**中国矿业大学计算机学院**

**2018 级本科生课程设计报告**

课程名称 系统软件开发实践

实验名称 Bison理论与练习实验1

报告时间 2021年3月10日

学生姓名 丁旭行

学 号 14184501

专 业 计算机科学与技术

任课教师 徐东红

**目 录**

[1 实验目的 1](#_Toc66222034)

[2 语法分析器自动生成工具Bison 1](#_Toc66222035)

[2.1 Bison简介 1](#_Toc66222036)

[2.2 Bison文件结构 1](#_Toc66222037)

[3 代码分析 3](#_Toc66222038)

[3.1 Lex源代码分析 3](#_Toc66222039)

[3.2 Bison源代码分析 4](#_Toc66222040)

[4 Windows环境下Bison的配置与使用 4](#_Toc66222041)

[4.1 下载安装Bison 4](#_Toc66222042)

[4.2 Bison源代码编写与执行 6](#_Toc66222043)

[4.3 Flex源代码编写与执行 9](#_Toc66222044)

[4.4 编写测试文件Name.txt并测试 11](#_Toc66222045)

[5 Linux-Ubuntu环境下Bison的配置与使用 12](#_Toc66222046)

[5.1 下载安装Bison 12](#_Toc66222047)

[5.2 Bison源代码编写与执行 13](#_Toc66222048)

[5.3 Flex源代码编写与执行 14](#_Toc66222049)

[5.4 编写测试文件Name.txt并测试 16](#_Toc66222050)

[6 实验体会 17](#_Toc66222051)

[6.1 实验中的困难 17](#_Toc66222052)

[6.2 程序评价 17](#_Toc66222053)

[6.3 实验收获 17](#_Toc66222054)

[6.4 Flex与Bison的关系 17](#_Toc66222055)

# 1 实验目的

使用 Bison 结合 Flex 编写语法分析程序，对一段文本进行编译，并输出结果。

# 2 语法分析器自动生成工具Bison

## 2.1 Bison简介

Bison是一种通用解析器生成器，它将带注释的上下文无关文法转换为使用LALR(1)解析器表的确定性LR或广义LR(GLR)解析器。作为一项实验性功能，Bison还可以生成IELR(1)或规范的LR(1)解析器表。Bison与Yacc向上兼容：所有正确编写的Yacc语法都应与Bison一起使用，而无需进行任何更改。

## 2.2 Bison文件结构

在结构上，Bison源程序具有与Flex源程序相同的三个部分：定义段，规则段和用户代码段。其中，第一部分及第三部分和第三部分之上的%%都可以省略，以%开头的符号、关键字和规则段的各个规则一般顶着行首来写， 前面没有空格。

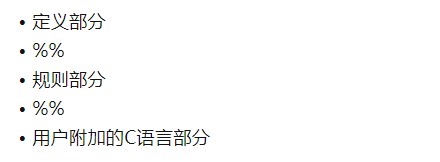


图1 Bison文件结构

### 2.2.1 定义段

定义段可以分为两部分：

1. 第一部分以符号%{和%}包裹，里面是以C语言写的一些定义和声明，譬如文件包含、宏定义、全局变量定义、函数声明等。
2. 第二部分主要是对文法的终结符和非终结符做一些相关声明。这些声明主要有如下一些：%token、%left、%right、%nonassoc、%union、%type、%start，其每一项均以%开头。%token 定义文法中使用了哪些终结符，终结符一般全大写。%left、%right、%nonassoc也是定义文法中使用的终结符，定义形式与%token类似，但是他们定义的终结符具有某种优先级和结合性，%left 表示左结合，%right表示右结合，%nonassoc表示不可结合(即它定义的终结符不能连续出现。例如<，如果文法中不允许出现形如 a<b<c的句子，则<就是不可结合的)。而优先级关系则是以他们定义出现的顺序决定的，先定义的优先级低，最后定义的优先级最高，同时定义的优先级相同。%start指定文法的开始符号(非终结符)，定义形式为：%start startsym，其中 startsym为文法的开始符号。如果不使用%start定义文法开始符号，则默认在第二部分规则段中定义的第一条产生式规则的左部非终结符为开始符号。%union 和%type用来处理文法中各符号所带的属性，其中%type用来定义非终结符的属性值类型。记号是由记号名和记号的属性值两部分组成的，文法中的终结符就是记号，他们有属性值，同样，非终结符也是可以有属性值的。

### 2.2.2 规则段

规则段定义了文法的非终结符及产生式集合，以及当归约整个产生式时应执行的操作。假如产生式为expr->expr plus term | term，则在规则段中应该写成expr：expr plus term {语义动作} | term {语义动作}；其中，产生式左部的非终结符expr应写在冒号左边，冒号表示产生符号->，产生式右部每个用 | 分隔的部分单独写一行并用 | 分隔，整组产生式写完后，结尾处应加分号；，分号表示同一左部的一组产生式结束。每一行产生式后面大括号括起的语义动作表示该条产生式归约时应该执行的操作，语义动作使用C语言的语法来写，他们将会被直接拷贝到Yacc编译器生成的C语言源程序文件中。在语义动作中，可以引用存放在属性值栈中的文法符号的属性值，$$符号可引用产生式左部非终结符的属性值，而$i则可以引用产生式右部第i个文法符号的属性值，当然，前提是你必须为这些文法符号的属性定义了类型，而且在你使用这些属性值之前，已经将这些属性值放入了属性值栈对应的地方。

### 2.2.3 用户代码段

用户代码段存放C语言语法写的辅助函数，辅助函数一般指在规则段中用到或者在语法分析器的其他部分用到的函数。这一部分一般会被直接拷贝到 Yacc编译器产生的C语言源文件中。一般来说，除规则段用到的函数外，辅助函数段一般包括如下一些例程：yylex()，yyerror()，main()。

int yylex()是词法分析程序，它返回记号。语法分析驱动程序yyparse()将会调用yylex()获取记号。如果不使用lex生成这个函数，则必须在辅助函数段用C语言写这个程序。记号由记号名和属性值构成，记号名一般作为yylex的返回值(注意，记号名是由%token等定义的终结符名，这些终结符名在Yacc内部会被宏定义成一些常数)，而属性值则由Yacc内部定义的变量yylval来传递。yylval的类型与属性值栈元素的类型相同，即默认状态下，yylval为int类型，若使用#define YYSTYPE double将属性值栈元素定义为double类型，则yylval就是double类型，如果你想让属性值栈可以存放多种类型的属性值，例如整型和字符串型等,则最好将属性值栈元素的类型定义为联合union类型，此时，你可以用%union来定义YYSTYPE的类型，然后逐个定义终结符和非终结符的属性值的类型。

