# CS 419 Compiler Projects Form

1. **Instructions to be Followed (for the Hard Copy):**
   1. Fill page **NO 4** with the required Fields:
      1. Project Idea: The **Idea** will be assigned to you .
      2. Team Members NO#: Number of team members **7**
      3. Table:
         * ID: Your FCIH ID.
         * Name: Your **Full Name** as registered on College’s Database.
         * Level & Department: Your **(Current)** level and department.
         * Section(Day-from-to): Your Section Day and time slot.
         * Role: Your role in project (**Team leader** OR **Member**).
         * Fill Page **NO 5 & 6** with your (**Regular Expressions**, **Finite automata**, **Parse trees** and **abstract syntax tree**) respectively.
2. **Minus Policies:**
   1. **Project Policy:** affects all projects members including team leader.
   2. **Member Policy:** affects a member of project’s members.
3. **General Notes:**
   1. Total grade of Project is 15
   2. **Deadline** to register yourself and your team on online form **Tuesday 05/04/2022 at 11:59 PM** after that **-2 Project Policy** will be applied**.**
   3. Once you Registered, **NO modifications** will be done.
   4. Allowed only on registration for team in form, duplication will got **-2 Project Policy.**
   5. Each group will be assigned **an Idea, ID and time slot** for **discussion**.
   6. Each team member and team leader in a team **must work in project’s coding phase** (including implementation of **finite automata and parse trees**).
4. **Discussion Notes:**
   1. **Copied Code** will be got ZERO Without Discussion.
   2. By references to Section 3 (General notes) Point V , **-5 Member policy** will be applied to each team member (including team leader) who does not participate in project coding phase **as well as the team leader who does not report this case**.
   3. Each team member must have **a complete knowledge** about the whole project
   4. **Evaluation** will be **Individual Evaluation** not project Evaluation.
   5. **-2 Project Policy** will be applied in case of being late for assigned discussion time slot
   6. **NO discussion will be repeated under any circumstances.**
   7. At Discussion day, in case of offline discussions, each team must have **Hard Copy form** including (**Finite automata and parse trees of team’s project**).
   8. Discussion Day will be **sent later** .

## Notes about Implementation:

* 1. **.Net or PHP** are only allowed.
  2. The project must be a Web (use latest technologies).
  3. Your code must be uploaded to github before discussion.
  4. **- 5 Project policy** will be applied in case of using **Built-in Method** within implementation of the scanner or parser, you must create your **own methods to match** for ex (your regular Expressions).
  5. Each Project must contain a full functional editor (comment, uncomment, put red line under wrong words, auto complete, navigation to function or class, line NO).
  6. Each Project must contain **two buttons** , one button called “**Scan**” to run scanner and other called “**Parse**” to run parser –parser must take output of scanner to do it’s task.
  7. Each project must contain a button named “**Browse**” that allows us to choose a **file from a disk** that allows us to parse or scan this file **Without Showing what is inside the file** and shows the output.
  8. – 3 Project policy will be applied if the content of the file that is mentioned in point V is opened or viewed.

## Notes about Discussion Testing:

* + There will be two types of Testing :

1. **White Box Testing:** This will be from **Editor**.
2. **Black Box Testing:** This will be from “**Browse**” Button

**Thanks,**

# CS 419 Compiler Project Form

**Project Idea:**

Project 3……………………………………………………………….

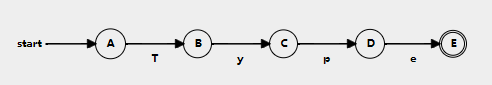
**Team Members NO#: 7** …………………………………..

