ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ПРОГРАММИРОВАНИЕ

Лексический и синтаксический анализ текста

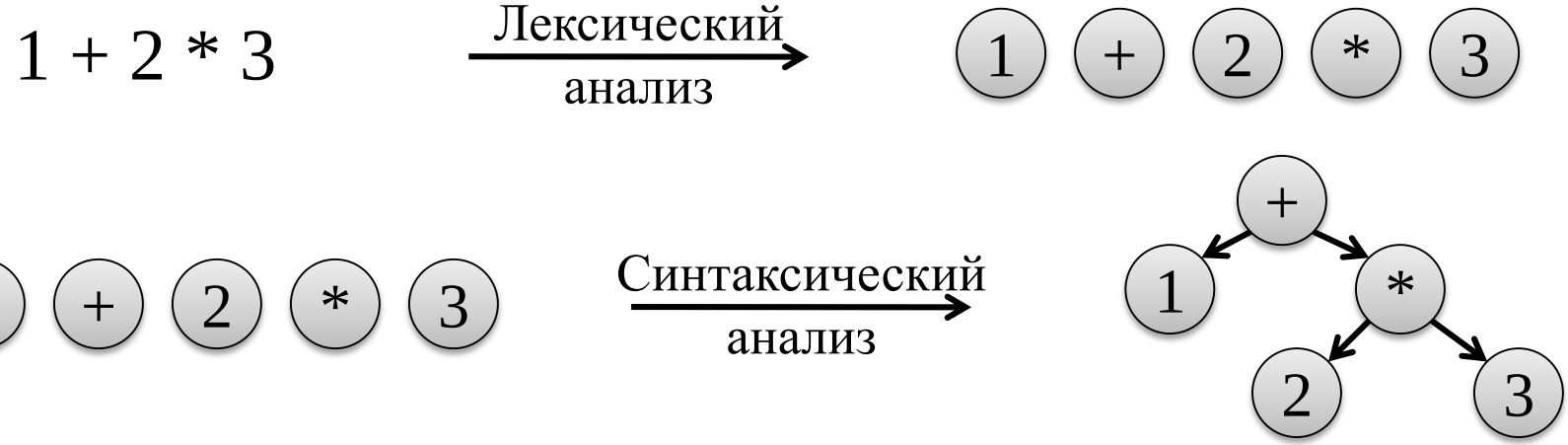
Преподаватель:

Перышкова Евгения Николаевна

Этапы анализа текста

Лексический анализ - процесс аналитического разбора входной последовательности символов с целью получения на выходе последовательности символов, называемых «токенами» (подобно группировке букв в словах).

Синтаксический анализ (жарг. *па́рсинг*) — это процесс сопоставления линейной последовательности лексем (*слов*, *токенов*) естественного или формального языка, получаемых в результате лексического анализа, с его формальной грамматикой. Результатом является структура данных, удобная для последующей обработки, например, синтаксическое дерево.



Лексический анализ

Группа символов входной последовательности, идентифицируемая на выходе процесса как *токен*, называется *лексемой*.

Грамматика языка, для которого проводится лексический анализ, задаёт определённый набор токенов, которые могут встретиться на входе.

Распознавание *токенов* в контексте грамматики предусматривает определение принадлежности к одному из возможных *классов*, задаваемых грамматикой языка. Например, в языке Си: целочисленная константа, идентификатор, ключевое слово, фигурные скобки и т.п.

Любая лексема, которая согласно грамматике не может быть идентифицирована как *токен* языка, обычно рассматривается как специальный *токен-ошибка*, например:

1abcd – *не константа и не идентификатор.*

Лексический анализ (2)

Каждый токен можно представить в виде структуры, содержащей *идентификатор токена* и, если нужно, последовательность символов лексемы, выделенной из входного потока (строку, число и т. д.).

Класс	tokenID
{	1
}	2
(	3
)	4
;	5
return	6
const10	7
identifier	8
type	9
string	10

```
int main(){
    printf("Hello world\n");
    return 0;
}
```

	0	1	2	3	4	5	6	7	8	9
0	i	n	t		m	a	i	n	(	)
1	{	\n	\t	p	r	i	n	f	(	"
2	H	e	l	l	o		w	o	r	l
3	d	\n	"	)	;	\n	\t	r	e	t
4	u	r	n		0	;	\n	}		

Лексический анализ (3)

Каждый токен можно представить в виде структуры, содержащей *идентификатор токена* и, если нужно, последовательность символов лексемы, выделенной из входного потока (строку, число и т. д.).

