编译原理

词法分析器实验报告

Lexical Analyzer Experimental Report

学　　院：计算机与信息技术

专　　业：计算机科学

学生姓名：刘宜进

学　　号：14282008

指导教师：徐金安

北京交通大学

2017 年 5 月

目 录

1 实验目的

以下为正则文法所描述的 C 语言子集单词符号的示例，请补充单词符号：

++，--， >>， <<, += , -= ,*=, /= ，&&（逻辑与），||（逻辑或），!（逻辑非）等等，给出补 充后描述 C 语言子集单词符号的正则文法，设计并实现其词法分析程序。

1、给出各单词符号的类别编码；

2、词法分析程序应能发现输入串中的错误；

3、词法分析作为单独一遍编写，词法分析结果为二元式序列组成的中间文件；

4、设计两个测试用例（尽可能完备），并给出测试结果。

2 实验内容

该实验运用词法分析基本原理，利用 Python 语言实现了一个 C 程序的词法分析器。下面，通过程序功能描述、程序结构、数据结构、主要函数等四方面展开详细介绍。

2.1 程序功能描述

该程序能够从本地读取 C 程序，并进行简单的词法分析，并以四元组的形式输出。同时，对于程序中的一些词法错误，能够进行识别。该程序最终生成图形用户接口(GUI)，方便操作。

2.2 程序结构

该程序主要有三大部分组成：

1、源程序读取

2、词法分析

3、界面展示

2.2.1 源程序读取

源程序的读取采用 Python 文件读取流。

```
root.filename =  filedialog.askopenfilename(
                    initialdir = "/",
                    title = "Select file",
                    filetypes = (("c files","*.c"),
                    ("all files","*.*")))
fin = open(root.filename, "r")
```

接着以换行符为分割符，将源程序的每一行进行分割，并存贮在一个列表 List 中。同时，提供一个读取每一个字符的函数 getchar(input_str)。

2.2.2 词法分析

词法分析又包括了扫描程序、错误报告、四元组生成三部分

扫描程序，每次读取一个字符，判断它是标识符、数字、分隔符或者操作符。

错误报告，每当检测到错误，将错误信息打印。

四元组生成，将分析正确的结果以四元组的形式生成。

2.2.3 界面展示

主要利用 Python 自带库 tkinter，主体由四部分组成：

CodeTest:该组件是可编辑文本，对源程序进行展示；

ErrorTest:该组件也是文本，对检测出的错误进行展示；

Analysis：该组件为分析的结果，即四元组。

Menu:该组件是菜单，由四个子菜单键 filemenu、lexmenu、windowsmenu 和 helpmenu。

Filemenu 是文件子菜单，包括文件打开、保存、退出等；

Lexmenu 是分析子菜单，只有一个下拉选项 lex，点击进行词法分析；

Windowsmenu 是窗口子菜单，点击 fullscreen 可全屏，或者按快捷键 F11；

Helpmenu 是帮助子菜单，目前没有实现。

2.3 数据结构

首先用三个列表(List)分别表示关键字、操作符、分隔符

用一个字典(Dict)表示字符表及其编码

两个全局变量 current_row、current_line 表示当前行和列，用来取字符和定位错误

2.4 主要函数

表 2-1 主要函数及功能介绍

函数名	参数	返回值	用途
getchar	input_str：string	下一个字符	读取一个字符
ungetchar	input_str：string	上一个字符	退回一个字符
error	msg：strign	no	错误
scanner	input_str：string	no	词法分析
fileloader	no	no	打开源程序
pre_interface	no	no	前端界面

3 程序测试

3.1 测试用例

采用两个 C 代码分别进行测试，第一是冒泡排序，其中存在错误如下：

```
float f = 2e10a;
void 2Bubble_Sort(int *num, int n)
```

