

методическое пособие

ОБРАБОТКА ТЕКСТОВЫХ ФОРМАТОВ ДАННЫХ И РЕАЛИЗАЦИЯ КОМПЬЮТЕРНЫХ ЯЗЫКОВ на Flex/Bison/C++/LLVM Java/C#/ANTLR

GitHub: <https://github.com/ponyatov/lexman>

© <dponyatov@gmail.com>

27 декабря 2015 г.

Оглавление

Применение	2
Необходимое программное обеспечение	2
1 Структура компилятора	4
1.1 Термины	4
1.2 Структура типового компилятора	6
1.3 Архитектура LLVM	7
2 Типичная структура проекта	8
2.1 README.md	9
2.2 Makefile	9
2.3 bat.bat	11
2.4 rc.rc	12
3 Лексер и утилита flex	13
3.1 Регулярные выражения	13
3.2 Примеры самостоятельного применения	14
3.2.1 Rij2D: загрузка файла числовых данных	14

Литература

20

Основы компиляторов	20
LLVM	20
Java/ANTLR	21
Утилиты	21
L ^A T _E X: система верстки для научных публикаций	21

Применение

- обработка текстовых форматов данных
файлы САПР, исходные данные для расчетных программ
- командный интерфейс для устройств на микроконтроллерах
управление человеко-читаемыми командами, передача пакетов данных любой структуры и типов
- реализация специализированных скриптовых языков
- обработка исходных текстов программ
модификация, трансляция на другие языки программирования,
универсальный язык шаблонов для ЯП с ограниченными или отсутствующими макросами

Необходимое программное обеспечение

- Windows

MinGW <http://www.mingw.org/>

пакет компилятора и утилит C^{++}

GNU GCC toolchain (g++, flex, bison, make)

git-scm <https://git-scm.com/>

git-клиент

gvim <http://www.vim.org/download.php#pc>

минималистичный редактор кода

с самой простой подсветкой синтаксиса

(на регулярках 3.1)

clang <http://llvm.org>

компилятор C/ C^{++} на базе LLVM

llvm ???

сама библиотека LLVM

- Linux (суперкластер СГАУ "Сергей Королев")

g++, flex, bison, make, git, llvm(-3.5), clang

Глава 1

Структура компилятора

1.1 Термины

исходный код , исходник: текстовое представление программы, предназначенное для чтения и написания человеком. Формат определяется синтаксисом используемого языка программирования или описания данных

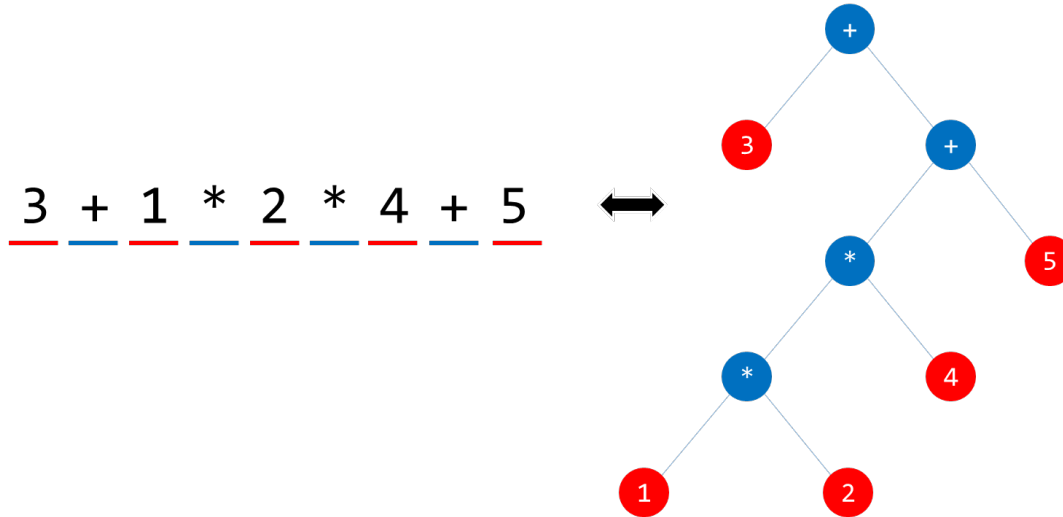
лексер **3** программный компонент, выполняющий выделение синтаксических элементов (токенов) из входного потока символов.

