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 17 листах

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

Исполнитель _____ студент гр. ИСБ-118 А.А. Королева

Владимир 2021

Оглавление

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-кода для виртуальной машины JVM (Java Virtual Machine).

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

- построение лексического анализатора;
- построение синтаксического анализатора;
- построение генератора байт-кода.

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 (строковый 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 Antlr4. Грамматика языка включает в себя элементы из популярных языков, таких как C, C++, Java и др.

Список зарезервированных слов:

- int
- float
- char
- bool
- true
- false
- print
- if
- else
- while
- break
- continue
- const
- and
- or

А также прочие символы наподобие *“{}”, “=”, “>”* и т.д.

1.3 Синтаксический 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 Parser, который возвращает классы модуля AST при совпадении правила из КС-грамматики языка.

КС-грамматика языка представлена ниже.

```
grammar cring;

program
    : function? mainProg EOF
    ;

function
    : type FUNC varname OPAR parametrList CPAR block (FUNC varname OPAR parametrList CPAR block)*
    ;
```

```
mainProg
    : MAIN block
    ;

block
    : OBRACE (statement)* CBRACE
    ;

statement
    : varDeclaration SCOL
    | callMethod SCOL
    | expression SCOL
    | print SCOL
    | ifStatement
    | assign
    | whileStatement
    | forStatement
    | breakStat SCOL
    | contStat SCOL
    | retStat SCOL
    ;
```

```
varDeclaration
    : type varname (ASSIGN expression)?
    ;

callMethod
    : varname OPAR expressionList CPAR
    ;

print
    : PRINT OPAR expressionList CPAR
    ;

ifStatement
    : IF OPAR parametrs CPAR block (ELSE block)?
    ;

assign
    : varname ASSIGN expression SCOL
    ;

whileStatement
    : WHILE OPAR parametrs CPAR block
    ;

forStatement
    : FOR OPAR forInside CPAR block
    ;

forInside
    : forInit SCOL parametrs SCOL expression
    ;
```

```
forInit
    : (varDeclaration | expression) (',' (varDeclaration | expression))*
    ;

breakStat
    : BREAK
    ;

contStat
    : CONT
    ;

retStat
    : RET expression?
    ;

parametr
    : expression                            #ExpConclusin
    | callMethod                            #CallMethodConlusin
    | expression op = (EQ | NE) expression #EqualityConclusion
    | expression op = (GT | GE) expression #MoreThenConclusion
    | expression op = (LT | LE) expression #LessThenConclusion
    ;

parametrs
    : parametr (pob = (AND | OR) parametr)*
    ;

parametrList
    : (varDeclaration (',' varDeclaration)*)?
    ;
```

```
expression
    : assign                                        #AssignExpr
    | callMethod                                    #CallMethodExpr
    | MINUS expression                              #UnaryMinusExpr
    | NOT expression                                #NotExpr
    | expression op = (MUL  | DIV | MOD) expression #MultiplicationExpr
    | expression op = (PLUS | MINUS) expression     #AdditiveExpr
    | varname                                       #VarnameExpr
    | literal                                       #LiteralExpr
    | OPAR expression CPAR                          #ParenExpr
    ;

expressionList
    : (expression (',' expression)*)?
    ;

literal
    : intLiteral
    | charLiteral
    | floatLiteral
    | boolLiteral
    ;

intLiteral
    : NUMBER
    ;

charLiteral
    : '\'' IDENTIFIER '\''
    ;

floatLiteral
    : NUMBER '.' NUMBER
    ;
```

```
boolLiteral
    : TRUE
    | FALSE
    ;

STRING
    : '"' (~["\r\n] | '""')* '"'
    ;

type
    : INT
    | CHAR
    | FLOAT
    | BOOL
    | 'string'
    ;

varname
    : IDENTIFIER (IDENTIFIER | NUMBER)*
    | STRING
    ;
```

```
OR    : '||' | 'or';
AND   : '&&' | 'and';
EQ    : '==';
NE    : '!=';
GT    : '>' ;
LT    : '<' ;
GE    : '>=';
LE    : '<=';
PLUS  : '+' ;
MINUS: '-' ;
MUL   : '*' ;
DIV   : '/' ;
MOD   : '%' ;
POW   : '^' ;
NOT   : '!' ;

SCOL   : ';';
ASSIGN : '=';
OPAR   : '(';
CPAR   : ')';
OBRACE : '{';
CBRACE : '}';

TRUE  : 'true'   ;
FALSE : 'false'  ;
```

```
IF    : 'if'      ;
ELSE  : 'else'    ;
WHILE : 'while'   ;

PRINT : 'print'   ;
FUNC  : 'function';
MAIN  : 'main()'  ;

BREAK : 'break'   ;
CONT  : 'continue';
RET   : 'return'  ;

INT    : 'int'  ;
CHAR   : 'char' ;
FLOAT  : 'float';
BOOL   : 'bool' ;

IDENTIFIER
    : [a-zA-Z] [a-zA-Z_0-9]*
    ;

NUMBER
    : [0-9]+ ([0-9])*
    ;

Newline
    : ( '\r' '\n'?
      | '\n'
      ) -> skip
    ;
// Пробелы и комменты
WS            : [ \t\r\n\u000C]+ -> skip;
COMMENTS      : '/*' .*? '*/'    -> skip;
LINE_COMM     : '//' ~[\r\n]*    -> skip;
```

