**电子科技大学 信息与软件工程 学院**

**标 准 实 验 报 告**

**（实验）课程名称 编译技术**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**组队排序1 学生姓名：李艳超 学 号：2022130102013**

**指导教师：周尔强**

**实验地点：信软楼 实验时间：2024.11.8**

**一、实验室名称：信软 303**

**二、实验项目名称：LR语法分析**

**三、实验学时：4学时**

**四、实验目的、原理、内容及步骤：**

**目的：**通过本实验加深对编译技术中重点算法和编译技术的理解，提高学生的编程能力培养好的程序设计风格。了解和掌握LR语法分析的基本原理，编写编写SysY语言所提供文法的LR语法分析程序。

**原理：**LR分析采用一个下推栈，用来存放已移进（的终结符）和规约（的非终结符）的符号串，该符号串看成“历史”；当前准备移进栈的符号看成“现在”；推测将来可能碰到的移进栈的符号串，将其看成“将来”。

LR分析器根据“历史”，“现在”和“将来”判断栈顶是否已经形成句柄，从而确定本次分析应该采取的动作是：规约，移进或语法出错。

**内容：**

1. 学习所提供的“表达式文法”的LR分析处理

理解 calc1.l, calc1.y, calc2.l, calc2.y的内容

在eclipse中建立工程,对calc3.l, calc3.y调试运行

2. 学习lrgram.txt所提供的文法

与递归下降分析所提供的文法作比较

4. 编写SysY语言所提供文法的LR语法分析程序

(1)编写生成“语法树”的相关程序，包括

bison源程序 lrparser.y

flex源程序 lrlex.l

语法树相关程序 ast.h 和 ast.c

(2)其它相关函数(如main函数)等 main.c,

(3)使编译得到的rdparser最终从命令行读取要分析的程序test.c,分析后调用showAst打印该程序的结构。

**实验步骤：**

学习所提供的“表达式文法”的LR分析处理，理解 calc1.l, calc1.y, calc2.l, calc2.y的内容，对calc3.l, calc3.y调试运行

