中国矿业大学计算机学院

2019 级本科生课程设计报告

课程名称 系统软件开发实践

报告时间 2022 年 3 月 1 日

学生姓名 胡钧耀

学　　号 06192081

专　　业 计算机科学与技术

任课教师 张博

成绩考核

编号	课程教学目标	占比	得分
1	**目标 1：**针对编译器中词法分析器软件要求，能够分析系统需求，并采用 FLEX 脚本语言描述单词结构。	15%	
2	**目标 2：**针对编译器中语法分析器软件要求，能够分析系统需求，并采用 Bison 脚本语言描述语法结构。	15%	
3	**目标 3：**针对计算器需求描述，采用 Flex/Bison 设计实现高级解释器，进行系统设计，形成结构化设计方案。	30%	
4	**目标 4：**针对编译器软件前端与后端的需求描述，采用软件工程进行系统分析、设计和实现，形成工程方案。	30%	
5	**目标 5：**培养独立解决问题的能力,理解并遵守计算机职业道德和规范，具有良好的法律意识、社会公德和社会责任感。	10%	
总成绩			

指导教师		评阅日期	

目　录

实验（三） 利用 Flex/Bison 构造编译器

3.1 实验目的

阅读参考书籍 *flex/bison.pdf* 实验教程，使用 bison 结合 flex 编写语法分析程序，对一段程序进行编译，并输出结果。要掌握移进/规约分析，掌握语法分析树，掌握抽象语法树。

3.2 相关概念简介

3.2.1 bison 简介

bison 是用来进行语法分析的，语法分析的任务是确定这些标记是如何彼此关联的。如在 `alpha = beta + gamma;`中，`beta + gamma` 是一个表达式，而该表达式的值被赋给 `alpha`。

bsion 是 YACC 的开源版本，需要配合 Flex 使用。

3.2.2 bison 语法规则简介

第一部分为定义部分，此部分主要包括选项、文字块、注释、声明符号、语义值数据类型的集合、指定开始符号及其它声明等等。文字块存在与`%{`和`%}`之间，它们将被原样拷贝到生成文件中。

第二部分，主要是语法规则和语义动作，如果没有指定语义动作，bison 将使用默认的动作。

第三部分，此部分的内容将直接逐字复制到生成的代码文件末尾。该部分主要用于对之前一些声明了的函数进行实现。

3.2.3 语法分析树与抽象语法树

语法分析树和语法树不是一种东西。习惯上，我们把前者叫做“具体语法树”，其能够体现推导的过程；后者叫做“抽象语法树”，其不体现过程，只关心最后的结果。

分析树能反映句型的推导过程，也能反映句型的结构。然而实际上，我们往往不关心推导的过程，而只关心推导的结果。因此，我们要对分析树进行改造，得到语法树。语法树中全是终结符，没有非终结符。而且语法树中没有括号。就一句话：叶子全是操作数，内部全是操作符，树里没有非终结符也不能有括号。

3.3 bison 的安装与配置

3.3.1 Windows

下载 *bison-2.4.1-setup.exe*。将 bison 和 flex 安装在同一目录 *D:\Software\GnuWin32\bin*。由于在同一个目录下，则不用配置环境变量，否则要配置一下环境变量的 Path。在命令行中输入 `bison --version` 查询 bison 版本，可验证安装和配置均已成功。安装配置过程截图如下所示。