# 3 代码分析

## 3.1 Lex源代码分析

Lex源代码及分析如图所示：

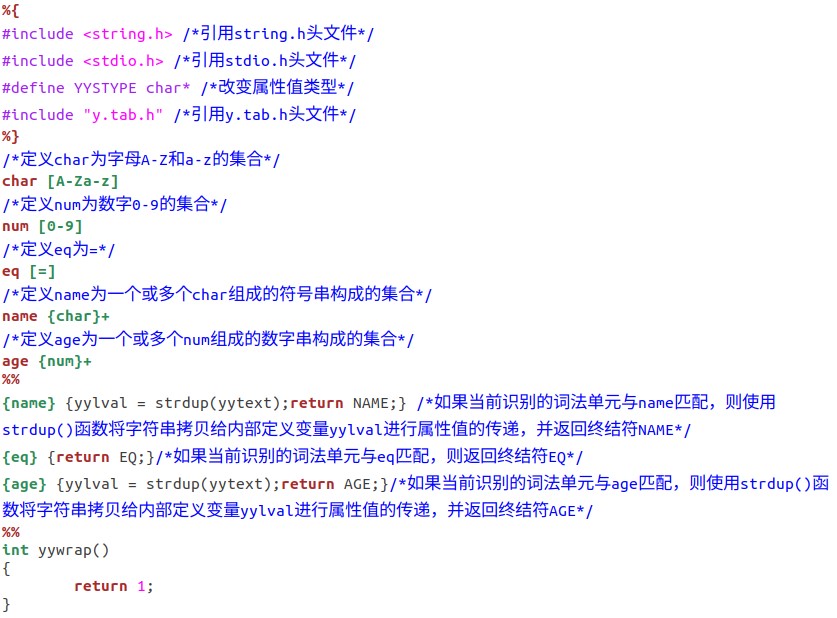


图2 Lex源代码及分析

## 3.2 Bison源代码分析

Bison源代码及分析如图所示：

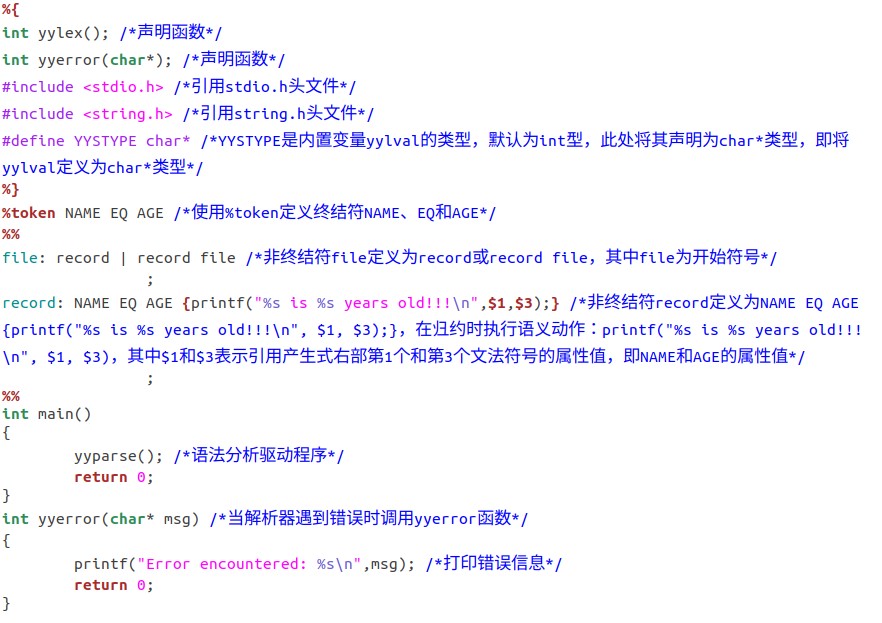


图3 Bison源代码及分析

# 4 Windows环境下Bison的配置与使用

## 4.1 下载安装Bison

在Windows环境下打开Bison安装包(老师提供的)，按照指示步骤进行安装，需要注意的是一定要将Bison与Flex安装在同一目录下。

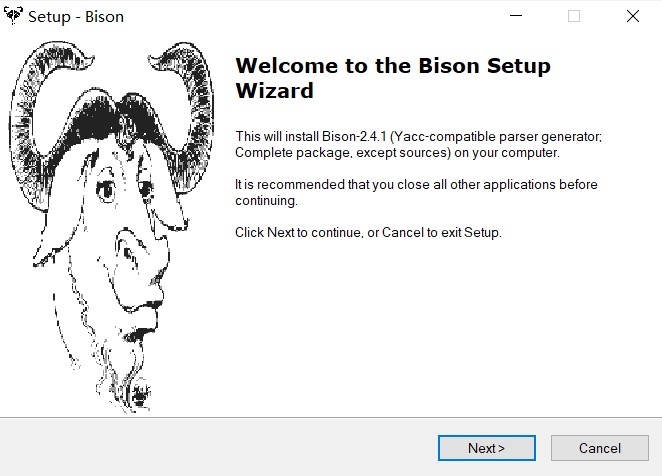


图4 Bison安装

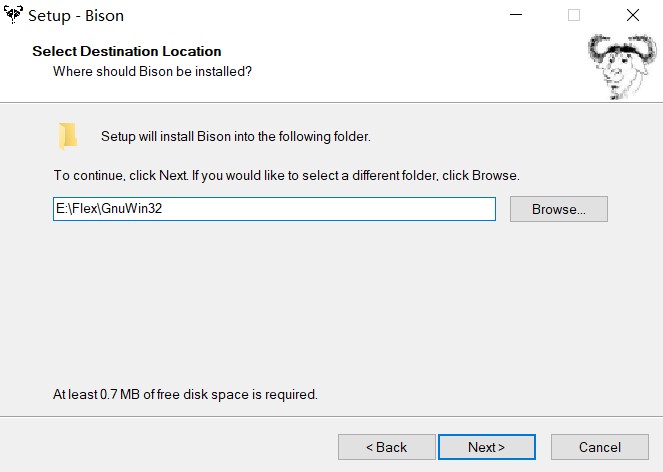


图5 设置安装目录

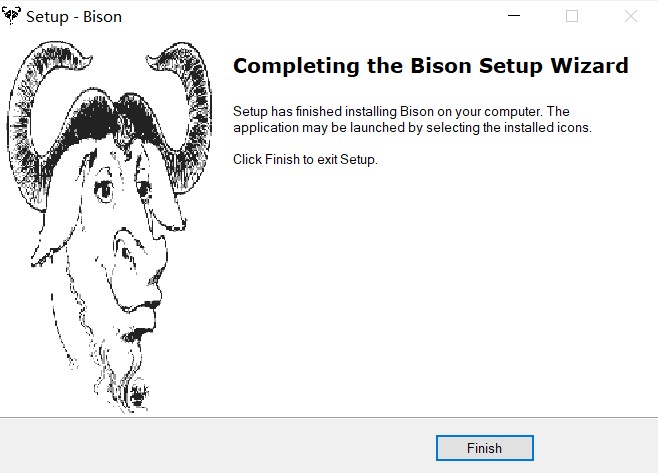


图6 Bison安装成功

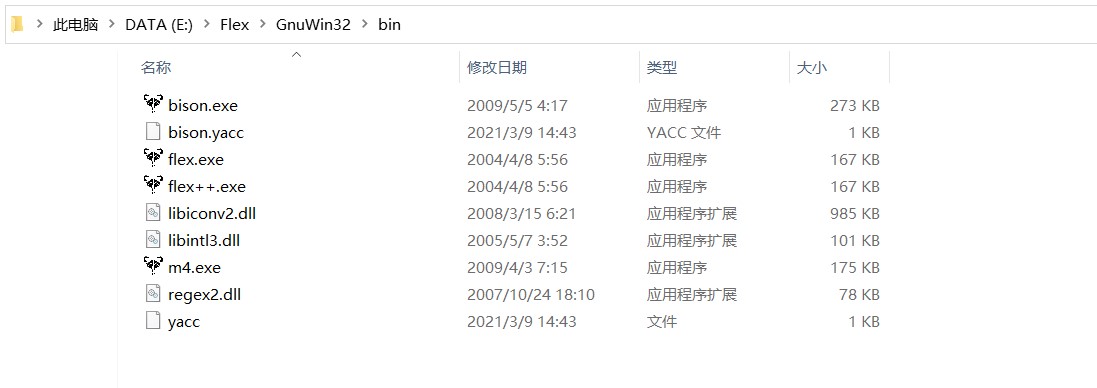


图7 安装结果

