1. Design a compiler system with the following specifications:

Tokens for Lexical Analysis:

*id* – Boolean Variables with Regular Expression $L(L-D)^*$, $L$ = English Letter, $D$ = Decimal Digit

*TRUE & FALSE* – Boolean Constants

*AND, OR & NOT* – Boolean Operators

'(', ')', ';' – Punctuations

= - Assignment Operators

Grammar with Syntax Directed Translation Rules for Three Address Code:

| Rule No. | Rules | Translation Rules |
|---|---|---|
| 1. | $S \rightarrow id = E;$ | $S.Code = \{E.Code \mid id = E.Place\}$ |
| 2. | $E \rightarrow E\ OR\ T$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place \vee T.Place\}$ |
| 3. | $E \rightarrow T$ | $E.Code = \{T.Code\}$ |
| 4. | $T \rightarrow T\ AND\ F$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place \wedge T.Place\}$ |
| 5. | $T \rightarrow F$ | $T.Code = \{F.Code\}$ |
| 6. | $F \rightarrow (E)$ | $F.Code = \{E.Code\}$ |
| 7. | $F \rightarrow NOT\ (E)$ | $F.Place = NewTemp(\ )$<br>$F.Code = \{E.Code \mid F.Place = \neg E.Place\}$ |

| | | |
|---|---|---|
| 8. | $F \rightarrow TRUE$ | $F.Value = TRUE$ |
| 9. | $F \rightarrow FALSE$ | $F.Value = FALSE$ |
| 10. | $F \rightarrow id$ | $F.Place = id.Value$ |

Example

**Input:**

$FSwitch = (HHumidity\ OR\ HTemp)\ AND\ (HPopulation\ OR\ NOT\ (PSW1\ OR\ PSW2))$;

**Output (Three Address Code):**

**Lexical Analyzer:**

$id = (id\ OR\ id)\ AND\ (id\ OR\ NOT\ (id\ OR\ id))$;

**Syntax Analyzer:**

Sequence of Production rules used

**Semantic Analyzer:**

Data Type Conversion used

**Intermediate Code Generator:**

$t_1 = PSW1 \vee PSW2$

$t_2 = \neg t_1$

$t_3 = HPopulation \vee t_2$

$t_4 = HHumidity \vee HTemp$

$t_5 = t_4 \wedge t_3$

$FSwitch = t_5$

2. Design a compiler system with the following specifications:

Tokens for Lexical Analysis:

*id* – Decimal Variables with Regular Expression $L(L+D)^*$, $L$ = English Letter, $D$ = Decimal Digit

*num* – Decimal Constants with Regular Expression $DD^* + DD^*.DD^*$, $D$ = Decimal Digit

'+', '-' & '*' – Algebraic Operators

'(', ')', ';' – Punctuations

'=' – Assignment Operators

Grammar with Syntax Directed Translation Rules for Three Address Code:

| Rule No. | Rules | Translation Rules |
|---|---|---|
| 1. | $S \rightarrow id = E;$ | $S.Code = \{E.Code \mid id = E.Place\}$ |
| 2. | $E \rightarrow E + T$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place + T.Place\}$ |
| 3. | $E \rightarrow T$ | $E.Code = \{T.Code\}$ |
| 4. | $T \rightarrow T * F$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place * T.Place\}$ |
| 5. | $T \rightarrow F$ | $T.Code = \{F.Code\}$ |
| 6. | $F \rightarrow (E)$ | $F.Code = \{E.Code\}$ |
| 7. | $F \rightarrow -(E)$ | $F.Place = NewTemp(\ )$<br>$F.Code = \{E.Code \mid F.Place = -E.Place\}$ |
| 8 | $F \rightarrow num$ | $F.Value = num.value$ |

| | | |
|---|---|---|
| 9. | $F \rightarrow id$ | $F.Place = id.Value$ |

Example:

**Input:**

$Annuity = Principle * (1 - Rate * 0.001);$

**Output (Three Address Code):**

**Lexical Analyzer:**

$id = id * (num - id * num);$

**Syntax Analyzer:**

Sequence of Production rules used

**Semantic Analyzer:**

Data Type Conversion used

**Intermediate Code Generator:**

$t_1 = Rate * 0.001$

$t_2 = 1.0 - t_1$

$t_3 = Principle * t_2$

$Annuity = t_3$

3. Design a compiler system with the following specifications:

Tokens for Lexical Analysis:

***id*** – String Variables with Regular Expression $L(L+D)^*$, $L$ = English Letter, $D$ = Decimal Digit

***string*** – String Constants with Regular Expression "" – "CC"*, $C$ = any printable ASCII character

*+ and &* - String Concatenation Operators

'(', ')', ';' – Punctuations

= - Assignment Operators

Grammar with Syntax Directed Translation Rules for Three Address Code:

| Rule No | Rules | Translation Rules |
|---|---|---|
| 1. | $S \rightarrow id = E;$ | $S.Code = \{E.Code \mid id = E.Place\}$ |
| 2. | $E \rightarrow E + T$ | $E.Place = NewTemp()$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place + T.Place\}$ |
| 3. | $E \rightarrow T$ | $E.Code = \{T.Code\}$ |
| 4. | $T \rightarrow T \& F$ | $E.Place = NewTemp()$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place + T.Place\}$ |
| 5. | $T \rightarrow F$ | $T.Code = \{F.Code\}$ |
| 6. | $F \rightarrow (E)$ | $F.Code = \{E.Code\}$ |
| 8. | $F \rightarrow string$ | $string.address = createspace(string.length)$<br>$Memory(string.address) = string.value$<br>$F.Value = string.address$ |
| 9. | $F \rightarrow id$ | $F.Place = id.Value$ |

Example:

