Документация по проекту «Компилятор языка Паскаль»

Подготовили Елфимова Екатерина, Мещеряков Илья и Стребкова Ольга

Синтаксический анализ

Компилятор языка Паскаль, написанный на языке python с испоьзованием библиотеки pyparsing.

На даммный момент в проекте реализованно построение ast-дерева для кода на языке Паскаль.

Структура проекта:

Класс PascalGrammar который отвечает за описание грамматики языка Паскаль. В нем присутствует метод _make_parser грамматику языка и правила ее разбора. Здесь мы возспользовались модулем pyparsing для создания грамматики, используя предоставляемую модулем библиотеку классов для построения грамматики непосредственно в коде Python.

main.py используется для вызова функции.

В файле nodes.py описаны узлы для постороения ast дерева:

AstNode - класс предок для всех последующих классов, определяет для узлов такие свойства как получние потомков узла, и рисование узла

LiteralNode - узел для описания литерала

IdentNode - узел для описания идентификатора

ArrayIdentNode - узел для описания элементов массива

BinOpNode - узел для описания бинарной операции

StmtNode - узел для описания выражения

IdentListNode - узел для описания списка идентификаторов

TypeSpecNode - узел для описания типа

VarDeclNode - узел для описания объявления переменной

ArrayDeclNode - узел для описания объявления массива

VarsDeclNode - узел для описания объявлений переменных

CallNode - узел для описания вызова функции или процедуры

AssignNode - узел для описания присваивания переменной значения

IfNode - узел для описания условного оператора if

WhileNode - узел для описания цикла while

ForNode - узел для описания цикла for

StmtListNode - узел для описания списка выражений

BodyNode - узел для описания тела (внутренности между begin и end) содержащий список выражений

ParamsNode - узел для описания параметров функции/процедуры

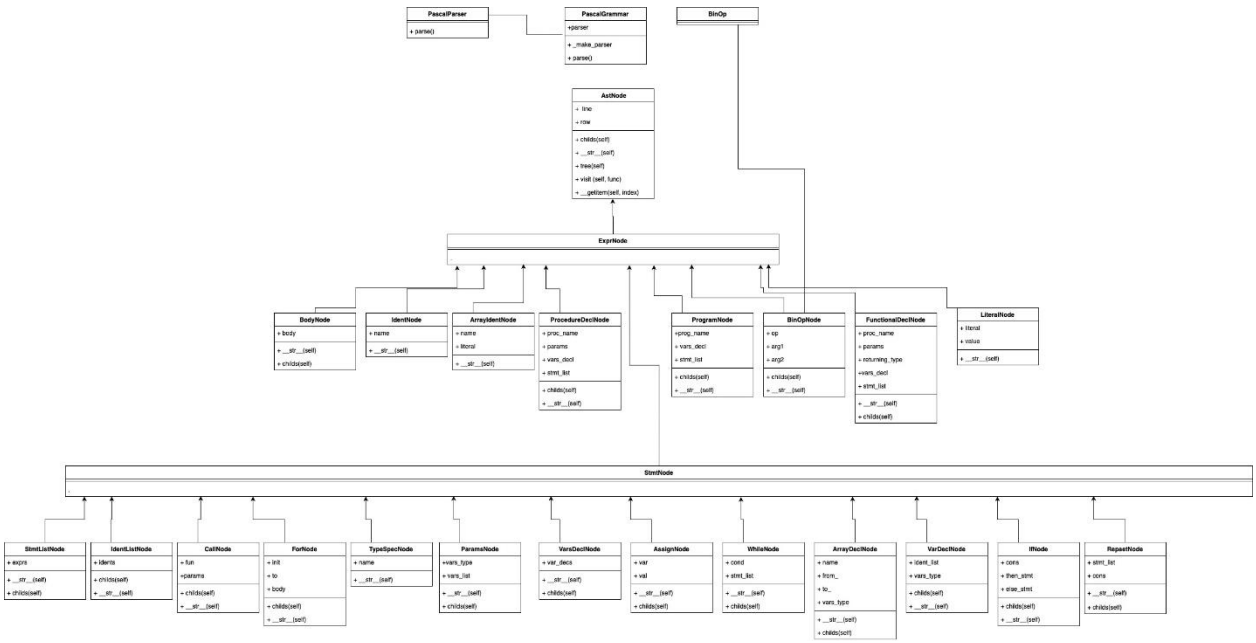
ProgramNode - узел для описания программы

ProcedureDeclNode - узел для описания процедуры

FunctionDeclNode- узел для описания функции

Так же в коде проекта присутствуют комментарии с пояснениями по ходу выполнения программы.