## 4.2 Bison源代码编写与执行

### 4.2.1 Name.y源代码编写与保存

新建文本文档编写Name.y源代码，编写完成后，将其保存到Bison的安装目录下。



图8 Name.y源代码



图9 Name.y保存目录

### 4.2.2 Visual Studio 2019命令行执行命令

打开Developer Command Prompt for VS 2019命令窗口，依次输入e:、cd E:\Flex\GnuWin32\bin进入安装目录下，再输入bison.exe -d Name.y来调用 bison.exe生成Name.tab.c和Name.tab.h文件。

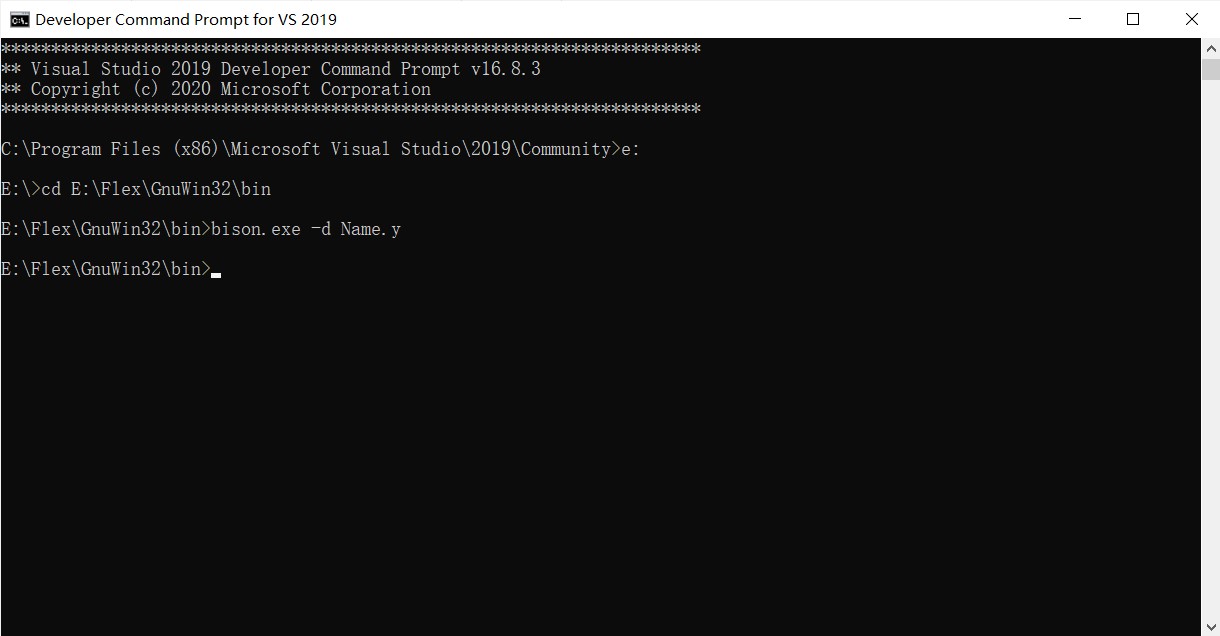


图10 生成Name.tab.c和Name.tab.h输入的命令



图11 生成Name.tab.c和Name.tab.h文件所在目录

### 4.2.3 Name.tab.c和Name.tab.h代码分析

打开Name.tab.c和Name.tab.h文件，由其中的代码可知，在Name.y中定义的三个终结符NAME、EQ和AGE在这两个文件中都是枚举类型的变量，并且 Name.y中的辅助函数段代码也全部拷贝进了Name.tab.c中。