Класс	tokenID
{	1
}	2
(	3
)	4
;	5
operator	6
const	7
identifier	8
type	9
string	10

	0	1	2	3	4	5	6	7	8	9
0	i	n	t		m	a	i	n	(	)
1	{	\n	\t	p	r	i	n	f	(	"
2	H	e	l	l	o		w	o	r	l
3	d	\n	"	)	;	\n	\t	r	e	t
4	u	r	n		0	;	\n	}		

⟨9,int⟩; ⟨8,main⟩; ⟨3⟩; ⟨4⟩; ⟨1⟩; ⟨8,printf⟩; ⟨3⟩; ⟨10,"Hello world"⟩; ⟨4⟩; ⟨5⟩; ⟨6,return⟩; ⟨7,0⟩; ⟨5⟩; ⟨2⟩;

Выходной формат лексического анализатора

100	0	1	2	3	4	5	6	7	8	9
0	i	n	t		m	a	i	n	(	)
1	{	\n	\t	p	r	i	n	f	(	"
2	H	e	l	l	o		w	o	r	l
3	d	\n	"	)	;	\n	\t	r	e	t
4	u	r	n		0	;	\n	}		

⟨9,int⟩; ⟨8,main⟩; ⟨3⟩; ⟨4⟩;
⟨1⟩; ⟨8,printf⟩; ⟨3⟩;
⟨10,"Hello world"⟩;
⟨4⟩; ⟨5⟩; ⟨6,return⟩;
⟨7,0⟩; ⟨5⟩; ⟨2⟩;

table[*i*]

```
struct token_t {
    int id;
    char *val;
    int len;
} table[1024];
```

i	id	val_{10}	len	Фрагмент
0	9	100	3	int
1	8	104	4	main
2	3	NULL	-	(
3	4	NULL	-	)
4	1	NULL	-	{
5	8	113	5	printf

Выходной формат лексического анализатора (2)

i	id	val	len	Фрагмент
0	9	100	3	int
1	8	104	4	main
2	3	NULL	-	(
3	4	NULL	-	)
4	1	NULL	-	{
5	8	113	5	printf
6	3	-	-	)
7	10	119	14	"Hello…\n"
8	4	-	-	)
9	5	-	-	;
10	6	137	6	return
11	7	144	1	0
12	5	-	-	;
13	2	-	-	}

100	0	1	2	3	4	5	6	7	8	9
0	i	n	t		m	a	i	n	(	)
1	{	\n	\t	p	r	i	n	f	(	"
2	H	e	l	l	o		w	o	r	l
3	d	\n	"	)	;	\n	\t	r	e	t
4	u	r	n		0	;	\n	}		

⟨9,int⟩; ⟨8,main⟩; ⟨3⟩; ⟨4⟩;
⟨1⟩; ⟨8,printf⟩; ⟨3⟩;
⟨10,"Hello world"⟩;
⟨4⟩; ⟨5⟩; ⟨6,return⟩;
⟨7,0⟩; ⟨5⟩; ⟨2⟩;

Обработка пробельных разделителей

Во многих текстовых форматах предусматривается несколько альтернативных символов, позволяющих визуально разделить элементы конфигурационного файла так, чтобы их удобно было читать человеку. Для этой цели используются символы *пробела* (" "), *табуляции* ("`\t`"), перевода на новую *строку* ("`\n`").

При этом большинство форматов допускает многократное использование и смешивание символов, например:

# стр.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	`{`	`\n`													
2					`"`	`k`	`e`	`y`	`"`	`:`	`"`	`I`	`D`	`"`	`\n`
3	`\t`	`"`	`v`	`a`	`l`	`"`	`:`	`"`	`2`	`0`	`"`	`\n`			
4	`\n`														
5	`}`														

Обработка пробельных разделителей (2)

Представление в текстовом редакторе:

# стр.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	{	\n													
2					"	k	e	y	"	:	"	I	D	"	\n
3	\t	"	v	a	l	"	:	"	2	0	"	\n			
4	\n														
5	}														

Представление оперативной памяти и в файле:

{	\n					"	k	e	y	"	:	"	I	D
"	\n	\t	"	v	a	l	"	:	"	2	0	"	\n	\n
}														

Обработка пробельных разделителей (3)

Представление оперативной памяти и в файле:

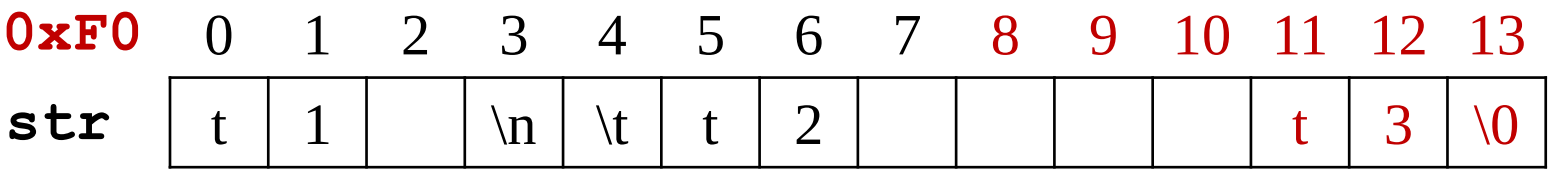
{	\n					"	k	e	y	"	:	"	I	D
"	\n	\t	"	v	a	l	"	:	"	2	0	"	\n	\n
}														

Функция пропуска пробельных символов в строке suf:

```
char *skip_spaces(char *suf)
{
    while( schr(" \t\n",*suf) >= 0 ){
        suf++;
    }
    return suf;
}
```

Пропустить разделители

```
char *skip_spaces(char *suf)
{
    while( schr(" \t\n",*suf) ){
        suf++;
    }
    return suf;
}
```




```
suf = str + 2;
suf = skipspaces(suf);            // suf == 0xF5
suf = suf + scspn(suf," \t\n"); // suf == 0xF7
suf = skipspaces(suf);            // suf == 0xFB
```

Синтаксический анализ

Целью *синтаксического анализа* является распознавание линейной последовательности лексем и формирование специализированной структуры, позволяющей дальнейшую обработку данных.

Любой текст, имеющий синтаксис поддается автоматическому анализу:

- Языки программирования – разбор исходного кода языков программирования, в процессе трансляции (компиляции или интерпретации);
- Структурированные данные — данные, языки их описания, оформления и т. д. Например, XML, HTML, CSS, ini-файлы, специализированные конфигурационные файлы и т. п.;
- SQL-запросы (DSL-язык);
- Математические выражения;
- Регулярные выражения (которые, в свою очередь, могут использоваться для автоматизации лексического анализа);

Файл passwd

Лексический анализ

```
passwd
        root:lZTB0KTrSKy8M:0:0:Acc. for root:/root:/bin/sh\n
        b@n:x:2:1ff:binary acc.:/bin:/bin/false\n
        user:x:20:20:Regular user:/home/user/:/bin\sh
```

Класс	ID	Комментарий
login	**0**	[0-9a-zA-Z]
password	**1**	[0-9a-zA-Z]
UID	**2**	Дес. число, уникальное
GID	**3**	Дес. число
GECOS	**4**	[0-9a-zA-Z/\!@#$%^&*()":;/?>.<,]
home	**5**	[0-9a-zA-Z/\!@#$%^&*()":;/?>.<,]
shell	**6**	[0-9a-zA-Z/\!@#$%^&*()":;/?>.<,]
error	**10**	Токен-ошибка

Идентификатор токена совпадает с номером поля в строке, разделенной знаком ':'

Обнаружение символа перевода строки или конца файла означает сброс нумерации на 0

Лексический анализ (2)

```
passwd
        root:lZTB0KTrSKy8M:0:0:Acc. for root:/root:/bin/sh\n
        b@n:x:2:1ff:binary acc.:/bin:/bin/false\n
        user:x:20:20:Regular user:/home/user/:/bin\sh
```

Класс	ID	Комментарий
login	**0**	[0-9a-zA-Z]
password	**1**	[0-9a-zA-Z]
UID	**2**	Дес. число, уникальное
GID	**3**	Дес. число
GECOS	**4**	[0-9a-zA-Z ,/\!@#$%^&*()":;/?>.<,]
home	**5**	[0-9a-zA-Z, /\!@#$%^&*()":;/?>.<,]
shell	**6**	[0-9a-zA-Z /\!@#$%^&*()":;/?>.<,]
error	**10**	Токен-ошибка

⟨0,root⟩; ⟨1,lZTB0KTrSKy8M⟩; ⟨2,0⟩; ⟨3,0⟩; ⟨4, Acc. for root⟩; ⟨6,/root⟩; ⟨7,/bin/sh⟩;