第二个是杨辉三角，其中存在错误如下：

```
3main()
```

分别进行测试，并对测试结果进行测评。

3.2 测试结果

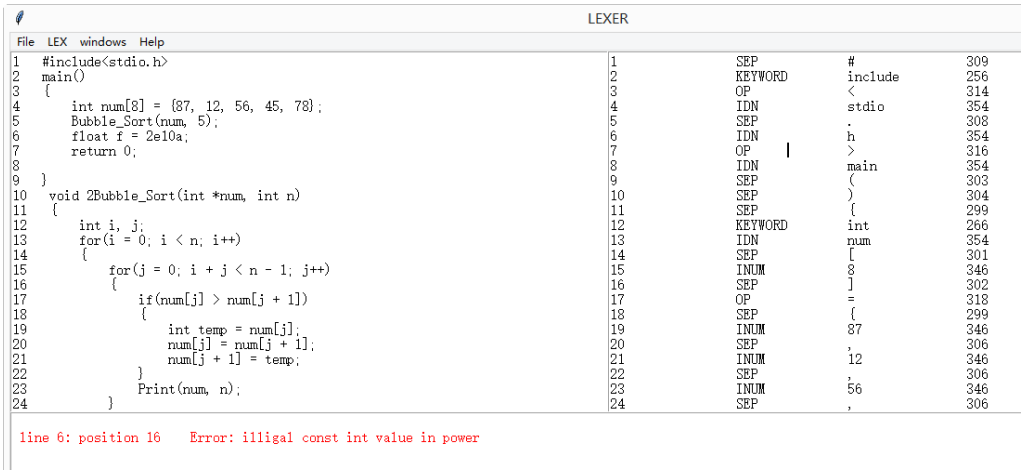

图 3-1 冒泡排序结果

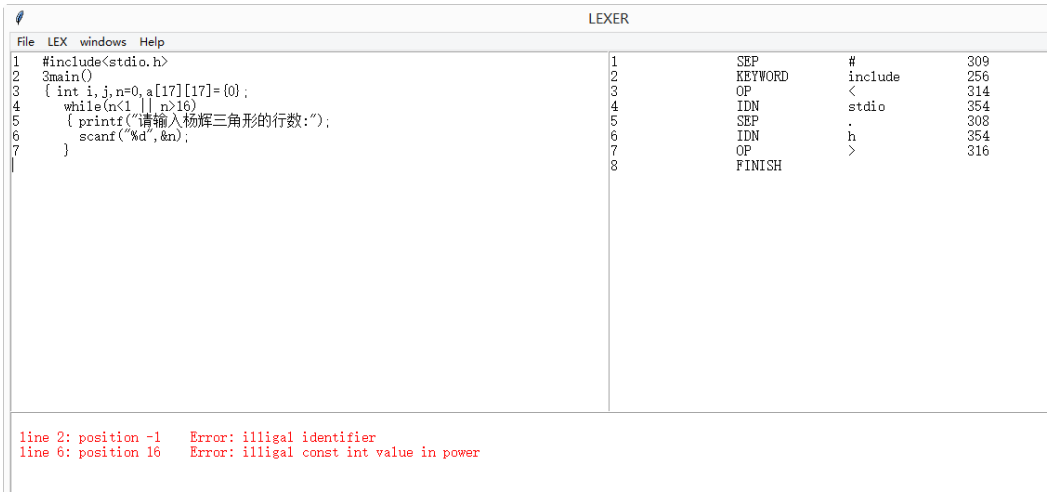

图 3-2 杨辉三角结果

3.3 结果分析

采用两个 C 代码分别进行测试，第一是冒泡排序，其中存在错误如下：

根据上图可以看出，程序正确的识别出了源程序中的词法错误，并提示出来。同时，对于分析结果四元组的形式进行展示。

附　　录

附录 A 程序代码

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import sys
from tkinter import filedialog
from tkinter import *

KEYWORD_LIST = ['if', 'else', 'while', 'break', 'continue', 'for',
'double', 'int', 'float', 'long', 'short', 'bool','switch', 'case',
'return', 'void','include']

SEPARATOR_LIST = ['{', '}', '[', ']', '(', ')', '~', ',', ';', '.', '?',
':', ' ','#']