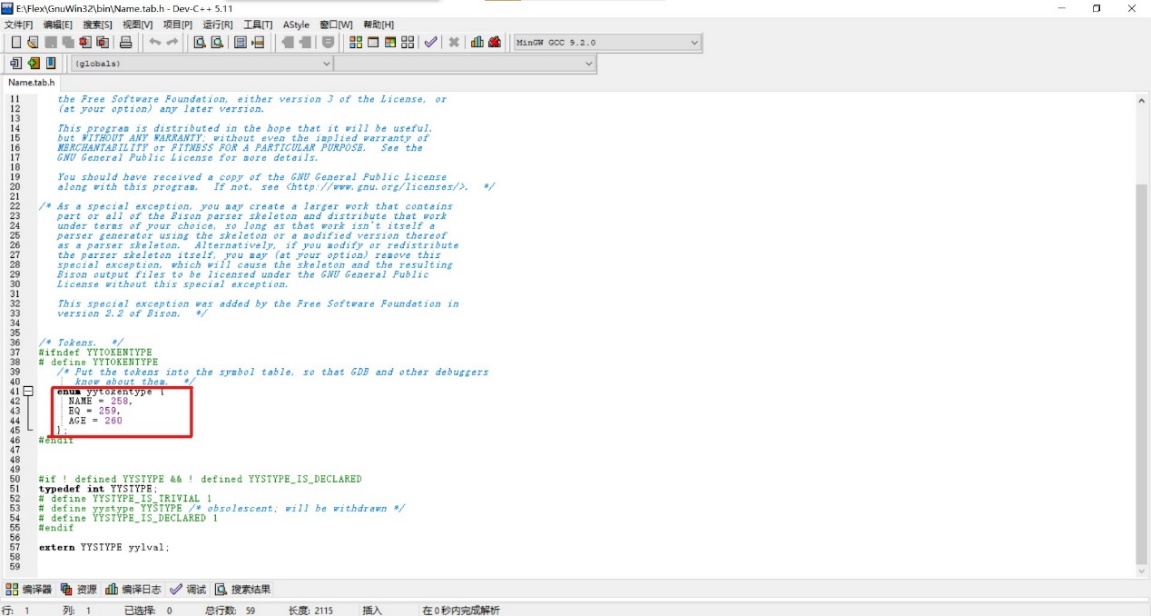


图12 Name.tab.h文件中的枚举变量

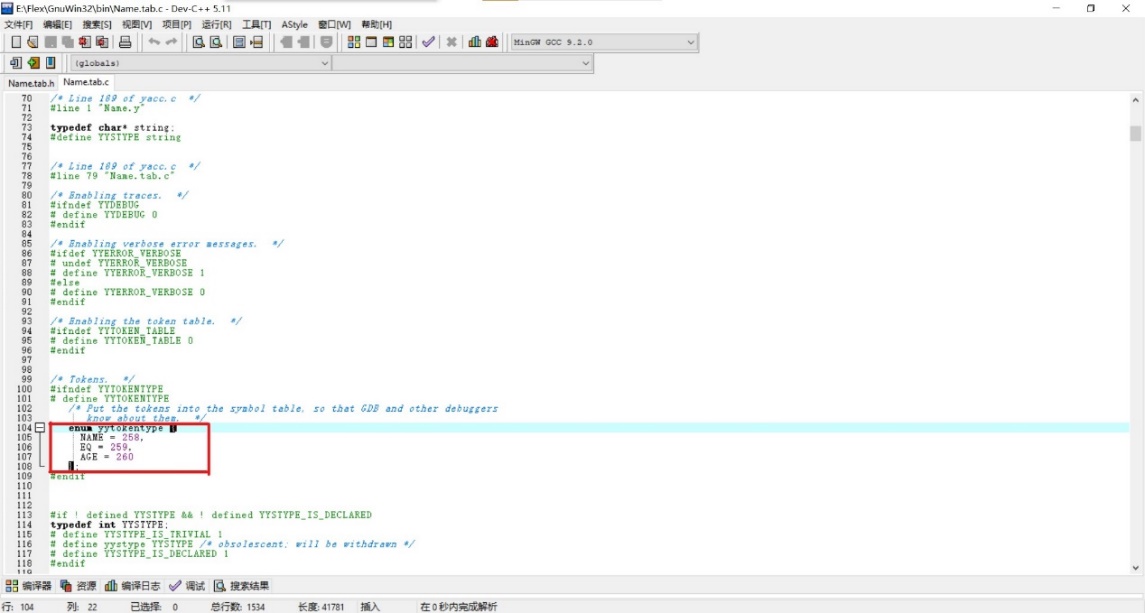


图13 Name.tab.c文件中的枚举变量

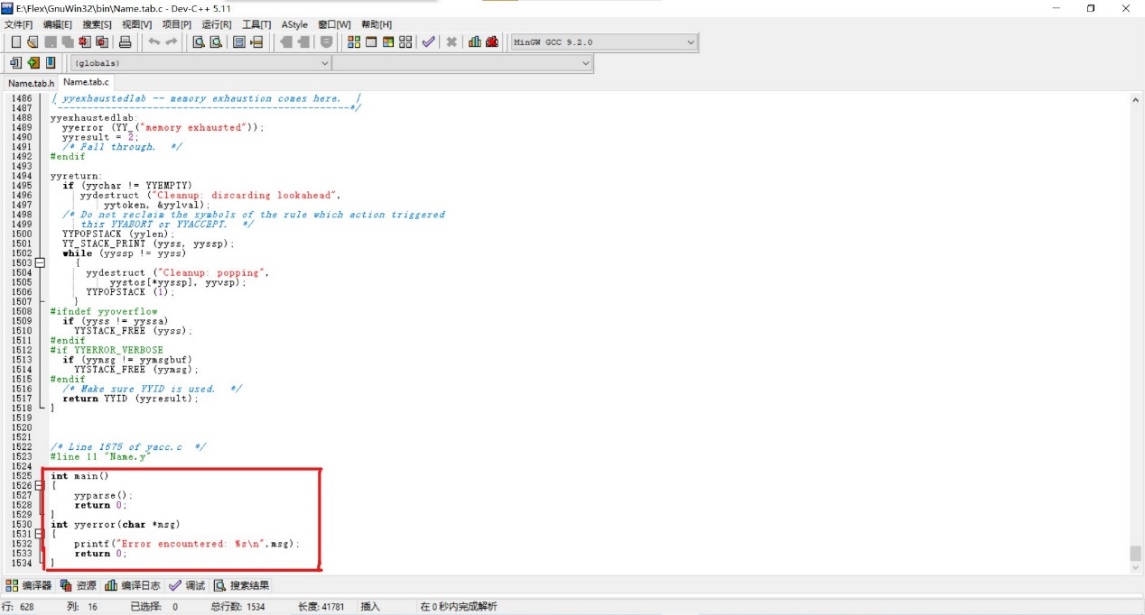


图14 Name.tab.c文件中的辅助函数段代码

## 4.3 Flex源代码编写与执行

### 4.3.1 Name.l源代码编写与保存

新建文本文档编写Name.l源代码，编写完成后，将其保存到Flex的安装目录下。



图15 Name.l源代码



图16 Name.l保存目录

### 4.3.2 Visual Studio 2019命令行执行命令

打开Developer Command Prompt for VS 2019命令窗口，依次输入e:、cd E:\Flex\GnuWin32\bin进入安装目录下，再输入flex.exe -o"Name.yy.c" Name.l来调用flex.exe生成Name.yy.c文件。

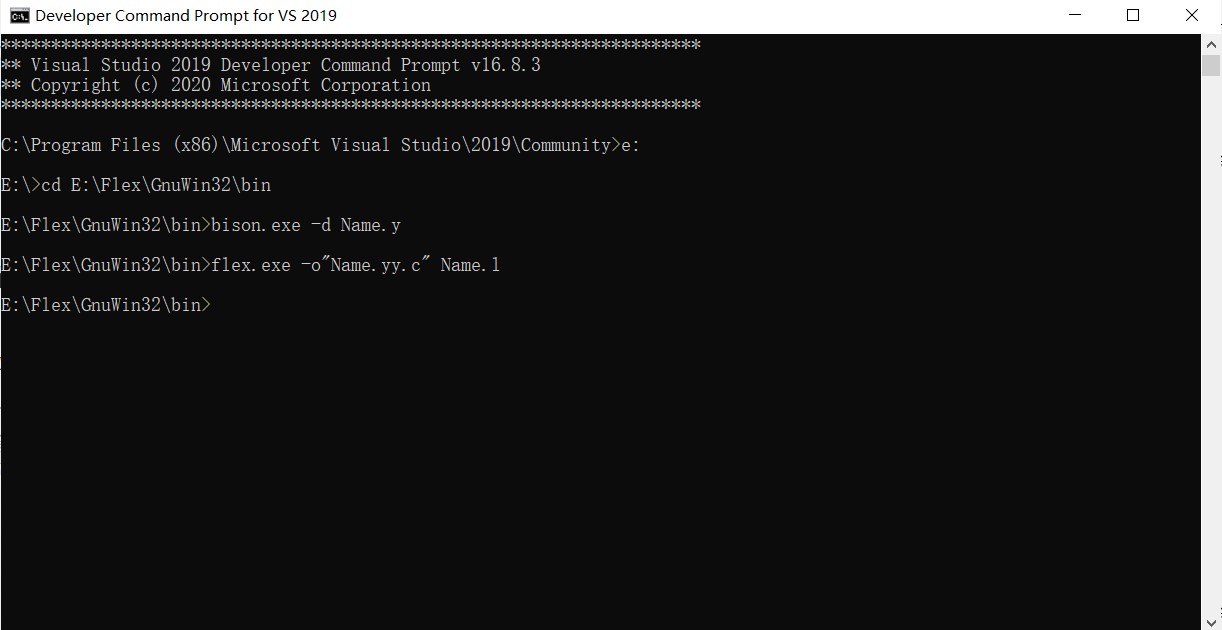


图17 生成Name.yy.c输入的命令



图18 生成Name.yy.c文件所在目录

## 4.4 编写测试文件Name.txt并测试

### 4.4.1 编写测试文件Name.txt

新建文本文档编写Name.txt测试文件并保存。



图19 测试文件Name.txt

### 4.4.2 测试Name.txt文件

打开Developer Command Prompt for VS 2019命令窗口，依次输入e:、cd E:\Flex\GnuWin32\bin进入安装目录下，再输入cl Name.yy.c Name.tab.c来调用cl.exe生成Name.yy.exe文件，最后输入Name.yy.exe < Name.txt命令进行语法分析，并显示结果。