⟨10⟩; ⟨1,x⟩; ⟨2,2⟩; ⟨10⟩; ⟨4, binary acc.⟩; ⟨6,/bin⟩; ⟨7,/bin/false⟩;

⟨0,user⟩; ⟨1,x⟩; ⟨2,20⟩; ⟨3,20⟩; ⟨4, Regular user⟩; ⟨6,/home/user⟩; ⟨7,/bin\sh⟩;

Синтаксический анализ

⟨0,root⟩; ⟨1,lZTB0KTrSKy8M⟩; ⟨2,0⟩; ⟨3,0⟩; ⟨4, Acc. for root⟩; ⟨6,/root⟩; ⟨7,/bin/sh⟩;
⟨10⟩; ⟨1,x⟩; ⟨2,2⟩; ⟨10⟩; ⟨4, binary acc.⟩; ⟨6,/bin⟩; ⟨7,/bin/false⟩;
⟨0,user⟩; ⟨1,x⟩; ⟨2,20⟩; ⟨3,20⟩; ⟨4, Regular user⟩; ⟨6,/home/user⟩; ⟨7,/bin\sh⟩;

Лексический анализ проверяет токены только на отсутствие недопустимых символов. Синтаксическому анализатору доступно больше информации, например известно, что поля home и shell содержат файловые пути, а поле UID (Unique IDentifier) должно быть уникально у каждого пользователя.

В процессе синтаксического анализа проводится проверка выполнения этих условий и построение таблицы, содержащей корректные записи:

login	Password	UID	GID	GECOS	home	shell
root	lZTB0KTrSKy8M	0	0	Acc. for root	/root	/bin/sh

Синтаксический анализ (2)

В процессе синтаксического анализа проводится проверка выполнения этих условий и построение таблицы, содержащей корректные записи:

login	Password	UID	GID	GECOS	home	shell
root	lZTB0KTrSKy8M	0	0	Acc. for root	/root	/bin/sh

```
struct passwd_record {
    char user[32]; // man useradd
    char passwd[256];
    int uid, gid;
    char gecos[256];
    char home[PATH_MAX];
    char shell[PATH_MAX];
} pwdrecs[1024];
```

Файл crontab

Лексический анализ

```
crontab
    5 0-20/2 * * * root /bin/echo
    15 1-20,22-40 1 * * alex /home/alex/test
    0 F2 * * 1-5 john /usr/local/bin/ls > /home/alex/out
    23 */2 * * * root /usr/var/lib/dhclient_fix.sh
```

Класс	ID	Комментарий
Minutes	**0**	[0-9\-,*/]
Hours	**1**	[0-9\-,*/]
Month day	**2**	[0-9\-,*/]
Month	**3**	[0-9\-,*/]
Week day	**4**	[0-9\-,*/]
user	**5**	[0-9a-zA-Z]
command	**6**	[0-9a-zA-Z /\!@#$%^&*()":;/?>.<,]
error	**10**	Токен-ошибка

Лексический анализ (2)

```
crontab
    5 0-20/2 * * * root /bin/echo
    15 1-20,22-40 1 * * alex /home/alex/test
    0 F2 * * 1-5 john /usr/local/bin/ls > /home/alex/out
    23 */2 * * * root /usr/var/lib/dhclient_fix.sh
```

⟨0,"5"⟩; ⟨1,"0-20/2"⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"root"⟩; ⟨6,"/bin/echo"⟩;

⟨0,"15"⟩; ⟨1,"1-20,22-40"⟩; ⟨2,"1"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"alex"⟩;
⟨6,"/home/alex/test"⟩;

⟨0,"0"⟩; ⟨10⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"1-5"⟩; ⟨5,"john"⟩;
⟨6,"/usr/local/bin/ls > /home/alex/out"⟩;

⟨0,"23"⟩; ⟨1,"*/2"⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"root"⟩;
⟨6,"/usr/var/lib/dhclient_fix.sh"⟩;

Синтаксический анализ

⟨⟨0,"5"⟩; ⟨1,"0-20/2"⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"root"⟩; ⟨6,"/bin/echo"⟩;
⟨0,"15"⟩; ⟨1,"1-20,22-40"⟩; ⟨2,"1"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"alex"⟩;
⟨6,"/home/alex/test"⟩;
⟨0,"0"⟩; ⟨10⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"1-5"⟩; ⟨5,"john"⟩;
⟨6,"/usr/local/bin/ls > /home/alex/out"⟩;
⟨0,"23"⟩; ⟨1,"*/2"⟩; ⟨2,"*"⟩; ⟨3,"*"⟩; ⟨4,"*"⟩; ⟨5,"root"⟩;
⟨6,"/usr/var/lib/dhclient_fix.sh"⟩;

На этапе синтаксического анализа проверяется принадлежность значений временных полей допустимым диапазонам. Также проверяется корректность файлового пути команды, расположенной первой в поле **command**. В результате строится таблица, состоящая из допустимых записей.

