Лабораторная работа №2. Ручное построение нисходящих синтаксических анализаторов

Иван Громаковский

22 марта 2014

1 Разработка грамматики

Описания переменных в Си. Сначала следует имя типа, затем разделенные запятой имена переменных. Переменная может быть указателем, в этом случае перед ней идет звездочка (возможны и указатели на указатели, и т. д.). Описаний может быть несколько.

1.1 Грамматика

$$
\begin{array}{lcl}
S & \to & \mathrm{n}VN;S' \\
S' & \to & S \\
S' & \to & \varepsilon \\
V & \to & *V \\
V & \to & \mathrm{n} \\
N & \to & ,VN \\
N & \to & \varepsilon
\end{array}
$$

1.2 Описания нетерминалов

Нетерминал	**Описание**
S	Первое описание переменных.
S'	Следующее описание.
V	Описание одной переменной.
N	Описание следующей переменной.

1.3 Удаление левой рекурсии и правого ветвления

В получившейся грамматике нет ни левой рекурсии, ни правого ветвления.

2 Построение лексического анализатора

В нашей грамматике четыре терминала: «n» (имя), «*», «,» и «;». Не забудем также про конец строки.

```
enum token_t
{
```

```
  NAME, ASTERISK, COMMA, SEMICOLON, END
};
```

Терминал	Токен
n	NAME
*	ASTERISK
,	COMMA
;	SEMICOLON
$	END

3 Построение синтаксического анализатора

Построим множества FIRST и FOLLOW для нетерминалов.

Нетерминал	FIRST	FOLLOW
S	«n»	$
$S^{'}$	«n», ε	$
V	«n», «*»	«,», «;»
N	ε, «,»	«;»

Заведём структуру для хранения дерева.

```
class node
{
   std::string s;
   std::vector<node *> children;
public:
   node(const std::string & s);

   void add_child(node * child);
   void add_child(const std::string & s);

   std::vector<node *> get_children() const;
   std::string get_name() const;
};
```

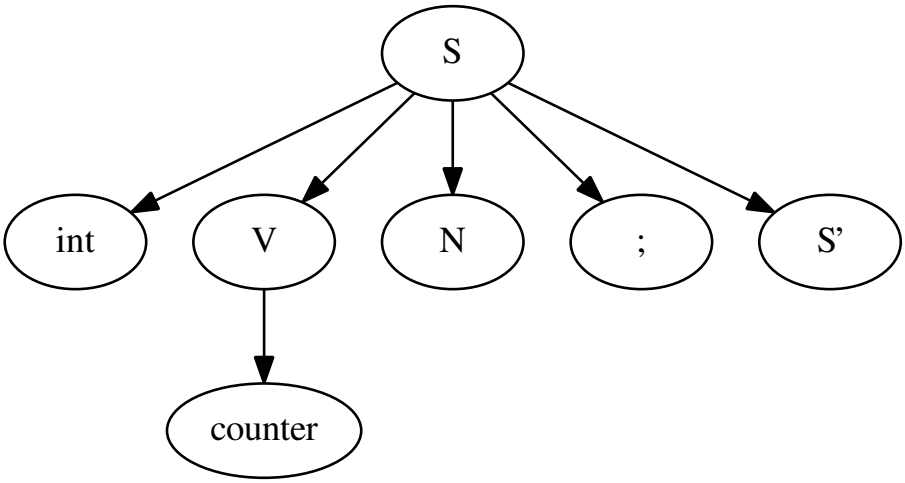
4 Визуализация и тестирование

Тест	**Описание**
int counter;	Примитивный тест с одним простым описанием.
int counter	Тест, в котором забыли про точку с запятой.
, counter;	Тест, в котором вместо имени типа запятая.
int counter; char c;	Тест с двумя описаниями.
int counter; char @ c;	Во втором описании неизвестный символ.
size_t counter;	Подчёркивание в имени типа.
int 9thousands;	Имя не может начинаться с цифры.
line2 counter2;	Но цифра может быть в имени.
size_t counter, index_of_string;	Несколько переменных.
char * buf, ** point_to_str, cur_char;	Указатели.
void buf *;	Не в том месте звёздочка.
point_3t * pointer, p, ** * * o_O; void * buf, ** a;	Большой тест.
point_3t * pointer, p, ** * * o_O; void * buf ** a;	Забыли запятую.

4.1 Результаты тестов

4.1.1 Тест 1

int counter;



Passed.

4.1.2 Тест 2

int counter

Failed. Wrong input format at position 12, comma or semicolon was expected, but end of file was found

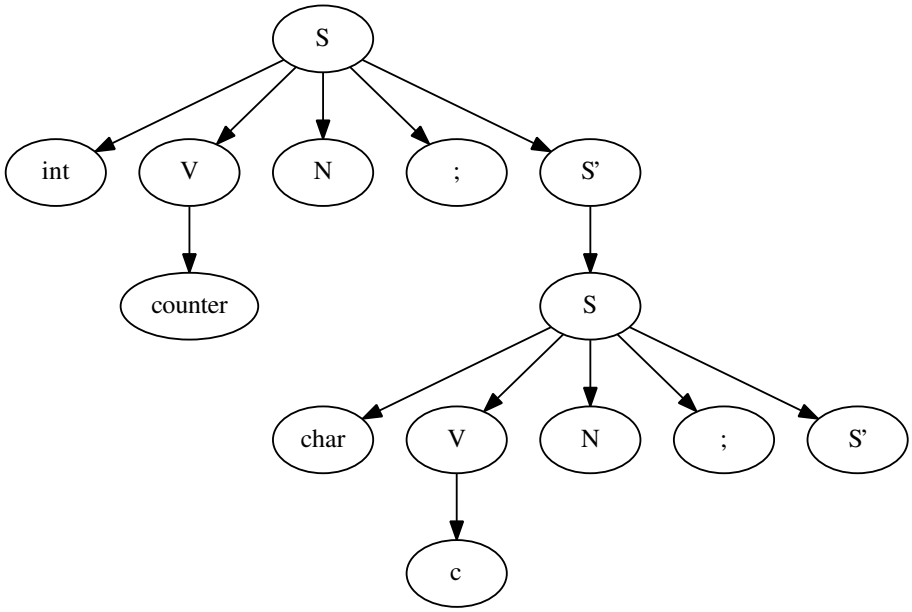
4.1.3 Тест 3

, counter;

Failed. Wrong input format at position 0, name was expected, but , was found

4.1.4 Тест 4

int counter; char c;



Passed.

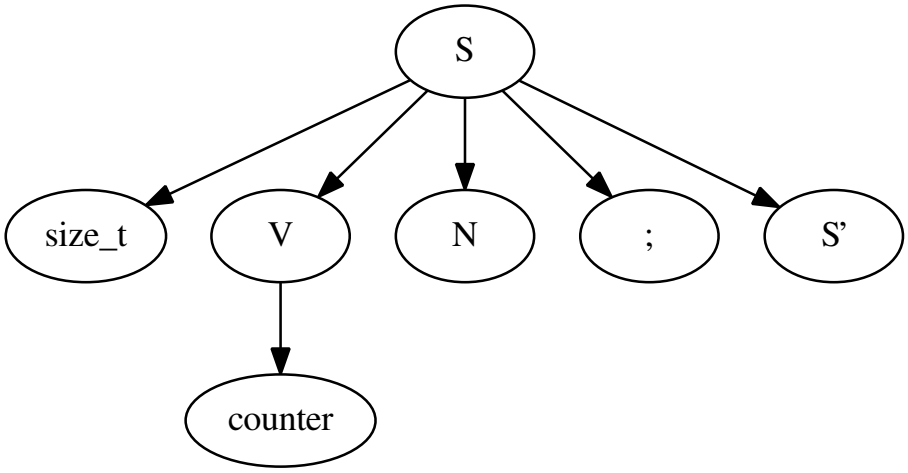
4.1.5 Тест 5

int counter; char @ c;

Failed. Parsing error. Bad symbol at position 18

4.1.6 Тест 6

size_t counter;



Passed.