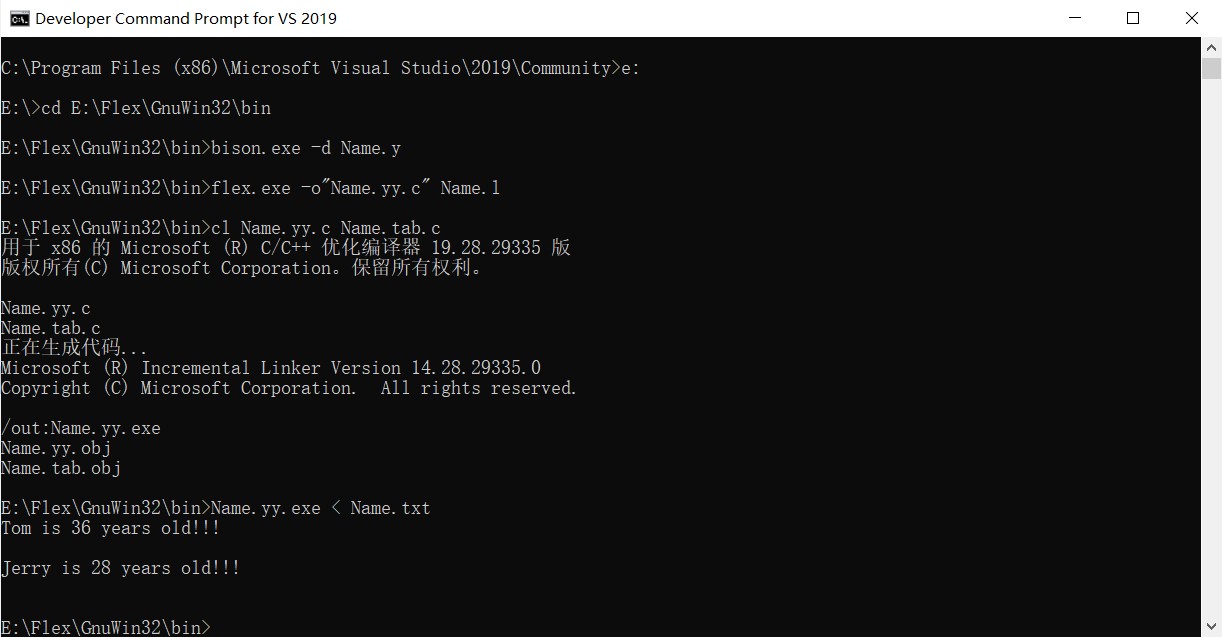


图20 命令输入及输出结果

### 4.4.3 分析程序输出结果

输出结果显示为：“Tom is 36 years old!!!\nJerry is 28 years old!!!\n”而在Name.txt文件中的内容为“Tom=36\nJerry=28\n”，而在源程序Name.y的规则段中定义的产生式“record：NAME EQ AGE {printf("%s is %s years old!!!\n", $1,$3);}”归约时执行的操作为：“printf("%s is %s years old!!!\n", $1, $3)”，与实际输出相符合，说明对Name.txt的语法分析成功。