Синтаксический анализ (2)

Представление временных интервалов (часы)

1. Временные интервалы можно сохранить в виде строки. Это допускается в рамках выполнения лабораторной работы.
2. Более удобным для последующей обработки является формат, позволяющий быстро получать необходимые интервалы. Для этого можно использовать массив целых чисел, размер которого соответствует диапазону допустимых значений, а элементы равны 0 или 1 в зависимости от того, используется данное значение для запуска команды или нет. Например для поля "часы":

"0-17/2"

0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	0	1	0	1	0	1	0	1	0

12	13	14	15	16	17	18	19	20	21	22	23
1	0	1	0	1	0	0	0	0	0	0	0

Синтаксический анализ (3)

Структура, описывающая таблицу синтаксического разбора выглядит следующим образом:

```
struct crontab_record {
    int min[60];
    int hour[24];
    int mday[31];
    int wday[7];
    char user[32];
    char command[1024];
} cronrecs[1024];
```

Файл syslog.conf

Лексический анализ

```
kern.*          /var/adm/kernel
kern.crit       @csc.sibsutis.fr
k0rn.crit       /dev/console
kern.enfo       /var/adm/kern.info

+192.168.300.1
kern.*          /var/remote/kernel

+csc.sibsutis.ru
auth.error      /var/csc/logs.txt
```

Класс	ID	Комментарий
service	0	[a-z]
priority	1	[a-z]
output	2	[0-9a-zA-Z /\!@#$%^&*()":;/?>.<,]
host	3	[0-9a-zA-Z/\@:.]
токен-ошибка	10	

⟨0,"kern"⟩; ⟨1,"*"⟩; ⟨2,"/var/adm/kernel"⟩; ⟨0,"kern"⟩; ⟨1,"crit"⟩;
⟨2,"@csc.sibsutis.fr"⟩; ⟨10⟩; ⟨1,"crit"⟩; ⟨2,"/dev/console"⟩; ⟨0,"kern"⟩; ⟨1,"enfo"⟩;
⟨2,"/var/adm/kern.info"⟩; ⟨3,"192.168.300.1"⟩; ⟨0,"kern"⟩; ⟨1,"*"⟩;
⟨2,"/var/remote/kernel"⟩; ⟨3,"csc.sibsutis.ru"⟩; ⟨0,"auth"⟩; ⟨1,"error"⟩;
⟨2,"/var/csc/logs.txt"⟩;

Синтаксический анализ

⟨0,"kern"⟩; ⟨1,"*"⟩; ⟨2,"/var/adm/kernel"⟩; ⟨0,"kern"⟩; ⟨1,"crit"⟩;
⟨2,"@csc.sibsutis.fr"⟩; ⟨10⟩; ⟨1,"crit"⟩; ⟨2,"/dev/console"⟩; ⟨0,"kern"⟩; ⟨1,"enfo"⟩;
⟨2,"/var/adm/kern.info"⟩; ⟨3,"192.168.300.1"⟩; ⟨0,"kern"⟩; ⟨1,"*"⟩;
⟨2,"/var/remote/kernel"⟩; ⟨3,"csc.sibsutis.ru"⟩; ⟨0,"auth"⟩; ⟨1,"error"⟩;
⟨2,"/var/csc/logs.txt"⟩;

На этапе синтаксического анализа проверяется допустимость полей селектора, IP адресов и доменных имен. В результате формируется таблица:

host	**serv**	**prio**	**target**
localhost	kern	*	**/var/adm/kernel**
csc.sibsutis.ru	auth	error	**/var/csc/logs.txt**

ini-файлы

Лексический анализ

```
[PostgreSQL]
Description = ODBC for PostgreSQL
Driver   = /usr/lib/psqlodbc.so
Driver   = /usr/lib/libodbcpsqlS.so
FileUs@ge  = 1
[MySQL]
Description = ODBC for MySQL
Driver = /usr/lib/libmyodbc3_r.so
Setup = /usr/lib/libodbcmyS.s@
FileUsage  = 1
```