Правила использования грамматики представлены ниже.

Присвоение

Блок присвоения представляется следующими правилами КС-грамматики:

```
varDeclaration
    : type varname (ASSIGN expression)?
    ;
```

```
assign
    : varname ASSIGN expression SCOL
    ;
```

Если переменная не определена в программа (т.е. ее нет в таблице символов), то указывается тип переменной (поддерживаются 2 типа: *int* – целочисленное и *float* – с плавающей точкой), имя и выражение, которое следует присвоить переменной.

Если переменная уже определена в программе (есть в таблице символов), то тип повторно не указывается, необходимо указать имя

переменной и выражение, которое ей присваивается.

Пример:

```
float c = 4.4;
c = 9.6;
int d = 5;
print(c);
```

Ветвление

Блок ветвления определяется следующим правилом КС-грамматики:

```
ifStatement
    : IF OPAR concList CPAR block (ELSE block)?
    ;
```

Блок обязательно содержит ключевое слово *if*, условие ветвления в скобках и тело в фигурных скобках. Опционально блок может содержать ветку *else*, которая выполняется в случае, если условие ветвления ложно.

Пример:

```
int aht = 0;
int ung = 1;
if (aht < 10) {
    ung = 2;
} else {
    ung = 0;
}
```

Циклы

Блок цикла определяется следующим правилом КС-грамматики:

```
whileStatement
    : WHILE OPAR concList CPAR block
    ;
```

```
block
    : OBRACE (statement)* CBRACE
    ;
```

Аналогично блоку ветвления, блок цикла содержит ключевое слово *while*, условие цикла в скобках и тело цикла в фигурных скобках.

Тело цикла будет выполняться до тех пор, пока условие цикла истинно. Кроме того, для данного блока имеются 2 специфичных оператора: *break* и *continue*. Оператор *break* позволяет досрочно выйти из тела цикла и

продолжить выполнение команд, идущих за ним, а оператор *continue* позволяет прервать текущую итерацию и начать выполнение тела цикла заново.

Пример:

```
int aht = 100;
while (aht < 10) {
    aht = aht + 1;
    if (aht == 5) {
        continue;
    }
    if (aht == 6) {
        break;
    }
}
```

Стандартный вывод

Встроенная функция *print(a)* позволяет выводить на стандартный вывод выражение *a*.

Пример:

```
int aht = 34;
print(aht);
print(aht + 1);
```

1.4 Построение генератора объектного кода

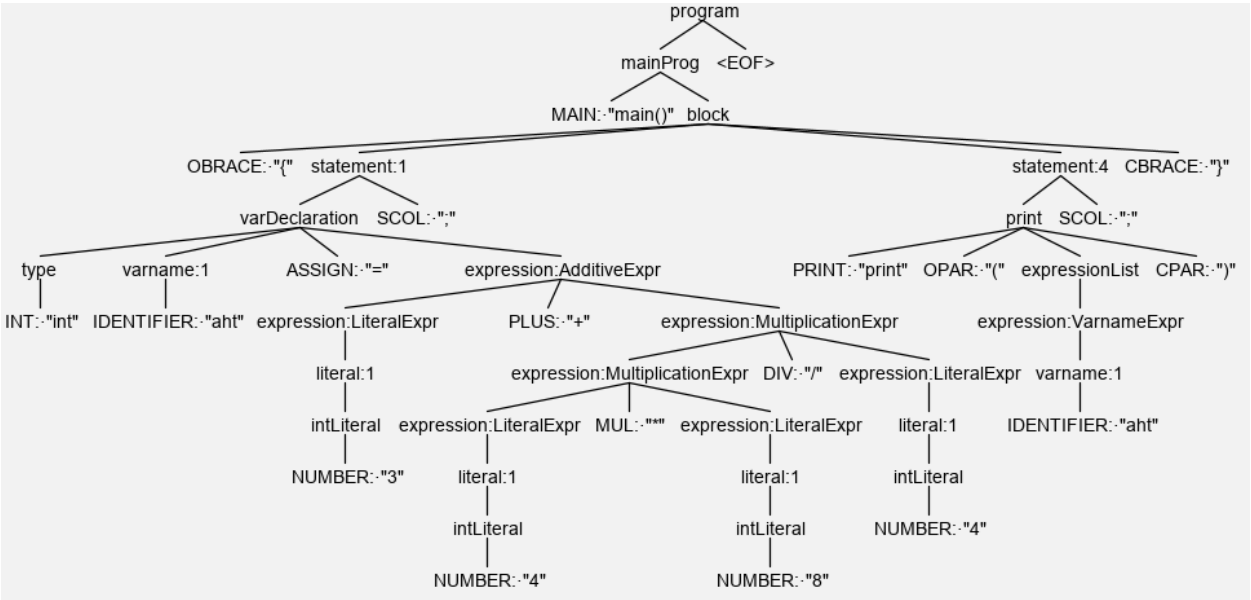
Генерация объектного кода выполняется объектами модуля AST, каждый из которых имеет метод генерации байт-кода (*class*).

Независимо от исходного кода в целевой код на языке jvm определяется функция стандартного вывода.

В модуле ASTопределены классы вершин синтаксического дерева. При синтаксическом анализе эти вершины добавляются к корневой вершине *main*.

Пример дерева разбора для программы:

```
int aht = 3 + 4 * 8 / 4;
print(aht)
```



2 ПРОВЕРКА НА СООТВЕТСТВИЕ ОСНОВНЫМ ТРЕБОВАНИЯМ

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

Далее приведено тестирование компилятора.

Проверка на игнорирование пробелов и индентацию:

```
main() {
            int aht= 3 + 4 * 8 / 4;
    print(aht
    );
}
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        int var1 = 3 + 4 * 8 / 4;
        System.out.print(var1);
    }
}
```

```
// class version 52.0 (52)
// access flags 0x1
public class cring {


  // access flags 0x1
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static main([Ljava/lang/String;)V
    BIPUSH 3
    BIPUSH 4
    BIPUSH 8
    IMUL
    BIPUSH 4
    IDIV
    IADD
    ISTORE 1
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ILOAD 1
    INVOKEVIRTUAL java/io/PrintStream.print (I)V
    RETURN
    MAXSTACK = 3
    MAXLOCALS = 2
}
```

