



الجمهورية العربية السورية
جامعة دمشق
كلية الهندسة المعلوماتية

مشروع مترجمات للغة Angular



مشروع أعد للقسم العملي من مادة المترجمات للسنة الرابعة – قسم هندسة البرمجيات

إعداد الطلاب:

محمد رياض عمر

مجد جميل لوقا

مجد سامر عروس

بثينه محمود السبيريج

إشراف:

د. باسم قصيبة – م. هدى الحوامدة

What is ANTLR?

ANTLR (ANother Tool For Language Recognition) هو مولد تحليل يُستخدم لإنشاء مترجمات ومحلات للغات برمجة وتنسيقات ملفات مختلفة. يتيح ANTLR للمطورين تحديد قواعد لغوية (Grammars) باستخدام قواعد تحليلية مبسطة، ثم يُنشئ ANTLR محلاً (Parser) ومولداً للمحلل (Lexer) يستند على هذه القواعد. في سياق مترجمات اللغات البرمجية:

Lexer (محلل الأقسام): يقوم بتحويل سلسلة من الرموز (tokens) من المدخلات إلى وحدات أساسية، مثل الكلمات المفتاحية، المعاملات، والرموز الترقيمية.

Parser (محلل الجمل): يقوم بفحص الترتيب اللغوي للرموز المُحللة من قبل المحلل، ويبني بنية بيانات تُعبر عن تركيب البرنامج وعلاقات بين مكوناته.

يقوم ANTLR بتوليد كود في لغة البرمجة المستهدفة (مثل جافا) للمحلل والمولد على أساس القواعد التي يتم تحديدها. القواعد تُعبر عن قاعدة اللغة المراد تحليلها.

مميزات ANTLR تشمل:

سهولة الاستخدام: تعتبر قواعد ANTLR أن تكون أكثر قرباً إلى الصياغة اللغوية المألوفة، مما يجعل من السهل على المطورين تحديد قواعد لغة جديدة.

دعم للغات متعددة: يمكن استخدام ANTLR لتحليل لغات متنوعة، سواء كانت لغات برمجة أو تنسيقات ملفات أخرى.

تكامل مع أنظمة متعددة: يمكن توليد المحلات باستخدام ANTLR لتناسب مجموعة متنوعة من الأنظمة والمشاريع.

مجتمع نشط: لديها مجتمع نشط من المستخدمين والمطورين، ويتم تحديث الأداة بانتظام.



What is Angular?

Angular هي إطار عمل مفتوح المصدر يُستخدم لتطوير تطبيقات ويب حديثة وأحادية الصفحة (Single Page Applications - SPA). تم تطويره وصيانته بواسطة فريق Google ويعتمد على لغة TypeScript التي تعتبر امتدادًا للغة JavaScript.

Angular يُسهّل بناء تطبيقات ويب ديناميكية، حيث يوفر أدوات متقدمة مثل مكونات قابلة لإعادة الاستخدام، التوجيه (Routing)، وحقن التبعية (Dependency Injection). يتميز بقوته وأدائه العالي في التطبيقات الكبيرة والمعقدة، مما يجعله خيارًا شائعًا بين المطورين.



بعض المفاهيم في Angular:

Modules (الوحدات): 

هي أجزاء أساسية تنظم الكود في Angular، حيث يتم تجميع المكونات والخدمات والتوجيهات المتعلقة ببعضها في وحدات مستقلة.

Components (المكونات): 

كل تطبيق Angular يتكون من مكونات. المكون يتكون من ملف TypeScript يحتوي على منطق العمل، وملف HTML يمثل واجهة المستخدم، وملف CSS للتنسيق.

Templates (القوالب): 

تستخدم لتحديد واجهة المستخدم، حيث يتم الجمع بين HTML مع توجيهات Angular مثل `ngIf` و `ngFor` لإضافة السلوك التفاعلي.

Directives (التوجيهات):

أدوات تُستخدم لتوسيع سلوك عناصر DOM، مثل إنشاء عناصر ديناميكية أو تطبيق منطق مخصص.