# 5 Linux-Ubuntu环境下Bison的配置与使用

## 5.1 下载安装Bison

在Linux环境下安装Bison，在命令窗口输入su后输入用户密码，之后在输入apt install bison，进行Bison的安装。

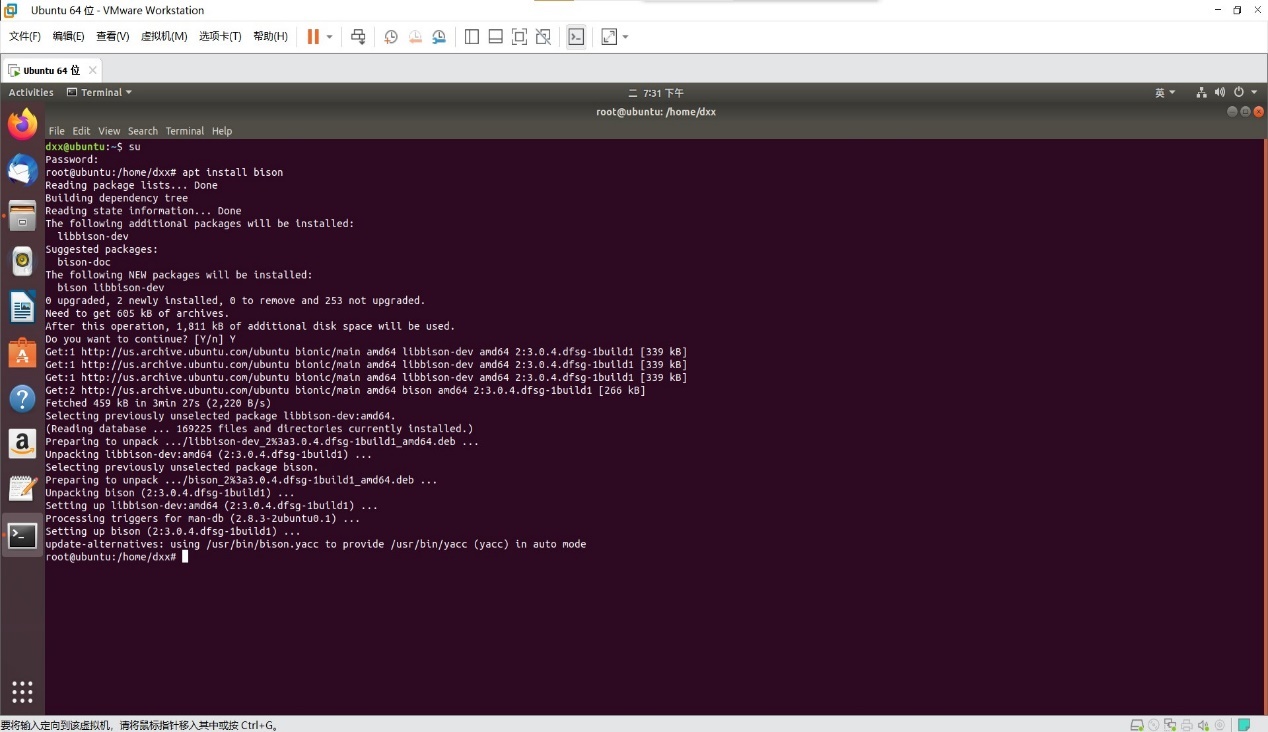


图21 Linux下安装Bison

## 5.2 Bison源代码编写与执行

### 5.2.1 Name.y源代码编写与保存

打开文本编辑器，输入Name.y源代码并保存到桌面。

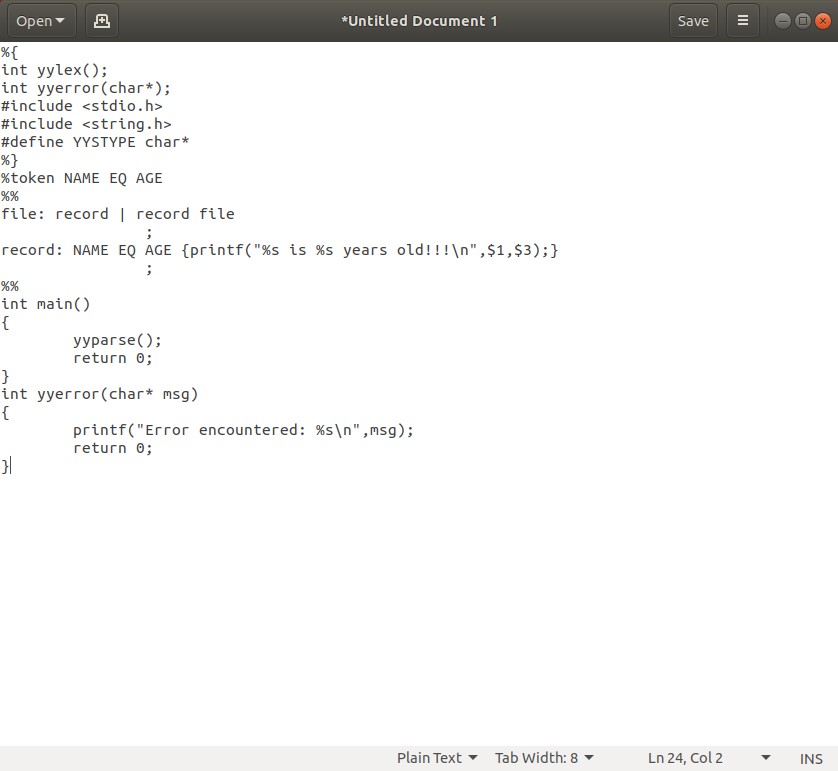


图22 编写Name.y文件

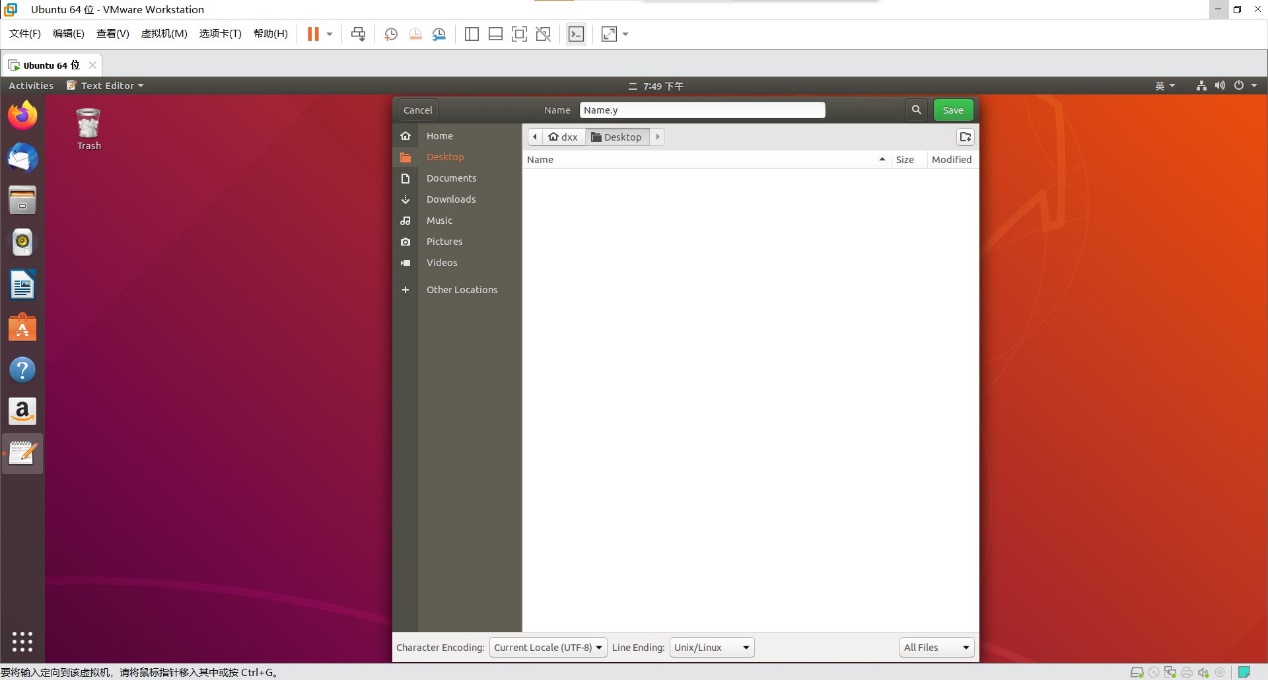


图23 保存Name.y文件至桌面

### 5.2.2 命令窗口执行命令

打开命令窗口，输入cd Desktop命令进入桌面目录，再输入bison -vdty Name.y命令生成y.tab.c、y.tab.h和y.output文件。

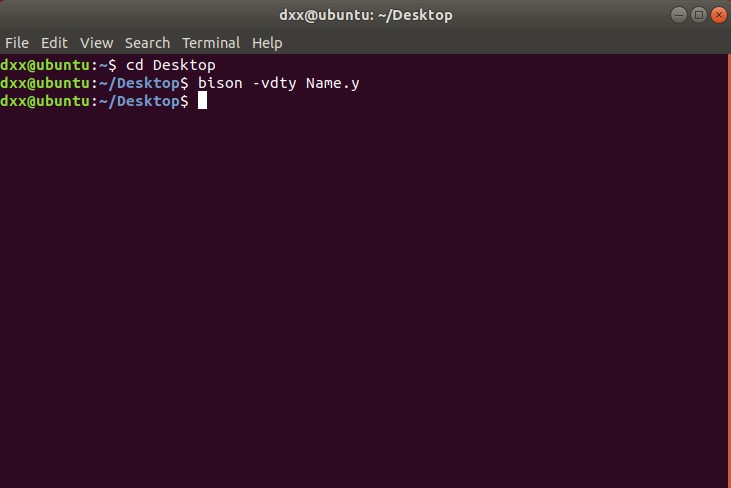


图24 生成y.tab.c和y.tab.h输入的命令

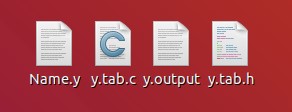


图25 生成y.tab.c和y.tab.h文件

## 5.3 Flex源代码编写与执行

### 5.3.1 Name.l源代码编写与保存

打开文本编辑器，输入Name.l源代码并保存到桌面。

