《编译原理》课设

文 档

学号：\_\_\_\_\_\_16041049\_\_\_\_\_\_\_\_\_

姓名：\_\_\_\_\_\_陈天宇\_\_\_\_\_\_\_\_\_\_\_

2018年 12 月 9 日

目录

[一、 需求说明 3](#_Toc532215574)

[1. 文法说明 3](#_Toc532215575)

[2. 目标代码说明 4](#_Toc532215576)

[二、 详细设计 5](#_Toc532215577)

[1. 程序结构 5](#_Toc532215578)

[2. 类/方法/函数功能 6](#_Toc532215579)

[三、 操作说明 10](#_Toc532215580)

[1. 运行环境 10](#_Toc532215581)

[2. 操作步骤 10](#_Toc532215582)

[四、 测试报告 12](#_Toc532215583)

[1. 测试程序及测试结果 12](#_Toc532215584)

[2. 测试结果分析 24](#_Toc532215585)

[五、 总结感想 25](#_Toc532215586)

## 需求说明

### 文法说明

关键字: const, int, void, if, else, while, main, return, printf, scanf

＜加法运算符＞ ::= +｜－

＜乘法运算符＞ ::= \* ｜／

＜关系运算符＞ ::= ＜｜＜＝｜＞｜＞＝｜!＝｜＝＝

＜字符＞ ::= ＿｜a｜．．．｜z｜A｜．．．｜Z

＜数字＞ ::= 0｜＜非零数字＞

＜非零数字＞ ::= 1｜．．．｜9

＜专用符号＞ ::= (｜)｜{｜}｜，｜；｜＝

<字符串> ::= "｛<合法字符> }"

＜程序＞ ::= 〔＜常量说明部分＞〕〔＜变量说明部分＞〕｛＜函数定义部分＞｝＜主函数＞

＜常量说明部分＞ ::= const ＜常量定义＞｛,＜常量定义＞};

＜常量定义＞ ::= ＜标识符＞＝＜整数＞

＜整数＞ ::= 〔＋｜－〕＜非零数字＞｛＜数字＞｝｜０

＜标识符＞ ::= ＜字符＞｛＜字符＞｜＜数字＞｝

＜声明头部＞ ::= int　＜标识符＞

＜变量说明部分＞ ::= ＜声明头部＞｛，＜标识符＞｝；

＜函数定义部分＞ ::= （＜声明头部＞｜void ＜标识符＞）＜参数＞＜复合语句＞

＜复合语句＞ ::= ‘{’〔＜常量说明部分＞〕〔＜变量说明部分＞〕＜语句序列＞‘}’

＜参数＞ ::= ‘(’＜参数表＞‘)’

＜参数表＞ ::= int ＜标识符＞｛，int ＜标识符＞} | 空

//参数表可以为空

＜主函数＞ ::= ( void ｜int ) main ＜参数＞＜复合语句＞

＜表达式＞ ::= 〔＋｜－〕＜项＞｛＜加法运算符＞＜项＞｝

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

＜因子＞ ::= ＜标识符＞｜‘（’＜表达式＞‘）’｜＜整数＞｜＜函数调用语句＞

＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜‘{’<语句序列>‘}’｜＜函数调用语句＞;

｜＜赋值语句＞; | <返回语句>;｜＜读语句＞;｜＜写语句＞;｜＜空＞

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞

＜条件语句＞ ::= if‘（’＜条件＞‘）’＜语句＞〔else＜语句＞〕

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

＜循环语句＞ ::= while‘（’＜条件＞‘）’＜语句＞

＜函数调用语句＞ ::= ＜标识符＞‘（’＜值参数表＞‘）’

＜值参数表＞ ::= ＜表达式＞｛，＜表达式＞｝｜＜空＞

＜语句序列＞ ::= ＜语句＞｛＜语句＞｝

＜读语句＞ ::= scanf‘(’＜标识符＞‘）’

＜写语句＞ ::= printf‘(’[<字符串>,][＜expression ＞]‘）’

　　　　　 //当出现字符串时，就加印字符串, 之后打印表达式的值；

＜返回语句＞ ::= return [ ‘(’＜表达式＞’)’ ]

### 目标代码说明

* **P-code**

一种栈式机的语言，这类栈式机没有累加器和通用寄存器，有四个控制寄存器（指令寄存器I， 指令寄存器P，栈式寄存器T和基址寄存器B），运算都在栈顶进行。

* **指令格式**

|  |  |  |
| --- | --- | --- |
| **f** | **x** | **y** |

* **f： 操作码**
* **x： x域，第一个操作数**
* **y： y域，第二个操作数**
* **设计使用指令：**

**表格 1 指令表**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **助记符** | **f** | **x** | **y** | **功能** |
| LDA | 0 | x | y | 把变量地址装入栈顶（x为层次，y为相对地址） |
| LOD | 1 | x | y | 装入值 |
| DIS | 3 | x | y | 更新display |
| JMP | 10 |  | y | 无条件转移到y |
| JPC | 11 |  | y | 如栈顶内容为假，转移到y |
| MKS | 18 |  | y | 标记栈，调用函数前使用，保存现场 |
| CAL | 19 |  | y | 调用用户过程或函数 |
| LDC | 24 |  | y | 装入字面常量 |
| RED | 27 |  | y | 读入整型数 |
| WRS | 28 |  | y | 写字符串 |
| WRW | 29 |  | y | 写栈顶整型数 |
| EXP | 32 |  |  | 退出过程 |
| EXF | 33 |  |  | 退出函数 |
| STO | 38 |  |  | 将栈顶内容存入以栈顶次高元为地址的单元 |
| EQL | 45 |  |  | 整型相等比较 |
| NEQ | 46 |  |  | 整型不等比较 |
| LSS | 47 |  |  | 整型小于比较 |
| LER | 48 |  |  | 整型小于等于比较 |
| GRT | 49 |  |  | 整型大于比较 |
| GEQ | 50 |  |  | 整型大于等于比较 |
| ADD | 52 |  |  | 整型加 |
| SUB | 53 |  |  | 整型减 |
| MUL | 57 |  |  | 整型乘 |
| DIV | 48 |  |  | 整型除 |

