词法分析程序的设计与实现——工具生成

任飞 2021210724

目录

1 概述

1.1 实验内容与要求

1. 选定源语言，比如：C、Pascal、Python、Java 等，任何一种语言均可；
2. 可以识别出用源语言编写的源程序中的每个单词符号，并以记号的形式输出每个单词符号。
3. 可以识别并跳过源程序中的注释。
4. 可以统计源程序中的语句行数、各类单词的个数、以及字符总数，并输出统计结果。
5. 检查源程序中存在的词法错误，并报告错误所在的位置。
6. 对源程序中出现的错误进行适当的恢复，使词法分析可以继续进行，对源程序进行一次扫描，即可检查并报告源程序中存在的所有词法错误。
7. 编写 LEX 源程序，利用 LEX 编译程序自动生成词法分析程序。

1.2 实现的内容

使用 lex 工具构建了一个 C 语言的词法分析程序，支持除了预处理器和替用记号外的 C17 标准中的所有词法元素，具有统计功能。

2 程序设计说明

2.1 设计规格

本程序参照 C17 标准文档编写，尽可能完整地实现了 C17 标准的所有内容，仅考虑实际情况做了以下简化：

1. 不支持与预处理器相关的内容。C 语言的基于文本替换的预处理器是一个比较特殊的设计。C 语言的 token 实际上分为预处理前的 preprocessing-token 和预处理后的 token，尤其是对数字的定义方面两者存在较大的差异。预处理时拼接行尾反斜杠的特性也破坏了词法的正则性。为了降低复杂性，我们只识别预处理后的 token，不考虑与预处理器相关的部分，也就是认为：
   - 所有的宏定义都已经被展开
   - 不存在行尾的反斜杠
   - 不存在三标符
   - 不存在不是合法整数或合法浮点数的预处理数字
   - `#` 直到行尾将被如同注释地忽略
2. 不支持替用记号，即 `<% %>` 等记号。

不同于手工编写的版本，本程序只接受 ascii 编码的源文件。

程序使用命令行交互，可通过参数指定输入文件（若不指定则为标准输入），token 流和统计信息输出到标准输出流。

程序将所有的 token 分为以下 7 类：

- 关键字 keyword
- 标识符 identifier
- 标点 punctuator
- 整数常量 integer constant
- 浮点数常量 floating constant
- 字符常量 character constant
- 字符串字面量 string literal

出现错误时会输出为一个类型为 error 的 token。

其中关键字包括：

`auto`	`break`	`case`	`char`	`const`	`continue`
`default`	`do`	`double`	`else`	`enum`	`extern`
`float`	`for`	`goto`	`if`	`inline`	`int`
`long`	`register`	`restrict`	`return`	`short`	`signed`
`sizeof`	`static`	`struct`	`switch`	`typedef`	`union`
`unsigned`	`void`	`volatile`	`while`	`_Alignas`	`_Alignof`
`_Atomic`	`_Bool`	`_Complex`	`_Generic`	`_Imaginary`	`_Noreturn`
`_Static_assert`	`_Thread_local`				

标点包括：

{	}	[	]	(	)	;	:	...	?	.	->
~	!	+	-	*	/	%	^	&	|	=	+=
-=	*=	/=	%=	^=	&=	|=	==	!=	<	>	<=
>=	&&	||	<<	>>	<<=	>>=	++	--	,		

2.2 程序结构

lex 源文件为 `lex.l` 经过 lex 程序编译后生成 `lex.yy.c` ，与 `main.cpp` 一起编译链接生成可执行文件 `lexer`。

`main` 函数中每次调用 `yylex` 函数即读出一个 token。最后输出统计信息。

2.3 词法分析器实现

参照标准中的词法定义编写 lex 正则表达式即可。

2.4 与手写版本的比较

如果输入的源文件在词法上是完全正确的，那么 lex 版本和手写版本的效果是一致的。但如果输入的源文件在词法上有误，两者的表现会有所不同。

lex 生成的词法分析器每次都会遵循最长匹配和优先匹配的原则匹配一个 token，有时不会如预期一般给出报错。而手写的词法分析程序会根据具体的情况给出具体的报错。

例如 `123abc`，手写版本认为是一个不合法的整数常量，给出“`abc` 不是合法的整数后缀”的报错。但是 lex 版本会将其识别为一个整数常量 `123` 和一个标识符 `abc`。

