Министерство науки и высшего образования Российской Федерации
ФГБОУ ВО «Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»
(ВлГУ)

Кафедра «Информатики и защиты информации»

Курсовая работа на тему:

Разработка компилятора подмножества процедурного языка в ассемблер

Специальность: 10.03.01 – Информационная безопасность

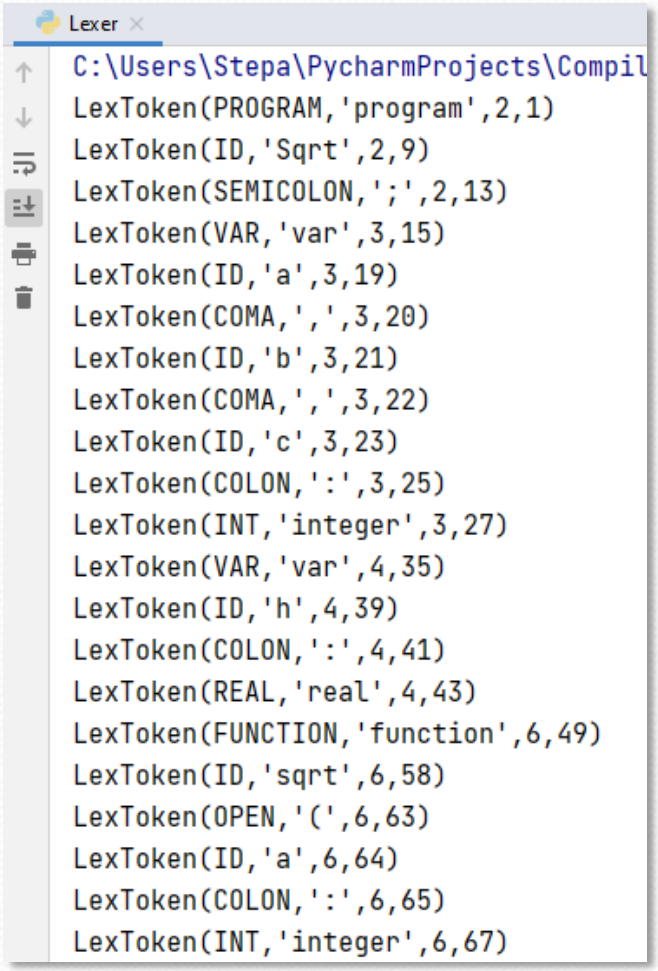
КОРОЧКИН Степан Владимирович,
ст. гр. ИБ-118

г. Владимир 2021

Компилятор – это программа, которая переводит текст, написанный на языке программирования, в набор машинных кодов.

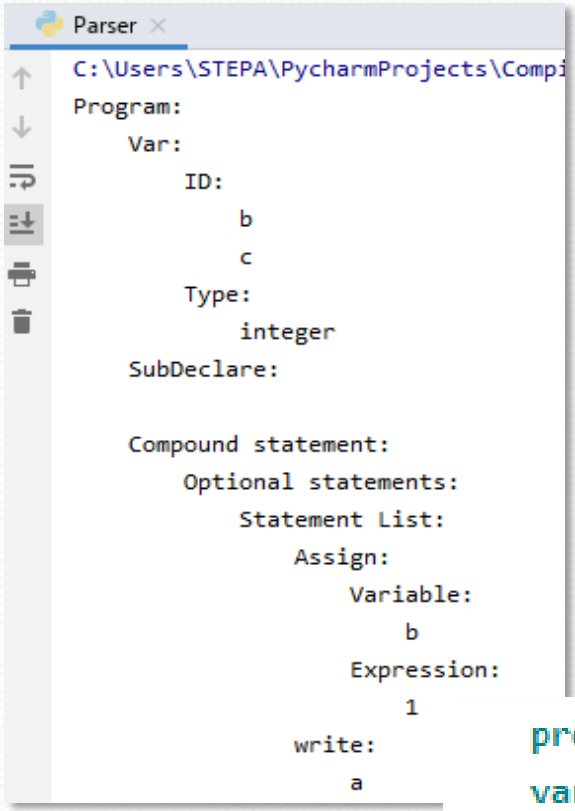
Для реализации компилятора подмножества языка Pascal был выбран язык программирования Python с использованием библиотек ply и llvmlite

Лексический анализатор



Лексический анализатор был реализован при помощи библиотеки ply. Для его работы необходим набор зарезервированных слов и набора регулярных выражений, при помощи которых лексер будет разбивать исходный код программы на токены.

Синтаксический анализатор




```
program Hello;
var b,c : integer
begin
    b := 1 ;
    write(a)
end.
```

Синтаксический анализатор был реализован при помощи библиотеки ply. На вход синтаксическому анализатору подается исходный код, поделенный на токены лексером. Результатом работы синтаксического анализатора является синтаксическое дерево разбора.