## 详细设计

### 程序结构

C0 Online编译程序采用以递归下降子程序法为语法分析核心，进行一遍扫描的编译。词法分析和语法分析程序相互独立，语法分析程序将词法分析程序的结果作为输入。在语法分析的同时，提供了出错报告的功能。在源程序没有错误，编译通过的情况下，将生成对应的P-code代码。

程序采用了Django作为模板引擎来渲染前端，用户在网页端提交代码或者提交代码文件之后，会向服务器发起POST请求，POST调用C0编译器程序对代码进行处理。处理后的结果将生成H5表格代码，通过Django引擎返回request，浏览器渲染页面，将表格呈现给用户。

首先，词法分析special\_lexer继承自词法分析作业中的C0lexer，因而初始化的时候先调用类C0lexer生成父类对象，如果是本地对象，需要指明测试代码文件的路径（默认为网络服务器调用服务），接着完成一系列私有属性的初始化。

1. **def** \_\_init\_\_(self, INPUT\_FILE\_NAME: str):
2. '''''
3. '''
4. self.TOKEN = ""
5. self.INPUT\_FILE\_NAME = INPUT\_FILE\_NAME
6. self.SOURCE\_TEXT = ""
7. self.p = 0
8. self.RESULT = []
9. self.REPLY = ""
10. self.KEYWORDS = ["const", "int", "void", "if", "else",
11. "while", "main", "return", "printf", "scanf"]

然后，我们开始对输入代码进行词法分析，如果是本地的词法分析的话，可以调用word\_analyze接口，如果是服务器端进程，则调用web\_word\_analyze接口，这两个接口都是自动执行getsym()函数直到出错或者读到文件结尾。

1. **def** word\_analyze(self):
2. '''''
3. 自动词法分析至出错或到文件末尾
4. '''
5. self.read()
6. length = len(self.SOURCE\_TEXT)
7. **while** self.p < length **and** self.getsym():
8. **continue**
9. **if** self.p >= length:
10. **print**("WordAnalysis Completed!")
11. **else**:
12. **print**("Unknown invalid syntax!")
14. **def** web\_word\_analyze(self):
15. '''''
16. 用于服务器的词法分析接口
17. '''
18. length = len(self.SOURCE\_TEXT)
19. self.RESULT = []
20. **while** self.p < length **and** self.getsym():
21. **continue**
22. **if** self.p >= length:
23. self.REPLY += "WordAnalysis Completed!" + "\n"
24. **else**:
25. self.REPLY += "Unknown invalid syntax!" + "\n"

getsym（）函数每次会对接下来的字符串进行分析处理，如果发现有符合条件的单词，就会把它和分析结果一起作为一条记录增加到RESULT这个list对象中。

当getsym读到保留字时，就会查找保留字字典，看该保留字是否有定义，如果有定义就判定为保留字，否则就把这个字符串识别为一个标识符。由于需求文法中的数字只有整型数字，因而程序也只识别整型数。对于字符串，则看字符串前后是否存在约定的专用符号，如果满足，则判断为字符串。其他类型的运算符合专用符号同理。

在分析完成后，lexer对象输出RESULT，可以调用output（）接口输出到指定文件，也可以直接将结果输入给下一个对象norm\_C0\_compiler()，compiler执行s\_program()过程，对程序进行递归下降分析，如果分析成功，就可以调用一系列表格生成接口输出HTML5代码，或者在shell里直接打印表格。如果分析失败，也可以输出详细的编译时错误信息（未定义，重复定义， 语法错误等）。

### 类/方法/函数功能

*class*grammar\_analysis.utils.**norm\_C0\_compiler**

C0编译器

**\_curword**()

返回当前指针所指的单词

**\_error**(*hint: str*)

添加报错信息

**\_gen\_Pcode**(*code: list*)

生成一行Pcode

**\_getword**()

返回下一个单词

**\_insert\_const**(*name*, *value*)

在rconst中插入常数

**\_insert\_display**(*name*, *typ*, *value*, *address*, *lev*)

在display区中插入一条记录

**\_lookup\_abp**(*lev*, *name*)

在某一个abp内非递归的查找一个变量

**\_lookup\_const**(*word\_name: str*)

查询rconst表，获取常量对应的值

**\_lookup\_variable**(*name: str*)

在display区查找一个变量

**\_new\_lev**(*pre\_lev: int*)

在display区中插入一个新的lev

**\_zipper\_fill**(*JPC\_id*, *JPC\_code*)

拉链回填

**print\_Pcode**()

将Pcode打印成表

**print\_display**()

打印变量表

**print\_error**()

打印报错信息

**print\_rconst**()

打印常量表

**report\_result**()

打印编译结果

**s\_assignment\_statement**()

＜赋值语句＞ ::= ＜标识符＞＝＜表达式＞

**s\_compound\_statement**()

＜复合语句＞ ::= ‘{’〔＜常量说明部分＞〕〔＜变量说明部分＞〕＜语句序列＞‘}’

**s\_condition**()

＜条件＞ ::= ＜表达式＞＜关系运算符＞＜表达式＞｜＜表达式＞

**s\_condition\_statement**()

＜条件语句＞ ::= if‘（’＜条件＞‘）’＜语句＞〔else＜语句＞〕

**s\_const\_definition**()

＜常量定义＞ ::= ＜标识符＞＝＜整数＞

**s\_constant\_description**()

＜常量说明部分＞ ::= const ＜常量定义＞｛,＜常量定义＞};

**s\_declaration\_head**()

＜声明头部＞ ::= int　＜标识符＞

**s\_expression**()

＜表达式＞ ::= 〔＋｜－〕＜项＞｛＜加法运算符＞＜项＞｝

**s\_factor**()

＜因子＞ ::= ＜标识符＞｜‘（’＜表达式＞‘）’｜＜整数＞｜＜函数调用语句＞

**s\_function\_call\_statement**()

＜函数调用语句＞ ::= ＜标识符＞‘（’＜值参数表＞‘）’