Services (الخدمات):

تُستخدم لمشاركة المنطق أو البيانات بين المكونات، حيث يتم إنشاء الخدمات ليتم حقنها عند الحاجة.

Dependency Injection (حقن التبعية):

نمط تصميم يُستخدم في Angular لتزويد المكونات والخدمات بما تحتاجه من تبعيات بشكل تلقائي.

Routing (التوجيه):

نظام لإدارة التنقل بين صفحات التطبيق باستخدام عناوين URL.

Two-way Data Binding (ربط البيانات ثنائي الاتجاه):

يتيح التزامن الفوري بين واجهة المستخدم وبيانات النموذج.

Pipes (الأنابيب):

تُستخدم لتحويل البيانات وعرضها بتنسيقات معينة في القوالب (مثل تحويل النصوص إلى حروف كبيرة أو تنسيق التواريخ).

Lifecycle Hooks (خطافات دورة الحياة):

مجموعة من الأحداث التي يمكن أن يتفاعل معها المطور أثناء دورة حياة المكون، مثل `ngOnInit` و `ngOnDestroy`.

Observables (القابلات للملاحظة):

جزء من مكتبة RxJS تُستخدم للتعامل مع البيانات غير المتزامنة مثل طلبات HTTP أو الأحداث.

الهدف من المشروع:

بناء مترجم لتحويل قواعد Angular إلى كود صالح للتنفيذ على المتصفح وذلك عبر مراحل عديدة نكتفي بها للقسم العملي لهذا الفصل (التحليل اللفظي واللغوي Lexical Analysis – التحليل القواعدي Syntactical Analysis – بناء شجرة AST – Visitor Functionc – بناء جدول Symbol Table). يتم تحقيق ذلك من خلال فهم قواعد كتابة Angular، التي تُستخدم لبناء وتنظيم واجهات المستخدم بشكل متكامل وفعال.



التحليل اللفظي واللغوي Lexical Analysis:

هي المرحلة الأولى عندما يقوم المترجم بمسح الكود المصدري من اليسار لليمين، حرفاً بحرف، وتجميع هذه الأحرف في الرموز المميزة.

الوظائف الأساسية لهذه المرحلة هي:

- + تحديد الوحدة
- + ات المعجمية في الكود المصدري.
- + تصنيف الوحدات المعجمية إلى فئات مثل الثوابت والكلمات المحجوزة، وإدخالها في جداول مختلفة، مع العلم أنه سيتجاهل التعليقات في البرنامج المصدري.
- + تحديد الرمز الذي ليس جزءاً من اللغة.

إن أداة **Lexer** وظيفتها التعرف على تسلسل الأحرف بتمثيل هذا التسلسل برمز **Token**. حيث تعرف الرموز بحروف كبيرة ضمن ملف ال **Lexer**، وترتيب تعريف الرموز مهم جداً في هذه المرحلة ولا يمكن تعريف أكثر من **Keyword** بنفس الاسم بها.

في مشروعنا نهتم بأهم الأساسيات الخاصة بلغة **Angular** والتي يتم تعريفها في ملف **Lexer**:

١. تعريف الكلمات المحجوزة الخاصة بتعريف الثوابت والمتغيرات والاستيراد والتصدير:

٢. تعريف الكلمات المحجوزة بهياكل التحكم والحلقات **Loops and Control Flow**

٣. تعريف الكلمات المحجوزة الخاصة بالصفوف **Classes** والبرمجة الكائنية **OOP**:

٤. تعريف أوامر الطباعة:

٥. تعريف التوابع:

٦. تعريف العمليات بأنواعها:

+ عمليات المقارنة:

+ العمليات الرياضية:

+ العمليات المنطقية:

+ عمليات الإسناد والزيادة والنقصان:

٧. تحليل السلاسل النصية:

٨. تعريف المتحولات ضمن سلسلة نصية:

٩. التعامل مع الأعداد:

١٠. تعريف علامات الترقيم:

١١. تعريف الأقواس بمختلف أنواعها:

١٢. المساحات البيضاء والأسطر الجديدة والتعليقات:

```
1 DoubleQuotationMark : '"';
2 MultiLineComment    : '/*' .*? '*/' -> channel(HIDDEN);
3 SingleLineComment   : '//' ~[\r\n\u2028\u2029]* -> channel(HIDDEN);
4 SingleQuote         : '\'' ~[ESC<'\n\t]* '\'';
5 DoubleQuote         : '"' ~[ESC<'\n\t]* '"';
6 BackTickQuote       : '`' ~[ESC<'\n\t]* '`';
```

```
1 /// Keywords
2
3 Break      : 'break';
4 Do         : 'do';
5 Instanceof : 'instanceof';
6 Typeof     : 'typeof';
7 Case       : 'case';
8 Else       : 'else';
9 New        : 'new';
10 Var        : 'var';
11 Catch      : 'catch';
12 Finally    : 'finally';
13 Return     : 'return';
14 Void       : 'void';
15 Continue   : 'continue';
16 For        : 'for';
17 Switch     : 'switch';
18 While      : 'while';
19 Debugger   : 'debugger';
20 Function_   : 'function';
21 This       : 'this';
22 With       : 'with';
23 Default    : 'default';
24 If         : 'if';
25 Throw      : 'throw';
26 Delete     : 'delete';
27 In         : 'in';
28 Try        : 'try';
29 As         : 'as';
30 From       : 'from';
31 ReadOnly   : 'readonly';
32 Async      : 'async';
33 Await      : 'await';
34 Yield      : 'yield';
35 YieldStar  : 'yield*';
```

```
1 Class      : 'class';
2 Enum       : 'enum';
3 Extends    : 'extends';
4 Super      : 'super';
5 Const      : 'const';
6 Export     : 'export';
7 Import     : 'import';
8
9 /// The following tokens are also considered to be FutureReservedWords
10 /// when parsing strict mode
11
12 Implements : 'implements';
13 Let        : 'let';
14 Private    : 'private';
15 Public     : 'public';
16 Interface  : 'interface';
17 Package    : 'package';
18 Protected  : 'protected';
19 Static     : 'static';
20
21 //keywords:
22 Any        : 'any';
23 Number     : 'number';
24 Never      : 'never';
25 Boolean    : 'boolean';
26 String     : 'string';
27 Int        : 'int';
28 Unique     : 'unique';
29 Symbol     : 'symbol';
30 Undefined  : 'undefined';
31 Object     : 'object';
32
33 Of         : 'of';
34 KeyOf      : 'keyof';
35
36 TypeAlias  : 'type';
37
38 Constructor : 'constructor';
39 Namespace   : 'namespace';
40 Require     : 'require';
41 Module      : 'module';
42 Declare     : 'declare';
43 Abstract    : 'abstract';
44 Is          : 'is';
45 At         : '@';
```

```

1 //TYPE SCRIPT
2 DoubleQuotationMark : '"';
3 MultilineComment : '/*' .*? '*/' -> channel(HIDDEN);
4 SingleLineComment : '//' ~[\r\n\u2028\u2029]* -> channel(HIDDEN);
5 SingleQuote : '\'' ~[ESC<'\n\t]* '\'';
6 DoubleQuote : '"' ~[ESC<'\n\t]* '"';
7 BackTickQuote : '`' ~[ESC<'\n\t]* '`';
8 OpenBracket : '[';
9 CloseBracket : ']';
10 OpenParen : '(';
11 CloseParen : ')';
12 OpenBrace : '{';
13 CloseBrace : '}';
14 SemiColon : ';';
15 Comma : ',';
16 Assign : '=';
17 QuestionMark : '?';
18 QuestionMarkDot : '?.';
19 Colon : ':';
20 Ellipsis : '...';
21 Dot : '.';
22 PlusPlus : '++';
23 MinusMinus : '--';
24 Plus : '+';
25 Minus : '-';
26 BitNot : '~';
27 Not : '!';
28 Multiply : '*';
29 Divide : '/';
30 Modulus : '%';
31 Power : '**';
32 NullCoalesce : '??';
33 Hashtag : '#';
34 LeftShiftArithmetic : '<<';
35 LessThan : '<';
36 MoreThan : '>' -> pushMode(HTML_TEXT_MODE);
37 LessThanEquals : '<=';
38 GreaterThanEquals : '>=';
39 Equals_ : '=';
40 NotEquals : '!=';
41 IdentityEquals : '===';
42 IdentityNotEquals : '!==';
43 BitAnd : '&';
44 BitXor : '^';
45 BitOr : '|';
46 And : '&&';
47 Or : '||';
48 MultiplyAssign : '*=';
49 DivideAssign : '/=';
50 ModulusAssign : '%=';
51 PlusAssign : '+=';
52 MinusAssign : '-=';
53 LeftShiftArithmeticAssign : '<<=';
54 RightShiftArithmeticAssign : '>>=';
55 RightShiftLogicalAssign : '>>>=';
56 BitAndAssign : '&=';
57 BitXorAssign : '^=';
58 BitOrAssign : '|=';
59 PowerAssign : '**=';
60 NullishCoalescingAssign : '??=';
61 ARROW : '=>';