编写SysY语言所提供文法的LR语法分析程序

​(1)编写生成“语法树”的相关程序，包括bison源程序 lrparser.y flex源程序 lrlex.l 语法树相关程序 ast.h 和 ast.c

​ (2)其它相关函数(如main函数)等 main.c,

​ (3)使编译得到的rdparser最终从命令行读取要分析的程序test.c,分析后调用showAst打印该程序的结构。

具体来说：主要是下面完成了这五个文件

flex源程序 lrlex.l

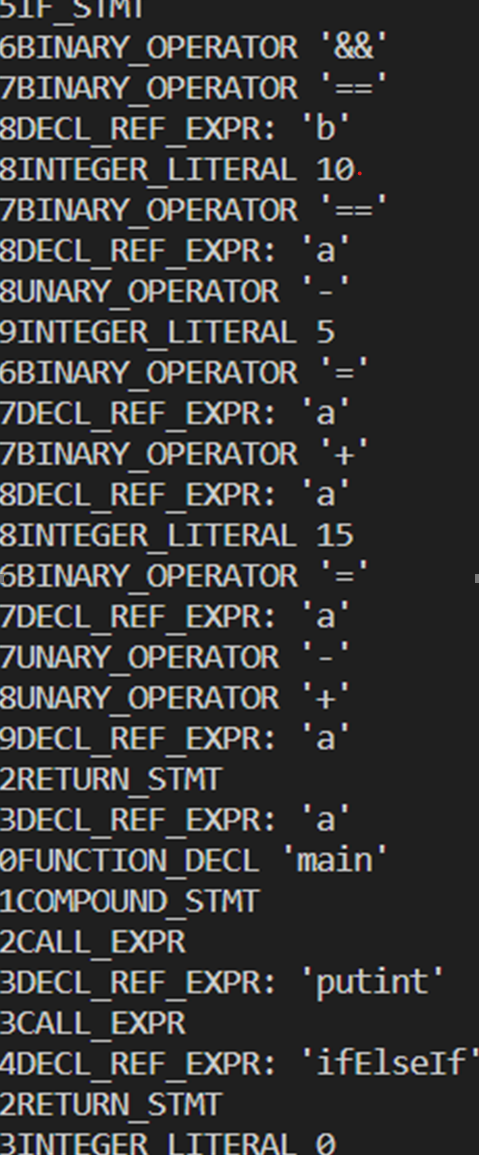
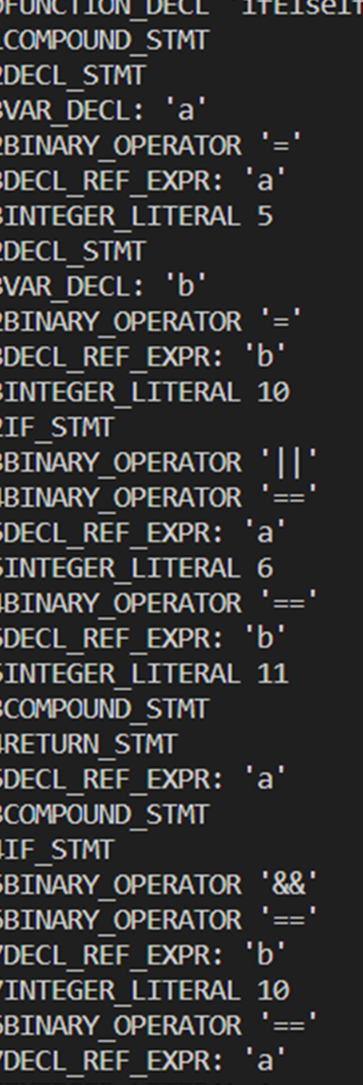
bison源程序 lrparser.y

生成语法树代码 ast.c 及头文件 ast.h

其它相关代码 main.c

实验报告 LR语法分析.docx

**五、实验运行结果：**



**六、实验结论与总结：**

完成了LR语法分析程序的构建，按照PPT要求能在本地虚拟机上输出一定AST的结果，另外在icoding测试中也可以正常通过。

主体代码的分析如下：

ast.c

刚开始时创建和初始化AST，具体来说定义了头结点等变量和一些基本函数

新建AST结点函数，分配新内存，初始化，获得结点

分配内存函数，分配内存，确定是否可以分配，然后返回结果

AST列表添加结点函数，列表为空就新建列表结点并添加，列表非空就添加新结点并更新

接下来就是一系列的函数，以新建数字结点为例说明，newAstNode获取一个新结点，设置相应的结点类型，再将传入数据赋给成员。其他类似

显示抽象语法树函数：获取结点类型，据此打印相应值

lrlex.l

还是词法分析器的东西，定义了一个词法分析器的规则集，用于将源代码文本分解为词法单元（tokens），包括忽略注释、识别关键字、运算符、数值和标识符等。

1. \*\*头文件和变量声明\*\*:

- `#include <stdlib.h>` 和 `#include "ast.h"` `#include "lrparser.tab.h"`：包含标准库和其他必要的头文件。

- `void yyerror(char const\* msg, ...);`：声明错误处理函数。

- `int lineCount = 1;`：初始化行计数器。

2. \*\*词法规则定义\*\*:

- \*\*注释忽略规则\*\*:

- `"/\*"([^\\*]|(\\*)+[^/])\*"\*/"`：多行注释，匹配 `/\*` 开始，`\*/` 结束的注释文本。

- `"//".\*`：单行注释，匹配 `//` 开始到行结束的文本。

- \*\*关键字匹配规则\*\*:

- `"int"`, `"const"`, `"void"`, `"if"`, `"else"`, `"while"`, `"break"`, `"continue"`, `"return"` 等：匹配特定的关键字，并返回对应的词法单元。

- \*\*运算符和分隔符匹配规则\*\*:

- `"+|-|\*|/|%|&|<|>|!|{ ... }|,|;|( ... )"`：匹配算术、逻辑运算符和各种括号、分号等。

- \*\*比较运算符和赋值运算符匹配规则\*\*:

- `"<"`, `">"`, `"<="`, `">="`, `"=="`, `"!="`, `"&&"`, `"||"`, `"="`：匹配比较和逻辑运算符。

- \*\*数值和标识符匹配规则\*\*:

- `\n`：行计数器增加。

- `[ \t]`：忽略空格和制表符。

- 数字匹配：匹配十进制、十六进制和八进制数。

- `[a-zA-Z\_][a-zA-Z\_0-9]\*`：匹配标识符。

- \*\*默认规则\*\*:

- `.`：任何其他字符会触发错误处理函数。

3. \*\*附加函数\*\*:

- `int yywrap()`：定义 `yywrap` 函数，返回 1 表示输入结束。

lrparser.y

定义了一个语法分析器的核心部分，包括它识别的语法元素（如标识符、数字、关键字）以及如何从这些元素构建出一个合法的语法结构（比如表达式、声明、语句等），以及在这个过程中如何构建抽象语法树。代码主要包含了文法规则和相关动作，用于解析和构建抽象语法树（AST）。下面是代码的逐块分析和总结：

1. \*\*引入头文件和声明\*\*:

- `#include <stdio.h>`: 引入标准输入输出头文件。

- `#include "ast.h"`: 引入关于抽象语法树的定义。

- `int yylex();`: 声明 `yylex` 函数，它是由 `lex`/`flex` 生成的词法分析器函数。

- `void yyerror(char \*);`: 声明错误处理函数。

2. \*\*联合体定义 `%union`\*\*:

- 定义了一个联合体，用于存储不同类型的语法元素值。这些元素包括整数 (`iValue`)、字符指针 (`eValue`) 和抽象语法树节点指针 (`pAst`)。

3. \*\*词法符号定义 `%token` 和 `%type`\*\*:

- 使用 `%token` 和 `<类型>` 定义了词法分析器可以识别的符号类型，如标识符、数字、关键字等。

- `%type <pAst>` 定义了那些需要作为抽象语法树节点处理的非终结符。

4. \*\*文法规则\*\*:

- 文法规则定义了如何从词法单元构建语法结构。每条规则后面的 `{}` 中包含了 C 代码，用于构建或操作抽象语法树。

- 例如，`CompUnit : CompUnit Decl` 规则意味着一个 `CompUnit` 可以由另一个 `CompUnit` 后跟一个 `Decl` 构成，`{ $$ = newCompUnit($2, $1); }` 是相关的动作代码，用于构建新的复合单元节点。

- 此部分代码使用了 `bison`的语法，其中 `$$` 表示产生式左侧的非终结符的值，`$1`, `$2` 等表示产生式右侧各符号的值。

main.c

读取、解析输入文件，并可能生成或显示一个抽象语法树。我将逐块分析这段代码：

1. \*\*头文件和宏定义\*\*:

```c

#include <stdarg.h>

#include <stdio.h>

#include <stdlib.h>

#include "ast.h"

#define BAD\_OUTPUT 0

```

- 引入了三个标准库`stdarg.h`, `stdio.h`, 和 `stdlib.h`，分别用于可变参数列表、标准输入输出和通用工具函数。

- 引入了一个自定义头文件`ast.h`，包含与抽象语法树（AST）相关的定义。

- 定义了一个宏`BAD\_OUTPUT`，其值为0，用于标记输出相关的错误。

2. \*\*外部变量声明\*\*:

```c

extern FILE\* yyin;

extern FILE\* yyout;

extern int yyparse();

```

- 声明了两个外部文件指针`yyin`和`yyout`，分别用于输入和输出。

- 声明了一个外部函数`yyparse()`.

