به نام خدا

درس:طراحی کامپایلر

پروژه فاز دوم:تحلیل گر نحوی

محمد کاظم هرندی (۴۰۰۳۶۲۳۰۳۹)

مسعود محمدزاده(۴۰۱۲۳۶۳۱۴۷)

باید برای متغیر ها و نوع های داده ای،دستورات شرطی،حلقه ها،توابع،دستور چاپ گرامر بنویسیم

مرحله اول ساخت گرامر:

ابتدا باید زبان مورد نظر و قواعد گرامری آن را به دقت تعریف کنید. این شامل تعیین نوع جملات، ساختار عبارات و ترتیب واژگان است چون میخواهیم از تجزیه کننده ll(1) استفاده کنیم باید از گرامر Context-Free Grammar استفاده کنیم و گرامری که مینویسیم باید سه شرط را باید رعایت کنیم:۱-مبهم نبودن۲-بازگشت چپ ۳-فاکتور چپ

که برای گرامر رو به رو این کار ها انجام شده است:

```
rules = {
    "Program": [["DeclarationBlock"]],

    "DeclarationBlock": [["SingleDec", "DeclarationExtension"]],
    "DeclarationExtension": [["SingleDec", "DeclarationExtension"], ["epsilon"]],

    "SingleDec": [["DataType", "t_id", "DeclarationType"]],
    "DeclarationType": [["VariableDeclaration"], ["FunctionDeclaration"]],

    "DataType": [["t_int"], ["t_bool"], ["t_char"]],

    "VariableDeclaration": [["VariableList", "t_semicolon"]],
    "VariableList": [["VariableInitializer", "VariableListExtension"]],
    "VariableListExtension": [["t_comma", "t_id", "VariableInitializer",
"VariableListExtension"], ["epsilon"]],

    "VariableInitializer": [["ArrayDeclaration", "VarInitializerExtension"]],
    "VarInitializerExtension": [["t_assign", "ExpressionStructure"],
["epsilon"]],

    "ArrayDeclaration": [["t_lb", "ArrayDimension", "t_rb"], ["epsilon"]],
    "ArrayDimension": [["ExpressionStructure"], ["epsilon"]],

    "FunctionDeclaration": [["t_lp", "FunctionParameters", "t_rp",
"ActionStatement"]],
    "FunctionParameters": [["ParametersList"], ["epsilon"]],
    "ParametersList": [["DataType", "t_id", "ParametersExtension"]],
    "ParametersExtension": [["t_comma", "DataType", "t_id", "ArrayDeclaration",
"ParametersExtension"], ["epsilon"]],

    "ActionStatement": [["CompoundStatement"], ["SimpleAction"],
["ConditionStatement"],
```

```
                        ["IterationStatement"], ["OutputStatement"],
["BreakAction"],
                        ["ReturnAction"], ["ContinueAction"],
["VariableDeclarationStatement"]
                       ],

    "CompoundStatement": [["t_lc", "StatementSequence", "t_rc"]],
    "StatementSequence": [["ActionStatement", "StatementSequence"], ["epsilon"]],

    "ConditionStatement": [["t_if", "ExpressionStructure", "CompoundStatement",
"AlternativeStatement"]],
    "AlternativeStatement": [["t_else", "CompoundStatement"], ["epsilon"]],

    "IterationStatement": [["t_for", "t_lp", "ForInitialization", "t_rp"]],
    "ForInitialization": [["LoopVarInitialization", "t_semicolon",
"LoopExpression", "t_semicolon", "LoopStep"]],

    "LoopVarInitialization": [["DataType", "t_id", "t_assign",
"ExpressionStructure"], ["t_id", "t_assign", "ExpressionStructure"],