```

```

1 //ANGULARSPECIFIC
2 Component: 'Component';
3 NgFor: 'NgFor' | 'ngFor' | 'ngfor';
4 NgIf: 'NgIf' | 'ngIf' | 'ngif';
5 Selector: 'selector';
6 templateUrl: 'templateUrl';
7 Template: 'template';
8 styleUrls: 'styleUrls';
9 styleUrls: 'styleUrl';
10 Styles: 'styles';
11 Imports: 'imports';
12 Standalone: 'standalone';
13 Get : 'get';

```


التحليل القواعدي Syntactical Analysis:

هو كل شيء عن اكتشاف البنية في الكود ويحدد ما إذا كان النص يتبع التنسيق المتوقع أم لا. الهدف الرئيسي من هذه المرحلة هو التأكد من شيفرة المصدر المكتوبة من قبل المبرمج صحيحة أم لا، حيث يعتمد التحليل النحوي على القواعد المستندة إلى لغة البرمجة المحددة عن طريق بناء شجرة القواعد Parse Tre بمساعدة الرموز Tokens. كما تحدد بنية لغة المصدر وقواعد اللغة أو تركيبها. المهام التي يتم تنفيذها في هذه المرحلة :

+ الحصول على الرموز المميزة Tokens من ملف ال Lexer.

+ التحقق مما إذا كان التعبير صحيحاً من الناحية التركيبية أم لا.

+ الإبلاغ عن جميع الأخطاء النحوية.

+ إنشاء البنية الهرمية أو شجرة القواعد Parse Tree.

على عكس ملف ال lexer فإن ملف ال Parser لا يهتم بالترتيب بالنسبة للقواعد فهو نعرف في ملف ال Parser القواعد المستندة إلى ملف ال Lexer:

❖ القاعدة الأساسية في البارسر:

```
prog: expression+ EOF # Program
;
expression:
    importStatement # Import
    | interface # InterfaceDecl
    | componentDeclaration # Component
    | classStructure # Class
    | genericStatement # Generic
    ;
```

❖ تعريف جملة import وتعريف interface بالإضافة إلى تسمية المتحول ونوعه:

```
ImportStatement: Import (OpenBrace | OpenBraceHTML) (Component | NgFor | NgIf | ID) (CloseBrace | CloseBraceHTML) From SingleQuote SemiColon ;  
  
interface: Interface ID (OpenBrace | OpenBraceHTML) (variableNaming SemiColon)* (CloseBrace | CloseBraceHTML);  
  
variableNaming: (Let | Var | Const)? ID (Colon varType (BitOr NullLiteral (Assign NullLiteral)?)?)? )? ;  
  
varType: String  
        | Int  
        | Boolean  
        | Void  
        | Any  
        | ID
```