Диаграмма классов, описывающая структуру проекта:



Примеры вход/выход:

```
вход 1:
Program prog1;
var
```

```
    j: integer;
BEGIN
    j:=9;
END.
```

выход 1:

```
Program
├ prog1
├ var
│ └ var_dec
│   ├ idents
│   │ └ j
│   └ integer
└ Body
  └ ...
    └ :=
      ├ j
      └ 9 (int)
```

вход 2:

```
Program prog1;
var
    k, d: integer;
    j: char;
    g, c: array [1 .. 100] of integer;
BEGIN
    while (i>3) do
        a:=k+2;
    a:=j mod k;
    while (i>3) do
    begin
        a:=k+2;
        a:=j mod k;
    end;
END.
```

выход 2:

```
Program
├ prog1
├ var
│ ├ var_dec
│ │ ├ idents
│ │ │ ├ k
│ │ │ └ d
│ │ └ integer
```

```
| ├ var_dec
| | ├ idents
| | | └ j
| | └ char
| └ arr_decl
|   ├ integer
|   ├ idents
|   | ├ g
|   | └ c
|   ├ 1 (int)
|   └ 100 (int)
└ Body
  └ ...
    ├ while
    | ├ >
    | | ├ i
    | | └ 3 (int)
    | └ :=
    |   ├ a
    |   └ +
    |     ├ k
    |     └ 2 (int)
    ├ :=
    | ├ a
    | └ mod
    |   ├ j
    |   └ k
    └ while
      ├ >
      | ├ i
      | └ 3 (int)
      └ ...
        ├ :=
        | ├ a
        | └ +
        |   ├ k
        |   └ 2 (int)
        └ :=
          ├ a
          └ mod
            ├ j
            └ k
```

вход 3:

```
Program prog1;
var
    k, d: integer;
```

```
    function t(j:integer; k: char):integer;
    var
        d: integer;
    begin
        for (i:=2 to 0 ) do
            g:=0;
        s:=0;
        if (k>2) then
        begin
            f:=9;
            h:=f;
        end;
        else
            v:=3;
    end;
BEGIN
    t(1,3);
END.
```

выход 3:

```
Program
├ prog1
├ var
│ ├ var_dec
│ │ ├ idents
│ │ │ ├ k
│ │ │ └ d
│ │ └ integer
│ └ function
│   ├ t
│   ├ params
│   │ ├ integer
│   │ ├ k
│   │ ├ char
│   │ └ j
│   ├ integer
│   ├ var
│   │ └ var_dec
│   │   ├ idents
│   │   │ └ d
│   │   └ integer
│   └ Body
│     └ ...
│       ├ for
│       │ ├ :=
│       │ │ ├ i
│       │ │ └ 2 (int)
```

```
│       ├ 0 (int)
│       └ :=
│         ├ g
│         └ 0 (int)
│       ├ :=
│       │ ├ s
│       │ └ 0 (int)
│       └ if
│         ├ >
│         │ ├ k
│         │ └ 2 (int)
│         ├ ...
│         │ ├ :=
│         │ │ ├ f
│         │ │ └ 9 (int)
│         │ └ :=
│         │   ├ h
│         │   └ f
│         └ :=
│           ├ v
│           └ 3 (int)
└ Body
  └ ...
    └ call
      ├ t
      ├ 1 (int)
      └ 3 (int)
```