["epsilon"]],
    "LoopExpression": [["ExpressionStructure"], ["epsilon"]],
    "LoopStep": [["SimpleAction2"], ["epsilon"]],

    "SimpleAction": [["t_id", "ArrayAccess", "t_assign", "ExpressionStructure",
"t_semicolon"]],
    "SimpleAction2": [["t_id", "ArrayAccess", "t_assign",
"ExpressionStructure"]],
    "ArrayAccess": [["t_lb", "ArrayDimension2", "t_rb"], ["epsilon"]],
    "ArrayDimension2": [["ExpressionStructure"]],

    "VariableDeclarationStatement": [["DataType", "t_id", "VariableList",
"t_semicolon"]],

    "ReturnAction": [["t_return", "t_semicolon"], ["t_return",
"ExpressionStructure", "t_semicolon"]],

    "BreakAction": [["t_break", "t_semicolon"]],

    "ContinueAction": [["t_continue", "t_semicolon"]],

    "OutputStatement": [["t_print", "t_lp", "OutputRules", "t_rp",
"t_semicolon"]],
    "OutputRules": [["ExpressionStructure", "PrintExtension"]],
    "PrintExtension": [["t_comma", "ExpressionStructure", "PrintExtension"],
["epsilon"]],
```

```
    "ExpressionStructure": [["LogicalExpression"]],

    "LogicalExpression": [["AndExpression", "OrExtension"]],
    "OrExtension": [["t_lop_or", "AndExpression", "OrExtension"], ["epsilon"]],

    "AndExpression": [["NegationExpression", "AndChain"]],
    "AndChain": [["t_lop_and", "NegationExpression", "AndChain"], ["epsilon"]],

    "NegationExpression": [["t_lop_not", "NegationExpression"],
["ComparisonExpression"]],

    "ComparisonExpression": [["SimpleExpression", "ComparisonExtension"]],
    "ComparisonExtension": [["ComparisonOps", "SimpleExpression",
"ComparisonExtension"], ["epsilon"]],

    "ComparisonOps": [["t_rop_l"], ["t_rop_g"], ["t_rop_le"], ["t_rop_ge"],
["t_rop_ne"], ["t_rop_e"]],

    "SimpleExpression": [["ArithmeticTerm", "ArithmeticChain1"]],
    "ArithmeticChain1": [["t_aop_pl", "ArithmeticTerm", "ArithmeticChain1"],
["t_aop_mn", "ArithmeticTerm", "ArithmeticChain1"], ["epsilon"]],

    "ArithmeticTerm": [["ArithmeticFactor", "ArithmeticChain2"]],
    "ArithmeticChain2": [["t_aop_ml", "ArithmeticFactor", "ArithmeticChain2"],
["t_aop_dv", "ArithmeticFactor", "ArithmeticChain2"], ["t_aop_rm",
"ArithmeticFactor", "ArithmeticChain2"], ["epsilon"]],

    "ArithmeticFactor": [["t_aop_pl", "Elementary"], ["t_aop_mn", "Elementary"],
["Elementary"]],

    "Elementary": [["t_id", "FunctionOrArrayAccess"], ["t_decimal"],
["t_hexadecimal"],
                   ["t_string"], ["t_char"], ["t_true"], ["t_false"],
                   ["t_lp", "ExpressionStructure", "t_rp"]],

    "FunctionOrArrayAccess": [["epsilon"], ["t_lp", "DetailedParameters",
"t_rp"]],

    "DetailedParameters": [["ParametersList2"], ["epsilon"]],

    "ParametersList2": [["t_id", "ParametersTail"]],

    "ParametersTail": [["t_comma", "t_id", "ArrayDeclaration", "ParametersTail"],
["epsilon"]]
```

```
}
```