❖ تعريف ال component:

```
componentDeclaration : At Component OpenParen (OpenBrace | OpenBraceHTML) componentInfo (Comma componentInfo)* (CloseBrace | CloseBraceHTML) CloseParen ;  
  
componentInfo: Selector Colon (SingleQuote | BackTickQuote) # Select  
              | templateUrl Colon (SingleQuote | BackTickQuote) # TempUrl  
              | styleUrls Colon OpenBracket (SingleQuote | BackTickQuote) CloseBracket # Styles  
              | standalone Colon BooleanLiteral # StandaloneStatus  
              | imports Colon OpenBracket ((NgFor | NgIf | ID) (Comma (NgFor | NgIf | ID))*?)? CloseBracket # Importss  
              ;
```

❖ تعريف المتحول والمصفوفة والتابع:

```
variableDeclaration: variableNaming (Assign value (BitOr NullLiteral (Assign NullLiteral)?)?)? (SemiColon)?;  
  
arrayDeclaration: variableNaming (OpenBracket CloseBracket)? (Assign arrayInfo*)?(SemiColon)?;  
arrayInfo: OpenBracket value (Comma value)* (Comma)? CloseBracket;  
  
functionDeclaration: ID functionBody;  
functionBody: OpenParen (variableNaming(Comma variableNaming))* CloseParen ARROW? (OpenBrace | OpenBraceHTML) genericStatement* (CloseBrace | CloseBraceHTML);  
functionCall: ID OpenParen(value (Comma value))* CloseParen;
```

```
genericStatement:  
    value # ValueType  
    | variableDeclaration # VariableDecl  
    | arrayDeclaration # ArrayDecl  
    | assignStatement # Assign  
    | returnStatement # Return  
    | ifStatement # If  
    | forLoop # For
```

```
assignStatement: (thisorId)? ID Assign value SemiColon;  
thisorId: ((ID|This) Dot);  
returnStatement: Return (thisorId)?value SemiColon;  
  
ifStatement: If OpenParen conditionalState CloseParen (OpenBrace|OpenBraceHTML) genericStatement* (CloseBrace|CloseBraceHTML) (Else genericStatement)*  
    ;  
  
conditionalState: LogicalStatement(logicalOp LogicalStatement)* # ConditionalStatement  
    | ID # VariableName  
    ;  
  
LogicalOp: LessThanEquals  
    | GreaterThanEquals  
    | Equals_  
    | NotEquals  
    | IdentityEquals  
    | IdentityNotEquals  
    | BitAnd  
    | BitXOr  
    | BitOr  
    | And  
    | Or  
    ;  
  
LogicalStatement: (value (logicalOp)value);
```

```
value:
  arrayInfo # ArrayInfoValue
  | functionDeclaration # Function
  | functionCall # FunctionSummoning
  | functionBody # FunctionStatement
  | jsonObject # JsonObjectValue
  | OpenParen value CloseParen # EventValue
  | value Dot value # ValueDotValue
  | value QuestionMarkDot value # NullableDotValue
  | value QuestionMark # NullableValue
  | value Or value # ValueOrValue
  | htmlTags # HtmlTagValue
  | string # StringValue
  | DecimalLiteral # DecimalNumberValue
  | ID # VariableValue
  | NullLiteral # NullValue
;
```

❖:jsonObject

```
jsonObject:(OpenBrace|OpenBraceHTML) ID Colon value (Comma ID Colon value)* (CloseBrace|CloseBraceHTML);
```