OPERATOR_LIST = ['+', '++', '-', '--', '+=', '-=', '*', '*=', '%', '%=',
'->', '|', '||', '|=','/', '/=', '>', '<', '>=', '<=', '=', '==', '!=',
'!', '&', '**']

CATEGORY_DICT = {
    # KEYWORD
    "include":256,
    "far": 257,
    "near": 258,
    "pascal": 259,
    "register": 260,
    "asm": 261,
    "cdecl": 262,
    "huge": 263,
    "auto": 264,
    "double": 265,
    "int": 266,
    "struct": 267,
    "break": 268,
    "else": 269,
    "long": 270,
    "switch": 271,
```

```
"case": 272,
"enum": 273,
"register": 274,
"typedef": 275,
"char": 276,
"extern": 277,
"return": 278,
"union": 279,
"const": 280,
"float": 281,
"short": 282,
"unsigned": 283,
"continue": 284,
"for": 285,
"signed": 286,
"void": 287,
"default": 288,
"goto": 289,
"sizeof": 290,
"volatile": 291,
"do": 292,
"if": 293,
"while": 294,
"static": 295,
"interrupt": 296,
"sizeof": 297,
"NULL": 298,
# SEPARATOR
"{": 299,
"}": 300,
"[": 301,
"]": 302,
"(": 303,
")": 304,
"~": 305,
",": 306,
";": 307,
".": 308,
"#": 309,
"?": 310,
":": 311,
# OPERATOR
"<<": 312,
">>": 313,
```

```
    "<": 314,
    "<=": 315,
    ">": 316,
    ">=": 317,
    "=": 318,
    "==": 319,
    "|": 320,
    "||": 321,
    "|=": 322,
    "^": 323,
    "^=": 324,
    "&": 325,
    "&&": 326,
    "&=": 327,
    "%": 328,
    "%=": 329,
    "+": 330,
    "++": 331,
    "+=": 332,
    "-": 333,
    "--": 334,
    "-=": 335,
    "->": 336,
    "/": 337,
    "/=": 338,
    "*": 339,
    "*=": 340,
    "!": 341,
    "!=": 342,
    "sizeof": 343,
    "<<=": 344,
    ">>=": 345,
    "inum": 346,
    "int16": 347,
    "int8": 348,
    "char": 350,
    "string": 351,
    "bool": 352,
    "fnum": 353,
    "IDN": 354,
    '**': 355
}

current_row = -1
```

```
current_line = 0
out_line = 1
global errorTest

# 读取一个字符
def getchar(input_str):
    global current_row
    global current_line
    current_row += 1
    if (current_row == len(input_str[current_line])):
        current_line += 1
        current_row = 0
    if current_line == len(input_str):
        return 'FINISH'
    return input_str[current_line][current_row]
# 退格
def ungetchar(input_str):
    global current_row
    global current_line
    current_row = current_row - 1
    if current_row < 0:
        current_line = current_line - 1
        current_row = len(input_str[current_row]) - 1
    return input_str[current_line][current_row]

# 错误报告
def error(msg, line=None, row=None):
    global out_line
    global errorTest
    if line is None:
        line = current_line + 1
    if row is None:
        row = current_row + 1
    errorTest.insert(str(out_line) + '.end', " line "+ str(line) + ':
position ' + str(row-4) + '\t\t\tError: ' + msg)
    errorTest.insert(str(out_line) + '.end', "\n")
    out_line = out_line + 1

# 扫描器
def scanner(input_str):
    global current_line
    global current_row

    current_char = getchar(input_str)
```

```
    if current_char == 'FINISH':
        return ('FINISH', '', '')
    if current_char.strip() == '':
        return
    # 数字
    if current_char.isdigit():
        int_value = 0
        while current_char.isdigit():
            int_value = int_value * 10 + int(current_char)
            current_char = getchar(input_str)

