном коде определяется за счет использования правил именования идентификаторов и регулируется их префиксами, что и обуславливает их локальность на уровне исходного кода.

## 1.18 Стандартная библиотека и её состав

В языке KVV-2023 присутствует стандартная библиотека, которая подключается автоматически при трансляции исходного кода в язык ассемблера. Содержимое библиотеки и описание функций представлено в таблице 1.8.

Таблица 1.8 Стандартная библиотека языка KVV-2023

|  |  |
| --- | --- |
| Функция | Описание |
| **string** concat(**string** result, **string** s1, **string** s2); | Строковая функция, выполняет объединение строк s1 и s2 в указанном порядке. Заполняет поле result результатом конкатенации строк s1 и s2 |
| **number** atoii(**string** str); | Целочисленная функция. Преобразует строку в число |
| **number** lenght(**string** str); | Целочисленная функция. Вычисляет и возвращает длину строки str |

Стандартная библиотека написана на языке С++, подключается к транслированному коду на этапе генерации кода. Вызовы стандартных функций доступны там же, где и вызов пользовательских функций. Также в стандартной библиотеке реализованы функции для манипулирования выводом, недоступные конечному пользователю. Для вывода предусмотрен оператор **write**. Эти функции представлены в таблице 1.9.

Таблица 1.9 Дополнительные функции стандартной библиотеки

|  |  |
| --- | --- |
| Функция на языке С++ | Описание |
| void outnum(int value) | Функция для вывода в стандартный поток значения целочисленного идентификатора/литерала. |
| void outstr(char\* line) | Функция для вывода в стандартный поток значения строкового идентификатора/литерала. |

## 1.19 Ввод и вывод данных

Вывод данных осуществляется с помощью оператора **write**. Допускается использование оператора **write** с литералами и идентификаторами.

Функции, управляющие выводом данных, реализованы на языке С++ и вызываются из транслированного кода, конечному пользователю недоступны. Пользовательская команда **write** в транслированном коде будут заменена вызовом нужных библиотечных функций. Библиотека, содержащая нужные процедуры, подключается на этапе генерации кода.

## 1.20 Точка входа

В языке KVV-2023 каждая программа должна содержать главную функцию (точку входа) **main**, с первой инструкции которой начнётся последовательное выполнение команд программы.

## 1.21 Препроцессор

Препроцессор, принимающий и выдающий некоторые данные на вход транслятору, в языке KVV-2023 отсутствует.

## 1.22 Соглашения о вызовах

В языке вызов функций происходит по соглашению о вызовах stdcall. Особенности stdcall:

– все параметры функции передаются через стек;

– память высвобождает вызываемый код;

– занесение в стек параметров идёт справа налево.

## 1.23 Объектный код

## 1.24 Классификация сообщений транслятора

Генерируемые транслятором сообщения определяют степень его информативности, то есть сообщения транслятора должны давать максимально полную информацию о допущенной пользователем ошибке при написании программы. Сообщения транслятора приведены в таблице 1.10.

Таблица 1.10 Классификация ошибок

|  |  |
| --- | --- |
| Номера ошибок | Характеристика |
| 0 – 200 | Системные ошибки |
| 200 – 299 | Ошибки лексического анализа |
| 300 – 399 | Ошибки семантического анализа |
| 600 – 699 | Ошибки синтаксического анализа |
| 400-499, 700-999 | Зарезервированные коды ошибок |

## 1.25 Контрольный пример

Контрольный пример демонстрирует главные особенности языка KVV-2023: его фундаментальные типы, основные структуры, функции, процедуры, использование функций статической библиотеки.

# 