相比之下，手写版本的错误处理更加友好。当然使用 lex 也能够编写提供友好错误处理的词法分析程序，但需要手动加入对各种典型错误场景的正则表达式。

3 测试

由于篇幅限制，仅包含较为简短的测试用例。

3.1 基本正常程序测试

输入

```
#include <stdio.h>
int main(void) {
    int sum = 0;
    for (int i = 0; i < 100; i++) {
        sum += i;
    }
    printf("sum: %d\n", sum);
}
```

输出

```
2:1: <keyword, int>
2:5: <identifier, main>
2:9: <punctuator, (>
2:10: <keyword, void>
2:14: <punctuator, )>
2:16: <punctuator, {>
3:5: <keyword, int>
3:9: <identifier, sum>
3:13: <punctuator, =>
3:15: <error, 0>
3:16: <punctuator, ;>
4:5: <keyword, for>
4:9: <punctuator, (>
4:10: <keyword, int>
4:14: <identifier, i>
4:16: <punctuator, =>
4:18: <error, 0>
4:19: <punctuator, ;>
4:21: <identifier, i>
4:23: <punctuator, <>
4:25: <integer constant, 100>
4:28: <punctuator, ;>
4:30: <identifier, i>
4:31: <punctuator, ++>
4:33: <punctuator, )>
4:35: <punctuator, {>
5:9: <identifier, sum>
5:13: <punctuator, +=>
5:16: <identifier, i>
5:17: <punctuator, ;>
6:5: <punctuator, }>
7:5: <identifier, printf>
7:11: <punctuator, (>
7:12: <string literal, "sum: %d\n">
7:23: <punctuator, ,>
7:25: <identifier, sum>
7:28: <punctuator, )>
7:29: <punctuator, ;>
8:1: <punctuator, }>

stat:
```

```
total: 144 characters, 8 lines
tokens:
  error:
    0                   2
    (total)             2
  identifier:
    i                   4
    main                1
    printf              1
    sum                 3
    (total)             9
  integer constant:
    100                 1
    (total)             1
  keyword:
    for                 1
    int                 3
    void                1
    (total)             5
  punctuator:
    (                   3
    )                   3
    ++                  1
    +=                  1
    ,                   1
    ;                   5
    <                   1
    =                   2
    {                   2
    }                   2
    (total)             21
  string literal:
    "sum: %d\n"         1
    (total)             1
  (total)               39
```

3.2 复杂数字测试

输入

```
int main(void) {
    -0ll+0x123u*0123;
    0xabcdefghijk/0912%0912.0;
```

```
    1.e+2+0e-2-0x0.4p-2;
    3.4ll+-322lul+3L-3lL;
    12.4qwq+0x12.3P43;
    0x124+000-111e;
}
```