        if current_char not in OPERATOR_LIST and current_char not in
SEPARATOR_LIST and current_char != 'e':
            line = current_line + 1
            row = current_row + 1
            ungetchar(input_str)
            error('illigal identifier', line, row)
            return ('FINISH', '', '')
            return None
        if current_char != '.' and current_char != 'e':
            ungetchar(input_str)
            return ('INUM', int_value, CATEGORY_DICT['inum'])
        if current_char == 'e':
            power_value = str(int_value) + 'e'
            current_char = getchar(input_str)
            if current_char == '+' or current_char == '-':
                power_value += current_char
                current_char = getchar(input_str)
            while current_char.isdigit():
                power_value += current_char
                current_char = getchar(input_str)
            if current_char not in OPERATOR_LIST and current_char not in
SEPARATOR_LIST:
                line = current_line + 1
                row = current_row + 1
                ungetchar(input_str)
                error('illigal const int value in power', line, row)
                return ('FINISH', '', '')
                return None

            ungetchar(input_str)
            return ('INUM', power_value, CATEGORY_DICT['inum'])
        if current_char == '.':
            float_value = str(int_value) + '.'
```

```
            current_char = getchar(input_str)
            while current_char.isdigit():
                float_value += current_char
                current_char = getchar(input_str)
            if current_char not in OPERATOR_LIST and current_char not in
SEPARATOR_LIST or current_char == '.':
                line = current_line + 1
                row = current_row + 1
                ungetchar(input_str)
                error('illigal const float value', line, row)
                return ('FINISH', '', '')
                return None
            ungetchar(input_str)
            return ('FNUM', float_value, CATEGORY_DICT['fnum'])
    # 标识符
    if current_char.isalpha() or current_char == '_':
        string = ''
        while current_char.isalpha() or current_char.isdigit() or
current_char == '_' and current_char != ' ':
            string += current_char
            current_char = getchar(input_str)
            if current_char == 'FINISH':
                break
        ungetchar(input_str)
        if string in KEYWORD_LIST:
            return ("KEYWORD", string, CATEGORY_DICT[string])
        else:
            return ('IDN', string, CATEGORY_DICT['IDN'])
    # 注释
    if current_char == '\"':
        str_literal = ''
        line = current_line + 1
        row = current_row + 1

        current_char = getchar(input_str)
        while current_char != '\"':
            str_literal += current_char
            current_char = getchar(input_str)
            if current_char == 'FINISH':
                error('missing terminating \"', line, row)
                current_line = line
                current_row = row
                return ('FINISH', '', '')
        return ('STRING_LITERAL', str_literal, CATEGORY_DICT['string'])
```

```
if current_char == '/':
    next_char = getchar(input_str)
    line = int(current_line) + 1
    row = int(current_row) + 1
    if next_char == '*':
        comment = ''
        next_char = getchar(input_str)
        while True:
            if next_char == 'FINISH':
                error('unteminated /* comment', line, row)
                return ('FINISH', '', '')
            if next_char == '*':
                end_char = getchar(input_str)
                if end_char == '/':
                    return None
                if end_char == 'FINISH':
                    error('unteminated /* comment', line, row)
                    return ('FINISH', '', '')
            comment += next_char
            next_char = getchar(input_str)
    else:          #/=
        ungetchar(input_str)
        op = current_char
        current_char = getchar(input_str)
        if current_char in OPERATOR_LIST:
            op += current_char
        else:   # /
            ungetchar(input_str)
        return ('OP', op, CATEGORY_DICT[op])

if current_char in SEPARATOR_LIST:
    return ('SEP', current_char, CATEGORY_DICT[current_char])

if current_char in OPERATOR_LIST:
    op = current_char
    current_char = getchar(input_str)
    if(current_char in OPERATOR_LIST):
        op += current_char
    else:
        ungetchar(input_str)
    return ('OP', op, CATEGORY_DICT[op])
else:
    error('unknown character: ' + current_char)
```

```
def fileloader():
    global root
    code.delete(1.0, END)
    root.filename =  filedialog.askopenfilename(
                     initialdir = "/",
                     title = "Select file",
                     filetypes = (("c files","*.c"),
                     ("all files","*.*")))
    fin = open(root.filename, "r")
    input_file = fin.read()
    input_line = input_file.split("\n")

    out_line = 1
    for each in input_line:
        code.insert(str(out_line) + '.end', str(out_line)+ "   " + each)
        code.insert(str(out_line) + '.end', "\n")
        out_line = out_line + 1
    fin.close()

