методическое пособие

Обработка текстовых форматов данных и реализация компьютерных языков на Flex/Bison/C++/LLVM

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

© <dponyatov@gmail.com>

26 декабря 2015 г.

Оглавление

Применение 2
Необходимое программное обеспечение 2

1 Структура компилятора **3**
1.1 Термины 3
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 **Pij2D**: загрузка файла числовых данных 14

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/releases/download.html	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

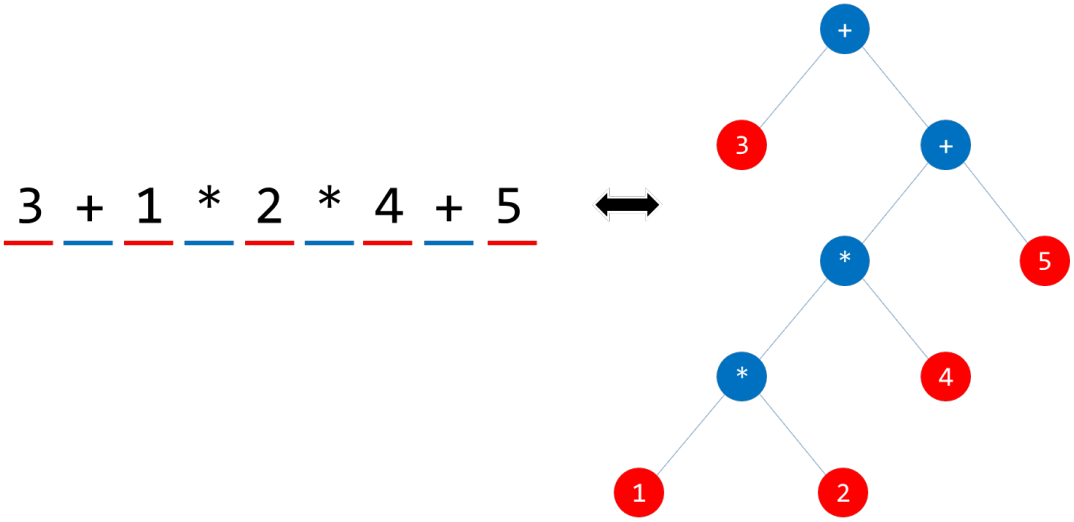
лексер 3 программный компонент, выполняющий выделение синтаксических элементов (токенов) из входного потока символов.

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.