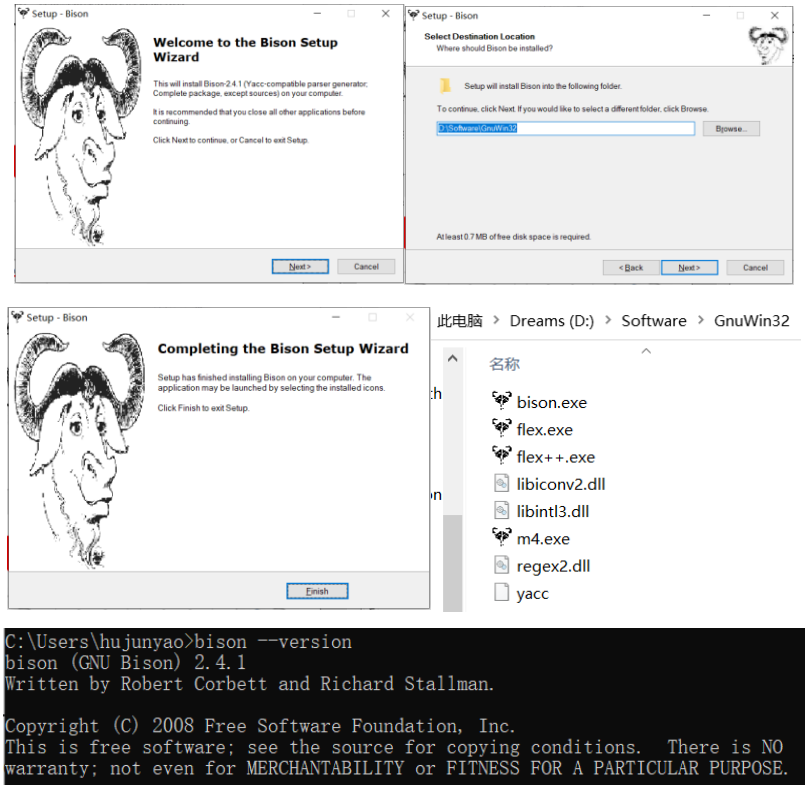

图 1 在 Windows 环境安装与配置 bison 操作截图

3.3.2 Linux

对于 Linux 系统使用如下代码进行安装并验证 bison 的安装结果。

```
sudo apt install bison
bison --version
```

图 2 在 Linux 环境安装与配置 bison 操作截图

3.4 源码分析

3.4.1 *Name.y*

```
%{
    typedef char* string;
    #define YYSTYPE string
    #include <stdio.h>
    int yylex();
    int yyerror(char *msg);
```

```
%}
%token NAME EQ AGE
%%
file :record | record file
    ;
record :NAME EQ AGE {printf("%s is %s years old!!!\n",$1,$3);}
    ;
%%
int main()
{
    yyparse();
    return 0;
}
int yyerror(char *msg)
{
    printf("Error encountered: %s \n",msg);
    return 0;
}
```

开头定义了 YYSTYPE 就是 string 类型，string 类型就是 char*类型,有助于传递所有字符串。引入相关函数库便于调用函数，显式声明 yylex 和 yyerror 函数防止编译报警。

```
%{
    typedef char* string;
    #define YYSTYPE string
    #include <stdio.h>
    int yylex();
    int yyerror(char *msg);
%}
```

token 声明了所定义的 V_T 终结符。

```
token NAME EQ AGE
```

第一条语义规则：file→record | record file | ε。

```
file :record | record file;
```

第二条语义规则：record→NAME EQ AGE | ε。

```
record :NAME EQ AGE {……};
```

第二条语义规则的语法动作是格式化输出 NAME 和 AGE。对应$1 和$3。

```
printf("%s is %s years old!!!\n",$1,$3);
```

主程序中调用 yyparse()开始语法分析。

```
int main(){yyparse();return 0;}
```

若出错，程序中会输出错误信息。

```
int yyerror(char *msg){printf("……%s \n",msg);return 0;}
```

3.4.2 *Name.l*

```
%{
    #include "Name.tab.h"
%}
char [A-Za-z]
num  [0-9]
eq   [=]
name {char}+
age  {num}+
%%
{name} { yylval = strdup(yytext); return NAME; }
{eq}   { return EQ; }
{age}  { yylval = strdup(yytext); return AGE; }
%%
int yywrap(){
    return 1;
}
```