токен объект, содержащий выделенный из исходного кода текст, имя файла/строку/столбец исходника, маркер типа данных (число, строка, оператор), и т.п.

парсер **??** компонент, выполняющий анализ структуры текстового файла данных, с учетом вложенных скобок, синтаксических блоков типа begin/end, условных конструкций, описаний числовых матриц и векторов, и т.п.

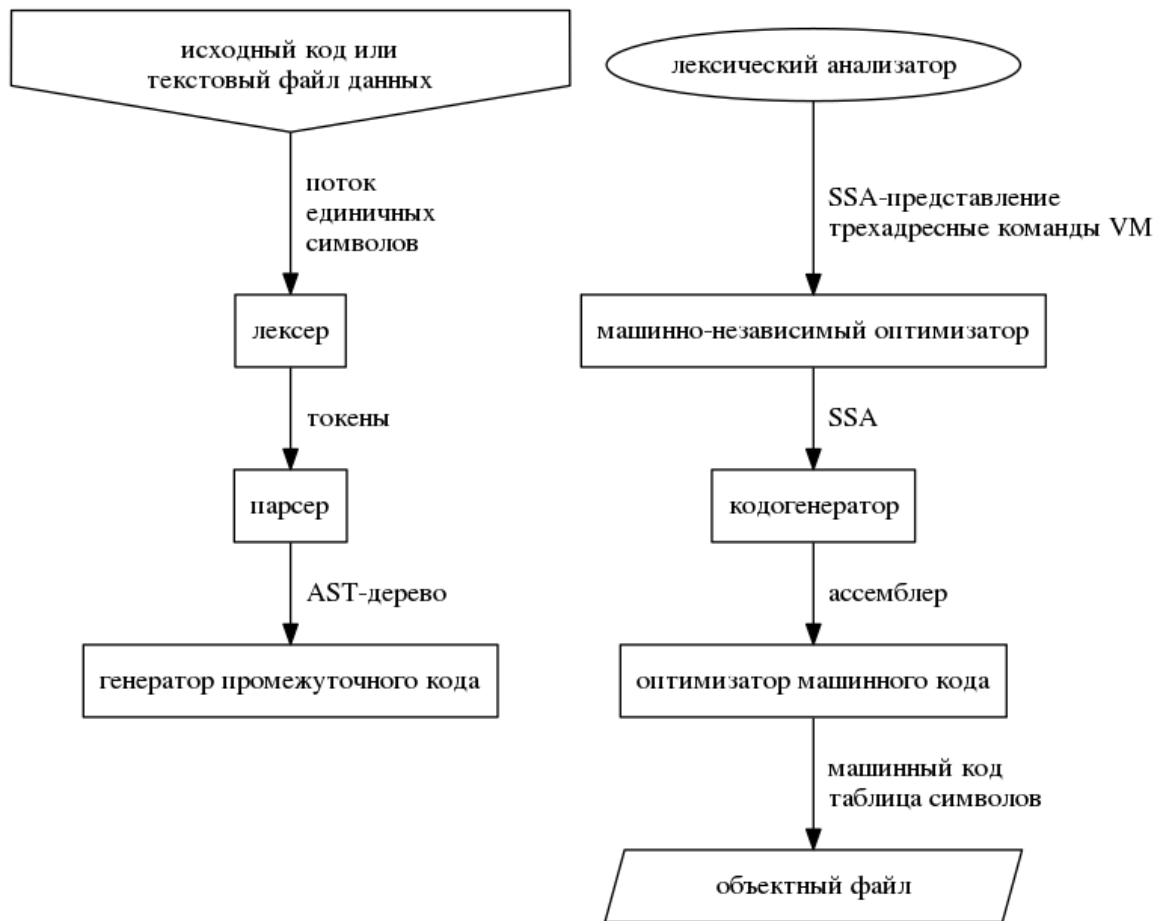
AST [A]bstract [S]yntax [T]ree, абстрактное синтаксическое дерево

вложенная структура данных, состоящая из синтаксических объектов: терминалы (целые, строки, символы,...) и нетерминалы (операторы ссылающиеся на операнды, блоки кода содержащие списки операций,...). AST хранит информацию о вложенности конструкций, порядке вычислений выражений, подчиненности элементов и т.п.



