**Tutorial 3**

Syntax Analysis 1

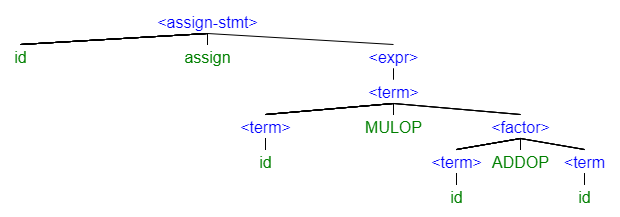
*Name: Huỳnh Nhất Long*

*ID: 1852522*

**Question 1:**

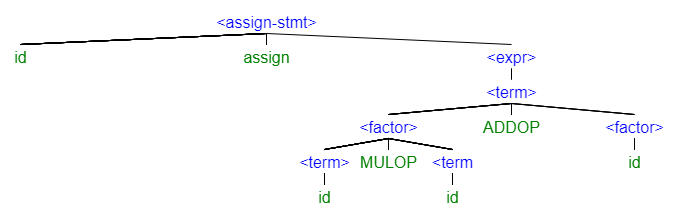
* assign → id "=" expr
* id → "A" | "B" | "C"
* expr → expr "+" term | term
* term → term "\*" factor | factor
* factor → "(" expr ")" | id

1. A = A\*(B+C)



Assign => id “=” expr => A = expr => A = term => A = term \* factor => A = factor \* factor => A = id \* factor => A = A \* factor => A = A \* (expr) => A = A \* (expr + term) => A = A \* (term + term) => A = A \*(factor +term) => A = A \* (id + term) => A = A \* (B + term) => A = A \* (B + factor) => A = A \*(B + id) => A = A \* (B + C).

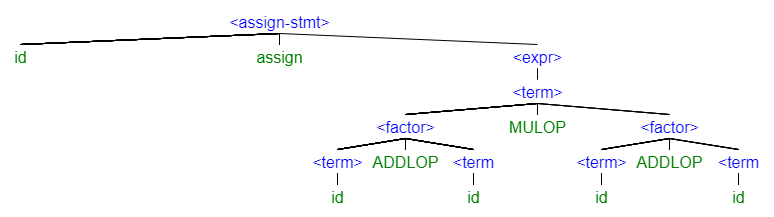
1. A = A \* B + C



Assign => id = expr => A = expr => A = expr + term => A = term + term => A = term \* factor + term => A = factor \* factor + term => A = id \* factor + term => A = A\*factor + term => A = A \* id + term => A = A \* B + term => A = A \* B + factor => A = A \* B + id => A = A \* B + C.

1. A = (A + C) \* (A + B)

Assign => id = expr => A = expr => A = term => A = term \* factor => A = factor \* factor => A = (expr) \* factor => A = (expr + term) \* factor => A = (term + term) \* factor => A = (factor + term) \* factor => A = (id + term) \* factor => A = (A + term) \* factor => A = (A +factor) \* factor => A = (A + id) \* factor => A = (A + C) \* factor => A = (A + C) \* (expr) => A = (A + C) \* (expr + term) => A = (A + C) \* (term + term) => A = (A + C) \* (factor + term) => A = (A +C) \* (id + term) => A = (A + C) \* (A + term) => A = (A + C) \* (A + factor) => A = (A +C) \* (A + id) => A = (A +C) \* (A + B).



**Question 2:**

main : stmt EOF

;

expr:

<assoc=right> ‘!’expr // 1, highest

| expr “==” expr // 2

|expr “!=” expr

|expr “<” expr // 3

|expr “<=” expr

|expr “>” expr

|expr “>=” expr

|expr “||” expr // 4 , left-assoc

|expr “&&” expr // 5, left-assoc

| INT

|ID

;

stmt : other

IF expr THEN stmt (ELSE stmt)?

;

**Question 3:**

s → a (Ba)\*

a → A (B)? a

Convert:

s → s B a

| a

a → a B a

| A

**Question 4:**

a)

s → a (1)

a → a + a | id (2)

id → A | B | C (3)

We can see in (2) that the non-ternimal element a exist in the left and the right side of the operator “+” or ADDOP, that makes the grammar ambiguous.

b) Left Recursion elimination: from (2) we change based on the formula:

A 🡪 beta A’

A’ 🡪 ϵ | alpha A’

Where A 🡪 A alpha | beta, thus:

a → id a’

a’ → ϵ | + a a’

**Question 5:**

stmt → IF expr THEN { stmt } ELSE { stmt }

| IF expr THEN { stmt }

| other

expr → TRUE | FALSE

Left factoring:

stmt 🡪 IF expr THEN stmt’

| other

stmt’ 🡪 ELSE stmt | ϵ

expr 🡪 TRUE | FALSE

**Question 6:**

**\***Question 4:

s 🡪 id a {a’}

a’ 🡪 {a}?

id 🡪 A | B | C

\*Question 5: