Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

РАЗРАБОТКА КОМПИЛЯТОРА

Пояснительная записка

На 15 листах

Руководитель _____ к.т.н. доцент кафедры ИЗИ Ю.М. Монахов

Исполнитель _____ студент гр. ИБ-117 А.А. Лоскутов

Владимир 2020

АННОТАЦИЯ

В данном программном документе приведён текст компилятора подмножества процедурно-ориентированного языка. Компилятор реализован на языке Java с использованием библиотек Antlr и Asm. Основная функция компилятора – проверка принадлежности исходной цепочки входному языку и генерация выходной цепочки символов виде байт-кодя для виртуальной машины JVM.

Разработка компилятора подмножества процедурного языка в ассемблер состоит из следующих стадий:

1) построение лексического анализатора;

2) построение синтаксического анализатора;

3) построение генератора объектного кода;

Оглавление

1 ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА

1.1 Основные требования

Разработка будет производиться в соответствии со следующими требованиями:

- Требования к входному языку:

1. Должны присутствовать операторные скобки;
2. Должна игнорироваться индентация программы;
3. Должны поддерживаться комментарии любой длины;
4. Входная программа должна представлять собой единый модуль, но поддерживать вызов функций.

- Требования к операторам:

1. Оператор присваивания;
2. Арифметические операторы;
3. Логические операторы (И, ИЛИ, НЕ);
4. Условный оператор (ЕСЛИ);
5. Оператор цикла (while);
6. Базовый вывод (строковой литерал, переменная);
7. Типы (целочисленный, вещественный).

1.2. Лексический анализатор

Лексический анализатор является первой фазой компилятора. Он преобразует входной поток символов в поток токенов.

Грамматика языка реализована с использованием библиотеки Antlr 4. Грамматика языка включает в себя элементы java/c++/python.

Список зарезервированных слов:

1. `while`
2. `if`
3. `else`
4. `const`
5. `def`
6. `print`
7. `main`
8. `and`
9. `or`
10. `true`
11. `false`
12. `int`
13. `float`
14. `char`
15. `bool`

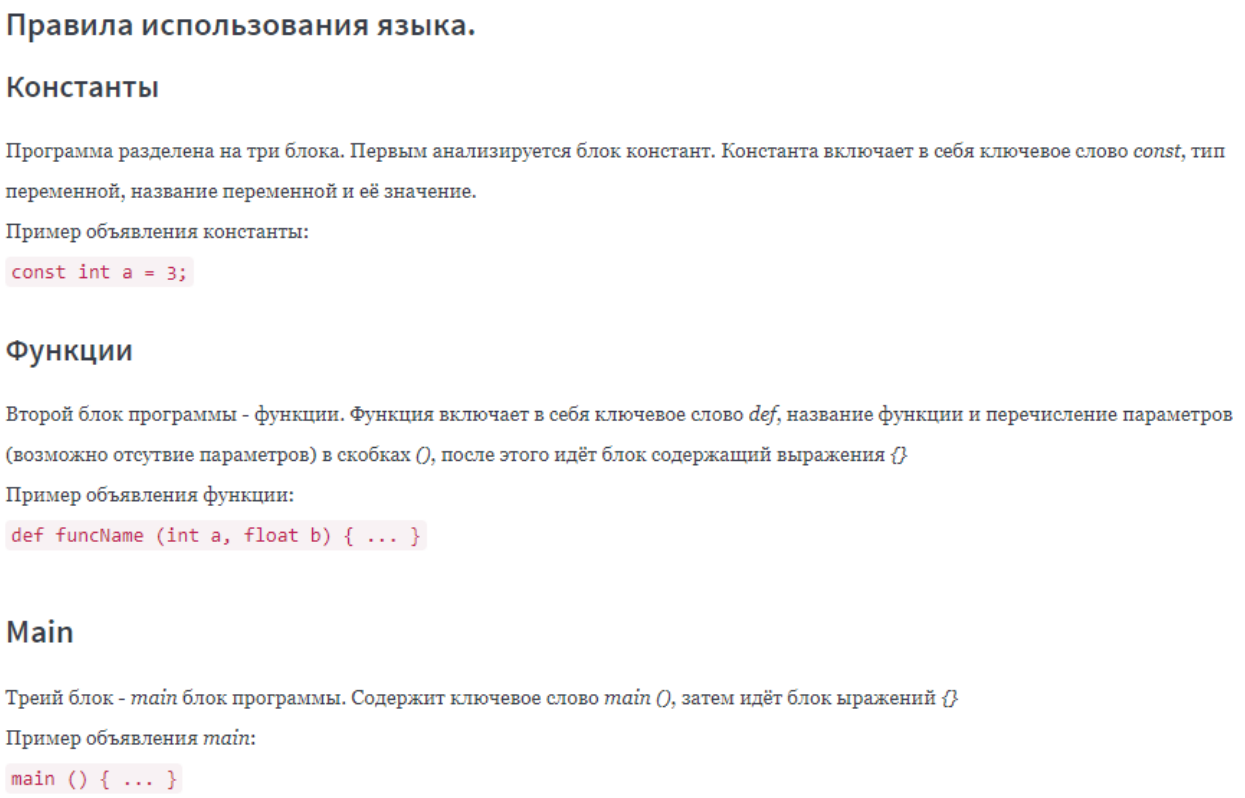
А также прочие символы на подобии *“{}”, “=”, “>”* и тд.