#  词法分析
def lexer_analysis(input_str):
    global current_row
    global current_line
    global out_line
    current_row = -1
    current_line = 0
    analysis_result = []
    r = ['','','']
    while (1):
        r = scanner(input_str)
        if r is not None:
            analysis_result.append(str(r[0]) + "\t\t" + str(r[1]) + "\t\t" +
str(r[2]))
            if  (r[0] == 'FINISH'):
                return analysis_result
    return  analysis_result

#  按键触发函数
def lexer():
    global out_line
    input_str = []
```

```
    analysis.delete(1.0, END)
    input_raw = code.get(1.0, END)
    input_str = input_raw.split("\n")
    temp = []
    for i in range(len(input_str)):
        input_str[i]=input_str[i][3:]      #remove the line number
        if (input_str[i]!= ""):
            temp.append(input_str[i])

    out_line = 0
    result = lexer_analysis(temp)
    for each in result:
        analysis.insert(str(out_line) + '.end', str(out_line) + " \t\t "+
each)
        analysis.insert(str(out_line) + '.end', "\n")
        out_line = out_line + 1


# 界面展示
def pre_interface():
    global root
    global code
    global analysis
    global errorTest
    root = Tk()
    menubar = Menu(root)
    filemenu = Menu(menubar, tearoff=0)
    filemenu.add_command(label="Open", command=fileloader,font = 26)
    filemenu.add_command(label="Save" ,font = 26)
    filemenu.add_command(label="Exit", command=root.quit,font = 26)
    menubar.add_cascade(label="File", menu=filemenu,font = 26)

    lexmenu = Menu(menubar, tearoff=0)
    lexmenu.add_command(label="lex", command=lexer,font = 26)
    menubar.add_cascade(label="LEX", menu=lexmenu,font = 26,command =
root.quit)

    windowsmenu = Menu(menubar, tearoff=0)
    windowsmenu.add_command(label="fullscreen",
command=toggle_fullscreen,font = 26)
    menubar.add_cascade(label="windows", menu=windowsmenu,font = 26)

    helpmenu = Menu(menubar, tearoff=0)
    helpmenu.add_command(label="Help Index",font = 26)
```

```
    menubar.add_cascade(label="Help", menu=helpmenu,font = 26)
    root.config(menu=menubar)

    code = Text(root,   font=26)
    analysis = Text(root, font=26)
    errorTest = Text(root, width = 10, font=26,foreground="red")
    root.title("LEXER")
    errorTest.pack(fill = X,side=BOTTOM,expand = YES)
    code.pack(side =LEFT,fill = Y,expand = YES)
    analysis.pack(side=RIGHT,fill = Y,expand = YES)

    root.bind("<F11>", toggle_fullscreen)

    root.mainloop()


def toggle_fullscreen(event=None):   # 增加全屏属性
    root.state("zoomed")


def main():
    pre_interface()

if __name__ == '__main__':
    main()
```

附录 B 测试代码

```
/****************ðŝŅʙ**********************/
#include<stdio.h>
main()
{
    int num[8] = {87, 12, 56, 45, 78};
    Bubble_Sort(num, 5);
    float f = 2e10a;
    return 0;

}
void 2Bubble_Sort(int *num, int n)
{
    int i, j;
    for(i = 0; i < n; i++)
    {
```

```
        for(j = 0; i + j < n - 1; j++)
        {
            if(num[j] > num[j + 1])
            {
                int temp = num[j];
                num[j] = num[j + 1];
                num[j + 1] = temp;
            }
            Print(num, n);
        }
    }
    return;
}
/******************杨辉三角*****************/
#include<stdio.h>
3main() {
    int i,j,n=0,a[17][17]= {0};
    while(n<1 || n>16) {
        printf("请输入杨辉三角形的行数:");
        scanf("%d",&n);
    }
```