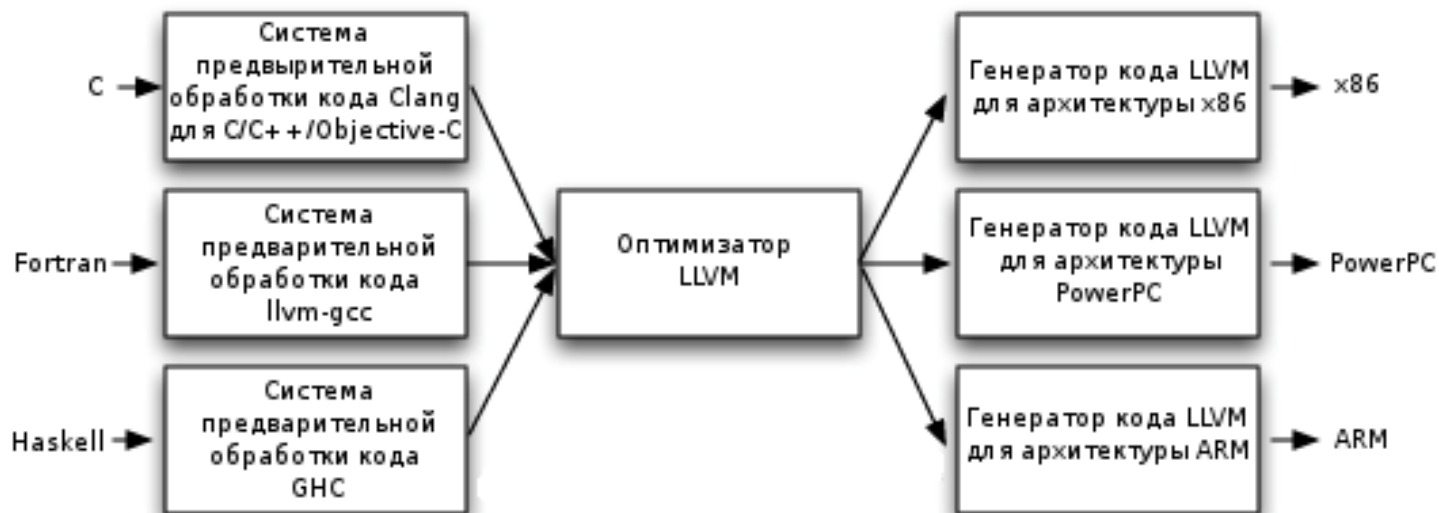
SSA ?? [S]ingle [S]tate [A]ssignment, однократное назначение: промежуточное представление, в котором каждой переменной значение присваивается лишь единожды. Переменные исходной программы разбиваются на версии, обычно с помощью добавления суффикса, таким образом, что каждое присваивание осуществляется уникальной версии переменной. В SSA используются [машинно-независимые трехадресные команды](#) абстрактной виртуальной машины.

1.2 Структура типового компилятора





1.3 Архитектура LLVM



Глава 2

Типичная структура проекта

README.md	2.1	github	описание проекта на https://github.com/
Makefile	2.2	make	зависимости между файлами и команды сборки
lpp.lpp	??	flex	лексер 3
ypp.ypp	??	bison	парсер ??
hpp.hpp	??	g++/clang++	заголовочные файлы C++
cpp.cpp	??	g++/clang++	C++-код: ядро интерпретатора, компилятор, реализация динамических типов, пользовательский код
bat.bat	2.3	win32	запускалка gvim
rc.rc	??	windres	описание ресурсов: иконки приложения, меню,..
logo.ico		windres	логотип в .ico формате
logo.png			логотип в .png (для github README)
filetype.vim	??	(g)vim	привязка расширения файлов скриптов
syntax.vim	??	(g)vim	синтаксическая подсветка для скриптов
.gitignore	??	git	список временных и производных файлов

2.1 README.md

```
# <логотип> <название>
(с) <имя> <email>
<лицензия>
<ссылка на проект на GitHub>
### <ссылки, дополнительная информация>
```

README.md

```
1 # ![logo](logo.png) Mega script language
2
3 (с) Vasya Pupkin <pupkin@gmail.com>, all rights reserved
4
5 license: http://www.gnu.org/copyleft/lesser.html
6
7 GitHub: https://github.com/pupkin/megascript
```

2.2 Makefile

Опции сборки (win32|linux):

EXE суффикс исполняемого файла
RES имя объектного файла ресурсов win32.exe
TAIL опция команды **tail** число последних строк **MODULE.log**