вход 4:

```
Program prog1;
var
    procedure t(j:integer; k: char);
        var
            d: integer;
            procedure j(d: integer);
            var
                h:integer;
            begin
                d:=h;
            end;
        begin


            a:=78;
        end;
BEGIN
    if (k>2 or l<0) then
```

```
    begin
        f:=9;
        if (k>2 and h>=0) then
            f:=4;
    end;
END.
```

выход 4:

```
Program
├ prog1
├ var
│ └ procedure
│   ├ t
│   ├ params
│   │ ├ integer
│   │ ├ k
│   │ ├ char
│   │ └ j
│   ├ var
│   │ ├ var_dec
│   │ │ ├ idents
│   │ │ │ └ d
│   │ │ └ integer
│   │ └ procedure
│   │   ├ j
│   │   ├ params
│   │   │ ├ integer
│   │   │ └ d
│   │   ├ var
│   │   │ └ var_dec
│   │   │   ├ idents
│   │   │   │ └ h
│   │   │   └ integer
│   │   └ Body
│   │     └ ...
│   │       └ :=
│   │         ├ d
│   │         └ h
│   └ Body
│     └ ...
│       └ :=
│         ├ a
│         └ 78 (int)
└ Body
  └ ...
    └ if
      ├ or
```

```
      | ├ >
      | | ├ k
      | | └ 2 (int)
      | └ <
      |   ├ l
      |   └ 0 (int)
      └ ...
        ├ :=
        | ├ f
        | └ 9 (int)
        └ if
          ├ and
          | ├ >
          | | ├ k
          | | └ 2 (int)
          | └ >=
          |   ├ h
          |   └ 0 (int)
          └ :=
            ├ f
            └ 4 (int)
```

вход 5:

```
Program t;
var
    k, d: integer;
    j: char;
    g, c: array [1 .. 100] of integer;
function t(j:integer; k: char):integer;
var
    d: integer;
begin
a:= 0;
end;
procedure t;
var
    d: integer;
begin
a:=78;
end;
    g: integer;
BEGIN
g[0]:=10;
g[1]:=c[0];
writeln(a, 3, "df", 7+9);
for ( i:=2 to 10  ) do
begin
```

```
    k:=2 mod 3;
    l:=j div 4;
    l:=i+8;
    k:=0;
end;
while (i>3) do
    a:=k+2;
for (i:=2 to 6 ) do
    g:=0;
    s:=0;
if (k>2 and j>=2) then
    begin
    f:=9;
    end;
else
    v:=3;
END.
```

выход 5:

```
Program
├ t
├ var
│ ├ var_dec
│ │ ├ idents
│ │ │ ├ k
│ │ │ └ d
│ │ └ integer
│ ├ var_dec
│ │ ├ idents
│ │ │ └ j
│ │ └ char
│ ├ arr_decl
│ │ ├ integer
│ │ ├ idents
│ │ │ ├ g
│ │ │ └ c
│ │ ├ 1 (int)
│ │ └ 100 (int)
│ ├ function
│ │ ├ t
│ │ ├ params
│ │ │ ├ integer
│ │ │ ├ k
│ │ │ ├ char
│ │ │ └ j
│ │ ├ integer
│ │ ├ var
```

```
| | | └ var_dec
| | |   ├ idents
| | |   | └ d
| | |   └ integer
| | └ Body
| |   └ ...
| |     └ :=
| |       ├ a
| |       └ 0 (int)
| ├ procedure
| | ├ t
| | ├ ...
| | ├ var
| | | └ var_dec
| | |   ├ idents
| | |   | └ d
| | |   └ integer
| | └ Body
| |   └ ...
| |     └ :=
| |       ├ a
| |       └ 78 (int)
| └ var_dec
|   ├ idents
|   | └ g
|   └ integer
└ Body
  └ ...
    ├ :=
    | ├ g [0 (int)]
    | └ 10 (int)
    ├ :=
    | ├ g [1 (int)]
    | └ c [0 (int)]
    ├ call
    | ├ writeln
    | ├ a
    | ├ 3 (int)
    | ├ "df" (str)
    | └ +
    |   ├ 7 (int)
    |   └ 9 (int)
    ├ for
    | ├ :=
    | | ├ i
    | | └ 2 (int)
    | ├ 10 (int)
    | └ ...
```

```
│   ├ :=
│   │ ├ k
│   │ └ mod
│   │   ├ 2 (int)
│   │   └ 3 (int)
│   ├ :=
│   │ ├ l
│   │ └ div
│   │   ├ j
│   │   └ 4 (int)
│   ├ :=
│   │ ├ l
│   │ └ +
│   │   ├ i
│   │   └ 8 (int)
│   └ :=
│     ├ k
│     └ 0 (int)
├ while
│ ├ >
│ │ ├ i
│ │ └ 3 (int)
│ └ :=
│   ├ a
│   └ +
│     ├ k
│     └ 2 (int)
├ for
│ ├ :=
│ │ ├ i
│ │ └ 2 (int)
│ ├ 6 (int)
│ └ :=
│   ├ g
│   └ 0 (int)
├ :=
│ ├ s
│ └ 0 (int)
└ if
  ├ and
  │ ├ >
  │ │ ├ k
  │ │ └ 2 (int)
  │ └ >=
  │   ├ j
  │   └ 2 (int)
  ├ ...
  │ └ :=
```

```
│   ├ f
│   └ 9 (int)
└ :=
  ├ v
  └ 3 (int)
```