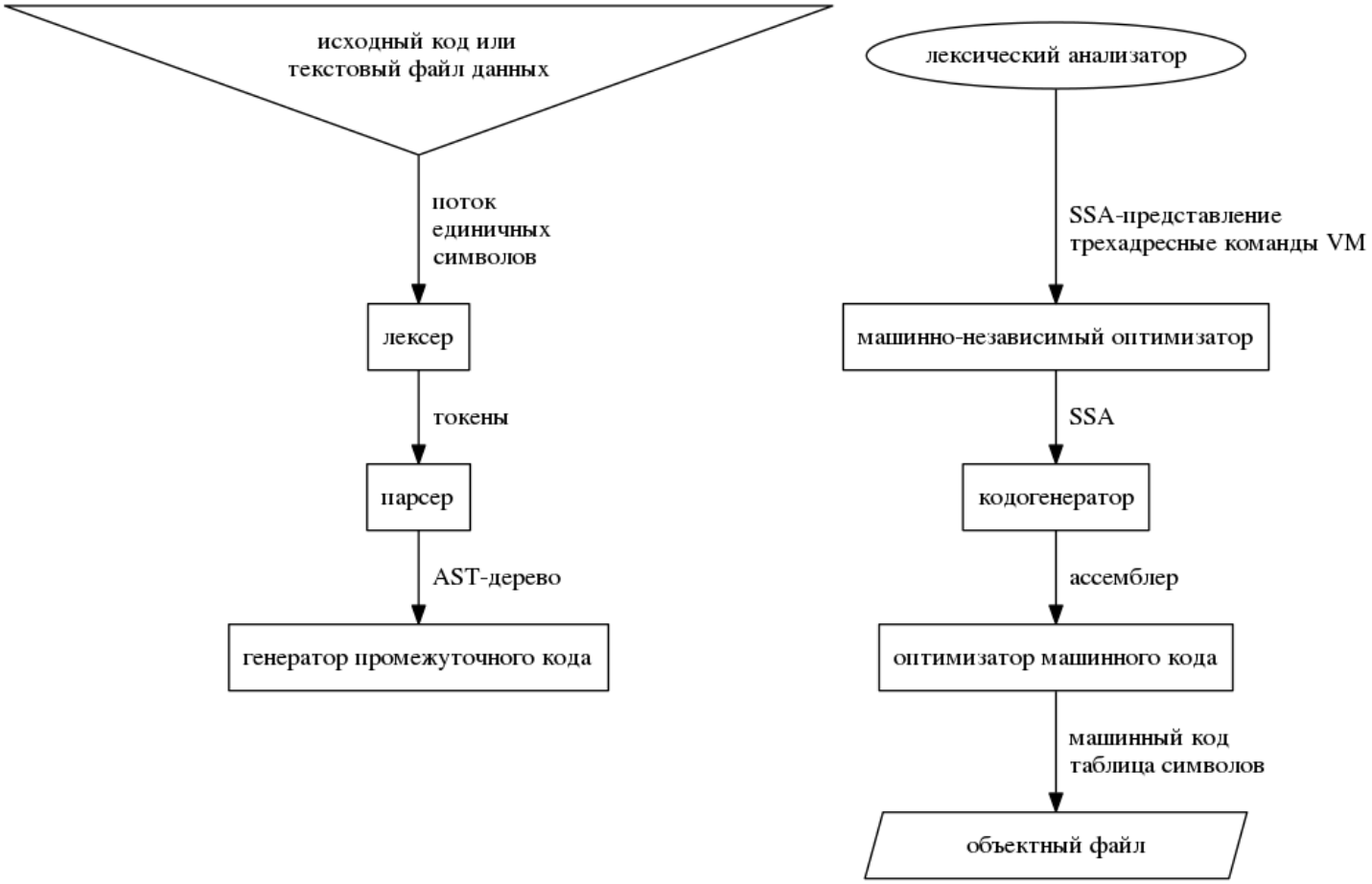
AST [A]bstract [S]yntax [T]ree, абстрактное синтаксическое дерево