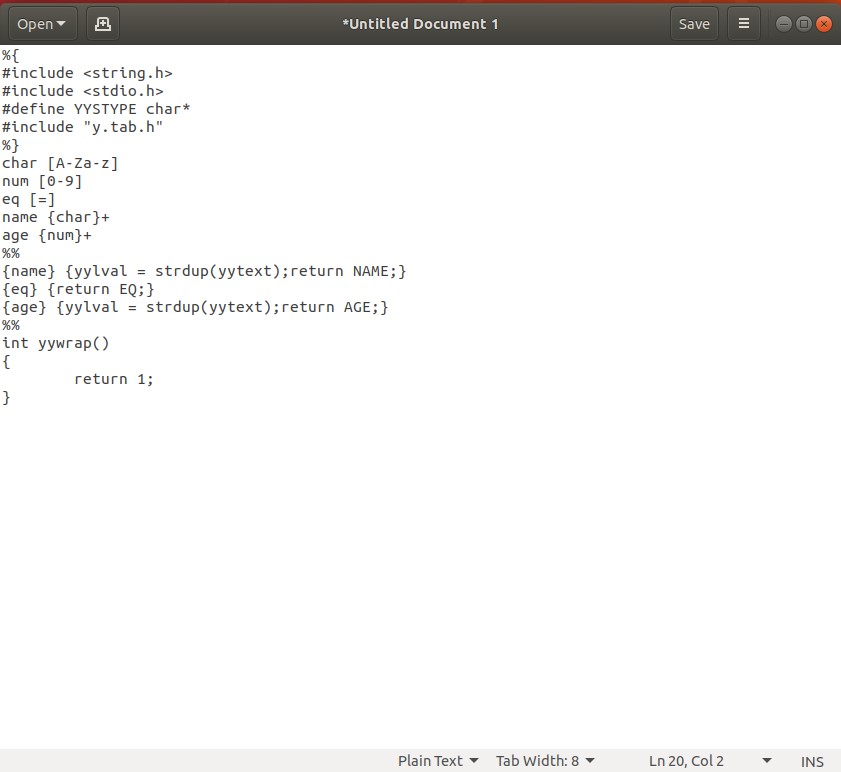


图26 编写Name.l文件

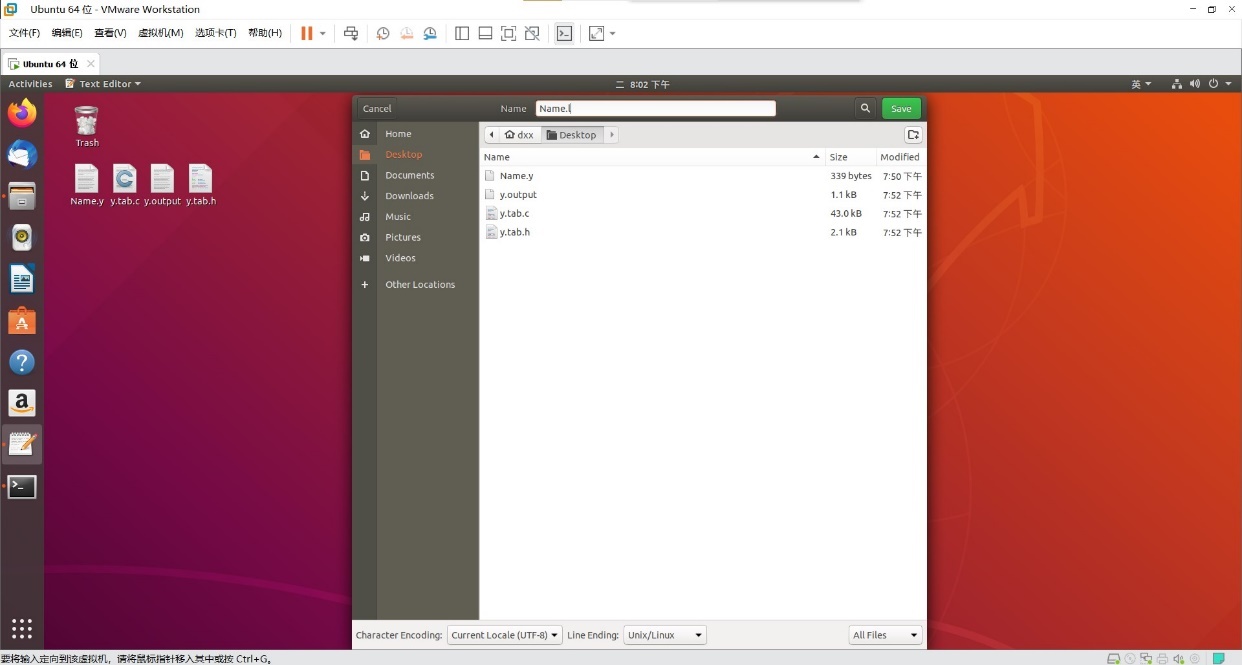


图27 保存Name.l文件至桌面

### 5.3.2 命令窗口执行命令

打开命令窗口，输入cd Desktop命令进入桌面目录，再输入flex Name.l命令生成lex.yy.c文件。

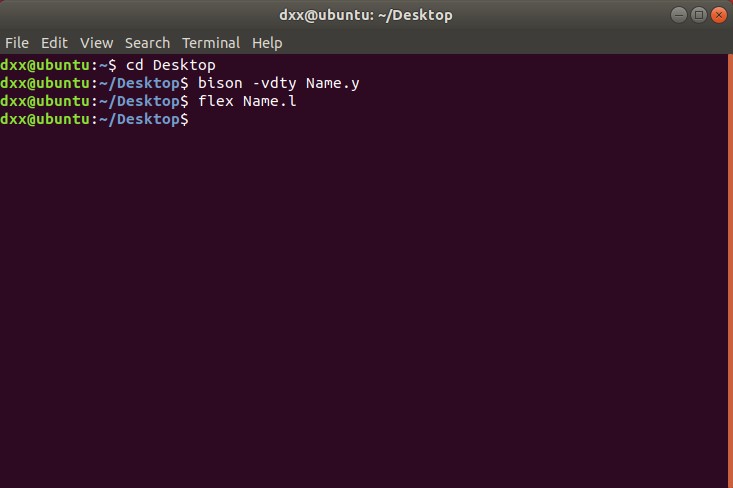


图28 生成lex.yy.c输入的命令

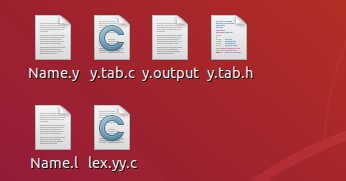


图29 生成lex.yy.c文件

## 5.4 编写测试文件Name.txt并测试

### 5.4.1 编写测试文件Name.txt

打开文本编辑器，编写Name.txt测试文件并保存到桌面。

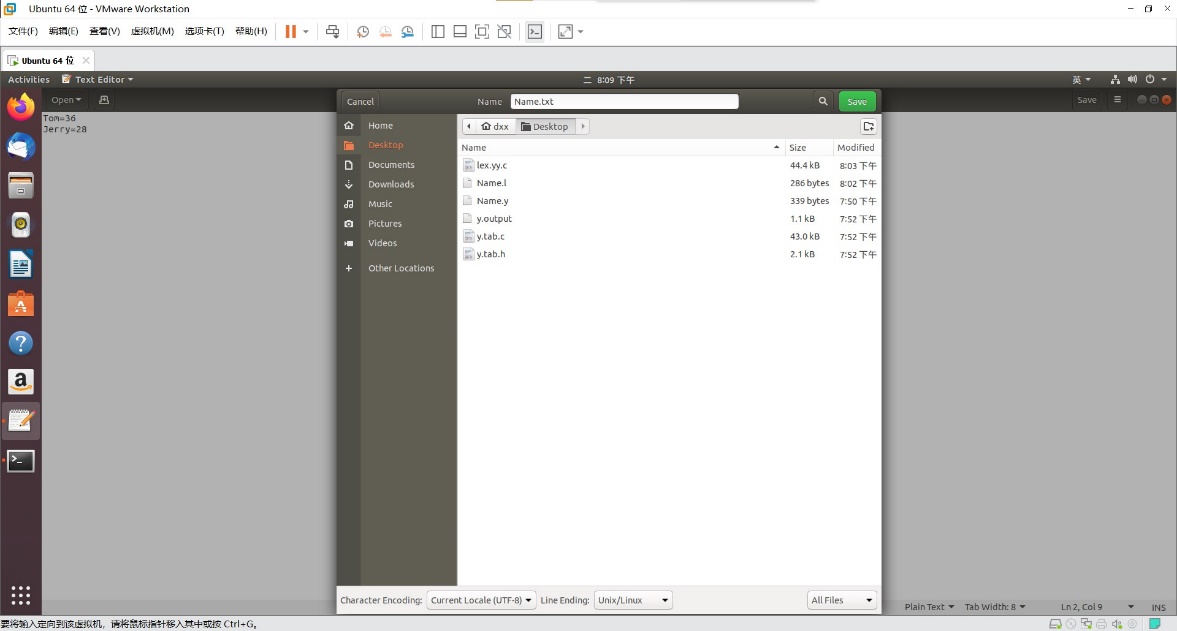


图30 编写Name.txt测试文件并保存

### 5.4.2 测试Name.txt文件

打开命令窗口，输入cd Desktop命令进入桌面目录，再输入cc -o parser y.tab.c lex.yy.c生成parser文件，最后输入./parser < Name.txt命令进行语法分析，并显示结果。

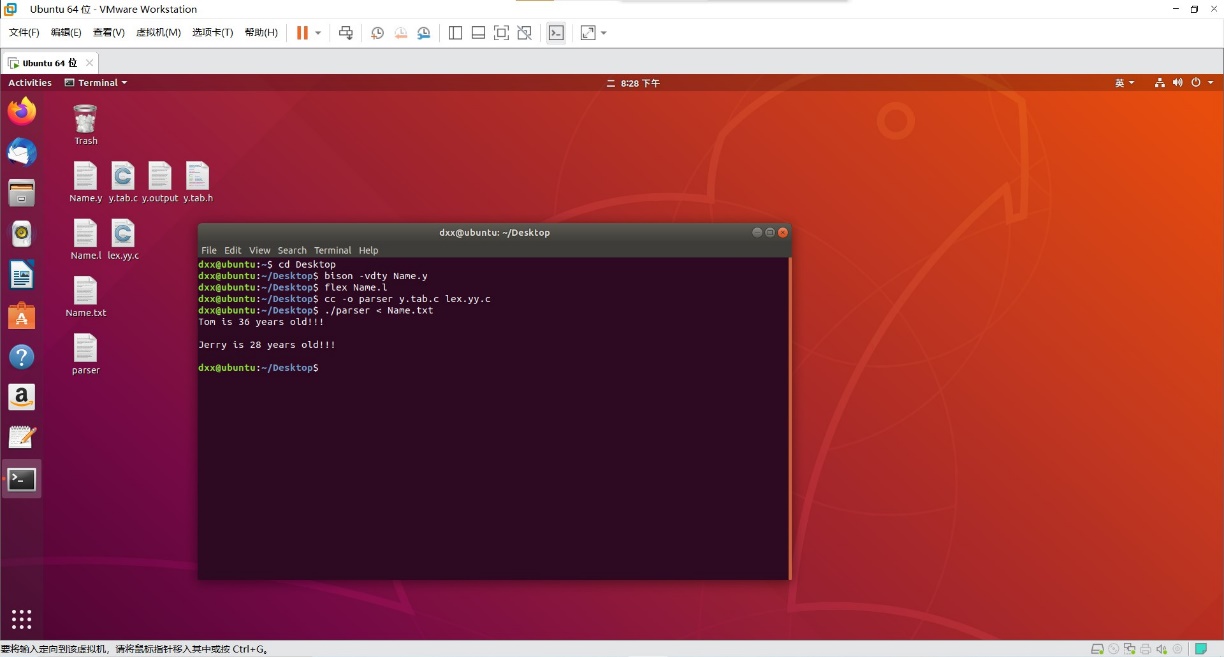


图31 命令输入和结果输出

### 5.4.3 分析程序输出结果

结果分析同Windows环境下的执行结果分析一致，在此不再赘述。

