## 1 语言说明

### 1.1 顶层定义top

Program -> ExtDefList //全局定义

ExtDefList -> ExtDef ExtDefList

ExtDef -> VarDec

| FuncDef

VarDec -> Specificer VarList ; //变量声明

VarList -> Var, VarList | Var

Var -> ID | ID = Exp

FuncDef -> Specificer ID(ParamList) CompSt //函数定义

ParamList -> Param, ParamList | Param | ε

Param -> Specificer ID

### 1.2 语句stmt

CompSt -> {StmtList} //复合语句 | 语句块

StmtList -> Stmt StmtList | ε

Stmt -> Exp;

| CompSt

| VarDec

| ID = Exp;

| ID OP\_ASSIGN Exp;

| IF ( Exp ) Stmt

| IF ( Exp ) Stmt ELSE Stmt

| WHILE ( Exp ) Stmt

| RETURN Exp;

### 1.3 表达式Exp

Exp -> ID

| C\_INT

| C\_FLOAT

| C\_CHAR

| TRUE | FALSE

| ( Exp )

| Exp , Exp

| Exp + Exp | Exp – Exp

| Exp \* Exp | Exp / Exp | Exp % Exp

| Exp REL\_OP Exp

| S\_PLUS Exp | S \_MINUS Exp | Exp S \_PLUS | Exp S \_MINUS

| Exp AND Exp

| Exp OR Exp

| ! Exp

| - Exp

| ID ( RParamList )

RParamList ->RParam, RParamList | RParam | ε //实参列表

RParam -> Exp

### 1.3 具体取值

Specificer -> INT

| FLOAT

| CHAR

| BOOL

## 2 词法分析

### 2.1 记号说明token

由于Bison中定义的记号默认从258开始编码，故单个字符的记号将其默认编码为其ASCII值。

1. 杂项

ID //关键字

C\_INT //int常量

C\_FLOAT //float常量

C\_CHAR //char 常量

1. 操作符//

//. //dot

//R\_ARROW //->

S\_PLUS //++

S\_MINUS //--

+, -, \*, /, %

REL\_OP // <, >, <=, >=, ==, !=

AND // &&

OR // ||

!

, //comma

=

OP\_ASSIGN // +=, -=, \*=, /=

1. 边界符

;

()

{}

1. 关键字

INT,

FLOAT,

CHAR,

VOID,

IF,

ELSE,

WHILE,

RETURN,

BOOL,

TRUE,

FALSE,

### 2.2 flex说明

1 在上面的正则表达式优先匹配

2 “ … ”双引号括起来的字符中除了转义字符之外其它的元字符（含特殊含义的字符如\*代表重复0个或多个）将失去其特殊意义。

3 flex处理冲突方式：1）对于同一个正则表达式，尽可能匹配多的字符。 2）对于不同正则表达式，优先匹配词法分析器中更早定义的模式。

4 辅助部分定义的正则表达式不能换行，或(|)之间不能有空格

5 规则部分不能有空行，使用/\*\*/时，要以一个空格开头

## 3 语法分析

### 3.1 非终结符

Program ExtDefList ExtDef VarDec FuncDef Specificer VarList Var ParamList CompSt Param

StmtList Stmt Exp RParamList Rparam

### 3.2 bison说明

1 编译参数-d 生成xxx.tab.h。包含终结符（记号）的宏定义（或枚举定义）

2 编译参数-v 生成parser.output。说明（LALR（1）分析法）哪几个状态有多少个冲突项

3 yylval是一个Flex和Bison共用的内部变量，类型为YYLVAL，按这样的方式，在Flex中通过yylval的成员保存单词属性值，在Bison中就可以通过yylval的成员取出属性值，实现了数据的传递。

4 冲突处理

Bison使用LALR分析法，会自动报告冲突。类似于Exp->Exp + Exp | Exp \* Exp的文法易于理解，但会产生二义性。在产生冲突后，可以通过指定结合性，和优先级来解决冲突。

%left %right %noassoc（没有结合性）%prec（对产生式指定优先级，默认为规则最右边记号的优先级）

5 Not found unistd.h错误：在windows下没有unistd头文件，使用VS编译会报错。（MinGW中有该头文件，编译不报错） 解决办法：create a dummy header file named unistd.h, include io.h and process.h

6 How to run it? 在make编译生成.exe文件后，直接运行，输入一些字符后输入EOF (windows下为ctrl-Z, linux下为ctrl-D)。或者使用重定向，’<’将文件内容作为输入，’>file 2>&2’将输出和错误输出到文件里。即：my\_gcc.exe <sample.c >output 2>&1

### 3.3 union

1 地址对齐。不同的变量定义顺序会导致不同的空间大小。

2 bision -d编译.y文件生成的xxx.tab.h中包含union YYSTYPE的定义，并声明了从其它文件导入yylval变量，因此我们需要在lex.l中定义yylval

typedef *union* YYSTYPE

{

/\*\*/

} YYSTYPE

extern YYSTYPE yylval;

3 文法符号对应的属性

|  |  |  |
| --- | --- | --- |
| **符 号** | **属 性** | **备注** |
| ID | 存放标识符字符串的字符数组 | 之后可改成符号表地址 |
| C\_INT | Int变量 |  |
| C\_FLOAT | Float变量 |  |
| C\_CHAR | Char变量 |  |
| 所有非终结符 | 抽象语法树的结点指针 | struct node \* |
| 其它终结符 | 可忽略 |  |

### 3.3 AST的表示

AST不同于语法分析树，忽略了一些用于表示层次关系的节点。

#### 孩子表示法

每个节点有固定个若干个子节点代表child

优点：实现简单

缺点：空间开销大

#### 孩子-兄弟表示法

每个节点有其第一个孩子和其右边第一个兄弟节点的指针（二叉链表）

优点：存储效率高

缺点：遍历时要复杂些。

采用孩子兄弟表示法。

内部节点定义：

struct node{

char name[20]; //标识节点名称，如Exp+Exp可产生一个名为"+"的内部节点

struct node \*child, \*sibling;

}

叶子节点定义：

struct leaf{

struct node \*child, \*sibling;

int leaf\_type; //用于标识采用下面联合体的哪个属性，true, false节点特殊，leaf\_type直接可以表示其值。

union{

int type\_int; //用于整数叶子节点

float type\_float;

char type\_char;

char type\_id[20];

}

}