文法：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S->S;S | L->E | E-> num |  |  |
| S-> id=E | L->L,E | E->id |  |  |
| S-> print(L) |  | E->E-E |  |  |
|  |  | E->E+E |  |  |
|  |  | E->E\*E |  |  |
|  |  | E->E/E |  |  |
|  |  | E->S,E |  |  |
|  |  | E->(E) |  |  |

消去二义性和左递归

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S->GS’ | L->EL’ | E->TE’ | T->FT’ | F->num |
| S’->;GS’ | L’->,EL’ | E’->+TE’ | T’->\*FT’ | F->(E) |
| S’-> | L’-> | E’->-TE’ | T’->/FT’ |  |
| G->id=E |  | E’-> | T’-> |  |
| G->print(L) |  | E->print(L)S’,E |  |  |
|  |  | E->id X | X->T’E’ |  |
|  |  |  | X->=ES’,E |  |
|  |  |  |  |  |

First、follow

|  |  |  |  |
| --- | --- | --- | --- |
|  | NULLABLE | FIRST | FOLLOW |
| S | N | print id |  |
| S’ | Y | ; | , |
| G | N | print id | ; , |
| L | N | Id num ( print | ) |
| L’ | Y | , | ) |
| E | N | Id num ( print | ) , ; |
| E’ | Y | + - | ) , ; |
| T | N | num ( | ) , + - ； |
| T’ | Y | \* / | ) , + - ； |
| F | N | num ( | ) , + - ；\* / |
| X | N | =,+-\*/ | ) , ; |

代码：

enmu token {EOF, ID, NUM, PLUS,MINUS, LPRAREN, PRINT, COMMA, RPRAREN, TIMES,DIV, SEMICOLON,ASSIGH };

union tokenval {string id; int num; }

enum token tok;

union tokenval tokval;

int F\_follow[ ]={RPRAREN, COMMA, PLUS,MINUS, SEMICOLON, TIMES,DIV ,-1};

int F(void) {switch (tok) {

case NUM: { int i=tokval.num; advance(); return i; }

case LPRAREN : eat(LPAREN); { int i=E();

eatOrSkipTo(RPAREN, F\_follow);

return i; }

case EOF:

default: printf( “expectedNUM, or left-paren”);

skipto(F\_follow);

return 0;

}

}

Int T\_follow[ ]= { RPRAREN, COMMA, PLUS,MINUS, SEMICOLON, -1};

Int T(void) { switch (tok) {

case NUM:

case LPRAREN : return Tprime ( F ( ) );

default: printf( “expected ID, NUM, or left-paren”);

skipto(T\_follow);

return 0;

}

}

Int Tprime\_follow[ ]= { RPRAREN, COMMA, PLUS,MINUS, SEMICOLON, -1};

Int Tprime( int a) { switch (tok) {

case TIMES: eat(TIMES); return Tprime( a\*F());

case DIV: eat(DIV); return Tprime( a/F());

case PLUS:

case MINUS:

case RPRAREN :

case COMMA:

case SEMICOLON:

case EOF: return a;

default: printf( “expected TIMES, DIV,PLUS,MINUS,right-paren, COMMA or SEMICOLON”);

skipto(Tprime\_follow);

return 0;

}

}

Int E\_follow[ ]={ RPRAREN, COMMA, SEMICOLON, -1};

Int E(void) { switch (tok) {

case NUM:

case LPRAREN : return Eprime (T ( ) );

case ID: int i=lookup(tokval.id)，advance( ); X(tokval.id),return i;

case PRINT: advance( ); eat(LPAREN); printf(“%d”,L( )) ; eat(RPAREN);

Sprime( ); eat(COMMA); return E();

default: printf( “expected ID, NUM, PRINT,or left-paren”);

skipto(E\_follow);

return 0;

}

}

Int Eprime\_follow[ ]={ RPRAREN, COMMA, SEMICOLON, -1};

Int Eprime(int a) { switch (tok) {

case PLUS: eat(PLUS); return Eprime( a+T());

case MINUS: eat(MINUS); return Eprime( a-T());

case RPRAREN :

case COMMA:

case SEMICOLON:

case EOF: return a;

default: printf( “expected PLUS, MINUS,COMMA,SEMICOLON,or right-paren”);

skipto(Eprime\_follow);

return 0;

}

}

Int L\_follow[ ]={ RPRAREN, -1};

Int L(void) { switch (tok) {

case ID:

case NUM:

case LPRAREN :

case PRINT: return Lprime (E ( ) );

default: printf( “expected ID, NUM, PRINT,or left-paren”);

skipto(L\_follow);

return 0;

}

}

Int Lprime\_follow[ ]={ RPRAREN, -1};

Int Lprime(int a) { switch (tok) {

case COMMA: eat(COMMA); return(Lprime (E ( ) );

case RPRAREN:

case EOF: return a;

default: printf( “expected COMMA or right-paren”);

skipto(Lprime\_follow);

return 0;

}

}

Int G\_follow[ ]={ COMMA,SEMICOLON, -1};

void G(void) { switch (tok) {

case PRINT: eat(PRINT); eat(LPRAREN); advance();

printf(“%d”,L( )) ; eat(RPAREN);L();

case ID : eat(ASSIGN); update(&table, tokval.id ,E( ));

default: printf( “expected PRINT or ID”);

skipto(G\_follow);

return 0;

}

}

Int S\_follow[ ]={ -1};

Int S(void) { switch (tok) {

case ID:

case PRINT: return Sprime (G ( ) );

default: printf( “expected ID or PRINT”);

skipto(S\_follow);

return 0;

}

}

Int Sprime\_follow[ ]={ COMMA,-1};

void Sprime(void) { switch (tok) {

case SEMICOLON: eat(SEMICOLON); return(Sprime (G ( ) );

case COMMA:

case EOF:

default: printf( “expected COMMA or SEMICOLON”);

skipto(Lprime\_follow);

return 0;

}

}

Int x\_follow[ ]={ -1};

Int x(string id) { switch (tok) {

case TIMES:

case DIV:

case PLUS:

case MINUS : int i=lookup(id); return Eprime (T prime (i ) );

case ASSIGH: advance( ); E(); SPrime( ); eat(COMMA);

update(&table,id, E());

default: printf( “expected TIMES, DIV, PLUS,MINUS,or ASSIGH”);

skipto(X\_follow);

return 0;

}

}