**s\_function\_declaration**()

＜函数定义部分＞ ::= （＜声明头部＞｜void ＜标识符＞）＜参数＞＜复合语句＞

**s\_item**()

＜项＞ ::= ＜因子＞{＜乘法运算符＞＜因子＞}

**s\_main\_function**()

＜主函数＞ ::= ( void ｜int ) main ＜参数＞＜复合语句＞

**s\_param**()

＜参数＞ ::= ‘(’＜参数表＞‘)’

**s\_param\_list**()

＜参数表＞ ::= int ＜标识符＞｛，int ＜标识符＞} | 空

//参数表可以为空

**s\_program**()

＜程序＞ ::= 〔＜常量说明部分＞〕〔＜变量说明部分＞〕｛＜函数定义部分＞｝＜主函数＞

**s\_read**()

<读语句> -> scanf’(‘<标识符>’)’

**s\_return**()

＜返回语句＞ ::= return [ ‘(’＜表达式＞’)’ ]

**s\_statement**()

＜语句＞ ::= ＜条件语句＞｜＜循环语句＞｜‘{’<语句序列>‘}’｜＜函数调用语句＞;

｜＜赋值语句＞; | <返回语句>;｜＜读语句＞;｜＜写语句＞;｜＜空＞

**s\_statement\_series**()

＜语句序列＞ ::= ＜语句＞｛＜语句＞｝

**s\_value\_param\_list**(*display\_index: int*)

＜值参数表＞ ::= ＜表达式＞｛，＜表达式＞｝｜＜空＞

**s\_variable\_description**()

＜变量说明部分＞ ::= ＜声明头部＞｛，＜标识符＞｝；

**s\_while\_statement**()

＜循环语句＞ ::= while‘（’＜条件＞‘）’＜语句＞

**s\_write**()

＜写语句＞ ::= printf‘(’[<字符串>,][＜expression ＞]‘）’

当出现字符串时，就加印字符串, 之后打印表达式的值；

**web\_output\_Pcode\_table**()

生成H5代码

**web\_output\_display\_table**()

生成H5表格代码

**web\_output\_error\_table**()

生成H5代码

**web\_output\_rconst\_table**()[¶](file:///D:\%E5%8C%97%E8%88%AA%E5%AD%A6%E4%B9%A0\%E5%A4%A7%E4%B8%89%E4%B8%8A\%E7%BC%96%E8%AF%91%E5%8E%9F%E7%90%86%E8%AF%BE%E7%A8%8B%E8%AE%BE%E8%AE%A1\OnlineC0\docs\_build\html\grammar_analysis\compiler.html#grammar_analysis.utils.norm_C0_compiler.web_output_rconst_table)

生成H5代码

*class*grammar\_analysis.utils.**special\_lexer**(*INPUT\_FILE\_NAME='web'*)

词法分析器,是C0lexer的子类

**\_error**()

增加报错信息

**\_isNewLine**()

判断是否是一个新的行

**\_retract**()

回退一个字符

**clearToken**()

清空目前已经读取的字符缓存

**curChar**()

返回当前指针指向的字符

**error**()

出错提示

**error\_report**()

错误分析报告

**getchar**()

将当前字符加入TOKEN末尾 文件指针+1

**isDigit**()

判断当前字符是否为数字

**isLetter**()

判断当前字符为字母

**isNewLine**()

**isSpace**()

判断当前字符是否为’/r’或者”/n”