Makefile

```
1 # EXE = .exe |
2 # RES = res.res |
3 # TAIL = -n17|-n7
```

Модуль заполняется автоматически по имени текущего каталога:

Makefile

```
1 MODULE = $(notdir $(CURDIR))
```

Цель команды **make** по умолчанию: сборка и интерпретация тестового файла

Makefile

```
1 .PHONY: exes
2 exes: ./$(MODULE)$(EXE)
3     ./$(MODULE)$(EXE) < $(MODULE).scr > $(MODULE).log && tail $(TAIL) $(MODULE).blog
```

Вторая (стандартная) цель **clean**: удаление временных и рабочих файлов

Makefile

```
1 .PHONY: clean
2 clean:
3     rm -rf *~ *.~ *.exe *.elf *.log ypp.tab.?pp lex.yy.c $(RES)
```

Сборка C++ части

Makefile

```
1 C = cpp.cpp ypp.tab.cpp lex.yy.c
2 H = hpp.hpp ypp.tab.hpp
3 # CXX = clang++
4 CXXFLAGS += -I. -std=gnu++11
```

```
5 ./$(MODULE)$(EXE): $(C) $(H) $(RES)
6     $(CXX) $(CXXFLAGS) -o $@ $(C) $(RES)
```

Генерация кода парсера

Makefile

```
1 ypp.tab.cpp: ypp.ypp
2     bison $<
```

Генерация кода лексера

Makefile

```
1 lex.yy.c: lpp.lpp
2     flex $<
```

Компиляция файла ресурсов (win32)

Makefile

```
1 res.res: rc.rc
2     windres $< -O coff -o $@
```

2.3 bat.bat

bat.bat

```
1 @start .
2 @gvim -c "colorscheme darkblue" -p lexman.scr lexman.log \
3     ypp.ypp lpp.lpp hpp.hpp cpp.cpp \
4     Makefile
```

2.4 rc.rc

rc.rc

1 logo ICON "logo .ico "

Глава 3

Лексер и утилита flex

3.1 Регулярные выражения

Регулярное выражение, или **regex** — текстовая строка, используемая в качестве шаблона для проверки другой строки на совпадение, или поиска подстрок по шаблону.

Большинство букв и символов соответствуют сами себе. Например, регулярное выражение `test` будет в точности соответствовать строке `test`. Некоторые символы это специальные **метасимволы**, и сами себе не соответствуют:

[] используются для определения набора символов, в виде отдельных символов или диапазона, например regex `[0-9A-F]` задает одну цифру шестнадцатеричного числа; набор `[abcd]` можно заменить на диапазон `[a-d]`.

3.2 Примеры самостоятельного применения

Лексер может быть использован как самостоятельный инструмент, если не требуется анализ синтаксиса, и достаточно выполнять заданный C^{++} код при срабатывании одного из регулярных выражений.

3.2.1 Pij2D: загрузка файла числовых данных

Формат файла:

- число строк матрицы `max=Rmax`
- число элементов в строке `max=Xmax`
- данные построчно

Fi.dat

```
1 20.0000 ...
2 90.0000 ...
3 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 ...
4 1.0060 1.0060 1.0060 1.0060 1.0059 1.0059 1.0058 1.0058 1.0057 1.0056 1.0054 1.0053 ...
5 1.0049 1.0049 1.0049 1.0048 1.0047 1.0046 1.0045 1.0043 1.0042 1.0040 1.0037 1.0034 ...
6 ...
7 0.9936 0.9936 0.9935 0.9935 0.9934 0.9933 0.9932 0.9931 0.9929 0.9927 0.9925 0.9922 ...
8 0.9932 0.9932 0.9932 0.9932 0.9931 0.9931 0.9930 0.9930 0.9929 0.9928 0.9927 0.9926 ...
9 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000
```

Pij2D.lpp

```
1 %{
2 #include "hpp.hpp"
```

```

3
4 int  item=0;
5 int  R=0,Rlimit;
6 int  X=0,Xlimit;
7
8 double  Fi[Rmax][Xmax];
9
10 %}
11 %option noyywrap
12 S [\+\-]?
13 N [0-9]+
14 %%
15 {S}{N}(\.{N})?      {
16     item++;
17     if (item==1) { Rlimit=atoi(yytext); cout << "R limit:\t" << Rlimit << "\n"; }
18     if (item==2) { Xlimit=atoi(yytext); cout << "X limit:\t" << Xlimit << "\n";
19         X=R=0; assert(Rlimit<Rmax); assert(Xlimit<Xmax);
20     cout << "\nFi[] phield data:"; }
21     if (item>2) { Fi[R][X++] = atof(yytext); }
22 }
23
24 [\r\n]+      { if (item>2) { X=0; R++; } }
25
26 <<EOF>> {
27     for (int r=0;r<=Rlimit;r++) {
28         cout << "\n\n" << r << ": ";
29         for (int x=0;x<Xlimit;x++) {
30             cout << Fi[r][x] << " ";