Документация по проекту «Компилятор языка Паскаль».
Подготовили Елфимова Екатерина, Мещеряков Илья и Стребкова Ольга

Семантический анализ

Для проведения семантического анализа были в программу были добавлены файлы semantic.py и symbols.py для определения логики необходимой при проведении семантического анализа.

Рассмотрим файл symbols.py. В данном файле определены классы Symbol и его наследники - BuiltinTypeSymbol, BuiltinFunction, ArraySymbol, VarSymbol, BlockSymbol, ProcedureSymbol, FunctionSymbol. Данные классы описывают структуру соответствующих символов, которые будут записываться в символьную таблицу, необходимую для проверки семантики программы. Классы BuiltinTypeSymbol и BuiltinFunction описывают встроенные типы и функции соответственно.

Рассмотрим файл semantic.py.

Файл содержит класс ScopedSymbolTable, который описывает символьную таблицу. Данный класс содержит набор символов данной таблицы, имя таблицы, уровень вложенности данной таблицы и ссылку на вышестоящую таблицу. В классе определены методы для записи в таблицу нового символа (define) и для поиска в таблице символа (lookup).

Файл содержит класс NodeVisitor, необходимый для совершения обхода дерева и проверки семантики. Его потомком является класс SemanticAnalyzer, в котором присутствует описание методов для посещения конкретных типов узлов:

- visit_BinOpNode
- visit_IdentNode
- visit_LiteralNode
- visit_ProgramNode
- visit_VarsDeclNode

- visit_VarDeclNode
- visit_ArrayDeclNode
- visit_ArrayIdentNode
- visit_BodyNode
- visit_StmtListNode
- visit_AssignNode
- visit_ProcedureDeclNode
- visit_FunctionDeclNode
- visit_CallNode
- visit_IfNode
- visit_WhileNode
- visit_ForNode

Для проверки семантики программы в классе присутствует поле current_scope, в котором хранится текущий уровень символьной таблицы. Благодаря такому построению структуры у нас есть возможность создавать несколько вложенных областей (в языке Pascal это вложенные процедуры и функции).

Кроме того, класс реализует логику для проверки типов для бинарных операций. Для этого в классе имеется поле BinOpArgTypes, в котором описано соответствие бинарных операций и допустимых для них типов.

Таким образом, в рамках семантического анализа нами была реализована логика по проверки совместимости типов, а также проверки объявлений переменных, функций и процедур.