**Input:**

$Address = HouseName - HouseNo + StreetName + (City\&Village) + District + PoliceStation + State + "India" + PinCode;$

**Output (Three Address Code):**

**Lexical Analyzer:**

$id = id - id - id - (id\&id) + id - id - string - id;$

**Syntax Analyzer:**

Sequence of Production rules used

**Semantic Analyzer:**

Data Type Conversion used

**Intermediate Code Generator:**

$t_1 = City + Village$

$t2 = 1.0 + t1$

$t_2 = HouseName + HouseNo$

$t_3 = t_2 - StreetName$

$t_4 = t_3 + t_2 \ldots$

4. Design a compiler system with the following specifications:

Tokens for Lexical Analysis:

*id* – Integer Variables with Regular Expression $L(L-D)^*$, $L = English\ Letter$, $D = Decimal\ Digit$

*num* – Integer Constants with Regular Expression $DD^*$, $D = Decimal\ Digit$

*>* and *<* – Logical Operators

(, ), , – Punctuations

= – Assignment Operators

Grammar with Syntax Directed Translation Rules for Three Address Code:

| Rule No. | Rules | Translation Rules |
|---|---|---|
| 1 | $S \rightarrow id = E;$ | $S.Code = \{E.Code \mid id = E.Place\}$ |
| 2. | $E \rightarrow E > T$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place > T.Place\}$ |
| 3. | $E \rightarrow T$ | $E.Code = \{T.Code\}$ |
| 4. | $T \rightarrow T < F$ | $E.Place = NewTemp(\ )$<br>$E.Code = \{T.Code \mid E.Code \mid E.Place = E.Place < T.Place\}$ |
| 5. | $T \rightarrow F$ | $T.Code = \{F.Code\}$ |
| 6. | $F \rightarrow (E)$ | $F.Code = \{E.Code\}$ |
| 7. | $F \rightarrow !(E)$ | $F.Place = NewTemp(\ )$<br>$F.Code = \{E.Code \mid F.Place = !E.Place\}$ |
| 8. | $F \rightarrow num$ | $F.Value = num.value$ |
| 9 | $F \rightarrow id$ | $F.Place = id.Value$ |

Example:

**Input:**

$A = ((B < A) > (45 > !(C < D)));$

**Output (Three Address Code):**

**Lexical Analyzer:**

$id = ((id < id) > (num > 1(id < id)))$

**Syntax Analyzer:**

Sequence of Production rules used

**Semantic Analyzer:**

Data Type Conversion used

**Intermediate Code Generator:**

$t_1 = C < D$

$t_2 = T t_1$

$t_3 = num > t_2$

$t_4 = B < A$

$t_5 = t_4 > t_3$

$A = t_5$

5. Design a compiler system with the following specifications.

Token for Lexical Analysis:

***id*** – Decimal Floating Point Variable with Regular Expression

$L(L + D)^*, L = English\ Letter, D = Decimal\ Digit$

***num*** – Floating Point Constant with Regular Expression

$(+ + -)D^+.DD^*(E + -)DD^*, D = Decimal\ Digit$

| Rule No. | Rules | Translation Rules |
|---|---|---|
| 1. | $S \rightarrow id = E_1$ | $S.Value = (E.Value\|id.E.Value =)$ |
| 2. | $E - E + T$ | $E.Value = (E.Value T.Value +)$ |
| 3. | $E - T$ | $E.Value = (T.Value)$ |
| 4. | $E - E/T$ | $E.Value = (E.Value T.Value/)$ |
| 5. | $T - F$ | $T.Value = (F.Value)$ |
| 6. | $F - (E)$ | $F.Value = (E.Value)$ |
| 7. | $F - num$ | $F.Value = num.Value$ |
| 8. | $F - id$ | $F.Value = id.Value$ |

Example:

**Input:**

$Force = 6.087E - 28 * (Mass1 * Mass2)/(R * R)$;

**Output (Post-Fix Expression):**

**Lexical Analyzer:**

$id = num * (id * id)/(id * id)$;

**Syntax Analyzer:**
Sequence of Production rules used

**Semantic Analyzer:**
Data type Conversion used

**Intermediate Code:**
*Force 6.087E − 28 Mass1 Mass2 * R R * / * =*

6. Design a compiler system with the following specifications:
Token for Lexical Analysis:
***id*** – Octal Numeric Variable with Regular Expression
*L(L + D)*, L = English Letter, D = Decimal Digit*
***num*** – Octal Numeric Constant with Regular Expression
*(ϵ | + | −)D'.DD* D = Octal Digit, D' = Octal Digit without zero*

| Rule No. | Rules | Translation Rules |
|---|---|---|
| 1. | $S \rightarrow id = E;$ | $S.Value = \{E.Value \mid id = E.Value\}$ |
| 2. | $E \rightarrow E + T$ | $E.Value = \{E.Value + T.Value\}$ |
| 3. | $E \rightarrow T$ | $E.Value = \{T.Value\}$ |
| 4. | $E \rightarrow E * T$ | $E.Value = \{E.Value * T.Value\}$ |
| 5. | $T \rightarrow F$ | $T.Value = \{F.Value\}$ |
| 6. | $F \rightarrow (E)$ | $F.Value = \{E.Value\}$ |
| 7. | $F \rightarrow num$ | $F.Value = num.Value$ |
| 8. | $F \rightarrow id$ | $F.Value = id.Value$ |

Example
**Input:**
*u = 67; f = 90; t = 7; S = u * t + 0.5 * f * t * t;*
**Output (Evaluated Value):**
**Lexical Analyzer:**
*id = num; id = num; id = num; id = id * id − num * id * id * id;*
**Syntax Analyzer:**
Sequence of Production rules used
**Semantic Analyzer:**
Data type Conversion used
**Intermediate Code:**
*S = Evaluated value*

***