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Name** | **Level& Department** | **Section(Day- from-to)** | **Role (Lead/Member)** | **Grade** |
| 201900894 | ندي صبري محمد موسي | level 3  CS | الاربعاء من 2 ل 4 | **Leader** |  |
| 201900749 | محمد وائل عبد الرحمن عبد الباسط محمد | level 3  CS | الاربعاء من 4 ل 6 | **Member** |  |
| 201900742 | محمد هاني سيد مهدي | level 3  CS | الاربعاء من 4 ل 6 | **Member** |  |
| 201900204 | ايمان اشرف اسماعيل محمد | level 3  CS | الاربعاء من 12 ل2 | **Member** |  |
| 201900745 | مروان نبيل صبحي الديب | level 3  CS | الأربعاء من 2 ل 4 | **Member** |  |
| 201900600 | ماريان موريس جرجس | level 3  CS | الاربعاء من 4 ل 6 | **Member** |  |
| 201900149 | اسماء محمد عابدين توفيق | level 3  CS | الاربعاء من 10 ل 12 | **Member** |  |

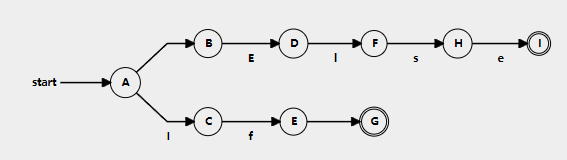
## Regular Expression, Finite automata and Conversion from RegX to NFA, NFA to DFA

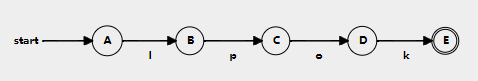
Regular Expression

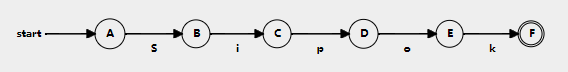
1. Class 🡪 Type
2. Inheritance 🡪 Infer
3. Condition 🡪 If |Else
4. Integer 🡪 Ipok
5. Sinteger 🡪 Sipok
6. Character 🡪 Craf
7. String 🡪 Sequence
8. Float 🡪 Ipokf
9. Sfloat 🡪 Sipokf
10. Void 🡪 Valueless
11. Boolean 🡪 Rational
12. Break 🡪 Endthis
13. Loop 🡪 However | When
14. Return 🡪 Respondwith
15. Struct 🡪 Srap
16. Switch 🡪 Scan | Conditionof
17. Start Symbol 🡪 @| ^
18. End Symbol 🡪 # | $
19. Arithmetic Operation 🡪 + | - | \* | /
20. Logic operators 🡪 && | || | ~
21. relational operators 🡪 == | < | > | != | <= |>=
22. Assignment operator 🡪 =
23. Access Operator 🡪 ->
24. Braces 🡪 { | } | [ | ]
25. Constant 🡪 [0-9]
26. Quotation Mark 🡪 ‘ | , | “
27. Inclusion 🡪 Require
28. Letter 🡪 [A-Z] | [a-z] | \_
29. Identifier 🡪 letter(letter|constant)\*

