

2016 级《编译原理课程设计》总结报告（12 组）

2019 年 5 月 23 日

姓名	性别	班级	学号	所占比例	个人成绩
陆子旭	男	04	51160420	100%	
任务分工： （请用小四号宋体填写）					
陆子旭： 1. iOS 界面； 2. 测试； 3. 词法分析； 4. 语法分析；					
成绩评定：					
词法分析		自底向上语 法分析		中间代码生 成	
自顶向下语 法分析		语义分析		目标代码生 成	
团队成绩			教师签章		
备注					
填写说明：					
1、请将首页红色部分信息填全，其中：班级为 2 位数字，保留首位的 0；学号为 8 位数字，软件学院以 54 开头；所占比例为百分数，精确到个位数，且所有人的所占比例之和为 100%；不足 3 人的分组请保留后面的多余空行，请勿修改该表的结构。 2、请根据实际情况填写任务分工部分，主要任务包括：编译系统的总体分析与设计，4 个具体功能的设计与实现，对应的测试与验证过程（报告正文需要列出若干组具体测试样例与对应结果），系统界面的设计与美工，以及辅助工具、视图和文件等。 3、成绩评定部分由指导教师填写，请勿填写和修改。					
报告正文 （请用小四号宋体填写，自行组织章节和段落）					

一、 界面处理：

界面使用 swift 开发在 iOS 平台上，主要做了以下处理：

1. 适配了全部 iOS 设备，所有的 iOS 设备均可以有良好的运行界面；
2. 主体显示部分的任意拖放和不使用自动换行。因为在显示时有时需要显示大的表格，自动换行会带来显示上的不便；
3. 将默认的比例字体调节为等宽字体以便制表符的整齐输出；
4. 使用表格形式和树形形式的 String 形象输出；
5. 控制 TextView 的可编辑性，来准许输入代码；
6. 控制按钮的可交互性，来显示处理数据的依赖性。

二、 词法分析：

(一) 程序设计语言中的单词分为下面几类：

1. 保留字：PROGRAM, PROCEDURE, TYPE, VAR, IF, THEN, ELSE, FI, WHILE, DO, ENDWH, BEGIN, END, READ, WRITE, ARRAY, OF, RECORD, RETURN, INTEGER, CHAR;
2. 标识符：ID;
3. 常数：如 INTC, CHARC
4. 运算符：算术运算
5. 界限符：如分号、逗号、括号、单引号
6. 异常出错：ERROR（无法识别成上面几类，标记为错误，便于后续语法分析处理）
7. 结束符：ENDFILE。（留待用）

(二) TOKEN 结构：

TOKEN	column	int	记录该单词在源程序中的行数
	Line	int	记录该单词在源程序中的行数
	type	Terminal Type	记录该单词的词法信息， 其中 TerminalType 为单词的类型枚举
	data	String	记录该单词的语义信息

表 1 SNL 的 Token 结构内部表示

本编译器将词法分析和语法分析分别进行。主程序将首先调用词法分析程序，通过扫描源程序字符序列，按语言的词法规则将一个一个的字符组合成各类单词符号，并填写 TOKEN 信息，然后将结果输出到图形化界面上，同时把结果传给语法分析部分以便读取具有独立意义的单词符号。

(三) 基本实现：

我使用了有限状态自动机(DFA)来识别程序设计语言的单词的二元运算符部分，用正则表达式处理其他部分。即：

1. 先去掉注释；
2. 扫描得出符号，二元运算符用 switch-case 写出自动机，一元直接匹配，通过符号分割字符；

3. 找保留字;
4. 正则表达式判断其余类型。

(四) 注意点:

1. 保留字和标识符名字的区分: 由于保留字的词法规则符合标识符的词法规则, 故在处理时应加以区分。当扫描完一个单词时, 先调用函数查保留字表, 以确定是否是保留字, 如果是则将类型标记为该单词在保留字表中对应的项。接着在扫描过程中只要遇到符合标识符词法规则的单词, 如果没有出错, 则统一标记为标识符三种中的一种;
2. 二元单词的处理: 在程序设计语言中, 有一类单词是由两个或两个以上符号组成, 这类单词的前部分也可以是一个独立的单词, 如:=等, 在处理到“:”时, 我还不能断定这个单词是“:”还是“:=”, 这取决于“:”的下一个字符, 如果下一个字符是“=”, 则为“:=”, 否则该单词为“:”。在处理这类单词时, 是无法简单使用正则表达式的, 要加以注意;
3. 注释的处理: 在SNL 源程序语言中, 符号“{”用作注释的开始符号, 注释以符号“}”的第一次出现作为结束。下面是一个注释的例子: { this is a single {comment } }。在第一个“{”和第一个“}”之间的部分均为注释部分, 词法分析程序将注释中的内容略过不输出。

(五) 基本代码:

1. Token 结构:

```
struct Token{
    var type: TerminalType
    var data: String
    var line: Int
    var column: Int
}
```
2. 保留字查找:

```
let reservedWord:[String:TerminalType] = [
    "program": .PROGRAM,
    //。。。省略部分
]
```
3. 符号查找:

```
let symbolWord:[Character:TerminalType] = [
    "=": .EQ,
    //省略部分
]
```
4. 类型判别:

```
let discriminateType:[String:TerminalType] = [
    "\\d+": .INTC,
    "\\'.{1}\\'": .CHARC,
    "[a-zA-Z_][a-zA-Z_0-9]*": .ID
]
```

三、 语法分析：

- (一) 语法分析的任务：根据语言的语法规则，对源程序进行语法检查，并识别出相应的语法成分。按照 SNL 编译程序的模型，语法分析的输入是从词法分析器输出的源程序的 Token 序列形式，然后根据语言的文法规则进行分析处理，语法分析的输出是无语法错误的语法成分，表示成语法树的形式。（如图一）

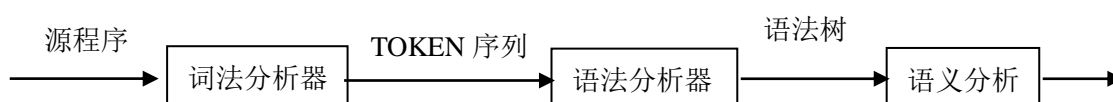


图1 语法分析器的功能

- (二) 重要集合（First 集，Follow 集，Predict 集）说明：在语法分析中，会用到以下三个集合：First 集，Follow 集，Predict 集。First(β)表示 β 串所能推导的所有可能的终极符串的头终极符集，如果能推导出空串，则令 First(β)包含 ϵ 。Follow(A)表示所有那些终极符的集合，这些终极符在某个句型（句型是指由文法开始符推导出来的符号串，可以包含终极符和非终极符）中出现在 A 的紧后面，这种集合主要用于求 Predict(A $\rightarrow \beta$) 集合。集合相关代码如下：

1. 产生式：

```
struct Production {  
    var productionLeft : NonTerminalType  
    var productionRight = [String]()  
}
```

2. 集合：

```
var firstSet = [NonTerminalType : Set<String>]()  
var followSet = [NonTerminalType : Set<String>]()  
var predictSet = [Int : Set<String>]()
```

- (三) 语法分析程序的输入输出：

1. 输入：利用词法分析产生的 Token 序列，每次从 Token 序列中取出一个 Token，作为程序正在分析的当前单词，其中存有其词法信息 Token 码和语义信息 Token 符号名，作为语法分析的输入。
2. 输出：SNL语法分析程序输出与源程序结构相对应的LL（1）语法分析树。

- (四) LL（1）矩阵：可以运用 LL（1）矩阵来通过进入的 Token 的终极符，和文法上的非终极符来找到相应的产生式，相关代码如下：

```
func showTable() -> [[Int]] {  
    var result : [[Int]] = [[Int]](repeating: [Int](repeating:  
-1, count: TerminalType.allCases.count), count:  
NonTerminalType.allCases.count)  
    for (i, set) in predictSet {  
        for p in set {
```

```

        result[NonTerminalType.allCases.firstIndex(of:
productionList[i].productionLeft)!][TerminalType.allCases.firstIn
dex(of: TerminalType.init(rawValue: p)!)] = i
    }
}
return result
}

```

(五) LL(1) 语法树定义：在语法树节点中，需要链接父子节点、需要表示出终极符或者非终极符的类型，也需要存储 data 值。故语法树节点定义如下：

```

class Node {
    var parentNode : Node?
    var children = [Node]()
    var tType : TerminalType?
    var nType : NonTerminalType?
    var data = ""
}

```

(六) LL(1) 语法分析程序：自顶向下的 LL(1) 分析程序需要维护两个表，一是读取的 Token 表，二是产生式的对应。在这里，我分了两个步骤：一是通过读取的 Token 来给产生式进行读取排序，最终会得到一个符合 Token 序列的自上而下的产生式序列。接着通过产生式序列深度遍历的方式来构造语法分析树，同时用一个总标记来标志 Token 序列的偏移，来记录读取 Token 序列中带有数据。代码如下：

// 产生式的排序

```

func productionListAnalyze() {
    var S = [String]()
    if let start = ProductionList.first {
        S.append(start.productionLeft.rawValue)
        var index = 0
        while !S.isEmpty {
            let s = S.last!
            if index < Tokens.count {
                if let nt = NonTerminalType.init(rawValue: s) {
                    let i = LLTable[NonTerminalType.allCases.firstIndex(of:
nt)!][TerminalType.allCases.firstIndex(of: Tokens[index].type)!]
                    if i > -1 {
                        Ps.append(ProductionList[i])
                        S.removeLast(1)
                        S += ProductionList[i].productionRight.reversed()
                    } else {
                        print("ERROR: Production")
                    }
                }
            }
            index += 1
        }
    }
}

```

```

        exit(1)
    }
    } else if let tt = TerminalType.init(rawValue:
s) {
        if Tokens[index].type == tt {
            S.removeLast(1)
            index += 1
        } else {
            print("ERROR: NOT Match")
            exit(-1)
        }
    } else {
        S.removeLast(1)
    }
    } else {
        print("ERROR: Tokens")
        exit(-1)
    }
}
}

//深度遍历构成树
func dfsBuildTree(node : Node) {
    for i in Ps[PsIndex].productionRight {
        if let t = TerminalType.init(rawValue: i) {
            node.children.append(Node.init(parentNode: node,
children: [Node](), tType: t, nType: nil, data: ""))
        } else if let t = NonTerminalType.init(rawValue: i) {
            node.children.append(Node.init(parentNode: node,
children: [Node](), tType: nil, nType: t, data: ""))
        } else if i == null{
        } else {
            print("ERROR: Production")
            exit(-1)
        }
    }
    PsIndex += 1
    for i in node.children {
        if let _ = i.tType {
            i.data = Tokens[TokensIndex].data
            TokensIndex += 1
        } else {
            dfsBuildTree(node: i)
        }
    }
}

```

```

    }
}

```

(七) 树形展示：我需要将最终树的形状用 String 显示出来，需要生成一个能形象显示树的 String。代码如下：

```

func showNode(lftstr : String, append : String, node : Node) ->
String {
    var b = append
    if let t = node.tType {
        b += t.rawValue
    } else if let t = node.nType {
        b += t.rawValue
    }
    if node.data != "" {
        b += "(" + node.data + ")"
    }
    b += "\n"

    if node.children.count > 0 {
        for (i, child) in node.children.enumerated() {
            if i == node.children.count - 1 {
                b += lftstr + showNode(lftstr: lftstr, append:
"|-", node: child)
            } else {
                b += lftstr + showNode(lftstr: lftstr + "| ",
append: "|-", node: child)
            }
        }
    }
    return b
}

```

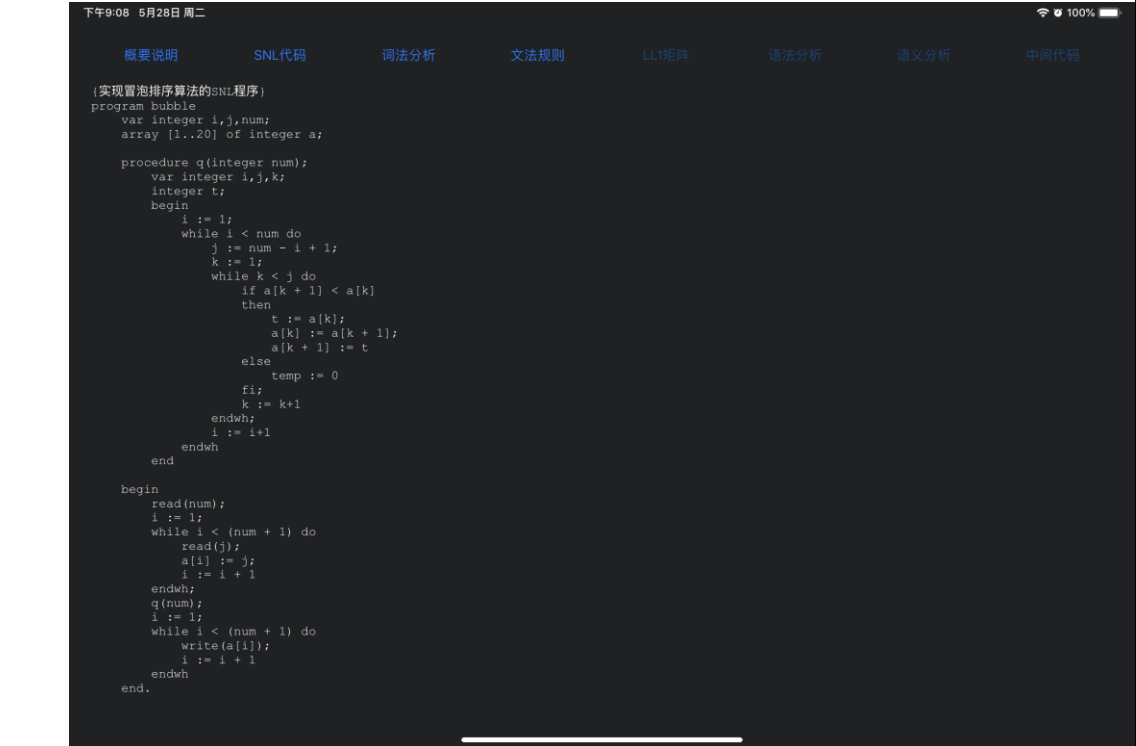
四、 效果展示：

注：实现了全设备的适配，但仅展示 iPad 11 英寸版。

(一) 概要说明



(二) SNL 代码



(三) 词法分析

下午9:08 5月28日 周二

概要说明	SNL代码	词法分析	文法规则	LL1矩阵	语法分析	语义分析	中间代码
------	-------	------	------	-------	------	------	------

INDEX	TYPE	DATA	LINE	COLUMN
0	PROGRAM		2	1
1	ID	bubble	2	2
2	VAR		3	1
3	INTEGER		3	2
4	ID	i	3	3
5	COMMA		3	4
6	ID	j	3	5
7	COMMA		3	6
8	ID	num	3	7
9	SEMI		3	8
10	ARRAY		4	1
11	LMIDPAREN		4	2
12	INTC	1	4	3
13	UNDERANGE		4	4
14	INTC	20	4	5
15	RMIDPAREN		4	6
16	OF		4	7
17	INTEGER		4	8
18	ID	a	4	9
19	SEMI		4	10
20	PROCEDURE		6	1
21	ID	q	6	2
22	LPAREN		6	3
23	INTEGER		6	4
24	ID	num	6	5
25	RPAREN		6	6
26	SEMI		6	7
27	VAR		7	1
28	INTEGER		7	2
29	ID	i	7	3
30	COMMA		7	4
31	ID	j	7	5
32	COMMA		7	6
33	ID	k	7	7
34	SEMI		7	8
35	INTEGER		8	1
36	ID	t	8	2
37	SEMI		8	3
38	BEGIN		9	1
39	ID	i	10	1
40	ASSIGN		10	2

(四) 文法规则

下午9:08 5月28日 周二

概要说明	SNL代码	词法分析	文法规则	LL1矩阵	语法分析	语义分析	中间代码
------	-------	------	------	-------	------	------	------

```

Program      ::= ProgramHead DeclarePart ProgramBody DOT
ProgramHead  ::= PROGRAM ProgramName
ProgramName  ::= ID

DeclarePart  ::= TypeDec VarDec ProcDec

TypeDec      ::= ε
| TypeDeclaration
TypeDeclaration ::= TYPE TypeDecList
TypeDecList  ::= TypeId EQ TypeName SEMI TypeDecMore
TypeDecMore  ::= ε
| TypeDecList
TypeId       ::= ID

TypeName     ::= BaseType
| StructureType
| ID
BaseType     ::= INTEGER
| CHAR
StructureType ::= ArrayType
| RecType
ArrayType    ::= ARRAY LMIDPAREN Low UNDERANGE Top RMIDPAREN OF BaseType
Low          ::= INTC
Top          ::= INTC
RecType     ::= RECORD FieldDecList END
FieldDecList ::= BaseType IdList SEMI FieldDecMore
| ArrayType IdList SEMI FieldDecMore
FieldDecMore ::= ε
| FieldDecList
IdList       ::= ID IdMore
IdMore       ::= ε
| COMMA IdList

VarDec       ::= ε
| VarDeclaration
VarDeclaration ::= VAR VarDecList
VarDecList   ::= TypeName VarIdList SEMI VarDecMore
VarDecMore   ::= ε
| VarDecList
VarIdList    ::= ID VarIdMore
VarIdMore    ::= ε
| COMMA VarIdList
-

```

(五) LL1 矩阵

下午9:08 5月28日 周二

概要说明

SNL代码

词法分析

文法规则

LL1矩阵

语法分析

语义分析

中间代码

	ENDFILE	ERROR	PROGRAM	PROCEDURE	TYPE	VAR	IF	THEN	ELSE	FI	WHILE	DO	ENDWH	BEGIN	END	READ
Program	-1	-1	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ProgramHead	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ProgramName	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
DeclarePart	-1	-1	-1	3	3	3	-1	-1	-1	-1	-1	-1	-1	3	-1	-1
TypeDec	-1	-1	-1	4	5	4	-1	-1	-1	-1	-1	-1	-1	4	-1	-1
TypeDeclaration	-1	-1	-1	-1	6	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
TypeDeclList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
TypeDecMore	-1	-1	-1	8	-1	8	-1	-1	-1	-1	-1	-1	-1	8	-1	-1
TypeId	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
TypeName	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
BaseType	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
StructureType	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ArrayType	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Low	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Top	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
RecType	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FieldDeclList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FieldDecMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	24	-1
IdList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
IdMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
VarDec	-1	-1	-1	29	-1	30	-1	-1	-1	-1	-1	-1	-1	29	-1	-1
VarDeclaration	-1	-1	-1	-1	-1	31	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
VarDeclList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
VarDecMore	-1	-1	-1	33	-1	-1	-1	-1	-1	-1	-1	-1	-1	33	-1	-1
VarIdList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
VarIdMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ProcDec	-1	-1	-1	39	-1	-1	-1	-1	-1	-1	-1	-1	-1	38	-1	-1
ProcDeclaration	-1	-1	-1	40	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ProcDecMore	-1	-1	-1	42	-1	-1	-1	-1	-1	-1	-1	-1	-1	41	-1	-1
ProcName	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ParamList	-1	-1	-1	-1	-1	45	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ParamDeclList	-1	-1	-1	-1	-1	46	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ParamMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
Param	-1	-1	-1	-1	-1	50	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FormList	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
FidMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
ProcDecPart	-1	-1	-1	54	54	54	-1	-1	-1	-1	-1	-1	-1	54	-1	-1
ProcBody	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	55	-1	-1
ProgramBody	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	56	-1	-1
StmList	-1	-1	-1	-1	-1	-1	57	-1	-1	-1	57	-1	-1	-1	-1	57
StmMore	-1	-1	-1	-1	-1	-1	-1	-1	-1	58	-1	-1	58	-1	58	-1

(六) 语法分析

下午9:08 5月28日 周二

概要说明	SNL代码	词法分析	文法规则	LL1矩阵	语法分析	语义分析	中间代码
Program							
-ProgramHead							
-PROGRAM							
-ProgramName							
-ID(bubble)							
-DeclarePart							
-TypeDec							
-VarDec							
-VarDeclaration							
-VAR							
-VarDeclList							
-TypeName							
-BaseType							
-INTEGER							
-VarIdList							
-ID(i)							
-VarIdMore							
-COMMA							
-VarIdList							
-ID(j)							
-VarIdMore							
-COMMA							
-VarIdList							
-ID(num)							
-VarIdMore							
-SEMI							
-VarDecMore							
-VarDeclList							
-TypeName							
-StructureType							
-ArrayType							
-ARRAY							
-LMIDPAREN							
-Low							
-INTC(1)							
-UNDERANGE							
-Top							
-INTC(20)							
-RMIDPAREN							
-OF							
-BaseType							
-INTEGER							
-VarIdList							
-ID(s)							

结论（请用小四号宋体填写）

在四周里，我实现了词法分析、LL（1）语法分析的基本功能，使得程序能正确在 iOS 上运行并产生预期的结果。其中，参照了书中和已有的源代码，在此基础上完善优化了代码结构。然而由于时间和个人能力有限，我的源程序还有一些细节不够完善，未能实现语义分析和生成中间代码，以后有空可以补全一下。