1.3. Синтаксического анализатор

Второй стадией компилятора является синтаксический анализ. На вход синтаксическому анализатору подаётся набор токенов из лексического анализатора. На основе грамматики языка строится дерево разбора грамматики.

Все взаимодействия во время построения дерева разбора обрабатываются через класс MyVisitor.

Привила использования грамматики представлены ниже на рисунках, а также можно посмотреть на https://github.com/irlyk/Compiler/blob/master/src/MLL/README.md:



Правила использования языка.

Константы

Программа разделена на три блока. Первым анализируется блок констант. Константа включает в себя ключевое слово *const*, тип переменной, название переменной и её значение.

Пример объявления константы:

```
const int a = 3;
```

Функции

Второй блок программы - функции. Функция включает в себя ключевое слово *def*, название функции и перечисление параметров (возможно отсутвие параметров) в скобках *()*, после этого идёт блок содержащий выражения *{}*

Пример объявления функции:

```
def funcName (int a, float b) { ... }
```

Main

Треий блок - *main* блок программы. Содержит ключевое слово *main ()*, затем идёт блок ыражений *{}*

Пример объявления *main*:

```
main () { ... }
```

Рисунок 1 Правила использования грамматики

Выражения

В грамматике определено несколько выражений: объявление переменной, присаивание, *if/else* конструкция, *while* конструкция, *for* конструкция, вызов функциии, вывод *print*.

1. Объявление переменной состоит из объявления типа, названия переменной и значения (может отсутвовать),
   пример: `int c; int a = 3; float b = 3.0 + 5.2;`
2. Присаивание состоит из объявления имени переменной и присваивания ей значения
   пример: `a = 3 + 3;`
3. *if/else* конструкция состоит из объявления ключевого слова *if*, логического выражения внутри фигурных скобок *()*, выражений внутри блока *{}* и опциональныъ ключевого слова *else* и блока выражений *{}*
   пример: `if (a > b) { ... } else { .... }`
4. *while* конструкция состоит из объявления ключевого слова *while*, логического выражения внутри фигурных скобок *()*, выражений внутри блока *{}*
   пример: `while (a > b) { ... }`
5. *for* конструкция состоит из объявления ключевого слова *for*, фигурных скобок *()* с счётчиком, логическим выражением и действием. Затем блок выражений *{}*
   пример: `for (int i = 0; i < 10; i = i + 1) { ... }`
6. Вызов функций осуществляется посредством указывания имени функции, и скобок с перменными если они необходимы *()*
   пример: `funcName(a);`
7. Вывод осуществляется посредстввом указывания ключевого слова *print* и скобок *()* внутри которых перечисление парметров через запятую, которые необходимо вывести
   пример: `print(a, b, c);`

Рисунок 2 Правила использования грамматики

1.4. Построение генератора объектного кода

Генерация объектного кода выполняется во время обхода дерева в классе MyVisitor, с помощью создания дерева элементов, каждый из которых имеет метод генерации байт-кода. Затем дерево передаётся классы Compiler, который добавляет необходимые элементы кода и вызывает метод genJVM у корня дерева.

Ключевые элементы дерева – Node (главный наследуемый класс), Stmt, Expr. Представляют собой наследуемые элементы, но не являются листьями.

- if, else, whiel, declare, set, print, seq – классы наследуемые от Stmt.
- op, constant, id, logical, elist – классы наследуемые от Expr.
- and, or, rel -классы наследуемые от Logical.