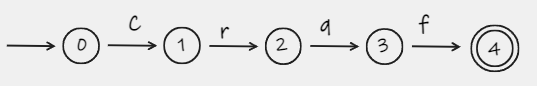


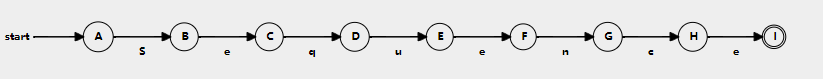












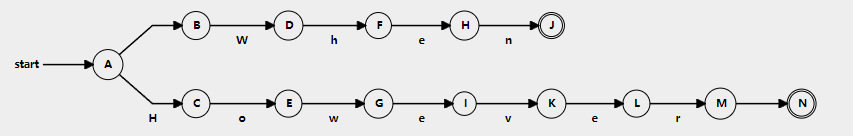


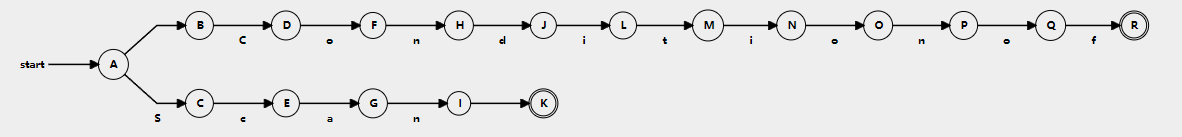


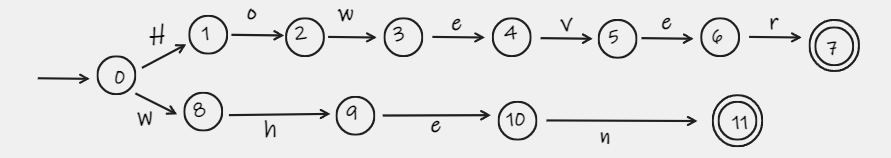


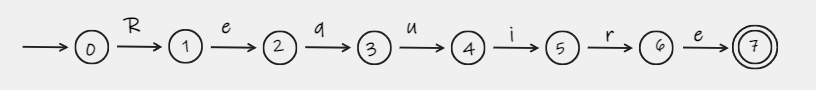


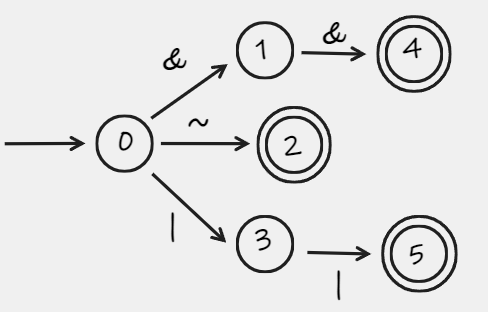


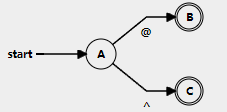


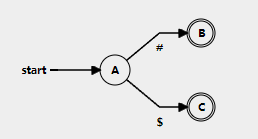


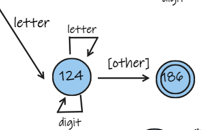


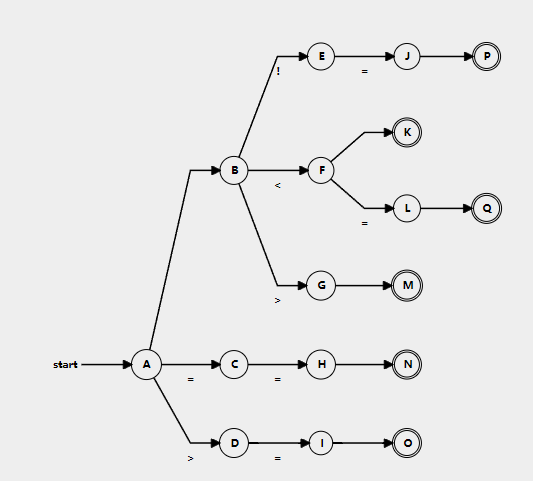


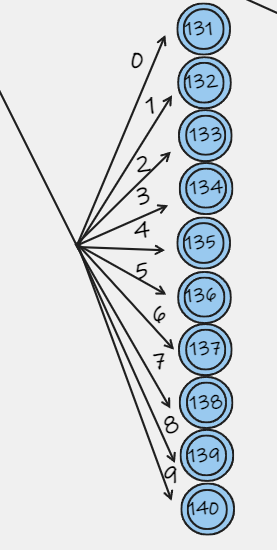




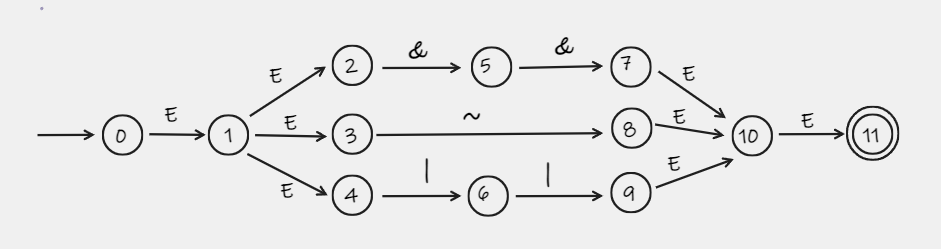


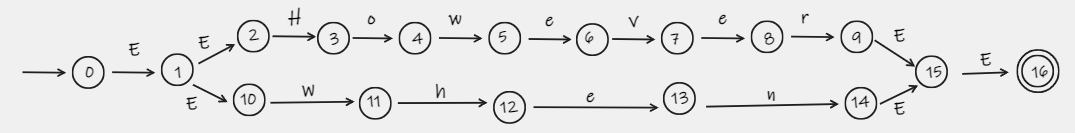


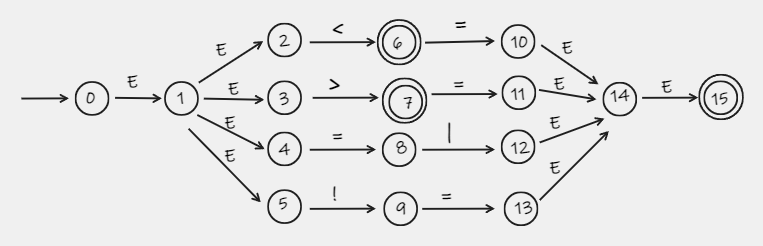


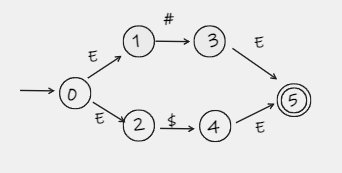