مرحله دوم به دست آوردن first**و** follow **متغییر ها را بدست می آوریم:**

Firstها:

```
all_first_sets = {
    'Program': {'t_bool', 't_int', 't_char'},
    'DeclarationBlock': {'t_bool', 't_int', 't_char'},
    'DeclarationExtension': {'t_bool', 't_int', 't_char', 'ε'},
    'SingleDec': {'t_bool', 't_int', 't_char'},
    'DeclarationType': {'t_comma', 't_assign', 't_lb', 't_semicolon', 't_lp'},
    'DataType': {'t_bool', 't_int', 't_char'},
    'VariableDeclaration': {'t_assign', 't_comma', 't_lb', 't_semicolon'},
    'VariableList': {'t_assign', 't_comma', 't_lb', 'ε'},
    'VariableListExtension': {'t_comma', 'ε'},
    'VariableInitializer': {'t_assign', 't_lb', 'ε'},
    'VarInitializerExtension': {'t_assign', 'ε'},
    'ArrayDeclaration': {'t_lb', 'ε'},
    'ArrayDimension': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_lop_not',
't_true', 't_lp', 't_char', 't_false', 'ε', 't_string', 't_hexadecimal'},
    'FunctionDeclaration': {'t_lp'},
    'FunctionParameters': {'t_bool', 't_int', 't_char', 'ε'},
    'ParametersList': {'t_bool', 't_int', 't_char'},
    'ParametersExtension': {'t_comma', 'ε'},
    'ActionStatement': {'t_bool', 't_for', 't_lc', 't_id', 't_return',
't_continue', 't_int', 't_break', 't_char', 't_print', 't_if'},
    'CompoundStatement': {'t_lc'},
    'StatementSequence': {'t_bool', 't_for', 't_lc', 't_id', 't_return',
't_continue', 't_int', 't_break', 't_char', 't_print', 'ε', 't_if'},
    'ConditionStatement': {'t_if'},
    'AlternativeStatement': {'t_else', 'ε'},
    'IterationStatement': {'t_for'},
    'ForInitialization': {'t_bool', 't_id', 't_char', 't_int', 't_semicolon'},
    'LoopVarInitialization': {'t_bool', 't_id', 't_char', 'ε', 't_int'},
    'LoopExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_lop_not',
't_true', 't_lp', 't_char', 't_false', 'ε', 't_string', 't_hexadecimal'},
    'LoopStep': {'t_id', 'ε'},
    'SimpleAction': {'t_id'},
    'SimpleAction2': {'t_id'},
```

```
    'ArrayAccess': {'t_lb', 'ε'},
    'ArrayDimension2': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_lop_not',
't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'VariableDeclarationStatement': {'t_bool', 't_int', 't_char'},
    'ReturnAction': {'t_return'},
    'BreakAction': {'t_break'},
    'ContinueAction': {'t_continue'},
    'OutputStatement': {'t_print'},
    'OutputRules': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_lop_not',
't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'PrintExtension': {'t_comma', 'ε'},
    'ExpressionStructure': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id',
't_lop_not', 't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'LogicalExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id',
't_lop_not', 't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'OrExtension': {'t_lop_or', 'ε'},
    'AndExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_lop_not',
't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'AndChain': {'t_lop_and', 'ε'},
    'NegationExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id',
't_lop_not', 't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'ComparisonExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id',
't_true', 't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'ComparisonExtension': {'t_rop_le', 't_rop_l', 't_rop_ne', 't_rop_ge',
't_rop_g', 'ε', 't_rop_e'},
    'ComparisonOps': {'t_rop_le', 't_rop_l', 't_rop_ne', 't_rop_ge', 't_rop_g',
't_rop_e'},
    'SimpleExpression': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_true',
't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'ArithmeticChain1': {'t_aop_pl', 't_aop_mn', 'ε'},
    'ArithmeticTerm': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_true',
't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'ArithmeticChain2': {'t_aop_rm', 't_aop_ml', 't_aop_dv', 'ε'},
    'ArithmeticFactor': {'t_aop_pl', 't_decimal', 't_aop_mn', 't_id', 't_true',
't_lp', 't_char', 't_false', 't_string', 't_hexadecimal'},
    'Elementary': {'t_decimal', 't_id', 't_char', 't_false', 't_string',
't_true', 't_hexadecimal', 't_lp'},
    'FunctionOrArrayAccess': {'t_lp', 'ε'},
    'DetailedParameters': {'t_id', 'ε'},
    'ParametersList2': {'t_id'},
    'ParametersTail': {'t_comma', 'ε'},
}
```