**output**()

输出词法分析结果文件

**print\_result**()

打印结果

**read**()

如果以文件形式输入，调用该函数执行读操作

**tobin**(*x: int*)

十进制转二进制

**web\_input**(*SOURCE\_TEXT*)

从表单输入文本

**web\_reply**()

用于服务器的分析应答接口

**web\_word\_analyze**()

用于服务器的词法分析接口

**word\_analyze**()

自动词法分析至出错或到文件末尾

源代码可访问我的github获得：<https://github.com/Tarpelite/OnlineC0>

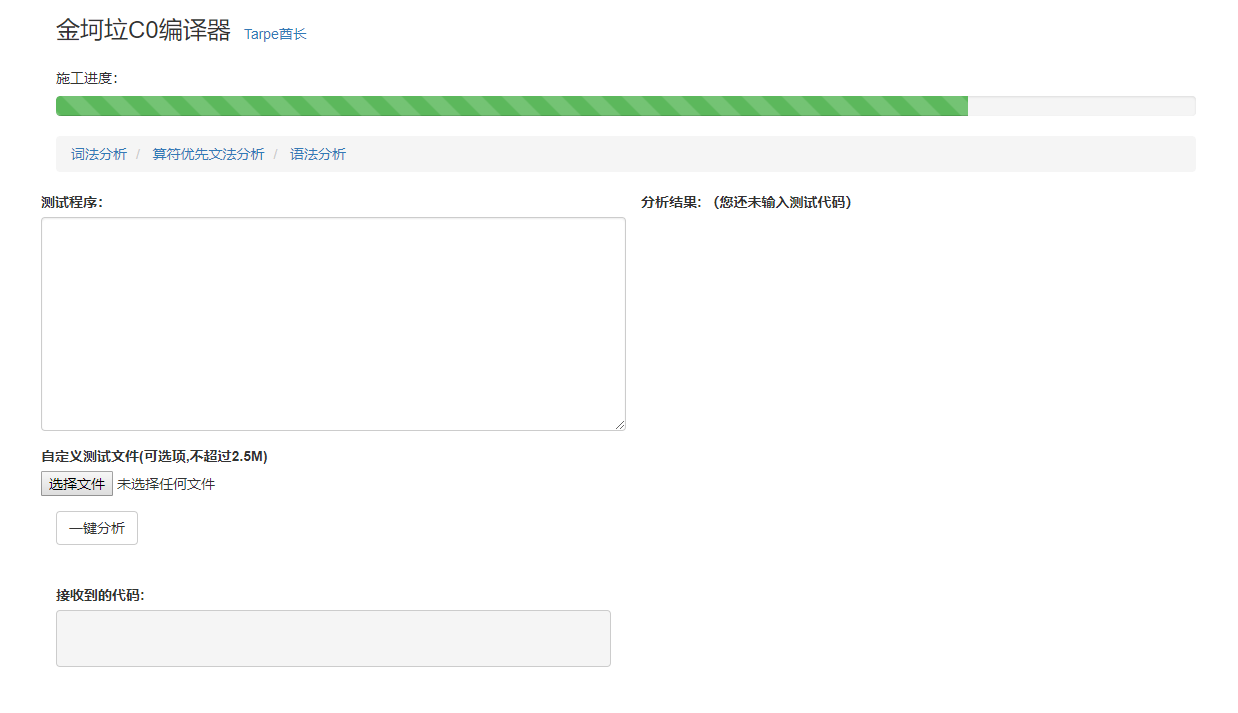
## 操作说明

### 运行环境

浏览器：chrome 66及以上

### 操作步骤

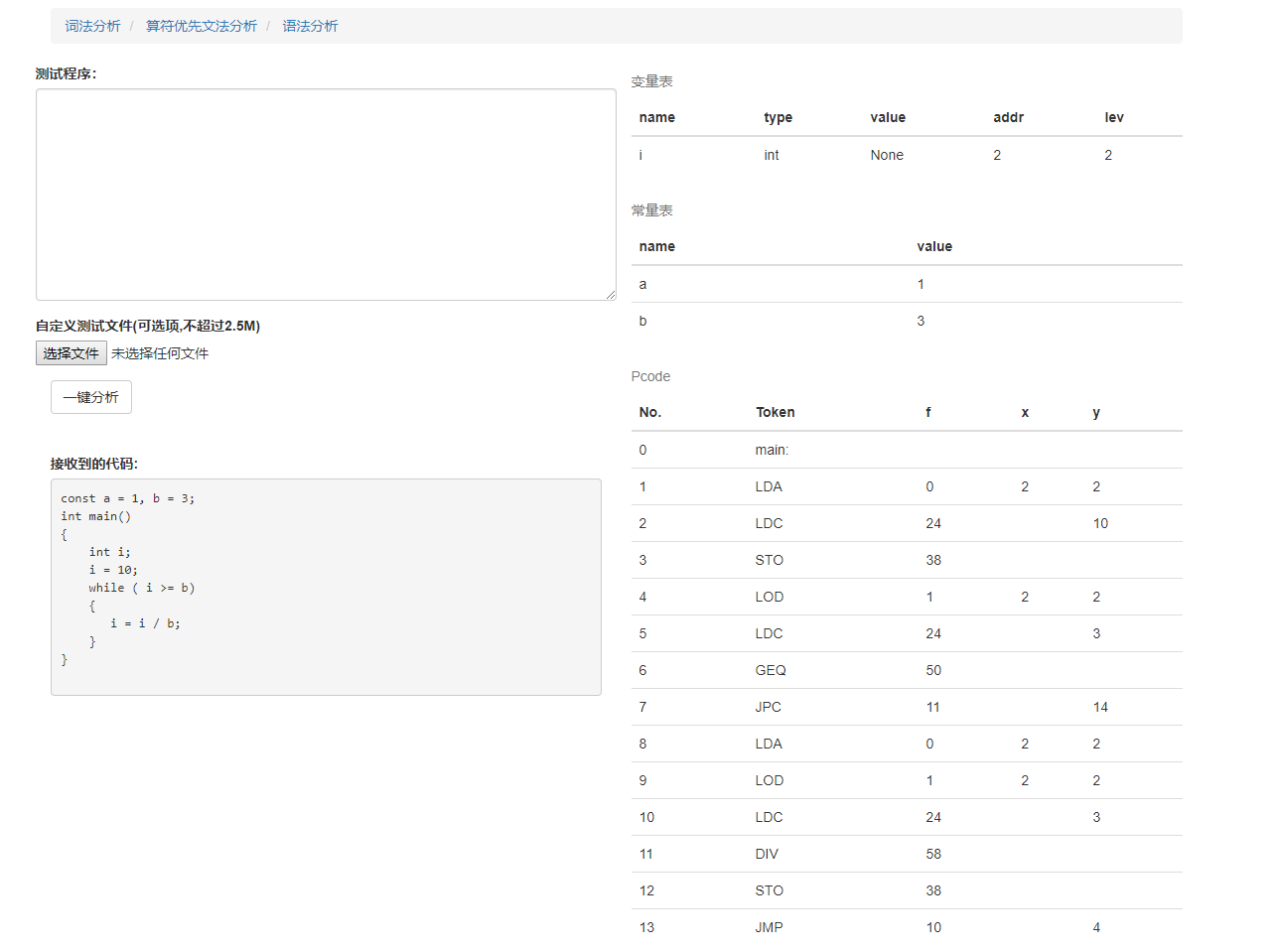
访问网址：<http://www.buaatech.top:8080/gramma/>



在测试程序中输入C0代码或者提交代码文件，然后点击一键分析。



然后就可以看到运行之后的输出结果了。



## 测试报告

### 测试程序及测试结果

* **Test 1：**

**int main()**

**{**

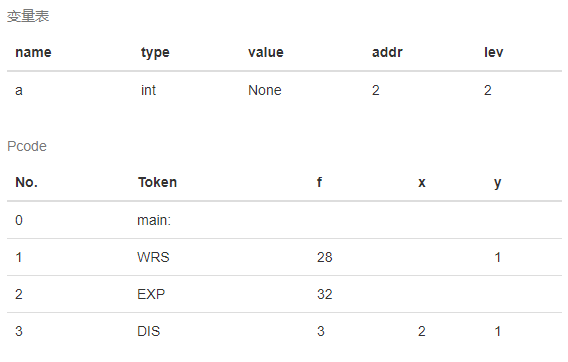
**int a;**

**printf("abcd");**

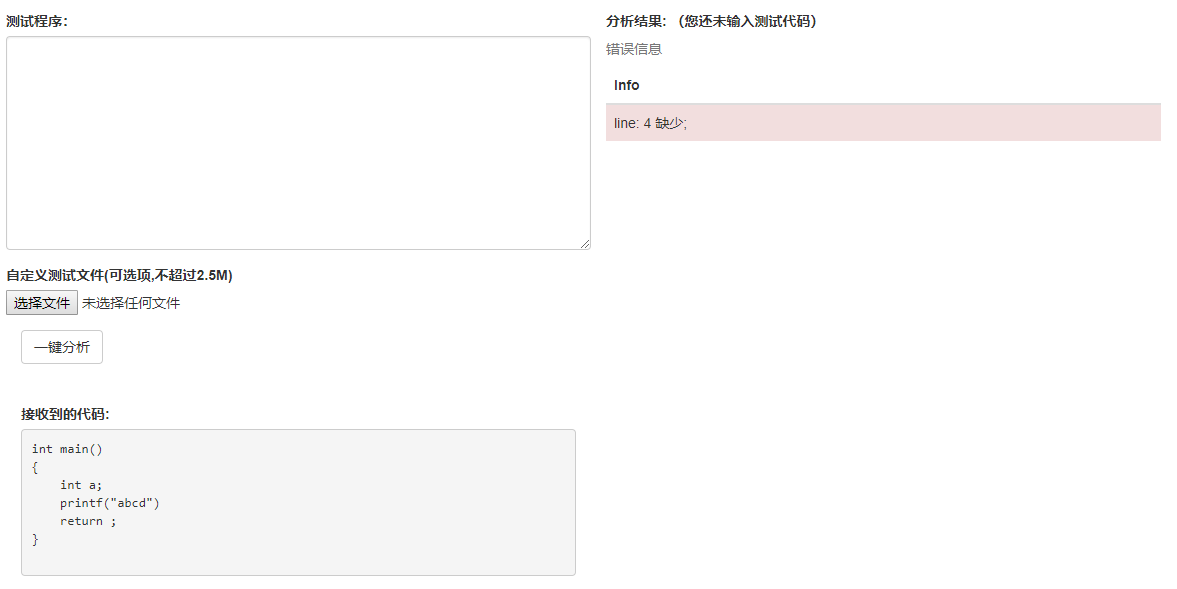
**return ;**

**}**

**Test 1 Result：**



**删去第4行分号时：**



* **Test 2:**

**const a = 1, b = 2;**

**int add(int i, int j)**

**{**

**return (i+j);**

**}**

**int sub(int k, int s)**

**{**

**return (add(k, s) - 2\*s);**

**}**

**int main()**

**{**

**int c;**

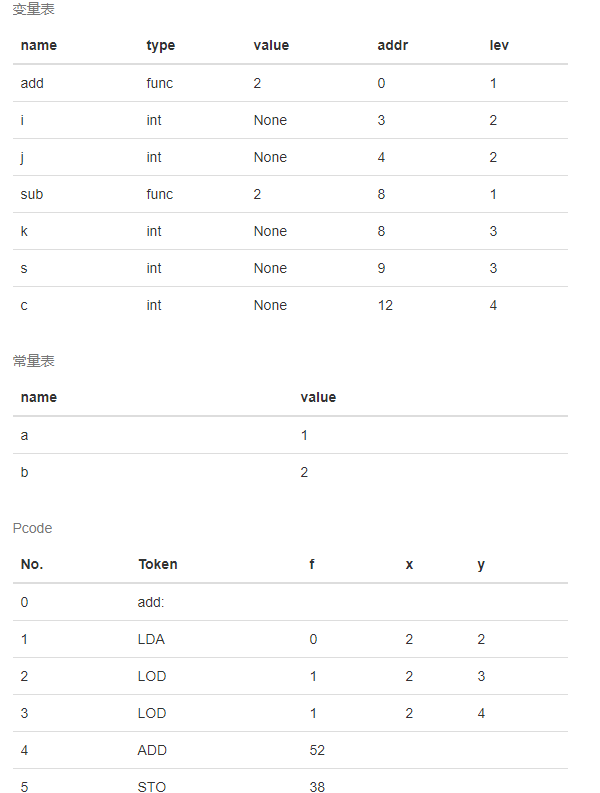
**c = add(a, b);**

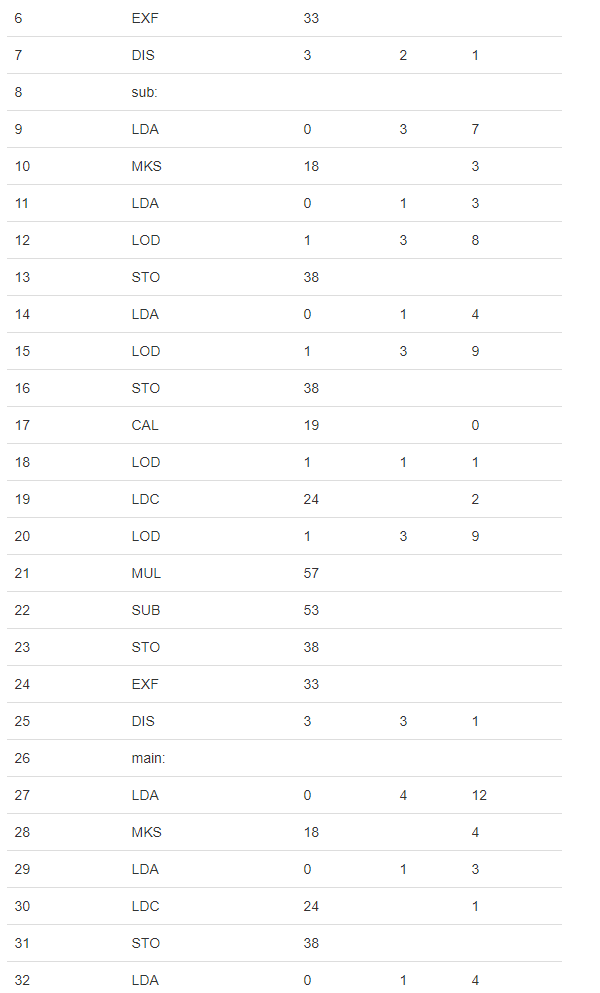
**printf("%d",c);**

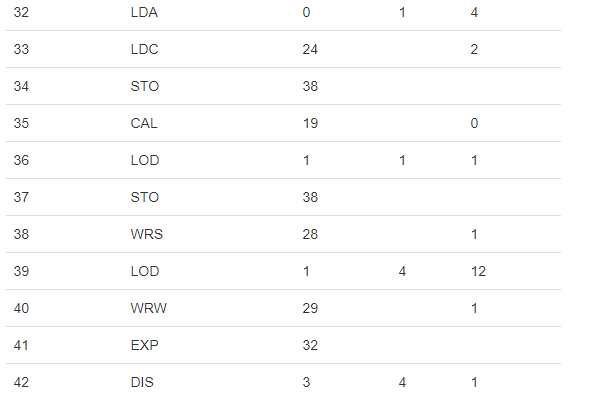
**return;**

**}**

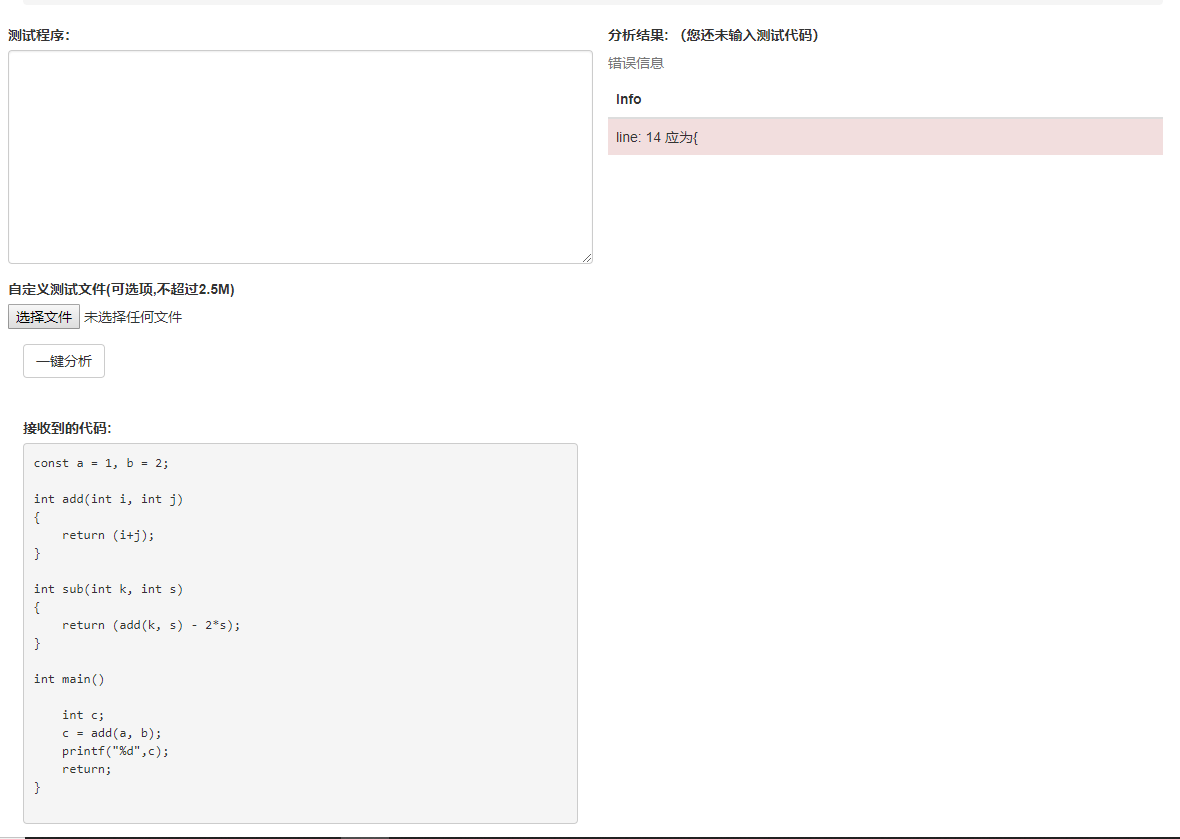
**Test 2 Result:**







**当删去一个括号之后：**



* **Test 3:**