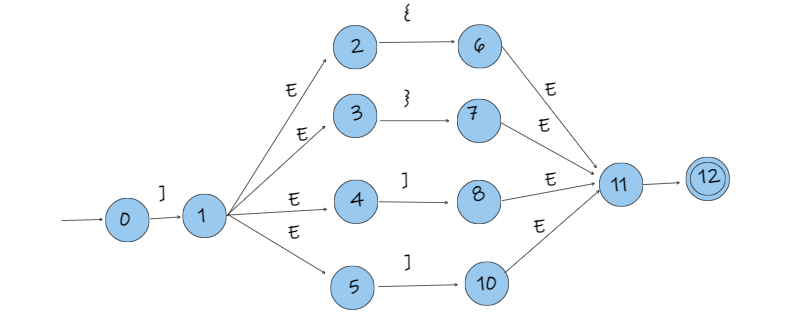
ε**-NFA**

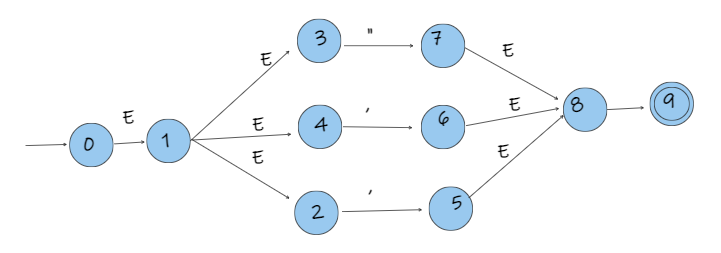


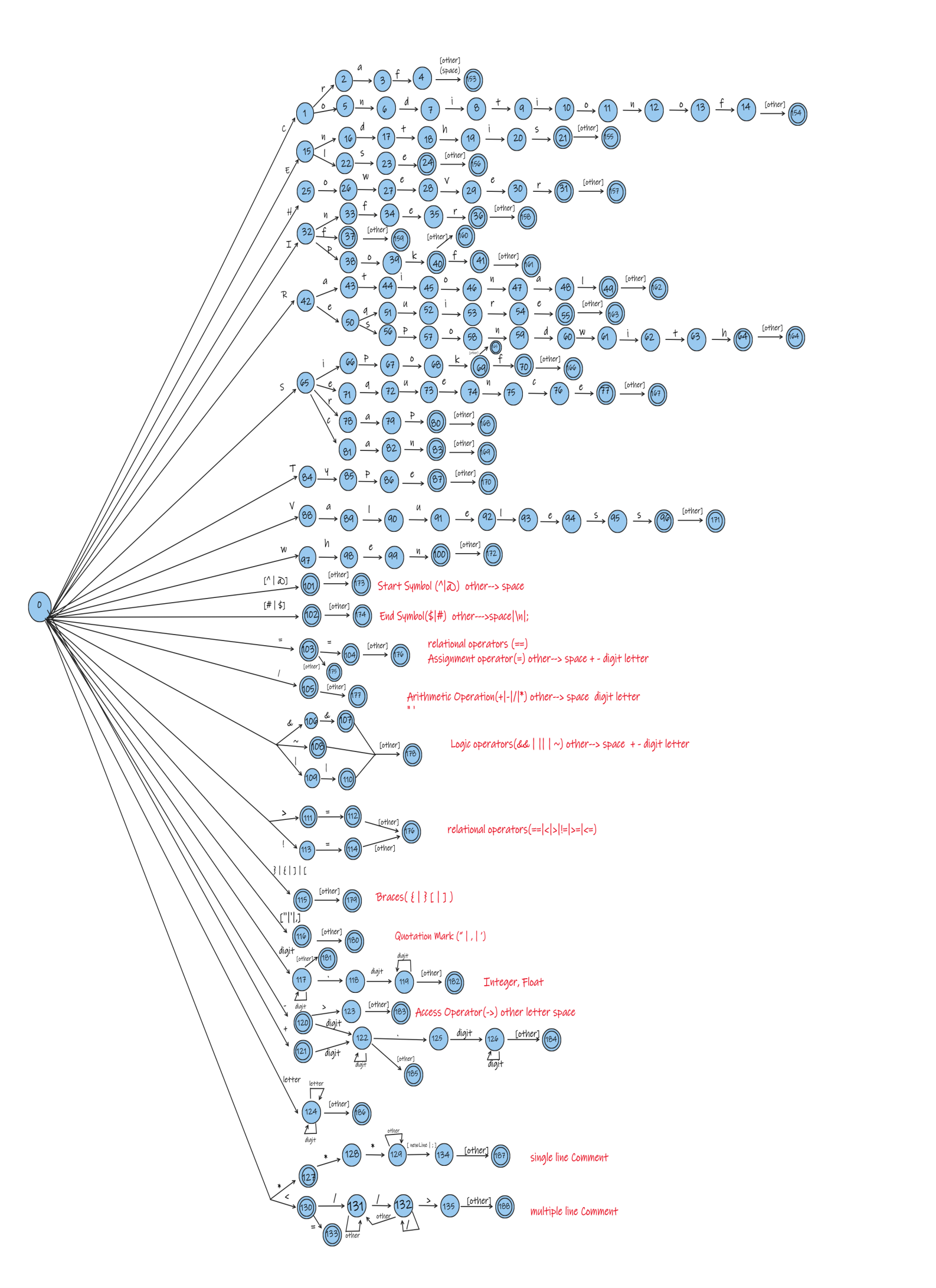












**Parse tree and Abstract syntax tree**

Parser grammar

1. (Program) → Start-Symbols ClassDeclaration End-Symbols.
2. (Start-Symbols) →@| ^
3. (End-Symbols) →$ |#
4. (ClassDeclaration) → Type ID (ClassDeclaration)' (left factoring)
5. (ClassDeclaration)' -> { Class\_Implementation} | Infer { Class\_Implementation}
6. (Class\_Implementation)→ Variable\_Decl Class\_Implementation| Method\_Decl Class\_Implementation | Comment Class\_Implementation | require\_command Class\_Implementation| Func \_Call Class\_Implementation |em
7. (Method\_Decl) → Func Decl (Method\_Decl)’
8. (Method\_Decl) → ( ; | { Variable\_Decl Statements } )
9. (Func Decl)→Type ID (ParameterList)
10. (Type) → Ipok |Sipok |Craf |Sequence |Ipokf |Sipokf |Valueless |Rational
11. (ParameterList)→em| None | Non-Empty List
12. (**Non-Empty List**)’ 🡪**, Type ID Non-Empty List’ |** Ꜫ
13. (**Non-Empty List**) 🡪 **Type ID (Non-Empty List)’**
14. (Variable\_Decl)→ em |Type ID\_List (Variable\_Decl)’
15. (Variable\_Decl)’→; Variable\_Decl | [ID] ; Variable\_Decl
16. (ID\_List)\ 🡪 ,ID (ID\_List)\ | Ꜫ
17. (ID\_List) 🡪 ID (ID\_List)\
18. (Statements)\ → (Statements)\ Statement | Ꜫ
19. (Statements) 🡪 em | Statement Statements
20. Statement → Assignment | If \_Statement | However \_Statement | when\_Statement