Followها:

```
all_follow_sets = {
    'Program': {'$'},
    'DeclarationBlock': {'$'},
    'DeclarationExtension': {'$'},
    'SingleDec': {'t_bool', 't_int', '$', 't_char'},
    'DeclarationType': {'t_int', '$', 't_bool', 't_char'},
    'DataType': {'t_id'},
    'VariableDeclaration': {'t_int', '$', 't_bool', 't_char'},
    'VariableList': {'t_semicolon'},
    'VariableListExtension': {'t_semicolon'},
    'VariableInitializer': {'t_comma', 't_semicolon'},
    'VarInitializerExtension': {'t_comma', 't_semicolon'},
    'ArrayDeclaration': {'t_semicolon', 't_assign', 't_comma', 't_rp'},
    'ArrayDimension': {'t_rb'},
    'FunctionDeclaration': {'t_int', '$', 't_bool', 't_char'},
    'FunctionParameters': {'t_rp'},
    'ParametersList': {'t_rp'},
    'ParametersExtension': {'t_rp'},
    'ActionStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'CompoundStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int',
't_id', 't_lc', 't_break', 't_bool', 't_continue', 't_else', 't_return',
't_char'},
    'StatementSequence': {'t_rc'},
    'ConditionStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int',
't_id', 't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'AlternativeStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int',
't_id', 't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'IterationStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int',
't_id', 't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'ForInitialization': {'t_rp'},
    'LoopVarInitialization': {'t_semicolon'},
    'LoopExpression': {'t_semicolon'},
    'LoopStep': {'t_rp'},
    'SimpleAction': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'SimpleAction2': {'t_rp'},
    'ArrayAccess': {'t_assign'},
    'ArrayDimension2': {'t_rb'},
    'VariableDeclarationStatement': {'t_rc', 't_print', '$', 't_if', 't_for',
't_int', 't_id', 't_lc', 't_break', 't_bool', 't_continue', 't_char',
't_return'},
    'ReturnAction': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
```

```
    'BreakAction': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'ContinueAction': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'OutputStatement': {'t_rc', 't_print', '$', 't_if', 't_for', 't_int', 't_id',
't_lc', 't_break', 't_bool', 't_continue', 't_char', 't_return'},
    'OutputRules': {'t_rp'},
    'PrintExtension': {'t_rp'},
    'ExpressionStructure': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb'},
    'LogicalExpression': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb'},
    'OrExtension': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb'},
    'AndExpression': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb',
't_lop_or'},
    'AndChain': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb', 't_lop_or'},
    'NegationExpression': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb',
't_lop_or', 't_lop_and'},
    'ComparisonExpression': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb',
't_lop_or', 't_lop_and'},
    'ComparisonExtension': {'t_lc', 't_semicolon', 't_comma', 't_rp', 't_rb',
't_lop_or', 't_lop_and'},
    'ComparisonOps': {'t_true', 't_aop_mn', 't_lp', 't_hexadecimal', 't_false',
't_id', 't_decimal', 't_aop_pl', 't_string', 't_char'},
    'SimpleExpression': {'t_comma', 't_rop_g', 't_lop_or', 't_rop_e', 't_rop_ge',
't_lc', 't_rop_l', 't_semicolon', 't_rp', 't_rb', 't_rop_le', 't_lop_and',
't_rop_ne'},
    'ArithmeticChain1': {'t_comma', 't_rop_g', 't_lop_or', 't_rop_e', 't_rop_ge',
't_lc', 't_rop_l', 't_semicolon', 't_rp', 't_rb', 't_rop_le', 't_lop_and',
't_rop_ne'},
    'ArithmeticTerm': {'t_aop_mn', 't_comma', 't_rop_g', 't_lop_or', 't_rop_e',
't_rop_ge', 't_lc', 't_rop_l', 't_semicolon', 't_aop_pl', 't_rp', 't_rb',
't_rop_le', 't_lop_and', 't_rop_ne'},
    'ArithmeticChain2': {'t_aop_mn', 't_comma', 't_rop_g', 't_lop_or', 't_rop_e',
't_rop_ge', 't_lc', 't_rop_l', 't_semicolon', 't_aop_pl', 't_rp', 't_rb',
't_rop_le', 't_lop_and', 't_rop_ne'},
    'ArithmeticFactor': {'t_aop_mn', 't_comma', 't_aop_dv', 't_aop_rm',
't_rop_g', 't_lop_or', 't_rop_e', 't_rop_ge', 't_lc', 't_rop_l', 't_semicolon',
't_aop_pl', 't_rp', 't_rb', 't_rop_le', 't_lop_and', 't_rop_ne', 't_aop_ml'},
    'Elementary': {'t_aop_mn', 't_comma', 't_aop_dv', 't_aop_rm', 't_rop_g',
't_lop_or', 't_rop_e', 't_rop_ge', 't_lc', 't_rop_l', 't_semicolon', 't_aop_pl',
't_rp', 't_rb', 't_rop_le', 't_lop_and', 't_rop_ne', 't_aop_ml'},
    'FunctionOrArrayAccess': {'t_aop_mn', 't_comma', 't_aop_dv', 't_aop_rm',
't_rop_g', 't_lop_or', 't_rop_e', 't_rop_ge', 't_lc', 't_rop_l', 't_semicolon',
't_aop_pl', 't_rp', 't_rb', 't_rop_le', 't_lop_and', 't_rop_ne', 't_aop_ml'},
    'DetailedParameters': {'t_rp'},
    'ParametersList2': {'t_rp'},
```

```
    'ParametersTail': {'t_rp'},
}
```