❖قواعد الHTML:

```
attribute: ngForStatement # NgFor
| ngIfStatement # NgIf
| (ID|Class) Assign DoubleQuote # DoubleQuotedAttribute
| OpenBracket (ID|Class) CloseBracket Assign DoubleQuote # OpenBracketAttribute // [src]
| OpenParen (ID|Class) CloseParen Assign DoubleQuote # OpenParenAttribute // (click)
;

ngForStatement: Multiply NgFor Assign (Let ID Of ID)
;

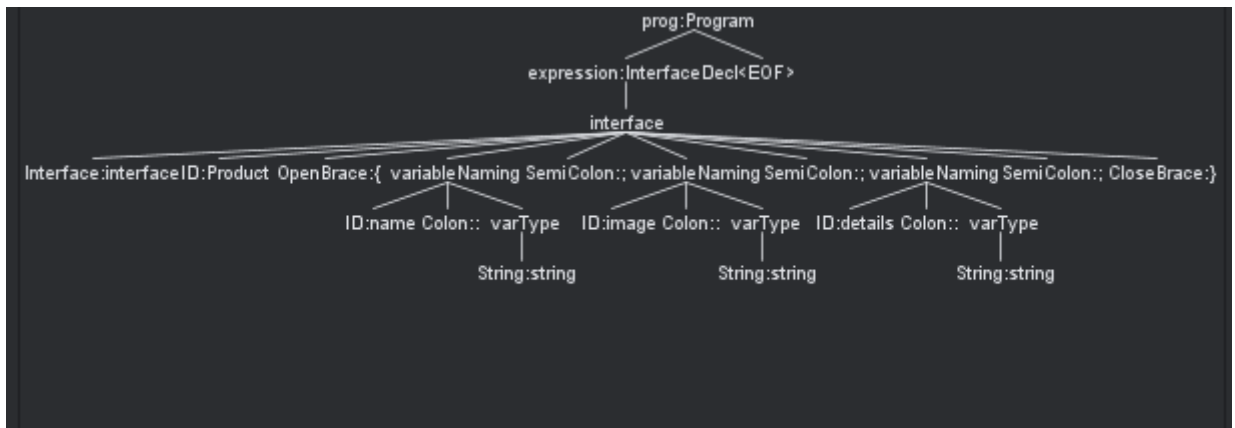
ngIfStatement: Multiply NgIf Assign (ID | logicalStatement)
;

htmlTags: openTag (htmlTags)* closeTag # PairedTag
| selfClosingTag # UnpairedTag
| TEXT # NormalHtmlText
| interpolation # HtmlInterpolation
;

interpolation: (OpenBrace|OpenBraceHTML)(OpenBrace|OpenBraceHTML) (value)* (CloseBrace|CloseBraceHTML)(CloseBrace|CloseBraceHTML);
openTag: (LessThan |OpenTag) ID (attribute)* (MoreThan|CloseTag);
closeTag: (LessThan |OpenTag) Divide ID (MoreThan|CloseTag);
selfClosingTag:(LessThan |OpenTag) ID (attribute)* Divide (MoreThan|CloseTag);
```

- عند كتابة عبارة من الكود في Parse Tree في IntelliJ تتولد الشجرة:

```
1 interface Product {
2   · name: string;
3   · image: string;
4   · details: string;
5 }
6
```



توليد شجرة ال AST:

- وتعرف ب اسم شجرة البناء المجردة **Abstract Syntax Tree**: هي بنية تستخدم للتمثيل جملة لغة البرمجة الذي يستخدم الحاقاً بشكل هرمي، بحيث يتم استدام هذه البنية لإنشاء Symbol Table في عملية التحقق الدلالي Semantic بالإضافة إلى توليد الكود Code Generation. تمثل AST جميع العناصر النحوية للغة البرمجة على غرار Parse Tree إلا أنها تركز على القواعد بدلاً عن العناصر مثل الأقواس والفواصل والفواصل المنقوطة.
- تحتوي شجرة ال AST على قواعد اللغة، وقواعد بالإضافة للرموز الرئيسية فقط (التي تمثل قيمة عددية أو تعطي معنى منطقي).
- بما أن ال AST تحتوي على قواعد اللغة ، وقواعد اللغة تحتوي على رموز tokens بالإضافة إلى قواعد أخرى فأفضل طريقة لتمثيل قاعدة هي باعتبارها كائن Object واعتبار الرمز token سلسلة String، واعتبار السلسلة من الكائنات list f Object وهكذا.

```

PairedTag {
  openTag:
  OpenTag{
    tagName: div,
    attributes: [
      QuotedAttribute{
        attributeName: style,
        attributeValue: "display: flex;"
      }
    ],
    innerTags: [
      PairedTag {
        openTag:
        OpenTag{
          tagName: div,
          attributes: [
            QuotedAttribute{
              attributeName: style,
              attributeValue: "margin-right: 20px;"
            }
          ],
          innerTags: [
            NormalHtmlText {
              text:

```

codesnap.dev

```

Class {
  GenericStatementList[
    VariableDecl {
      varName: VariableNaming {
        VariableName: selectedProduct
      }, VariableType: any
    }
    , varValue: null
  ],
  VariableDecl {
    varName:
    VariableNaming {
      VariableName: images
    }
    , varValue:
    ArrayInfoValue {
      arrayValues: [
        JsonObjectValue{
          {src:StringValue: 'assets/Image/1.jpg', name:StringValue: 'Product 1', details:StringValue: 'Details of Product 1.'}
        }
      ]
    }
  },

```

ASTVisitor:

- ✚ ASTVisitor هو كائن يستخدم في تحليل شجرة جملة (AST) المولدة من قبل المحلل (Parser) في ASTVisitor. ANTLR يُستخدم للتنقل عبر الشجرة وتنفيذ إجراءات مخصصة على العقد (الأفرع) المختلفة في الشجرة. يتم توليد الزوار (Visitor) بواسطة ANTLR بناءً على قواعد الزوار التي تحددها في ملف القواعد (Grammar file).
- ✚ ASTVisitor يحتوي على طرق مخصصة (override methods) لزيارة كل نوع من أنواع العقد (القواعد) في الشجرة، ويمكن للمستخدم تخصيص هذه الطرق لتنفيذ السلوك المرغوب.
- ✚ على سبيل المثال، يمكن أن يكون لدينا زائر لتوليد رمز المصدر من AST، حيث يُطلب منه زيارة كل نوع من العقد وتوليد الكود المعادل له. كما يمكن أن يتم استخدام زائر آخر لجمع معلومات أو إجراء تحليل آخر.
- ✚ في سياق ANTLR، يُستخدم ال (#) Labels لتعيين أو تسمية القواعد والبديهيات في القواعد.
- تُستخدم هذه العلامات لتعيين أسماء فريدة لكل قاعدة أو بديهية، وهذا يُسهل في تحديد نوع العقد (AST Node) الذي يتم توليده عند مطابقة القاعدة.
- ✚ على سبيل المثال، إذ كان هناك زائر لتوليد رمز المصدر، يمكن أن يستخدم الزائر العلامات (#) للتحقق من نوع العقد التالي وتوليد الكود المصدر المناسب وفقاً لذلك.
- يسهل هذا الاستخدام تقديم أكواد مولدة متنوعة وقابلة للتكوين لاستخدامات مختلفة.

❖ **Visitor** للبرنامج يزور كل جملة في Program ويضعها في قائمة ليتم طباعتها باستخدام :to string

```
public class AntlrToProgram extends AngularParserBaseVisitor<Program> { 3 usages

    public SymbolTable symbolTable = new SymbolTable(); 3 usages
    public void setSymbolTable(SymbolTable symbolTable) { this.symbolTable = symbolTable; }

    @Override 1 usage
    public Program visitProgram(AngularParser.ProgramContext ctx) {
        Program program = new Program();
        AntlrToExpression exprVisitor = new AntlrToExpression();
        for(int i = 0; i < ctx.getChildCount(); i++){
            if(i == ctx.getChildCount() - 1){
                // the last child is EOF, we don't visit it
            }else{
                program.addExpression(exprVisitor.visitExpression(ctx.expression(i)));
            }
        }
        this.symbolTable = exprVisitor.symbolTable;
        return program;
    }
}
```