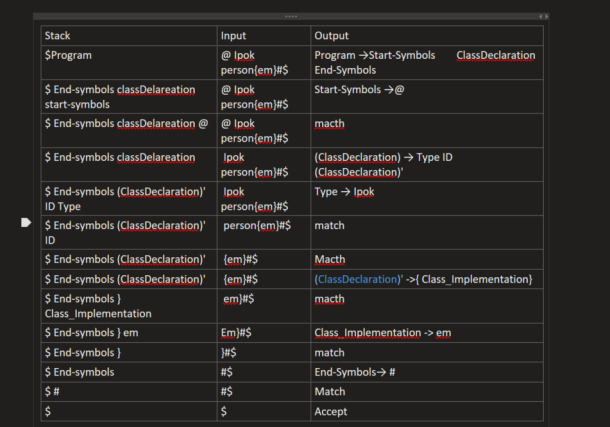
Respondwith \_ Statement | Endthis \_Statement|Scanvalur (ID ); | Print (Expression);

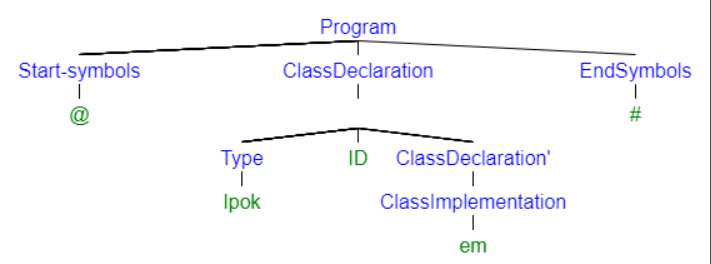
1. Assignment→ Variable\_Decl = Expression;
2. Func \_Call → ID (Argument\_List);
3. Argument\_List →em | NonEmpty\_Argument\_List
4. (NonEmpty\_Argument\_List)\ → , Expression (NonEmpty\_Argument\_List)\ | Ꜫ
5. ( NonEmpty\_Argument\_List) 🡪 Expression ( NonEmpty\_Argument\_List)\
6. Block Statements→{ statements }
7. If \_Statement→ if (Condition \_Expression) (If \_Statement)'
8. (If \_Statement)'→Block Statements | Block Statements else Block Statements
9. Condition \_Expression→ Condition (Condition \_Expression)'
10. (Condition \_Expression)' → (Ꜫ |Condition \_Op Condition)
11. Condition \_Op → && | ||
12. Condition→ Expression Comparison \_Op Expression
13. Comparison \_Op → == | != | > | >= | < | <=
14. However \_Statement → However (Condition \_Expression) Block Statements
15. when \_Statement → when ( expression ; expression ; expression ) Block Statements
16. Respondwith \_Statement→ Respondwith Expression ; | return ID ;
17. Endthis \_Statement→ Endthis;
18. Expression → Term Expression\
19. (Expression)\ → Add\_Op Term Expression\ | ε
20. Add\_Op → + | -
21. Term→Factor Term\
22. (Term)\ → Mul\_Op Factor Term\  | ε
23. Mul\_Op→ \* | /
24. Factor→ ID| Number
25. Comment → | \*\*\*STR
26. Require\_command →Require(F\_name.txt);
27. F\_name →STR

-----------------------------------------------------------------------------------------------------------------------------------------------