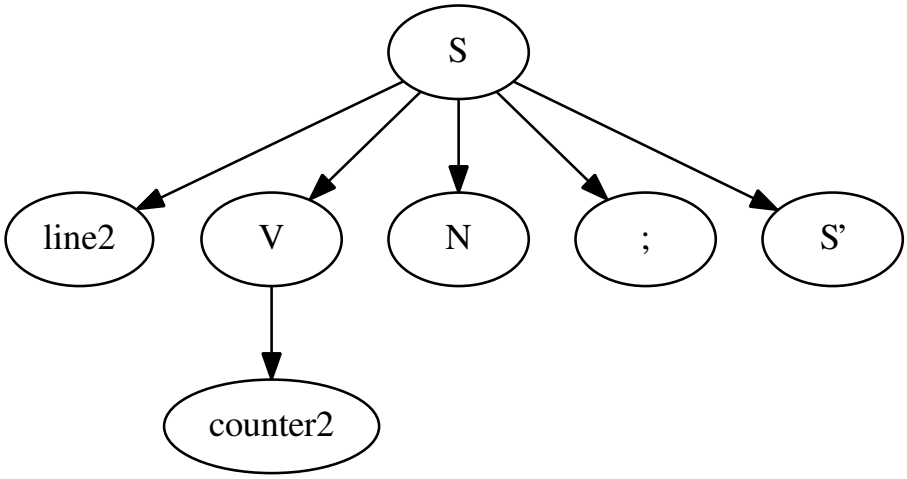
4.1.7 Тест 7

int 9thousands;

Failed. Parsing error. Bad symbol at position 4

4.1.8 Тест 8

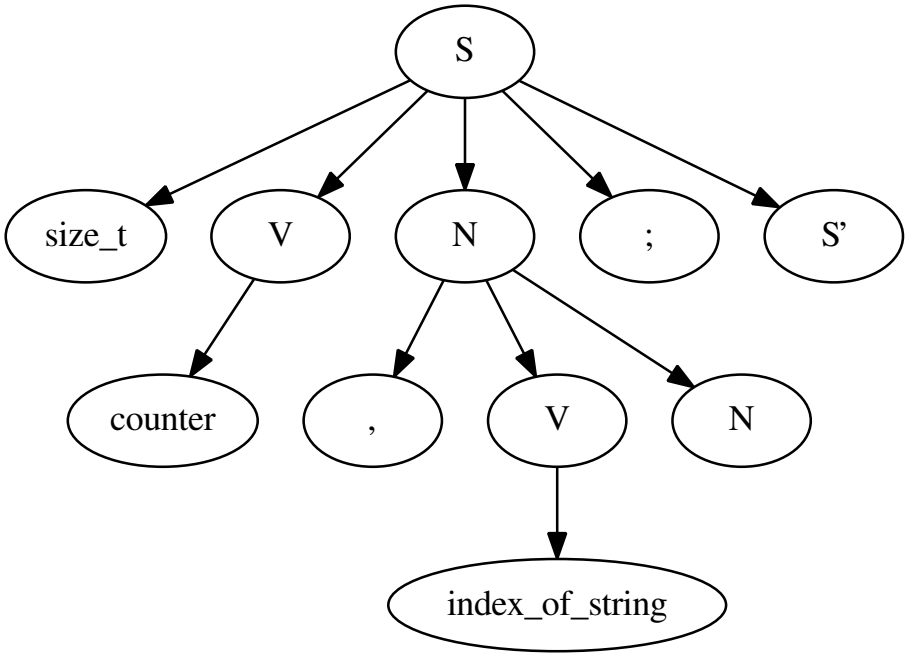
line2 counter2;



Passed.