Пример дерева из кода:

```
int a = 0;
a = 3 + 5;
```

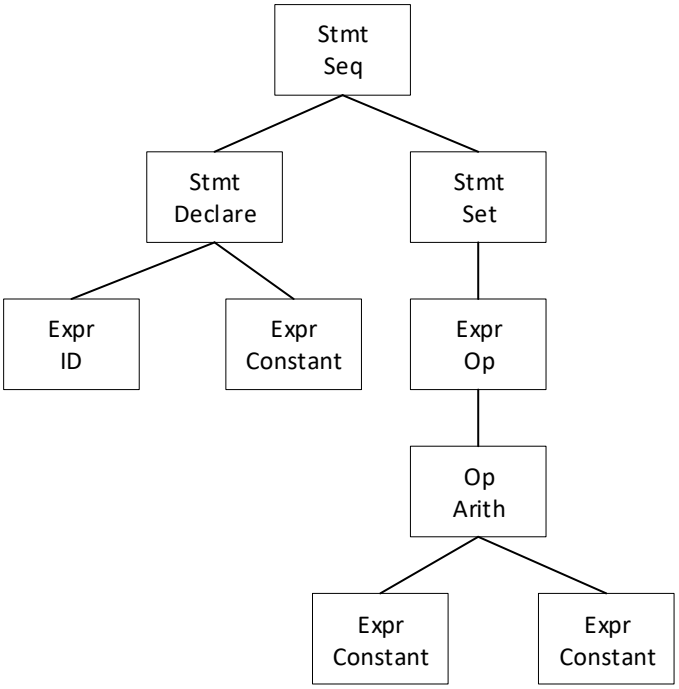


Рисунок 3 дерево для генерации кода

2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ

При проектировании компилятора к основному языку были установлены следующие минимальные требования: наличие операторных скобок, игнорирование пробелов и идентации программы, поддержка многострочных комментариев и вызова функций. Наличие операторов присваивания, условных, цикла, арифметических, логических. Должны присутствовать два типа данных.

Далее приведено тестирование компилятора.

Рисунки 4-7 – Проверка областей видимости

```
main() {
    int a = 0;
    int b = 10;
    if (true) {
        int a = 1;
        print(a);
        print(b);
    }
    print(a);
}
```

Рисунок 4 Код на исходном языке

```
public class test {
    public test() {
    }
    public static void main(String[] var0) {
        byte var1 = 0;
        byte var2 = 10;
        if (true) {
            byte var3 = 1;
            System.out.println(var3);
            System.out.println(var2);
        }
        System.out.println(var1);
    }
}
```

Рисунок 5 Декомпилированный файл

```
0: bipush        0
       2: istore_1
       3: bipush        10
       5: istore_2
       6: iconst_1
       7: ifne          13
      10: goto          32
      13: bipush        1
      15: istore_3
      16: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
      19: iload         3
      21: invokevirtual #22                 // Method java/io/PrintStream.println:(I)V
      24: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
      27: iload         2
      29: invokevirtual #22                 // Method java/io/PrintStream.println:(I)V
      32: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
      35: iload         1
      37: invokevirtual #22                 // Method java/io/PrintStream.println:(I)V
      40: return
```

Рисунок 6 Код на языке JVM

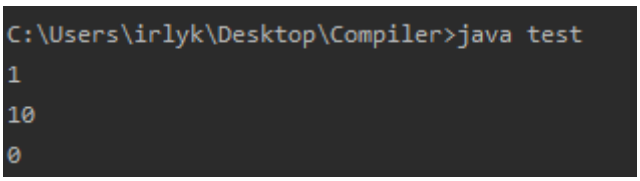


Рисунок 7 Результат программы

Рисунки 8-10 – проверка условного оператора

```
main() {
     int a = 1;
     int b = 3;
     if (a < b) {
        print ('y');
     } else {
        print ('n');
     }
     if ( true and false) {
        print ('y');
     } else {
         print ('n');
      }
     if ( true or false) {
        print ('y');
     } else {
         print ('n');
      }
}
```

Рисунок 8 Код на исходном языке

```
 0: bipush            1;   2: istore_1;     3: bipush        3;
 5: istore_2;         6: iload         1;   8: iload         2;
10: if_icmple     17; 13: iconst_0          14: goto          18;
17: iconst_1         18: ifne          32
21: getstatic;     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
24: bipush        110;
26: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
29: goto          40;
32: getstatic;     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
35: bipush        121;
37: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
40: iconst_1;        41: iconst_0;           42: imul;
43: iconst_0;        44: iadd;               45: ifne          59;
48: getstatic;     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
51: bipush        110;
53: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
56: goto          67;
59: getstatic;     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
62: bipush        121;
64: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
67: iconst_1;        68: iconst_0;       69: iadd
70: ifne          84
73: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
76: bipush        110;
78: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
81: goto          92;
84: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
87: bipush        121;
89: invokevirtual; #22                 // Method java/io/PrintStream.println:(C)V
92: return;
```

Рисунок 9 Код на языке JVM (сдвинут для наглядности)

```
C:\Users\irlyk\Desktop\Compiler>java test
y
n
y
```

Рисунок 10 Результат выполнения программы

Рисунки 11-13 – проверка цикла while

```
main() {
     int a = 1;
     int b = 3;
     while (a < b) {
        a = a + 1;
     }
     print (a);
 }
```

Рисунок 11 Код на исходном языке

```
 0: bipush        1
 2: istore_1
 3: bipush        3
 5: istore_2
 6: iload         1
 8: iload         2
10: if_icmple     17
13: iconst_0
14: goto          18
17: iconst_1
18: ifne          24
21: goto          33
24: iload         1
26: bipush        1
28: iadd
29: istore_1
30: goto          6
33: getstatic     #16                 // Field java/lang/System.out:Ljava/io/PrintStream;
36: iload         1
38: invokevirtual #22                 // Method java/io/PrintStream.println:(I)V
41: return
```

Рисунок 12 Код на языке JVM

```
C:\Users\irlyk\Desktop\Compiler>java test
4
```

Рисунок 13 Результат выполнения программы

Реквизиты к курсовой работе:

https://github.com/irlyk/Compiler