```



```
31     }}
32     yyterminate();
33 }
34
35 .    {}
36 %%
```

hpp.hpp

```
1 #ifndef _H_PIJ2D
2 #define _H_PIJ2D
3
4 #define TE "te.log"
5
6 #define Rmax 20+1
7 #define Xmax 90+1
8
9 // #include <mpi.h>
10
11 #include <iostream>
12 #include <fstream>
13 #include <iomanip>
14 #include <cmath>
15 #include <cmath>
16 #include <cstdlib>
17 #include <cstdio>
18 #include <cassert>
19 using namespace std;
20
```

```

21 extern int doit();
22
23 extern int yylex();
24
25 extern double Fi[Rmax][Xmax];
26
27 #endif // _H_PIJ2D

```

cpp.cpp

```

1 int main (int argc, char *argv[]) {
2     // command line processing
3     assert (argc==4+1); // pij.exe [V] [Qm] [Alpha]
4     V = atof(argv[1]); assert(V >0); cout << "V:\t\t\t" << V << "\n";
5     Qm = atof(argv[2]); assert(Qm >0); cout << "Qm:\t\t\t" << Qm << "\n";
6     Alpha = atof(argv[3]); assert(Alpha>0); cout << "Alpha:\t\t\t" << Alpha << "\n";
7     r = atof(argv[4])/1000; assert(r >0); cout << "r:\t\t\t\t" << r << "\n";
8     // Fi.txt field data parsing
9     while (yylex()); // Fi.txt parser loop from stdin
10    // compute tracks
11    return doit();
12 }

```

```

Rmax // строк, не более чем
Xmax // столбцов, не более чем
double Fi[Rmax][Xmax] // массив под данные
int item // общий счетчик прочитанных чисел

```

```
argc, argv // часть исходных данных задается с командной строки  
doit() // функция обработки данных
```

```
while (yylex()); // цикл опроса лексера,  
yylex() // на каждый вызов возвращается один токен
```

`item` используется для определения, какой тип имеет текущее прочитанное число: `Rlimit`, `Xlimit` или данные.

Конец строки в обработке не участвует, факт перехода на следующую строку матрицы определяется по превышению `Xlimit`.

Глава 4

ANTLR

Литература

Основы компиляторов

- [1] [Книга Дракона \(Dragon Book\)](#):
Альфред Ахо, Моника С. Лам, Рави Сети, Джефффри Ульман
Компиляторы: принципы, технологии и инструментарий
- [2] Habrahabr: [Компиляция. 1: лексер](#)
- [3] Habrahabr: [Компиляция. 2: грамматики](#)

LLVM

- [4] [LLVM tutorial](#)

Java/ANTLR

- [5] Грамматика арифметики или пишем калькулятор на ANTLR

Утилиты

- [6] Emden Gansner and Eleftherios Koutsofios and Stephen North
Drawing graphs with dot

Л^AT_EX: система верстки для научных публикаций

- [7] Википедия: **система верстки Л^AT_EX**
- [8] Котельников И. А., Чеботаев П. З.
Л^AT_EX по-русски.
— СПб.: «Корона-Век», 2011. — 496 с. — 2000 экз. — ISBN 978-5-7931-0878-2.
- [9] Львовский С. М.
Набор и верстка в системе L^AT_EX.
— М.: МЦНМО, 2006. — С. 448. — ISBN 5-94057-091-7.
- [10] Балдин Е. М.
Компьютерная типография L^AT_EX.
— «БХВ-Петербург», 2008. — 304 с. — 2000 экз. — ISBN 978-5-9775-0230-6. Книга доступна в электронном виде на сайте CTAN под лицензией CC-BY-SA.

[11] Столяров А. В.

Сверстай диплом красиво: L^AT_EX за три дня.

— Москва: МАКС Пресс, 2010. — 100 с. — 200 экз. — ISBN 978-5-317-03440-5.