引入 *Name.tab.h* 头文件为 bison 编译之后生成的头文件，用于协作。

```
#include "Name.tab.h"
```

定义了一些正则表达式的命名，name 由一个或者多个大小写字符构成，age 由一个或者多个数字构成，eq 就是等号。

```
char [A-Za-z]
num  [0-9]
eq   [=]
name {char}+
age  {num}+
```

规定了三条语义规则。

识别到 name，将数值存入 yylval，向符号表中传递响应标识符的属性值，同时 return 标识符。识别到 eq，return 标识符。识别到 age，将数值存入 yylval，向符号表中传递响应标识符的属性值，同时 return 标识符。而 bison 会读取

yylval 之中的值。

yylval 为了向符号表中传递响应 yytext 的属性值，使用了 strdup 方法，该 f 方法主要是拷贝字符串 yytext 的一个副本，由函数返回值返回，这个副本有自己的内存空间，和 yytext 不相干。

```
{name} { yylval = strdup(yytext); return NAME; }
{eq}   { return EQ; }
{age}  { yylval = strdup(yytext); return AGE; }
```

yywrap 指明分析器到达文件末尾时的下一步操作，yywrap 返回 1，词法分析器将返回一个零记号来表明文件结束。

```
int yywrap(){return 1;}
```

3.5 双系统实验步骤与结果

3.5.1 Windows

输入以下代码，可得到下图 Windows 系统执行结果。

```
bison -d Name.y
flex -o"Name.yy.c" Name.l
gcc -o"Name" Name.yy.c Name.tab.c -std=c89
Name < Name.txt
```

```
D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验1>bison -d Name.y

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验1>flex -o"Name.yy.c" Name.l

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验1>gcc -o"Name" Name.yy.c Name.tab.c -std=c89

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验1>Name < Name.txt
Tom is 40 years old!!!

JERRY is 30 years old!!!

D:\Docs\CUMT_StudyFiles\3-2-系统软件开发实践\Bison实验1>
```

图 3 Windows 系统运行结果截图

3.5.2 Linux

输入以下代码，可得到下图 Linux 系统执行结果。注意：在第三步之前，需要手动修改 *Name.tab.h* 第 60 行的 `typedef int YYSTYPE;`为 `typedef char* YYSTYPE;`，否则会报警，但不影响最终结果。

```
bison -d Name.y
flex -o"Name.yy.c" Name.l
gcc -o"Name" Name.yy.c Name.tab.c
Name < Name.txt
```

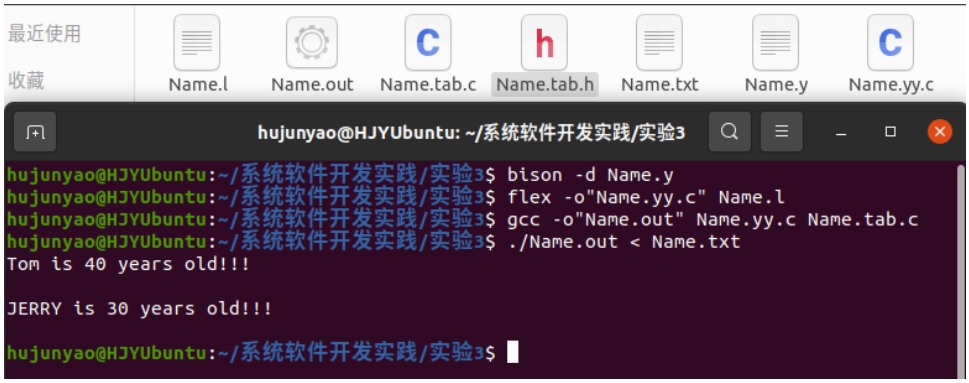

图 4 Linux 系统运行结果截图

3.6 结果分析与 flex/bison 的协同工作过程

bison 使用的是 LALR(1)分析方法，手工模拟识别，结果如下。

NAME	移进
NAME EQ	移进
NAME EQ AGE	规约 record→NAME EQ AGE
record NAME	移进
record NAME EQ	移进
record NAME EQ AGE	规约 record→NAME EQ AGE
record record	规约 file→record
record file	规约 file→file record
file	结束

绘制语法分析树和抽象语法树如下。

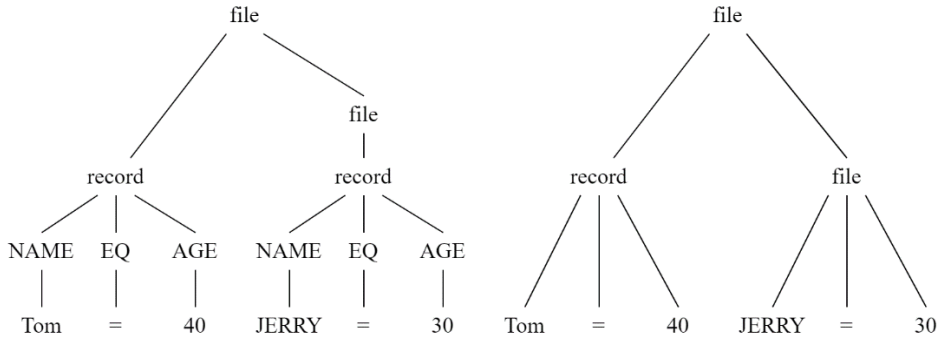

图 5 本实验语法分析树和抽象语法树实例

flex/bison 协同工作编译文件，理解如下。

flex 是用来处理分词，也就是词法分析，识别 V_T 符号，把他们输出切分成一段一段的 token 标记。这些 token 可以交给 bison 处理，当然也可以自己直接处理。bison 的任务是语法分析，把传过来的 V_T 符号确定他们如何关联，构建语法树，进行分析，从而理解语句。

以本次实验为例，通过词法分析找出了 Tom、=、40、JERRY、=、30，一共六个 token，通过语法分析，Tom 和 JERRY 是 NAME，=是 EQ，30 和 40 是 AGE，最后，在词法分析作出相应的输出。

3.7 实验总结

3.7.1 typedef 和 define 的联系与区别

typedef 和 define 都是替一个对象取一个别名，以此增强程序的可读性。

#define 是 C 语言中定义的语法，是预处理指令，在预处理时进行简单而机械的字符串替换，不作正确性检查，只有在编译已被展开的源程序时才会发现可能的错误并报错。

typedef 是关键字，在编译时处理，有类型检查功能。它在自己的作用域内给一个已经存在的类型一个别名，但不能在一个函数定义里面使用 typedef。用 typedef 定义数组、指针、结构等类型会带来很大的方便，不仅使程序书写简单，也使意义明确，增强可读性。

3.7.2 报警：显式声明

在使用 gcc 编译时报警，提示 yylex 和 yyerror 没有显式声明。

修改方法：在 *Name.l* 开头显式声明 yylex 和 yyerror

```
    int yylex();
    int yyerror(char *msg);
```

3.7.3 报警：YYSTYPE 类型不对

strdup(yytext)返回值的类型为 char*，而在链接程序时，默认的 yylval 的类型为整型，因而出现了赋值时类型不匹配的错误。但是手动在头部文件修改也没有用，需要在 *Name.tab.h* 手动修改。

```
typedef char* YYSTYPE;
```

或者在编译的时候，给 gcc 编译器设定标准，使用 c89 标准。

```
gcc -o"Name" Name.yy.c Name.tab.c -std=c89
```

3.7.4 程序评价与收获

本次实验完成的程序是一个简单的分析器，增加了语法分析的环节，回顾了一下上学期编译原理的相关内容，感觉还有很多知识要学，要多实践。通过本次实验，我对 flex 和 bison 如何结合在一起，进行联合编译，完成语法分析有了更深入的理解，对之前所学知识有了更深层次的认识。