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

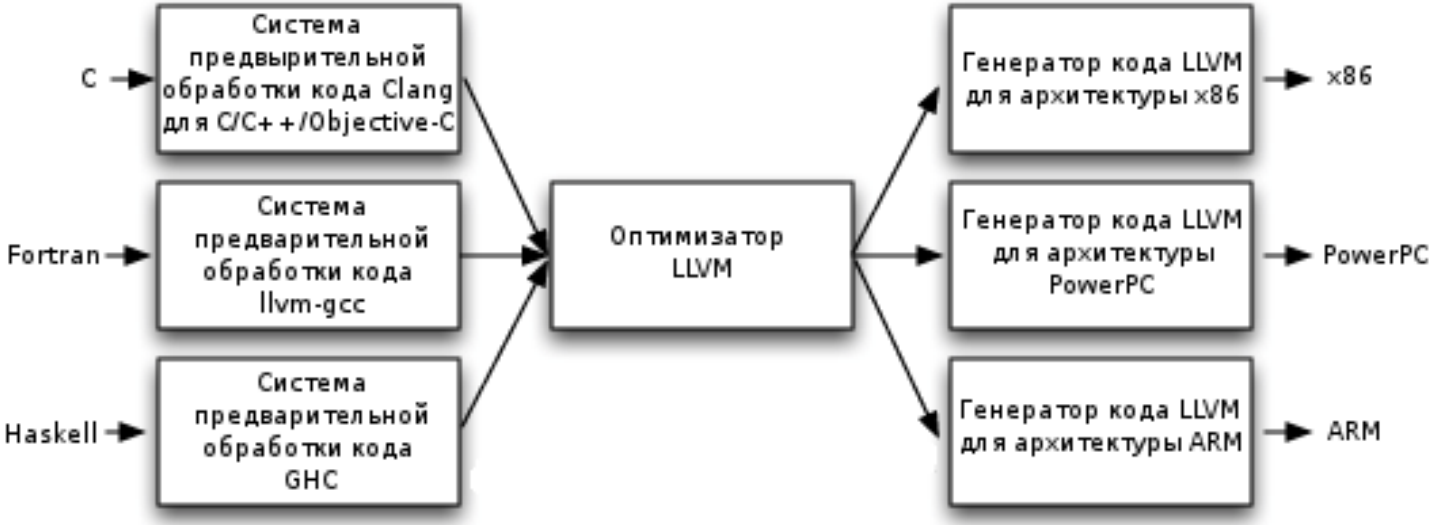


SSA ?? [S]ingle [S]tate [A]ssignment, однократное назначение: промежуточное представление, в котором каждой переменной значение присваивается лишь единожды. Переменные исходной программы разбиваются на версии, обычно с помощью добавления суффикса, таким образом, что каждое присваивание осуществляется 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>
(c) <имя> <email>
<лицензия>
<ссылка на проект на GitHub>
### <ссылки, дополнительная информация>
```

README.md

```
1 # ![logo](logo.png) Mega script language
2
3 (c) 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

```
# EXE = .exe/
# RES = res.res/
# TAIL = -n17/-n7
```

Модуль заполняется автоматически по имени текущего каталога:

Makefile

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

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

Makefile

```
.PHONY: exec
exec: ./$(MODULE)$(EXE)
	./$(MODULE)$(EXE) < $(MODULE).scr > $(MODULE).log && tail $(TAIL) $(MODULE).blog
```

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

Makefile

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

Сборка C^{++} части

Makefile

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

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

Генерация кода парсера

Makefile

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

Генерация кода лексера

Makefile

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

Компиляция файла ресурсов (win32)

Makefile

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

2.3 bat.bat

bat.bat

```
@start .
@gvim -c "colorscheme darkblue" -p lexman.scr lexman.log \
    ypp.ypp lpp.lpp hpp.hpp cpp.cpp \
    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 regexp — 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 regexp `[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

```
 20.0000
 90.0000
1.0000000000000000 1.0000000000000000 1.0000000000000000 1.0000000000000000 1.000000000000
1.0060432746261565 1.0060313466284774 1.0060108458681973 1.0059812667766372 1.005941879954
1.0049168503192238 1.0048934556222540 1.0048532468757054 1.0047952324802623 1.004717981823

...

0.9943152531710587 0.9942749775032367 0.9942057486531359 0.9941058575138396 0.99397283776
0.9939305900736830 0.9938997031202880 0.9938466120169097 0.9937700059670758 0.99366799339
0.9935726728158867 0.9935517984519751 0.9935159176911521 0.9934641446048677 0.99339520071
0.9932342207102822 0.9932237285702180 0.9932056936789868 0.9931796707237957 0.99314501707
1.0000000000000000 1.0000000000000000 1.0000000000000000 1.0000000000000000 1.000000000000
```

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

int item=0;
int R=0,Rlimit;
int X=0,Xlimit;

double Fi[Rmax][Xmax];

%}
%option noyywrap
S [\+\-]?
N [0-9]+
%%
{S}{N}(\.{N})?       {
     item++;
     if (item==1) { Rlimit=atoi(yytext); cout << "R limit:\t" << Rlimit <<"\n"; }
     if (item==2) { Xlimit=atoi(yytext); cout << "X limit:\t" << Xlimit <<"\n";
          X=R=0; assert(Rlimit<Rmax); assert(Xlimit<Xmax);
     cout << "\nFi[] phield data:"; }
     if (item>2) { Fi[R][X++] = atof(yytext); }
     }

[\r\n]+      { if (item>2) { X=0; R++; } }

<<EOF>> {
     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 <cfloat>
16 #include <cstdlib>
17 #include <cstdio>
```

```
#include <cassert>
using namespace std;

extern int doit();

extern int yylex();

extern double Fi[Rmax][Xmax];

#endif // _H_PIJ2D
```

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

```
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.

Литература

[1] Книга Дракона (Dragon Book):
Альфред Ахо, Моника С. Лам, Рави Сети, Джеффри Ульман
Компиляторы: принципы, технологии и инструментарий

[2] LLVM tutorial

Серия статей на Habrahabr.ru

[3] Компиляция. 1: лексер

[4] Компиляция. 2: грамматики

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