Пример построения символьной таблицы.

```
Program t;
var
g : integer;
c : boolean;
a : char;
d,n: array [1 .. 100] of integer;
h : array [1 .. 7] of boolean;
```

```
function Alpha(a,b: integer, g:char):integer;
var y: integer;
begin
g:='c';
c:=True;
ReadLn(g);
if (c) then
c:=False;
for (a:=2 to 6) do
begin
end;
end;
BEGIN
h[1]:=d[2]=1;
g:=g;
END.
```

```
Program
├ t
├ var
│ ├ var_dec
│ │ ├ idents
│ │ └ g
│ │ └ integer
│ ├ var_dec
│ │ ├ idents
│ │ └ c
│ │ └ boolean
│ ├ var_dec
│ │ ├ idents
│ │ └ a
│ │ └ char
│ ├ arr_decl
│ │ ├ integer
│ │ ├ idents
│ │ │ ├ d
│ │ │ └ n
│ │ ├ 1 (int)
│ │ └ 100 (int)
│ ├ arr_decl
│ │ ├ boolean
│ │ ├ idents
│ │ │ └ h
│ │ ├ 1 (int)
│ │ └ 7 (int)
│ └ function
│   ├ Alpha
│   ├ params
│   │ ├ var_dec
│   │ │ ├ idents
│   │ │ │ ├ a
│   │ │ │ └ b
│   │ │ └ integer
│   │ └ var_dec
│   │   ├ idents
│   │   │ └ g
│   │   └ char
│   ├ integer
│   ├ var
│   │ └ var_dec
│   │   ├ idents
│   │   │ └ y
│   │   └ integer
```

```
|    └ Body
|      └ ...
|        ├ :=
|        │ ├ g
|        │ └ 'c' (str)
|        ├ :=
|        │ ├ c
|        │ └ True (bool)
|        ├ call
|        │ ├ ReadLn
|        │ └ g
|        ├ if
|        │ ├ c
|        │ └ :=
|        │   ├ c
|        │   └ False (bool)
|        └ for
|          ├ :=
|          │ ├ a
|          │ └ 2 (int)
|          ├ 6 (int)
|          └ ...
└ Body
  └ ...
    ├ :=
    │ ├ h [1 (int)]
    │ └ =
    │   ├ d [2 (int)]
    │   └ 1 (int)
    └ :=
      ├ g
      └ g
```

```
SCOPE (SCOPED SYMBOL TABLE)
===========================
Scope name     : global
Scope level    : 1
Enclosing scope: None
Scope (Scoped symbol table) contents
------------------------------------
integer: 'integer'
   char: 'char'
boolean: 'boolean'
   Read: 'Read'
 ReadLn: 'ReadLn'
  Write: 'Write'
WriteLn: 'WriteLn'
      g: '<g:integer>'
      c: '<c:boolean>'
      a: '<a:char>'
      d: '<d:integer[1 .. 100]>'
      n: '<n:integer[1 .. 100]>'
      h: '<h:boolean[1 .. 7]>'
  Alpha: "<FunctionSymbol(name=Alpha, parameters=['a: integer', 'b: integer',
'g: char']): integer>"


SCOPE (SCOPED SYMBOL TABLE)
===========================
Scope name     : Alpha
Scope level    : 2
Enclosing scope: global
```

```
Scope (Scoped symbol table) contents
-----------------------------------
integer: 'integer'
   char: 'char'
boolean: 'boolean'
   Read: 'Read'
 ReadLn: 'ReadLn'
  Write: 'Write'
WriteLn: 'WriteLn'
      a: '<a:integer>'
      b: '<b:integer>'
      g: '<g:char>'
```

Кодогенерация (В java byte code)

Выполнена Елфимовой Екатериной

Для реализации кодогенерации в программе добавлены инструкции, собирающие код по исходной поданной программе на языке Pascal. Программа генерирует ассемблер для виртуальной машины Java (Jasmin). После генерации кода необходимо с помощью сборщика Jasmin скомпилировать ассемблер сгенерированный программой в java byte code. Для этого необходимо выполнить:

java -jar jasmin.jar target_file.j

После выполнения данной команды будет target_file.j будет скомпилирован в файл .class, который может быть исполнен JVM.