مرحله سومparsing table:

باید از الگوریتم سازی پیش بینی کننده غیر بازگشتی استفاده کنیم:

الگوریتم تجزیه کننده پیش‌بینی کننده

- الگوریتم ساخت جدول تجزیه کننده پیش‌بینی کننده به صورت زیر است. این الگوریتم گرامر G را دریافت و جدول تجزیه M را تولید می‌کند.
- برای هر یک از قوانین $A \to \alpha$ از گرامر به صورت زیر عمل می‌کنیم.

۱. به ازای هریک از ترمینال‌های a در $\text{First}(\alpha)$ قانون $A \to \alpha$ را به $M[A, a]$ اضافه می‌کنیم.

۲. اگر ϵ در $\text{First}(\alpha)$ باشد، آنگاه به ازای هریک از ترمینال‌های a در $\text{Follow}(A)$ قانون $A \to \alpha$ را به $M[A, a]$ اضافه می‌کنیم. اگر ϵ در $\text{First}(\alpha)$ باشد و \$ در $\text{Follow}(A)$ باشد آنگاه $A \to \alpha$ را به $M[A, \$]$ اضافه می‌کنیم.

- اگر پس از عملیات بالا هیچ قانونی در $M[A, a]$ قرار نگرفت، آنگاه در $M[A, a]$ مقدار خطا (error) قرار می‌دهیم. برای سادگی، خطاها را با خانه‌های خالی در جدول نمایش می‌دهیم.

تجزیه کننده پیش‌بینی کننده غیربازگشتی

- الگوریتم زیر عملیات تجزیه پیش‌بینی کننده را نشان می‌دهد.

let a be the first symbol of w;
let X be the top stack symbol;
while ($X \neq \$$) { /* stack is not empty */
    if ($X = a$) pop the stack and **let** a be the next symbol of w;
    else if (X is a terminal) *error*();
    else if ($M[X, a]$ is an error entry) *error*();
    else if ($M[X, a] = X \to Y_1 Y_2 \cdots Y_k$) {
        output the production $X \to Y_1 Y_2 \cdots Y_k$;
        pop the stack;
        push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with Y_1 on top;
    }
    let X be the top stack symbol;
}

نحوه کار LL(1):

- ابتدا یک جدول پیش‌بینی برای گرامر ساخته می‌شود. این جدول به پارسر کمک می‌کند تا با نگاه کردن به توکن فعلی و نماد غیرترمینال فعلی، تصمیم بگیرد که از کدام قاعده تولید استفاده کند.

