Análise Sintática

Analisador Descendente Recursivo

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Expressões pré-fixas

Considere a cadeia **+b*ab**

Como é sua derivação mais à esquerda?

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

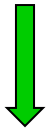
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

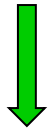
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

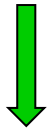
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

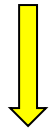
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

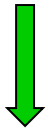
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

$+b*aE$

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

$+b*aE$

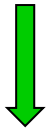
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **+b*ab**

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

$+b*aE$

$+b*ab$

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **+b*ab** ✓ `Cadeia Aceita`

Como é sua derivação mais à esquerda?

E

$+EE$

$+bE$

$+b*EE$

$+b*aE$

$+b*ab$

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **ba**

Como é sua derivação mais à esquerda?

E

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **ba**

Como é sua derivação mais à esquerda?

E

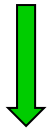
Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$



Cadeia: **ba**

Como é sua derivação mais à esquerda?

E

b

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **ba**

Como é sua derivação mais à esquerda?

E

b

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **ba**

Como é sua derivação mais à esquerda?

E

b

Análise Descendente (Predictive Parsing)

$E \rightarrow +EE$

$E \rightarrow *EE$

$E \rightarrow a$

$E \rightarrow b$

Cadeia: **ba** ✗ `Cadeia Recusada`

Como é sua derivação mais à esquerda?

E

b

Análise Descendente (Predictive Parsing)

Também chamada de *recursive-descent* ou *top-down*

É um algoritmo simples, capaz de fazer o *parsing* de gramáticas LL

Cada produção se torna uma cláusula em uma função recursiva

Tem-se uma função para cada não-terminal

A análise descendente produz uma derivação à esquerda

Ela precisa determinar a produção a ser usada para expandir o não-terminal corrente

Análise Descendente (Predictive Parsing)

$S \rightarrow \textit{if}\ E\ \textit{then}\ S\ \textit{else}\ S$

$S \rightarrow \textit{begin}\ S\ L$

$S \rightarrow \textit{print}\ E$

$L \rightarrow \textit{end}$

$L \rightarrow ;\ S\ L$

$E \rightarrow \textit{num} = \textit{num}$

Como seria um *parser* para essa gramática?

Analisador Léxico

```
/* Nao usar a biblioteca do flex*/
%option noyywrap

%%

"if"          { return 1; /*IF*/    }
"then"        { return 2; /*THEN*/  }
"else"        { return 3; /*ELSE*/  }
"begin"       { return 4; /*BEGIN*/ }
"end"         { return 5; /*END*/   }
"print"       { return 6; /*PRINT*/ }
";"           { return 7; /*SEMI*/  }
[+-]?[0-9]+   { return 8; /*NUM*/   }
"="           { return 9; /*EQ*/    }
\n            { /*quebra de linha*/ }
" "           { /*espaço em branco*/}

%%
int getToken();
{
    return yylex();
}
```

$S \rightarrow if\ E\ then\ S\ else\ S$

$S \rightarrow begin\ S\ L$

$S \rightarrow print\ E$

$L \rightarrow end$

$L \rightarrow ;\ S\ L$

$E \rightarrow num = num$

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
        case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
        case BEGIN: eat(BEGIN); S(); L(); break;
        case PRINT: eat(PRINT); E(); break;
        default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
        case SEMI: eat(SEMI); S(); L(); break;
        default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow \textit{if}\ E\ \textit{then}\ S\ \textit{else}\ S$
$S \rightarrow \textit{begin}\ S\ L$
$S \rightarrow \textit{print}\ E$
$L \rightarrow \textit{end}$
$L \rightarrow ;\ S\ L$
$E \rightarrow \textit{num} = \textit{num}$

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

$S \rightarrow$ *if* E *then* S *else* S
$S \rightarrow$ *begin* $S\ L$
$S \rightarrow$ *print* E
$L \rightarrow$ *end*
$L \rightarrow$ ***;*** $S\ L$
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

S → *if E then S else S*
S → *begin S L*
S → *print E*
L → *end*
L → *; S L*
E → *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

S → *if E then S else S*
S → *begin S L*
S → *print E*
L → *end*
L → *; S L*
E → *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if* E *then* S *else* S

$S \rightarrow$ *begin* $S\ L$

$S \rightarrow$ *print* E

$L \rightarrow$ *end*

$L \rightarrow$ ***;*** $S\ L$

$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if* E *then* S *else* S

$S \rightarrow$ *begin* $S\ L$

$S \rightarrow$ *print* E

$L \rightarrow$ *end*

$L \rightarrow$ *;* $S\ L$

$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

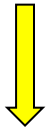
$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```



print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

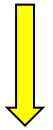
$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```



print num = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if* E *then* S *else* S

$S \rightarrow$ *begin* $S\ L$

$S \rightarrow$ *print* E

$L \rightarrow$ *end*

$L \rightarrow$ ***;*** $S\ L$

$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print num = num ✓ Cadeia Aceita

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

$S \rightarrow$ *if* E *then* S *else* S
$S \rightarrow$ *begin* S L
$S \rightarrow$ *print* E
$L \rightarrow$ *end*
$L \rightarrow$ ***;*** S L
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if* E *then* S *else* S

$S \rightarrow$ *begin* $S\ L$

$S \rightarrow$ *print* E

$L \rightarrow$ *end*

$L \rightarrow$ ***;*** $S\ L$

$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;
```

$S \rightarrow$ *if* E *then* S *else* S

$S \rightarrow$ *begin* $S\ L$

$S \rightarrow$ *print* E

$L \rightarrow$ *end*

$L \rightarrow$ ***;*** $S\ L$

$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num

```
void S() -> case PRINT: eat(PRINT); E(); break;

void E() -> { eat(NUM); eat(EQ); eat(NUM); }
```

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Análise Descendente (Predictive Parsing)

```
int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6, SEMI=7, NUM=8, EQ=9;

int token = getToken();
void advance() {token=getToken();}
void eat(int t) {if (token==t) advance(); else error();}

void S(){
    switch(token) {
          case IF: eat(IF); E(); eat(THEN); S(); eat(ELSE); S(); break;
          case BEGIN: eat(BEGIN); S(); L(); break;
          case PRINT: eat(PRINT); E(); break;
          default: error(); }
}
void L(){
    switch(token) {
        case END: eat(END); break;
          case SEMI: eat(SEMI); S(); L(); break;
          default: error(); }
}
void E(){ eat(NUM); eat(EQ); eat(NUM); }
```

print begin = num ✘ Cadeia Recusada

$S \rightarrow$ *if E then S else S*
$S \rightarrow$ *begin S L*
$S \rightarrow$ *print E*
$L \rightarrow$ *end*
$L \rightarrow$ *; S L*
$E \rightarrow$ *num = num*

Lista de Exercícios

Lista 9

- Exercícios de Implementação