Для исполнения полученного файла необходимо выполнить команду:

java target_file

Для кодогенерации ассемблера были изучена структура такой программы и выделены некоторые инструкции, которые были использованы при генерации кода:

- *ldc 0* – кладет на стек значение 0
- *iload_1* – берет из локальной переменной с адресом 1 значение и кладет его на стек (префикс i – целое число)
- *istore_1* – снимает со стека значение и сохраняет его в локальную переменную с адресом 1 (префикс i – целое число)

Бинарные операции, которые снимают со стека два значения производят соответствующую операцию и возвращают результат на стек:

- *iadd*
- *isub*
- *imul*
- *idiv*
- *iand*

- *ior*

Префикс i обозначает что операция будет выполнена над целыми числами.

Ключевое слово Program из языка Pascal интерпретируется как класс. Для этого прописывается:

.class public ClassName

.super java/lang/Object

для описания класса и вызова конструктора суперкласса.

Для описания методов используются конструкции вида:

.method public static MethName(II)V

.limit stack 10

.limit locals 10

//some code

return

.end method

Здесь приведено описание процедуры. Символы II говорят о том, что метод будет принимать два параметра целочисленного типа. Символ V описывает тип возвращаемого значения, в данном случае – void. Limit stack и limit locals заявляют выделяемый размер стека и максимальное количество локальных переменных.

Для вызова методов используется инструкция:

invokestatic ClassName/MethName(II)V

Для вызова библиотечных функций:

- *getstatic java/lang/System/in Ljava/io/InputStream;*
  invokevirtual java/io/InputStream/read()I (для ввода)
- *invokevirtual java/io/PrintStream/println(I)V*
  getstatic java/lang/System/out Ljava/io/PrintStream; (для вывода)

Глобальные переменные программы Pascal интерпретируются как

статические поля класса. Их объявление выглядит так:

.field public static field1 I

символ I указывает тип поля, в данном случае поле целочисленное.

Для операций с данным полем используются инструкции:

- *putstatic ClassName/field1 I* – снимает значение со стека и записывает его в поле
- *getstatic ClassName/field1 I* – кладет значение из поля в стек

Для описания условного оператора и цикла были использованы такие конструкции как label и инструкция goto. Label представляет из себя инструкцию к которой можно вернуться с помощью инструкции *goto some_label*. Кроме того использовались инструкции для проверки условия:

- if_icmpeq
- if_icmpne
- if_icmge
- if_icmle
- if_icmgt
- if_icmlt

Массивы описываются с помощью следующей инструкции:

.field public static mass [I

После чего они должны быть проинициализированы, что исполняется с помощью инструкций:

```
ldc 100                 //100-число эл-ов массива
newarray int            //инициализация
putstatic ClassName/mass [I     //сохранение в соответствующее поле
```

Тело программы Pascal исполняется в методе *main*, который является точкой входа в программу.

Примеры генерации:

Вход 1:

```
Program pr1;
            var
            g, b : integer;
            d, n: array [1 .. 100] of integer;

            function Alpha(a: integer):integer;
            var y: integer;
            begin
            a:=a*10;
            end;

            BEGIN
            g:=0;
            while (g<10) do
            begin
                d[1]:=g;
                Write(d[1]);
                g:=g+1;
            end;

            g:=Alpha(4);
            WriteLn(g);

            END.
```

Выход 1:

```
.class public pr1
.super java/lang/Object
.field public static g I
.field public static b I
.field public static d [I
.field public static n [I
.method public static Alpha(I)I
.limit stack 100
.limit locals 100
iload_0
ldc 10
imul
istore_0
iload_0
ireturn
.end method
.method                   public static main([Ljava/lang/String;)V
.limit stack          100
.limit locals         100
ldc 100
newarray int
putstatic pr1/d [I
ldc 100
newarray int
putstatic pr1/n [I

ldc 0
putstatic pr1/g I
while_0:
```

```
getstatic pr1/g I
ldc 10
swap
if_icmplt done_0
getstatic pr1/d [I
ldc 1
getstatic pr1/g I
iastore
getstatic              java/lang/System/out Ljava/io/PrintStream;
getstatic pr1/d [I
ldc 1
iaload
invokevirtual          java/io/PrintStream/println(I)V
getstatic pr1/g I
ldc 1
iadd
putstatic pr1/g I
goto while_0
done_0:
ldc 4
invokestatic pr1/Alpha(I)I
putstatic pr1/g I
getstatic              java/lang/System/out Ljava/io/PrintStream;
getstatic pr1/g I
invokevirtual          java/io/PrintStream/println(I)V
   return
.end method
```