Класс	ID	Комментарий
section	**0**	[0-9a-zA-Z]
key	**1**	[0-9a-zA-Z]
value	**2**	[0-9a-zA-Z /\!@#$%^& *()":;/?>.<,]
токен-ошибка	**10**	

⟨0,"PostgreSQL"⟩; ⟨1,"Description"⟩; ⟨2,"ODBC for PostgreSQL"⟩; ⟨1,"Driver"⟩; ⟨2,"/usr/lib/psqlodbc.so"⟩; ⟨1,"Driver"⟩; ⟨1,"/usr/lib/libodbcpsqlS.so"⟩; ⟨10⟩; ⟨2,"1"⟩; ⟨0,"MySQL"⟩; ⟨1,"Description"⟩; ⟨2,"ODBC for MySQL"⟩; ⟨1,"Driver"⟩; ⟨2,"/usr/lib/libmyodbc3_r.so"⟩; ⟨1,"Setup"⟩; ⟨2,"/usr/lib/libodbcmyS.s@"⟩; ⟨1,"FileUsage"⟩; ⟨2,"1"⟩;

Синтаксический анализ

⟨0,"PostgreSQL"⟩; ⟨1,"Description"⟩; ⟨2,"ODBC for PostgreSQL"⟩; ⟨1,"Driver"⟩; ⟨2,"/usr/lib/psqlodbc.so"⟩; ⟨1,"Driver"⟩; ⟨1,"/usr/lib/libodbcpsqlS.so"⟩; ⟨10⟩; ⟨2,"1"⟩; ⟨0,"MySQL"⟩; ⟨1,"Description"⟩; ⟨2,"ODBC for MySQL"⟩; ⟨1,"Driver"⟩; ⟨2,"/usr/lib/libmyodbc3_r.so"⟩; ⟨1,"Setup"⟩; ⟨2,"/usr/lib/libodbcmyS.s@"⟩; ⟨1,"FileUsage"⟩; ⟨2,"1"⟩;

На этапе синтаксического анализа проверяется корректность всех файловых путей. Исключаются секции с одинаковым названиями и ключи внутри одной секции, имеющие одинаковые имена. В результате заполняется таблица

section	key	value
PostgreSQL	Description	ODBC for PostgreSQL
PostgreSQL	Driver	/usr/lib/libodbcpsqlS.so
MySQL	Description	ODBC for MySQL
MySQL	Driver	/usr/lib/libmyodbc3_r.so
MySQL	FileUsage	1

XML-файлы

Формат XML

XML (англ. eXtensible Markup Language) активно используется для как хранения конфигурации, например, GNOME Gconf, CUPS, Diasoft Framework.

Элемент – базовое понятие XML. Каждый XML-документ содержит один или несколько элементов. Границы элементов представлены начальным и конечным **тегами**. Имя элемента в начальном и конечном тегах элемента должно совпадать:

Начальный тег: <name1> **Конечный тег**: </name1>

В начальном теге и теге пустого элемента могут быть заданы **атрибуты:**

<**mytag attr1**=”value1” **attr2**=”value2” … > Body </**mytab**>

Тег пустого элемента (не имеющего тела) обозначается так:

<**myetag attr1**=”avalue1” **attr2**=”avalue2” … / >

Лексический анализ

<**mytag attr1** = value1 **attr2**="Str with space" attr3>
Value of element
<**mytag1**> Value of element </**mytag1**>
</**mytab**>

Наличие значения атрибута не обязательно: key [= value]

Класс	ID	Комментарий
open tag begin	**0**	<
close tag begin	**1**	</
tag end	**2**	>
tag name	**3**	[a-zA-Z0-9]
attr name	**4**	между tag name и tag end [a-zA-Z0-9]
attr val	**5**	После attr name: 1) без кавычек (attr1) – до первого разделителя; 2) с кавычками (attr2) – все что внутри " … "
element value	**6**	строка от tag end открывающегося тэга до: 1)close tag begin того же элемента(mytag) 2) open tag begin вложенного элемента

Лексический анализ

```
<recipe name="хлеб" preptime="5" cooktime="180">
  <composition>
    <ingr amount="3" unit="ст">Мука</ingr>
    <ingr amount="0.25" unit="гр">Дрожжи</ingr>
  </composition>
  <instructions>
    <step>Смешать все ингредиенты и тщательно замесить.</step>
    <step>Закрыть и оставить на один час в теплом помещении.</step>
</instructions>
</recipe>
```