•پشته:(Stack)

- پارسر از یک پشته استفاده می‌کند که ابتدا شامل نماد شروع (Start Symbol) گرامر و علامت پایان ورودی (معمولاً $ یا #) است.

•ورودی:(Input)

- ورودی پارسر یک رشته از توکن‌ها است که با علامت پایان (مثلاً $) خاتمه می‌یابد.

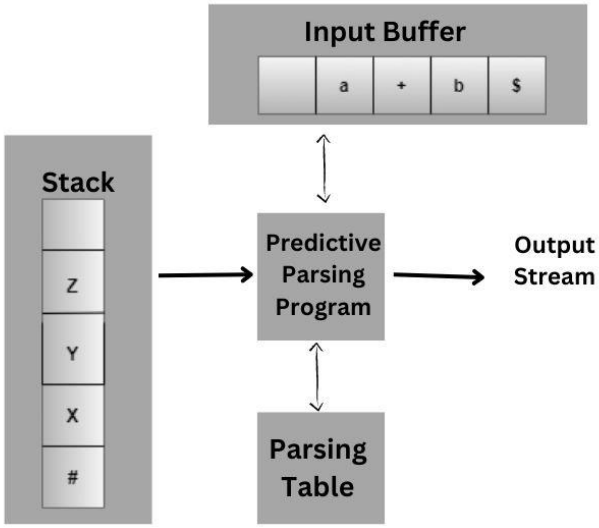
•عملکرد پارسر:

- پارسر عملیات خود را با خواندن اولین توکن از ورودی شروع می‌کند و این فرآیند را تا رسیدن به علامت پایان ورودی ادامه می‌دهد.
- در هر مرحله، پارسر نماد بالای پشته و توکن فعلی ورودی را مقایسه می‌کند و یکی از این حالات رخ می‌دهد:

۱. **حالت تطابق:(Match)**
   - اگر نماد بالای پشته با توکن فعلی ورودی یکسان باشد، پارسر هر دو را از پشته و ورودی حذف می‌کند و به توکن بعدی ورودی می‌رود.

۲. **حالت قاعده تولید:(Production Rule)**
   - اگر نماد بالای پشته یک غیرترمینال باشد، پارسر از جدول پیش‌بینی استفاده می‌کند تا قاعده تولید مربوطه را پیدا کند و غیرترمینال بالای پشته را با نمادهای سمت راست قاعده تولید جایگزین کند (به ترتیب معکوس).

۳. **حالت خطا:(Error)**
   - اگر هیچ قاعده تولیدی پیدا نشود و یا تطابقی وجود نداشته باشد، پارسر به خطا می‌رسد و فرآیند تجزیه متوقف می‌شود.



این مراحل را در کد پیاده سازی کردیم.

الگوریتم تجزیه LR

- در ابتدا تجزیه‌کننده s_0 را برروی پشته قرار می‌دهد و $w\$$ برروی بافر ورودی قرار می‌گیرد. سپس الگوریتم زیر اجرا می‌شود.

```
let a be the first symbol of w$;
while(1) {
        let s be the state on top of the stack;
        if ( ACTION[s, a] = shift t ) {
                push t onto the stack;
                let a be the next input symbol;
        } else if ( ACTION[s, a] = reduce A → β ) {
                pop |β| symbols off the stack;
                let state t now be on top of the stack;
                push GOTO[t, A] onto the stack;
                output the production A → β;
        } else if ( ACTION[s, a] = accept ) break;
        else call error-recovery routine;
}
```

مرحله چهارم ساخت درخت:

با استفاده از کتابخانه any tree درختمان را ساختیم.