4.1.9 Тест 9

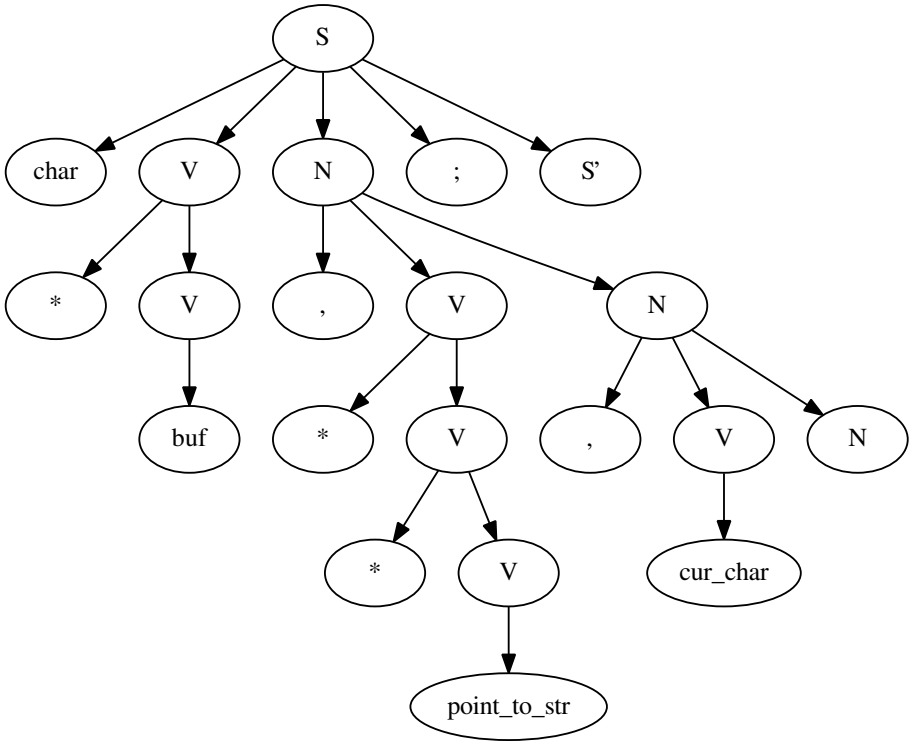
size_t counter, index_of_string;



Passed.

4.1.10 Тест 10

char * buf, ** point_to_str, cur_char;



Passed.

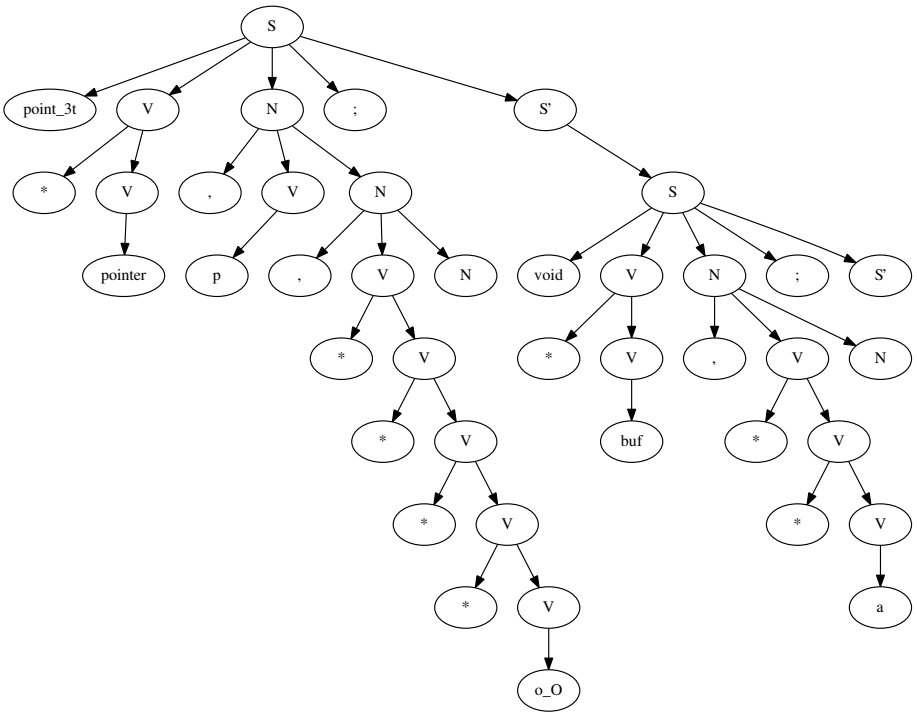
4.1.11 Тест 11

void buf *;

Failed. Wrong input format at position 9, comma or semicolon was expected, but * was found

4.1.12 Тест 12

point_3t * pointer, p, ** * * o_O; void * buf, ** a;



Passed.

4.1.13 Тест 13

point_3t * pointer, p, ** * * o_O; void * buf ** a;

Failed. Wrong input format at position 46, comma or semicolon was expected, but * was found