⟨0⟩; ⟨3,"recipe"⟩; ⟨4,"name"⟩; ⟨5,"хлеб"⟩; ⟨4,"preptime"⟩; ⟨5,"5"⟩; ⟨4,"cooctime"⟩; ⟨5,"180"⟩; ⟨2⟩; ⟨0⟩; ⟨3,"composition"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"3"⟩; ⟨4,"unit"⟩; ⟨5,"ст"⟩; ⟨2⟩;⟨6,"Мука"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"0.25"⟩; ⟨4,"unit"⟩; ⟨5,"гр"⟩; ⟨2⟩; ⟨6,"Дрожжи"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;

Класс	ID
open tag begin	0
close tag begin	1
tag end	2
tag name	3
attr name	4
attr val	5
element value	6

Лексический анализ (2)

Класс	ID
open tag begin	0
close tag begin	1
tag end	2
tag name	3
attr name	4
attr val	5
element value	6

```
<recipe name="хлеб" preptime="5" cooktime="180">
  <composition>
    <ingr amount="3" unit="ст">Мука</ingr>
    <ingr amount="0.25" unit="гр">Дрожжи</ingr>
  </composition>
  <instructions>
    <step>Смешать все ингредиенты и тщательно замесить.</step>
    <step>Закрыть и оставить на один час в теплом помещении.</step>
</instructions>
</recipe>
```

⟨0⟩; ⟨3,"recipe"⟩; ⟨4,"name"⟩; ⟨5,"хлеб"⟩; ⟨4,"preptime"⟩; ⟨5,"5"⟩; ⟨4,"cooctime"⟩; ⟨5,"180"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"composition"⟩; ⟨2⟩; ⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"3"⟩; ⟨4,"unit"⟩; ⟨5,"ст"⟩; ⟨2⟩;⟨6,"Мука"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩; ⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"0.25"⟩; ⟨4,"unit"⟩; ⟨5,"гр"⟩; ⟨2⟩; ⟨6,"Дрожжи"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩; ⟨1⟩; ⟨3,"composition"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"instructions"⟩; ⟨2⟩; ⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"Смешать все ингредиенты и тщательно замесить."⟩; ⟨1⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"Закрыть и оставить на один час в теплом помещении"⟩; ⟨1⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨1⟩; ⟨3,"instructions"⟩; ⟨2⟩;

⟨1⟩; ⟨3,"recipe"⟩; ⟨2⟩;

Простейший синтаксический анализ

⟨0⟩; ⟨3,"recipe"⟩; ⟨4,"name"⟩; ⟨5,"хлеб"⟩; ⟨4,"preptime"⟩; ⟨5,"5"⟩; ⟨4,"cooctime"⟩; ⟨5,"180"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"composition"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"3"⟩; ⟨4,"unit"⟩; ⟨5,"ст"⟩; ⟨2⟩;⟨6,"Мука"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"0.25"⟩; ⟨4,"unit"⟩; ⟨5,"гр"⟩; ⟨2⟩; ⟨6,"Дрожжи"⟩; ⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;

⟨1⟩; ⟨3,"composition"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"instructions"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"Смешать все ингредиенты и тщательно замесить."⟩; ⟨1⟩; ⟨3,"step"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"Закрыть и оставить на один час в теплом помещении"⟩; ⟨1⟩; ⟨3,"step"⟩; ⟨2⟩;

⟨1⟩; ⟨3,"instructions"⟩; ⟨2⟩;

⟨1⟩; ⟨3,"recipe"⟩; ⟨2⟩;

Простейший синтаксический анализ (2)

⟨0⟩; ⟨3,"recipe"⟩; ⟨4,"name"⟩; ⟨5,"хлеб"⟩; ⟨4,"preptime"⟩; ⟨5,"5"⟩; ⟨4,"cooktime"⟩; ⟨5,"180"⟩; ⟨2⟩;	**recipe**
⟨0⟩; ⟨3,"composition"⟩; ⟨2⟩;	**composition | recipe**
⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"3"⟩; ⟨4,"unit"⟩; ⟨5,"ст"⟩; ⟨2⟩;	**ingr | composition | recipe**
⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;	**~~ingr~~ | composition | recipe**
⟨0⟩; ⟨3,"ingr"⟩; ⟨4,"amount"⟩; ⟨5,"0.25"⟩; ⟨4,"unit"⟩; ⟨5,"гр"⟩; ⟨2⟩;	**ingr | composition | recipe**
⟨1⟩; ⟨3,"ingr"⟩; ⟨2⟩;	**~~ingr~~ | composition | recipe**
⟨1⟩; ⟨3,"composition"⟩; ⟨2⟩;	**recipe**

⟨0⟩; ⟨3,"instructions"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"

⟨1⟩; ⟨3,"step"⟩; ⟨2⟩;

⟨0⟩; ⟨3,"step"⟩; ⟨2⟩; ⟨6,"

⟨1⟩; ⟨3,"step"⟩; ⟨2⟩;

...