输出

```
1:1: <keyword, int>
1:5: <identifier, main>
1:9: <punctuator, (>
1:10: <keyword, void>
1:14: <punctuator, )>
1:16: <punctuator, {>
2:5: <punctuator, ->
2:6: <error, 0>
2:7: <identifier, ll>
2:9: <punctuator, +>
2:10: <integer constant, 0x123u>
2:16: <punctuator, *>
2:17: <integer constant, 0123>
2:21: <punctuator, ;>
3:5: <integer constant, 0xabcdef>
3:13: <identifier, ghijk>
3:18: <punctuator, />
3:19: <error, 0>
3:20: <integer constant, 912>
3:23: <punctuator, %>
3:24: <floating constant, 0912.0>
3:30: <punctuator, ;>
4:5: <floating constant, 1.e+2>
4:10: <punctuator, +>
4:11: <floating constant, 0e-2>
4:15: <punctuator, ->
4:16: <floating constant, 0x0.4p-2>
4:24: <punctuator, ;>
5:5: <floating constant, 3.4l>
5:9: <identifier, l>
5:10: <punctuator, +>
5:11: <punctuator, ->
5:12: <integer constant, 322lu>
5:17: <identifier, l>
```

```
5:18: <punctuator, +>
5:19: <integer constant, 3L>
5:21: <punctuator, ->
5:22: <integer constant, 3l>
5:24: <identifier, L>
5:25: <punctuator, ;>
6:5: <floating constant, 12.4>
6:9: <identifier, qwq>
6:12: <punctuator, +>
6:13: <floating constant, 0x12.3P43>
6:22: <punctuator, ;>
7:5: <integer constant, 0x124>
7:10: <punctuator, +>
7:11: <integer constant, 000>
7:14: <punctuator, ->
7:15: <integer constant, 111>
7:18: <identifier, e>
7:19: <punctuator, ;>
8:1: <punctuator, }>

stat:
total: 165 characters, 8 lines
tokens:
  error:
    0                   2
    (total)             2
  floating constant:
    0912.0              1
    0e-2                1
    0x0.4p-2            1
    0x12.3P43           1
    1.e+2               1
    12.4                1
    3.4l                1
    (total)             7
  identifier:
    L                   1
    e                   1
    ghijk               1
    l                   2
    ll                  1
    main                1
```

```
    qwq                 1
    (total)             8
  integer constant:
    000                 1
    0123                1
    0x123u              1
    0x124               1
    0xabcdef            1
    111                 1
    322lu               1
    3L                  1
    3l                  1
    912                 1
    (total)             10
  keyword:
    int                 1
    void                1
    (total)             2
  punctuator:
    %                   1
    (                   1
    )                   1
    *                   1
    +                   6
    -                   5
    /                   1
    ;                   6
    {                   1
    }                   1
    (total)             24
  (total)               53
```

3.3 复杂字符常量、字符串字面量与通用字符名测试

输入

```
int main(void) {
    char *\u1234 = u8"\a\b\f\n\r\t\v\012\xab\U12345678";
    mm\u123 = L"\qabc";
    U'3' == '';
    "this is a
    '\t
}
```

输出

```
1:1: <keyword, int>
1:5: <identifier, main>
1:9: <punctuator, (>
1:10: <keyword, void>
1:14: <punctuator, )>
1:16: <punctuator, {>
2:5: <keyword, char>
2:10: <punctuator, *>
2:11: <identifier, \u1234>
2:18: <punctuator, =>
2:20: <identifier, u8>
2:22: <error, ">
2:23: <error, \>
2:24: <identifier, a>
2:25: <error, \>
2:26: <identifier, b>
2:27: <error, \>
2:28: <identifier, f>
2:29: <error, \>
2:30: <identifier, n>
2:31: <error, \>
2:32: <identifier, r>
2:33: <error, \>
2:34: <identifier, t>
2:35: <error, \>
2:36: <identifier, v>
2:37: <error, \>
2:38: <integer constant, 012>
2:41: <error, \>
2:42: <identifier, xab\U12345678>
2:55: <error, ">
2:56: <punctuator, ;>
3:5: <identifier, mm>
3:7: <error, \>
3:8: <identifier, u123>
3:13: <punctuator, =>
3:15: <identifier, L>
3:16: <error, ">
3:17: <error, \>
```

```
3:18: <identifier, qabc>
3:22: <error, ">
3:23: <punctuator, ;>
4:5: <char constant, U'3'>
4:10: <punctuator, ==>
4:13: <error, '>
4:14: <error, '>
4:15: <punctuator, ;>
5:5: <error, ">
5:6: <identifier, this>
5:11: <identifier, is>
5:14: <identifier, a>
6:5: <error, '>
6:6: <error, \>
6:7: <identifier, t>
7:1: <punctuator, }>

stat:
total: 138 characters, 7 lines
tokens:
  char constant:
    U'3'                1
    (total)             1
  error:
    "                   5
    '                   3
    \                   12
    (total)             20
  identifier:
    L                   1
    \u1234              1
    a                   2
    b                   1
    f                   1
    is                  1
    main                1
    mm                  1
    n                   1
    qabc                1
    r                   1
    t                   2
    this                1
```

```
    u123                1
    u8                  1
    v                   1
    xab\U12345678       1
    (total)             19
  integer constant:
    012                 1
    (total)             1
  keyword:
    char                1
    int                 1
    void                1
    (total)             3
  punctuator:
    (                   1
    )                   1
    *                   1
    ;                   3
    =                   2
    ==                  1
    {                   1
    }                   1
    (total)             11
  (total)               55
```

4 总结

这次实验让我初步掌握了 lex 这个工具。