**const a = 1, b = 2;**

**int c, d;**

**int main()**

**{**

**c = 2;**

**scanf(d);**

**if ( c>1 )**

**{**

**printf("%d", a);**

**if ( d > 0)**

**{**

**printf(d);**

**}**

**else**

**{**

**printf(c);**

**}**

**}**

**return;**

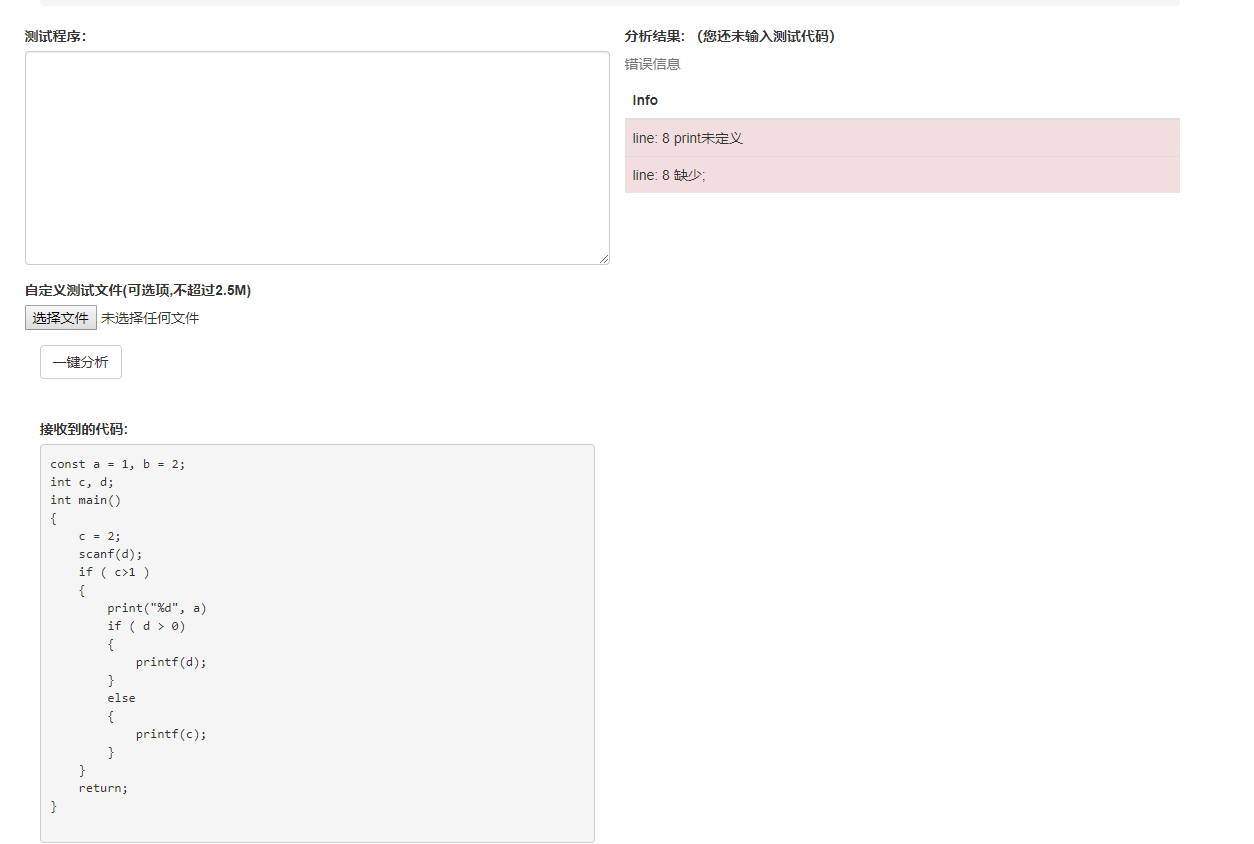
**}**

**Test 3 Result:**

| 变量表 | | | | | |
| --- | --- | --- | --- | --- | --- |
| **name** | **type** | **value** | | **addr** | **lev** |
| c | int | None | | 0 | 1 |
| d | int | None | | 1 | 1 |
| 常量表 | | | | | |
| **name** | | | **value** | | |
| a | | | 1 | | |
| b | | | 2 | | |

| Pcode | | | | |
| --- | --- | --- | --- | --- |
| **No.** | **Token** | **f** | **x** | **y** |
| 0 | main: |  |  |  |
| 1 | LDA | 0 | 1 | 0 |
| 2 | LDC | 24 |  | 2 |
| 3 | STO | 38 |  |  |
| 4 | LDA | 0 | 1 | 1 |
| 5 | RED | 27 |  | 1 |
| 6 | LOD | 1 | 1 | 0 |
| 7 | LDC | 24 |  | 1 |
| 8 | GRT | 49 |  |  |
| 9 | JPC | 11 |  | 22 |
| 10 | WRS | 28 |  | 1 |