Вход 2:

```
Program pr2;
        var
        g, b : integer;
        d,n: array [1 .. 100] of integer;

        BEGIN
        g:=4*4/2+2-5;
        b:=0;
        WriteLn(g);
        WriteLn(b);
        if (b>g) then
            b:=0;
        else
            b:=1;
        WriteLn(b);
        while (b<g) do
        begin
            b:=b+1;
            WriteLn(b);
        end;
        END.
```

Выход 2:

```
.class public pr2
.super java/lang/Object
.field public static g I
.field public static b I
.field public static d [I
.field public static n [I
.method                  public static main([Ljava/lang/String;)V
```

```
.limit stack          100
.limit locals         100
ldc 100
newarray int
putstatic pr2/d [I
ldc 100
newarray int
putstatic pr2/n [I

ldc 4
ldc 4
imul
ldc 2
idiv
ldc 2
iadd
ldc 5
isub
putstatic pr2/g I
ldc 0
putstatic pr2/b I
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr2/g I
invokevirtual         java/io/PrintStream/println(I)V
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr2/b I
invokevirtual         java/io/PrintStream/println(I)V
getstatic pr2/b I
getstatic pr2/g I
swap
if_icmpgt else_0
ldc 0
putstatic pr2/b I
goto endif_0
else_0:
ldc 1
putstatic pr2/b I
endif_0:
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr2/b I
invokevirtual         java/io/PrintStream/println(I)V
while_1:
getstatic pr2/b I
getstatic pr2/g I
swap
if_icmplt done_1
getstatic pr2/b I
ldc 1
iadd
putstatic pr2/b I
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr2/b I
invokevirtual         java/io/PrintStream/println(I)V
goto while_1
done_1:
   return
.end method
```

Вход 3:

```
Program pr3;
var x, y, i: integer;
function Alpha(a,b: integer):integer;
```

```
    var y: integer;
begin
a:=b+a;
end;
BEGIN
x:=1;
y:=x;
y:=(y+45)*(1+1);
x:=3+y-1;
Write(y);
y:=Alpha(3,5);
while (y<100) do
begin
    Write(y);
    y:=y+1;
    Write(y);
end;
if (y>1) then
    x:=1;
else
    x:=2;
END.
```

Выход 3:

```
.class public pr3
.super java/lang/Object
.field public static x I
.field public static y I
.field public static i I
.method public static Alpha(II)I
.limit stack 100
.limit locals 100
iload_1
iload_0
iadd
istore_0
iload_0
ireturn
.end method
.method                   public static main([Ljava/lang/String;)V
.limit stack          100
.limit locals         100

ldc 1
putstatic pr3/x I
getstatic pr3/x I
putstatic pr3/y I
getstatic pr3/y I
ldc 45
iadd
ldc 1
ldc 1
iadd
imul
putstatic pr3/y I
ldc 3
getstatic pr3/y I
iadd
ldc 1
isub
putstatic pr3/x I
getstatic             java/lang/System/out Ljava/io/PrintStream;
```

```
getstatic pr3/y I
invokevirtual         java/io/PrintStream/println(I)V
ldc 3
ldc 5
invokestatic pr3/Alpha(II)I
putstatic pr3/y I
while_0:
getstatic pr3/y I
ldc 100
swap
if_icmplt done_0
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr3/y I
invokevirtual         java/io/PrintStream/println(I)V
getstatic pr3/y I
ldc 1
iadd
putstatic pr3/y I
getstatic             java/lang/System/out Ljava/io/PrintStream;
getstatic pr3/y I
invokevirtual         java/io/PrintStream/println(I)V
goto while_0
done_0:
getstatic pr3/y I
ldc 1
swap
if_icmpgt else_1
ldc 1
putstatic pr3/x I
goto endif_1
else_1:
ldc 2
putstatic pr3/x I
endif_1:
   return
.end method
```