При обнаружении знака open tag begin значение tag name, следующее за ним поместить в стек. Если за open tag begin следует другой токен – ***ошибка***.
При обнаружении close tag begin вытолкнуть из стека элемент и сравнить его значение с tag name следующим за close tag begin. Если не равны – ***ошибка***.

JSON-файлы

Формат JSON

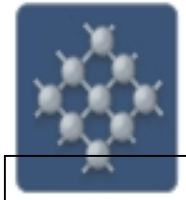
JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными. Формат считается языконезависимым и может использоваться практически с любым языком программирования. В качестве значений в JSON используются структуры:

Объект — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки "{ }". Ключ описывается строкой, между ним и значением стоит символ ":". Пары ключ-значение отделяются друг от друга запятыми.

Массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки "[ ]". Значения разделяются запятыми.

Значение может быть строкой в двойных кавычках, числом, объектом, массивом, одним из литералов: true, false или null. Т.о. структуры могут быть вложены друг в друга.

Строка — это упорядоченное множество из нуля или более символов юникода, заключенное в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты "\".



Формат JSON (2)

address.json

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
     "street": "Московское ш., 101, кв.101",
     "city": "Ленинград",
  },
  "phoneNumbers": [
     "812 123-1234",
     "916 123-4567"
  ]
}
```

address.xml

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <street>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
</address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

Формат JSON (3)

address.json

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
     "street": "Московское ш., 101, кв.101",
     "city": "Ленинград",
  },
  "phoneNumbers": [
     "812 123-1234",
     "916 123-4567"
  ]
}
```

address.xml

```
<person firstName="Иван" lastName="Иванов">
  <address street="Московское ш., 101, кв.101"
           city="Ленинград" postalCode="101101" />
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

Лексический анализ

address.json

```
{
  "firstName": "Иван",
  "address": {
     street : "Московское ш., 101, кв.101", "city": "Ленинград",
  },
  "phoneNumbers": [ "812 123-1234", "916 123-4567" ]
}
```

Класс	ID	Комментарий
open brace	**0**	**{**
close brace	**1**	**}**
open bracket	**2**	**[**
close bracket	**3**	**]**
colon	**4**	**:**
comma	**5**	**,**
string	**6**	1) набор символов между кавычками 2) набор символов между разделителями

Лексический анализ (2)

address.json

```
{
  "firstName": "Иван",
  "address": {
     street : "Московское ш., 101, кв.101", "city": "Ленинград",
  },
  "phoneNumbers": [ "812 123-1234", "916 123-4567" ]
}
```

⟨0⟩; ⟨6,"firstName"⟩; ⟨4⟩; ⟨6,"Иван"⟩; ⟨5⟩;
⟨6,"address"⟩; ⟨4⟩; ⟨0⟩;
⟨6,"street"⟩; ⟨4⟩; ⟨6,"Московское ш…"⟩; ⟨5⟩;
⟨6,"city"⟩; ⟨4⟩; ⟨6,"Ленинград"⟩; ⟨5⟩;
⟨1⟩; ⟨5⟩;
⟨6,"phoneNumbers"⟩; ⟨4⟩; ⟨2⟩; ⟨6,"812 123-1234"⟩; ⟨5⟩;
⟨6,"916 123-4567"⟩; ⟨3⟩; ⟨0⟩; ⟨5⟩;
⟨1⟩;

Класс	ID
open brace	0
close brace	1
open bracket	2
close bracket	3
colon	4
comma	5
string	6

Простейший синтаксический анализ

address.json

```
{
  "firstName": "Иван",
  "address": {
     street : "Московское ш., 101, кв.101", "city": "Ленинград",
  },
  "phoneNumbers": [ "812 123-1234", "916 123-4567" ]
}
```

1. Пара ключ-значения обязательно разделяется двоеточием.
2. Между парами ключ-значения ставятся запятые.
3. После последней пары в объекте запятая не обязательна.
4. Проверка корректности расстановки фигурных и квадратных скобок производится с помощью стека (как для формата XML).