Генератор промежуточного кода

```
program Hello;
var a,b,c,d : integer
var h : real
  begin
  while ( a < 20 ) do begin
  a := a + 1;
  write(a)


  end
end.
```

```
::::::::::::::::::::::::::::::::::::
   __init   | void []
      ('global_int', 'a')
      ('literal_int', 0, '__int_0')
      ('store_int', '__int_0', 'a')
      ('global_int', 'b')
      ('literal_int', 0, '__int_1')
      ('store_int', '__int_1', 'b')
      ('global_int', 'c')
      ('literal_int', 0, '__int_2')
      ('store_int', '__int_2', 'c')
      ('global_int', 'd')
      ('literal_int', 0, '__int_3')
      ('store_int', '__int_3', 'd')
      ('global_float', 'h')
      ('literal_float', 0.0, '__float_0')
      ('store_float', '__float_0', 'h')
      ('return_void ',)
   main   | void []
      WhileBlock
         WHILEcondition
            ('load_int', 'a', '__int_4')
            ('literal_int', 20, '__int_5')
            ('lt_int', '__int_4', '__int_5', '__bool_0')
         WHILEbody
            default
               ('load_int', 'a', '__int_6')
               ('literal_int', 1, '__int_7')
               ('add_int', '__int_6', '__int_7', '__int_8')
               ('store_int', '__int_8', 'a')
               ('load_int', 'a', '__int_9')
               ('print_int', '__int_9')

Process finished with exit code 0
```

Чтобы сгенерировать промежуточный код мною был реализован алгоритм обхода синтаксического дерева, который встречая определенные узлы дерева создавал для них инструкции. Для хранения промежуточного кода был реализован класс Block, который хранит в себе списки инструкций, название блока инструкций, его возвращаемый тип и параметры (если блок используется для хранения промежуточного кода функции) и может состоять из других вложенных в него блоков.

Генератор объектного кода

```
----------------------LLVM CODE-------------------
; ModuleID = "module"
target triple = "i686-pc-windows-msvc"
target datalayout = ""

declare i32 @"printf"(i8* %".1", ...)

define void @"__init"()
{
entry:
  store i32 0, i32* @"a"
  store i32 0, i32* @"b"
  store i32 0, i32* @"c"
  store double              0x0, double* @"h"
  br label %"exit"
exit:
  ret void
}

@"a" = global i32 0
@"b" = global i32 0
@"c" = global i32 0
@"h" = global double              0x0
define void @"main"()
{
entry:
  store i32 1, i32* @"a"
  br label %"whiletest"
exit:
  ret void
```

Для реализации трансляции кода в целевую машину мною была использована библиотека llvmlite. Получая промежуточный код из генератора, транслятор обрабатывает их и создает файл с объектным кодом с расширением ll и выполняет его.

Пример работы компилятора

```
program Fibonacci;
var a,b,c : integer
var h : real

procedure Fib (a: integer);
   var num,c,d,e : integer
   begin
       c := 1;
       d := 1;
       while ( num < a ) do
       begin
            write(d);
            e := c;
            c := c + d;
            d := e;
            num := num + 1
       end
   end;

begin
    write("the first a numbers of the Fibonacci sequence");
    a := 10;
    Fib(a)

end.
```

```
target triple = "x86_64-pc-windows-msvc"
target datalayout = ""

declare i32 @"printf"(i8* %".1", ...)

define void @"__init"()
{
entry:
  store i32 0, i32* @"a"
  store i32 0, i32* @"b"
  store i32 0, i32* @"c"
  store double              0x0, double* @"h"
  br label %"exit"
exit:
  ret void
}

@"a" = global i32 0
@"b" = global i32 0
@"c" = global i32 0
@"h" = global double              0x0
define void @"Fib"(i32 %".1")
{
entry:
  %"a" = alloca i32
  store i32 %".1", i32* %"a"
  %"num" = alloca i32
  store i32 0, i32* %"num"
  %"c" = alloca i32
  store i32 0, i32* %"c"
  %"d" = alloca i32
  store i32 0, i32* %"d"
  %"e" = alloca i32
  store i32 0, i32* %"e"
  store i32 1, i32* %"c"
  store i32 1, i32* %"d"
```

```
--------------------RUNNING THE PROGRAMM--------------------
the first a numbers of the Fibonacci sequence
1
1
2
3
5
8
13
21
34
55
```

Пример работы компилятора

```
program Sqrt;
var a,b,c : integer
var h : real

function sqrt (a: integer) : real;
   var b,c : integer
   var d,e,f,g : real
   begin
       c := b * b;
       while ( c <= a ) do
       begin
            b := b + 1;
            c := b * b
       end;
       b := b - 1;
       d := a - b * b;
       e := a * 2;
       f := d / e;
       g := b + f;
       d := f * f;
       e := 2 * g;
       f := g - d / e;
       sqrt := f

   end;

begin
    h := sqrt(9);
    write(h)

end.
```

```
@"a" = global i32 0
@"b" = global i32 0
@"c" = global i32 0
@"h" = global double              0x0
define double @"sqrt"(i32 %".1")
{
entry:
  %"return" = alloca double
  %"a" = alloca i32
  store i32 %".1", i32* %"a"
  %"sqrt" = alloca double
  store double              0x0, double* %"sqrt"
  %"b" = alloca i32
  store i32 0, i32* %"b"
  %"c" = alloca i32
  store i32 0, i32* %"c"
  %"d" = alloca double
  store double              0x0, double* %"d"
  %"e" = alloca double
  store double              0x0, double* %"e"
  %"f" = alloca double
  store double              0x0, double* %"f"
  %"g" = alloca double
  store double              0x0, double* %"g"
  %"__int_5" = load i32, i32* %"b"
  %"__int_6" = load i32, i32* %"b"
  %"__int_7" = mul i32 %"__int_5", %"__int_6"
  store i32 %"__int_7", i32* %"c"
  br label %"whiletest"
exit:
  %".28" = load double, double* %"return"
  ret double %".28"
whiletest:
  %"__int_8" = load i32, i32* %"c"
  %"__int_9" = load i32, i32* %"a"
  %"__bool_0" = icmp sle i32 %"__int_8", %"__int_9"
```

```
----------------------RUNNING THE PROGRAMM--------------------
3.000000
```

Спасибо за внимание!

КОРОЧКИН Степан Владимирович,
ст. гр. ИБ-118
stepakorochkin@yandex.ru