❖ مثال من ال **Visitor** الذي يقوم بزيارة جملة من import:

```
@Override 2 usages
public Import visitImport(AngularParser.ImportContext ctx) {
    return this.visitImportStatement(ctx.importStatement());
}

@Override 2 usages
public Import visitImportStatement(AngularParser.ImportStatementContext ctx) {
    Import imp = new Import(ctx.getChild(1).getText());
    Row row = new Row();
    row.type = "Imported";
    row.value = imp.type;
    this.symbolTable.addRow(row);
    return imp;
}
```

Symbol Tabl:

هو بنية معطيات Structure Data لمعرفة دلالة المتغيرات، كل رمز معناه أو القيمة التي يحملها يستخدم في عملية التحقق الدلالي Semantic Check وتوليد الكود Code Generation .

✚ إن كل عنصر مخزن في جدول الرموز له أهمية في عملية توليد الكود والتحقق المنطقي.
✚ ما يتم تخزينه في جدول الرموز: المتغيرات والثوابت – التوابع والإجرائيات – الثوابت والسلاسل – التسميات بلغة المصدر.

✚ يمكن أن نبنيه بالطريقة التي نراها مناسبة والتي قد تساعدنا لاحقاً بعملية توليد الكود والتحقق المنطقي.

قد يكون: Binary Search – Hash Map – Hash Table – Linked List – List Object

يحتوي جدول الرموز على سجل لكل معرف مع حقول لسمات المعرف.

هذا المكون يسهل على المترجم البحث في سجل المعرف واسترجاعه بسرعة.

يتفاعل جدول الرموز ومعالج الأخطاء مع جميع المراحل وتحديث جدول الرموز في المقابل.

يحتوي ال **Symbol Table** قائمة من الصفوف وتابع الطباعة:

```
public class SymbolTable { 26 usages
    public List<Row> rowList=new ArrayList<>(); 5 usages
    public void addRow(Row row){ this.rowList.add(row); }
    public void printTable(){ 1 usage
        System.out.println("Type\t\t\t\t\t\t\t\t\t\tValue");
        System.out.println("-----");
        for(int i=0;i<rowList.size();i++){
            if(rowList.get(i)!=null){
                String type=rowList.get(i).type;
                String value=rowList.get(i).value;
                type=String.format("%-20s",type);
                value=String.format("%-20s",value);
                System.out.println(type + "\t\t\t\t\t" + "|" + "\t\t\t\t\t" + value);
                System.out.println("-----");
                System.out.println();
            }
        }
    }
}
```

مثال Symbol Table لتعريف متحولات وجمل Import :

Type	Value
Imported	Component
Imported	CommonModule
Imported	NgFor
Imported	NgIf
VariableName	selectedProduct
VariableType	any
VariableName	images

يمثل صف في ال symbol Table:

```
package Classes.SymbolTable;

public class Row { 18 usages
    public String type;
    public String value;
}
```

:Main ال كلاس

```
public class Main {
    public static void main(String[] args) {
        {

            AngularParser parser = getParser("tests\\anotherWayForTheSameInterface.txt");

            ParseTree antlrAST = parser.prog();

            AntlrToProgram progVisitor = new AntlrToProgram();
            Program prog = progVisitor.visit(antlrAST);
            ExpressionProcessor ep = new ExpressionProcessor(prog.expressionList);
            for(String evaluation : ep.getEvaluationResults()){
                System.out.println(evaluation);
            } // for printing AST
            progVisitor.symbolTable.printTable(); //For printing symbol table
        }
    }

    private static AngularParser getParser(String fileName){
        AngularParser parser = null;

        try {
            CharStream input = CharStreams.fromFileName(fileName);
            AngularLexer lexer = new AngularLexer(input);
            CommonTokenStream tokens = new CommonTokenStream(lexer);
            parser = new AngularParser(tokens);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        return parser;
    }
}
```