3. \*\*yyerror函数\*\*

```c

void yyerror(char const\* msg, ...) {

va\_list va\_args;

va\_start(va\_args, msg);

if (yyout) {

fprintf(yyout, msg, va\_args);

return;

}

fprintf(yyout, msg, va\_args);

va\_end(va\_args);

}

```

- 这是一个错误处理函数，用于打印错误消息。

- 它使用了C语言的可变参数功能来处理不定数量的参数。

- 如果`yyout`非空，错误信息会被写入`yyout`，否则会尝试再次写入`yyout`

4. \*\*input\_switcher函数\*\*:

```c

int input\_switcher(int argc, char\*\* argv) {

// ...

}

```

- 此函数处理命令行参数来决定如何打开输入和输出文件。

- 如果提供了一个参数（`argc == 2`），它尝试打开这个文件作为输入（`yyin`）。

- 如果提供了两个参数（`argc == 3`），它同时打开第一个文件作为输入，第二个文件作为追加模式下的输出（`yyout`）。

- 如果文件打开失败，程序将打印错误并退出。

5. \*\*main函数\*\*:

```c

int main(int argc, char\*\* argv) {

// ...

}

```

- `main`函数是程序的入口点。

- 首先调用`input\_switcher`来设置输入输出文件。

- 如果`input\_switcher`返回`BAD\_OUTPUT`，则将输出重定向到标准输出（`stdout`）。

- 调用`yyparse()`进行解析操作。

- 调用`showAst`函数显示抽象语法树，这个函数可能在`ast.h`中定义。

- 程序最后返回0，表示正常结束。

makefile

Makefile，于自动化编译和清理一个包含多个源文件的项目,其实Makefile 就是一种组织编译步骤的方法用于简化和自动化编译过程。具体来说，文件主要作用是编译包含词法分析器（由 `flex` 生成）和语法分析器（由 `bison` 生成）的项目。下面是对这段代码的逐块分析和总结：

1. `all: p clean`

- 这一行定义了一个伪目标 `all`，它依赖于两个其他目标：`p` 和 `clean`。当运行 `make all` 时，这会导致首先执行 `p` 目标的命令，然后执行 `clean` 目标的命令。

2. `p : lrlex.c lrparser.tab.c ast.c main.c`

- 这定义了一个名为 `p` 的目标，它依赖于四个源文件：`lrlex.c`、`lrparser.tab.c`、`ast.c` 和 `main.c`。

- 当这些文件中的任何一个发生更改时，`p` 目标下面的命令将被执行。

3. `gcc -o p lrlex.c lrparser.tab.c ast.c main.c`

- 这是 `p` 目标的命令。它使用 `gcc` 编译器编译四个源文件，并生成一个名为 `p` 的可执行文件。

4. `lrlex.c : lrlex.l`

- 这定义了一个目标 `lrlex.c`，它依赖于文件 `lrlex.l`。

- 当 `lrlex.l` 文件更改时，会执行下面的命令以重新生成 `lrlex.c`。

5. `flex -olrlex.c lrlex.l`

- 这个命令使用 `flex` 工具根据 `lrlex.l` 文件生成 `lrlex.c`。

6. `lrparser.tab.c : lrparser.y`

- 这定义了一个目标 `lrparser.tab.c`，它依赖于 `lrparser.y` 文件。

- 当 `lrparser.y` 更改时，会执行下面的命令。

7. `bison -d lrparser.y`

- 这个命令使用 `bison` 工具处理 `lrparser.y` 文件，生成 `lrparser.tab.c` 和 `lrparser.tab.h` 文件。

8. `clean:`

- 这是一个清理目标，用于删除生成的文件和中间文件。

9. `-del /F /Q lrlex.c lrparser.tab.c lrparser.tab.h`

- 这个命令在 Windows 系统上使用 `del` 命令删除指定的文件。参数 `/F` 表示强制删除只读文件，而 `/Q` 表示安静模式，即不提示确认。

**报告评分：**

**指导教师签字：**