# 6 实验体会

## 6.1 实验中的困难

在实验过程中，遇到的主要困难是对Bison的语法规则不熟悉以及Flex与Bison之间的代码联系不清楚。以至于在刚开始编程时无从下手，编写完成后在生成最后的可执行文件时出现警告报错，导致出现输出结果乱码的现象。通过仔细查阅相关资料，阅读相关教程，逐渐学会了Bison的语法规则，理解了Name.y和 Name.l中的代码以及二者的关联，对语法分析的过程也有了更多的认识。

## 6.2 程序评价

本次实验所编写的程序实现了一个简单的语法分析器并正确输出了分析结果，代码清晰易懂，但是过于简单，实际意义不大，对于复杂的语法分析仍有很多东西要学习。

## 6.3 实验收获

在本次实验中，我学习了利用Bison结合Flex在Windows和Linux环境下对文本进行语法分析的方法。通过此次实验，掌握了Bison的源程序结构和语法，理解了语法分析的过程，为今后的学习打下了基础。

## 6.4 Flex与Bison的关系

Flex和Bison分别完成编译过程中的两个阶段，Flex负责词法分析，Bison负责语法分析，二者是互不相同却又相互协作。词法分析的主要任务是从左到右依次对源程序进行扫描，产生一个个单词序列token，输入到语法分析部分；而语法分析是在词法分析确定每个token单词符号正确的基础上，识别由单词符号构成的句子是否符合语言的语法规范，并执行相应的操作。比如此次实验中：”Tom=36”,由词法分析取出的单词Tom为NAME，=为EQ，36为AGE与规则record->NAME EQ AGE匹配，进而进行归约并执行相应操作。