| 11 | LDC | 24 |  | 1 |
| 12 | WRW | 29 |  | 1 |
| 13 | LOD | 1 | 1 | 1 |
| 14 | LDC | 24 |  | 0 |
| 15 | GRT | 49 |  |  |
| 16 | JPC | 11 |  | 20 |
| 17 | LOD | 1 | 1 | 1 |
| 18 | WRW | 29 |  | 1 |
| 19 | JMP | 10 |  | 21 |
| 20 | LOD | 1 | 1 | 0 |
| 21 | WRW | 29 |  | 1 |
| 22 | EXP | 32 |  |  |
| 23 | DIS | 3 | 2 | 1 |

**将关键字printf删去f之后，顺手删一个分号：**



* **Test 4:**

**const a = 1, b = 2, n = 10;**

**int i, j;**

**int main()**

**{**

**i = 0;**

**while(i <= n)**

**{**

**i = i + 1;**

**}**

**return;**

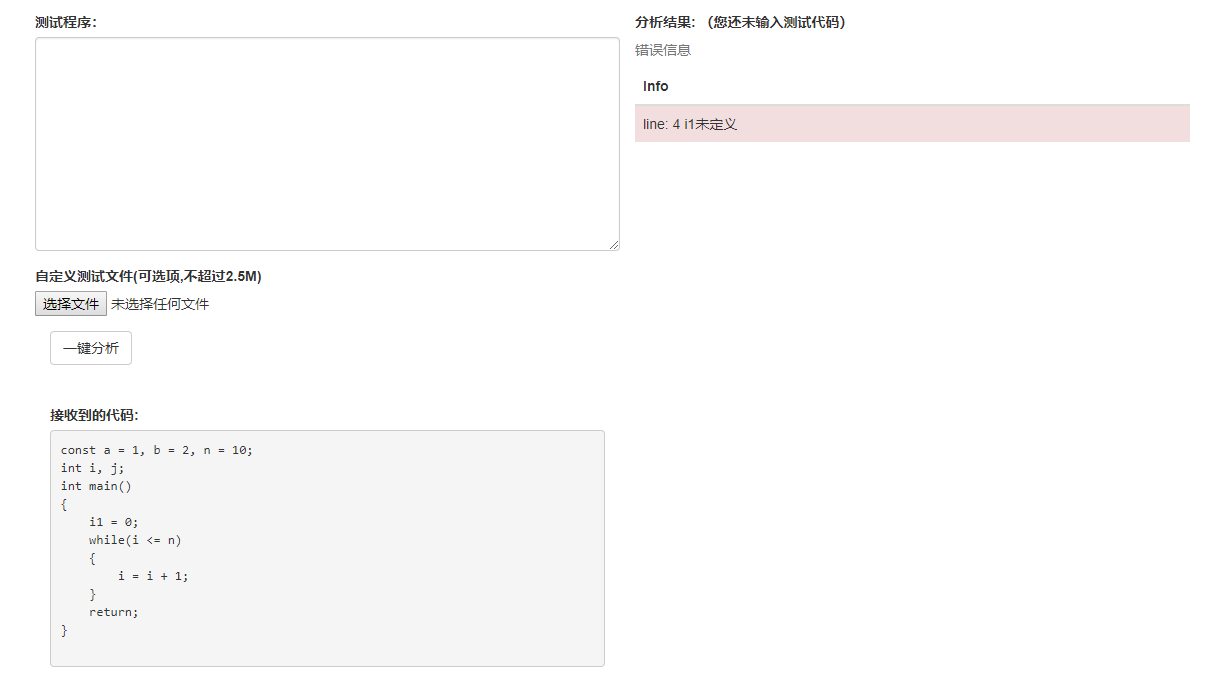
**}**

**Test 4 Result:**

| 变量表 | | | | | |
| --- | --- | --- | --- | --- | --- |
| **name** | **type** | **value** | | **addr** | **lev** |
| i | int | None | | 0 | 1 |
| j | int | None | | 1 | 1 |
| 常量表 | | | | | |
| **name** | | | **value** | | |
| a | | | 1 | | |
| b | | | 2 | | |
| n | | | 10 | | |