Проверка многострочных комментариев

```
main() {
    int aht = 3 + 4 * 8 / 4;
    print(aht);
    /* It`s
    Multi
    Lines
    Comments*/
}
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        int var1 = 3 + 4 * 8 / 4;
        System.out.print(var1);
    }
}
```

Проверка оператора присваивания:

```
int aht = 35;
int brin = 45;
brin = aht + 10 * aht;
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        byte var1 = 35;
        boolean var2 = true;
        int var3 = var1 + 10 * var1;
    }
}
```

```
// class version 52.0 (52)
// access flags 0x1
public class cring {


  // access flags 0x1
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static main([Ljava/lang/String;)V
    BIPUSH 35
    ISTORE 1
    BIPUSH 45
    ISTORE 2
    ILOAD 1
    BIPUSH 10
    ILOAD 1
    IMUL
    IADD
    ISTORE 2
    RETURN
    MAXSTACK = 3
    MAXLOCALS = 3
}
```

Проверка условного оператора:

```
int aht = 35;
if (aht > 30) {
    aht = 0;
} else {
    aht = 1;
}
```

```java
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        byte var1 = 35;
        boolean var2;
        if (var1 < 30) {
            var2 = true;
        } else {
            var2 = false;
        }

    }
}
```

```
// class version 52.0 (52)
// access flags 0x1
public class cring {


  // access flags 0x1
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static main([Ljava/lang/String;)V
    BIPUSH 35
    ISTORE 1
    ILOAD 1
    BIPUSH 30
    IF_ICMPGE L0
    ICONST_0
    GOTO L1
   L0
   FRAME APPEND [I]
    ICONST_1
   L1
   FRAME SAME1 I
    IFNE L2
    BIPUSH 1
    ISTORE 1
    GOTO L3
    L2
    FRAME SAME
     BIPUSH 0
     ISTORE 1
    L3
    FRAME SAME
     RETURN
     MAXSTACK = 2
     MAXLOCALS = 2
}
```

Проверка оператора цикла:

```
main() {
    int aht = 35;
    while (aht > 10) {
        aht = aht / 5;
        print(aht);
        break;
    }
}
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        int var1 = 35;

        while(var1 >= 10) {
            var1 /= 5;
            System.out.print(var1);
        }

    }
}
```

```
// class version 52.0 (52)
// access flags 0x1
public class cring {


  // access flags 0x1
  public <init>()V
    ALOAD 0
    INVOKESPECIAL java/lang/Object.<init> ()V
    RETURN
    MAXSTACK = 1
    MAXLOCALS = 1

  // access flags 0x9
  public static main([Ljava/lang/String;)V
    BIPUSH 35
    ISTORE 1
   L0
   FRAME APPEND [I]
    ILOAD 1
    BIPUSH 10
    IF_ICMPGE L1
    ICONST_0
    GOTO L2
   L1
   FRAME SAME
    ICONST_1
   L2
   FRAME SAME1 I
    IFNE L3
    GOTO L4
```

```
   L3
   FRAME SAME
    ILOAD 1
    BIPUSH 5
    IDIV
    ISTORE 1
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ILOAD 1
    INVOKEVIRTUAL java/io/PrintStream.print (I)V
    GOTO L0
   L4
   FRAME SAME
    RETURN
    MAXSTACK = 2
    MAXLOCALS = 2
}
```

Проверка арифметики:

```
print(2 * 9 - (4 + 3 / 2) + 23);
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        System.out.print(2 * 9 - (4 + 3 / 2) + 23);
    }
}
```

Проверка типов данных:

```
int stroka = 34;
float fros = 45.7;
```

```
public class cring {
    public cring() {
    }

    public static void main(String[] var0) {
        boolean var1 = true;
        float var2 = 45.7F;
    }
}
```

Реквизиты к курсовой работе:
https://github.com/Allelua23/CompilerCring