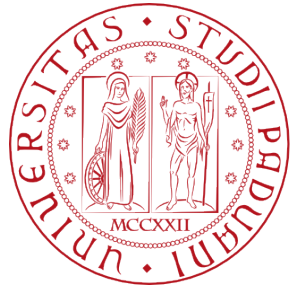
Automi e Linguaggi Formali

Terza Esercitazione di Laboratorio

LT in Informatica
a.a. 2020/2021



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

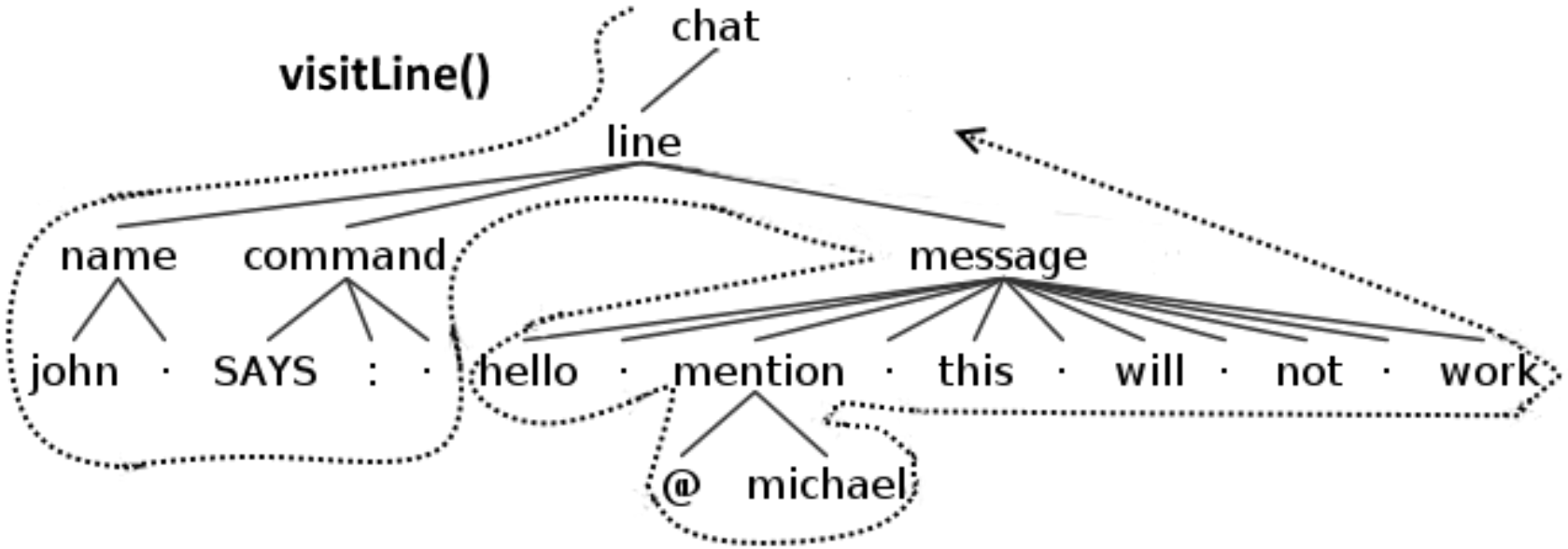
Questo strumento può produrre in automatico il codice relativo a:

- Lexer: parte lessicale dove si definiscono i token e i separatori da ignorare
- Parser: parte grammaticale contenente le regole di produzione
- Visitor: parte semantica che definisce le azioni da eseguire sul testo

Visitor

- Un Visitor consente di visitare l'albero sintattico eseguendo delle azioni per ognuno dei nodi visitati
- È flessibile: ci permette di controllare quali nodi visitare, quante volte visitarli, ed in che ordine.
- Le azioni da eseguire dipendono dalle regole della grammatica
- Le azioni possono ritornare un valore di qualsiasi tipo

BaseVisitor: implementazione di default che visita in profondità l'albero sintattico



Definisce un metodo **visit*Regola*** per ogni regola della grammatica

Visitor: schema di implementazione

- Si parte dal BaseVisitor generato da ANTLR
- Si ridefiniscono alcuni dei suoi metodi per cambiare il comportamento della visita
- I metodi ridefiniti possono:
  - eseguire delle azioni (esempio: lettura da tastiera)
  - ritornare un valore al chiamante (esempio: valore di una espressione)
  - scegliere su quali nodi figli continuare la visita e quali no (esempio: if-then-else)

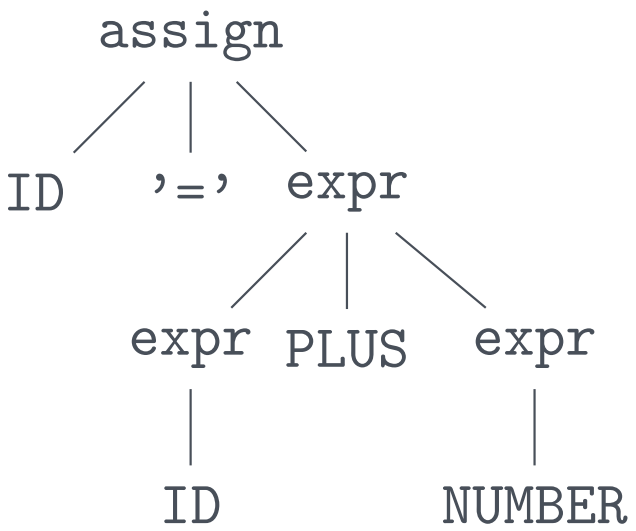
Esempio: assegnamento

Regola:

```
assign : ID '=' expr ;
```

Esempio:

```
x = y + 3
```



- Implementare l'assegnamento significa ridefinire il metodo `visitAssign` del `pascalBaseVisitor`
- Il metodo ottiene il nome della variabile da assegnare visitando il figlio `ID`
- Il valore dell'espressione si ottiene visitando il figlio `expr`
- Il nuovo valore della variabile viene memorizzato in una mappa

Codice di `visitAssign`


```
antlrcpp::Any runtimeVisitor::visitAssign(pascalParser::AssignContext *ctx) {
    string varname = ctx->ID()->getText();
    // controllo che la variabile sia stata dichiarata
    if(this->vars.find(varname) == this->vars.end()) {
        cerr << "Error: Undefined variable '" << varname << "'" << endl;
        exit(EXIT_FAILURE);
    }
    int value = visitExpr(ctx->expr());
    this->vars[varname] = value;
    return NULL;
}
```

- `antlrcpp::Any` permette di ritornare qualsiasi tipo di valore
- il parametro `pascalParser::AssignContext *ctx` è il contesto della regola:
  - contiene i puntatori ai figli `ctx->ID()` e `ctx->expr()`
- `this->vars`: mappa `string` $\rightarrow$ `int` con i valori delle variabili

Valutazione delle espressioni

Regola:

```
expr        : expr MOD expr | expr DIV expr | expr MULT expr
            | expr MINUS expr | expr PLUS expr |
            | '(' expr ')' | NUMBER | ID ;
```

- Implementare l'assegnamento significa ridefinire il metodo `visitExpr` del `pascalBaseVisitor`
- Il metodo ritorna un intero con il valore calcolato dell'espressione
- Si procede per casi
- Se necessario, si chiama ricorsivamente `visitExpr` per calcolare il valore delle sottoespressioni