| Pcode | | | | |
| --- | --- | --- | --- | --- |
| **No.** | **Token** | **f** | **x** | **y** |
| 0 | main: |  |  |  |
| 1 | LDA | 0 | 1 | 0 |
| 2 | LDC | 24 |  | 0 |
| 3 | STO | 38 |  |  |
| 4 | LOD | 1 | 1 | 0 |
| 5 | LDC | 24 |  | 10 |
| 6 | LER | 48 |  |  |
| 7 | JPC | 11 |  | 14 |
| 8 | LDA | 0 | 1 | 0 |
| 9 | LOD | 1 | 1 | 0 |
| 10 | LDC | 24 |  | 1 |
| 11 | ADD | 52 |  |  |
| 12 | STO | 38 |  |  |
| 13 | JMP | 10 |  | 4 |
| 14 | EXP | 32 |  |  |
| 15 | DIS | 3 | 2 | 1 |

**在标识符i后误加一个1，变为i1:**



* **Test 5:**

**const a = 1, b = 2;**

**int c, d;**

**int main()**

**{**

**f =? 2;**

**scanf(d);**

**if ( c>1 )**

**{**

**printf("%d", a);**

**if ( d > 0)**

**{**

**printf(d);**

**}**

**else**

**{**

**printf(c);**

**}**

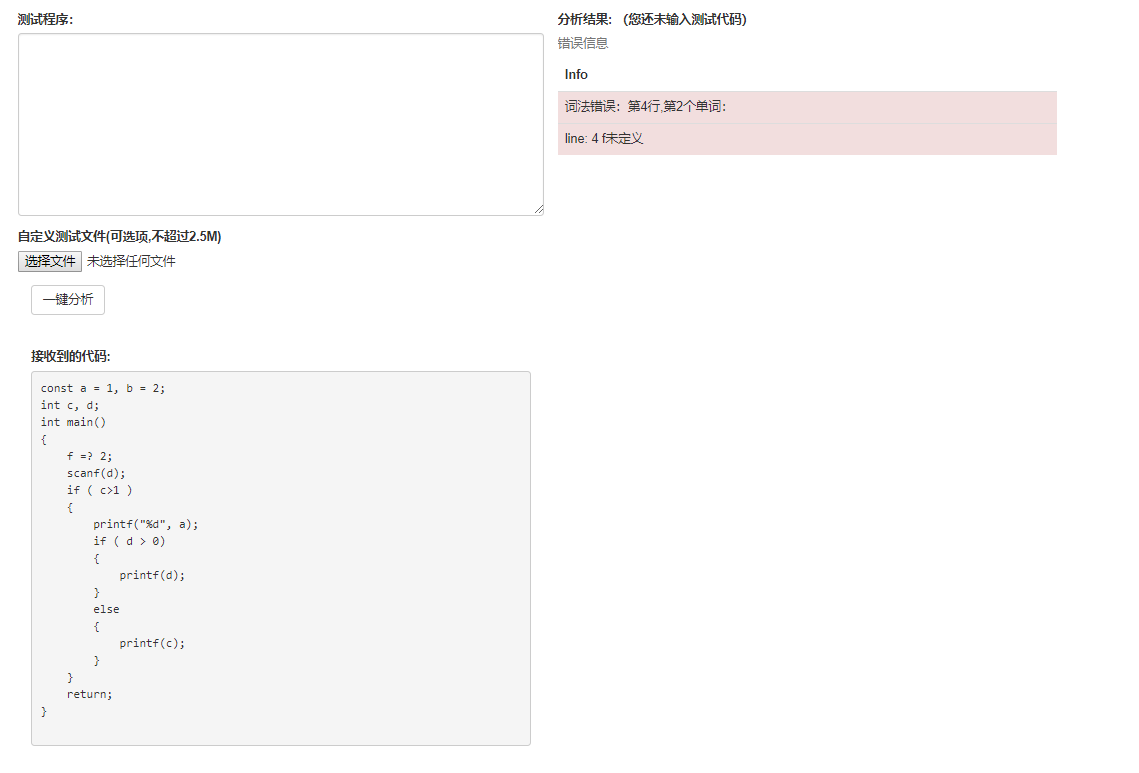
**}**

**return;**

**}**

**使用了未定义的符号?，同时也写错了变量名。**

**Test 5 Result:**



### 测试结果分析

|  |  |  |
| --- | --- | --- |
| 测试程序编号 | 正确程序覆盖点 | 错误程序覆盖点 |
| 1 | 变量声明，写语句 | 语句缺少分号 |
| 2 | 函数定义，函数调用 | 语句缺少“{” |
| 3 | 嵌套if语句，带else跳转 | 关键字错误 |
| 4 | while语句 | 变量未定义 |
| 5 |  | 运算符未定义 |
|  |  |  |

## 总结感想

我们都知道，编译器是任何通用操作系统的基础设施软件，所以编译原理这门课程非常关注底层，是一门真正深入浅出的核心专业课。设计编译器用到的不止是编译原理课堂上所传授的原理和算法，还有许多我们在前两年积累的工程经验和编程技巧，以及对计算机运行机制的深入理解。这次的课程设计可以说是一次比较大的综合工程了。

大学以来每一次课程设计我都在push自己向着更高的目标发起挑战，而编译器可以说是我非常有成就感的作品了。从后端的算法数据结构，前端的交互模型设计到整体的品控和项目规划，最终的测试。我第一次真正集合了算法工程师，前端工程师，测试工程师，产品经理于一身，是一次非常棒的锻炼。

最后要感谢邵老师的细致入微，清晰明了的讲解，把这门如此难的课程讲授的如此通俗易懂，也要感谢两位助教姐姐和许多学长给予的帮助和支持，尤其是技术和要求上的一些启发。