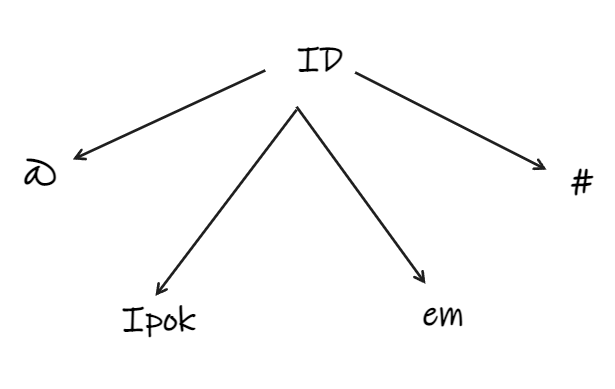
|  |  |  |
| --- | --- | --- |
|  | First | follow |
| **Program →Start-Symbols ClassDeclaration End-Symbols.** | **{@ , ^ }** | **{$}** |
| **Start-Symbols →@| ^** | **{@, ^ }** | **{** **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational}** |
| **End-Symbols→$ |#** | **{$,#}** | **{$}** |
| 1. (ClassDeclaration) → Type ID (ClassDeclaration)' (left factoring) 2. (ClassDeclaration)' -> { Class\_Implementation} | Infer { Class\_Implementation} | **{** **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational}** | **{$}** |
| **Class\_Implementation→ Variable\_Decl Class\_Implementation|**  **Method\_Decl** **Class\_Implementation | Comment Class\_Implementation |**  **require\_command Class\_Implementation| Func \_Call**  **Class\_Implementation |em** | **{em,** **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** | **{$}** |
| 1. (Method\_Decl) → Func Decl (Method\_Decl)’ 2. (Method\_Decl) → ( ; | { Variable\_Decl Statements } ) | **{ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** | **{** **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** |
| **Func Decl →Type ID (ParameterList)** | **{ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** | **{ ; , { }** |
| **Type → Ipok |Sipok |Craf |Sequence |Ipokf |Sipokf |Valueless |Rational** | **{ Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational}** | **{ID}** |
| **ParameterList →em| None | Non-Empty List** | **{em, None, Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** | **{ ) }** |
| **(Non-Empty List)\ 🡪**  **Type ID ,(Non-Empty List)\  | Ꜫ** | Ipok  Sipok  Craf  Sequence  Ipokf  Sipokf  Valueless  Rational  **Ꜫ** | ) |
| **(Non-Empty List) 🡪**  **Type ID , (Non-Empty List)\** | Ipok  Sipok  Craf  Sequence  Ipokf  Sipokf  Valueless  Rational | ) |
| 1. (Method\_Decl) → Func Decl (Method\_Decl)’ 2. (Method\_Decl) → ( ; | { Variable\_Decl Statements } ) | em  Ipok  Sipok  Craf  Sequence  Ipokf  Sipokf  Valueless Rational | = |
| **(ID\_List)\  🡪, ID (ID\_List)\ | Ꜫ** | **,**  **Ꜫ** | ; [ |
| **(ID\_List) 🡪 ID (ID\_List)\** | **ID** | ; [ |
| **Statements → em | Statement Statements** | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print**  **else** |
| **Statement →**  **Assignment | If \_Statement | However \_Statement | when\_Statement | Respondwith \_ Statement | Endthis \_Statement|Scanvalur (ID ); | Print (Expression);** | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **Print** | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** |
| **Assignment→ Variable\_Decl = Expression;** | Em  Ipok  Sipok  Craf  Sequence  Ipokf  Sipokf  Valueless  Rational | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** |
| **Func\_Call → ID (Argument\_List) ;** | **ID** | **{em, Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational }** |
| **Argument\_List →em | NonEmpty\_Argument\_List** | **em ID Number** | ) |
| **( NonEmpty\_Argument\_List)\  🡪**  **, Expression**  **( NonEmpty\_Argument\_List)\  | Ꜫ** | **, | Ꜫ** | **)** |
| **( NonEmpty\_Argument\_List) 🡪**  **Expression ( NonEmpty\_Argument\_List)\** | **ID Number** | **)** |
| **Block\_Statements→{ statements }** | **{** | **else**  **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** |
| 1. If \_Statement→ if (Condition \_Expression) (If \_Statement)' 2. (If \_Statement)'→Block Statements | Block Statements else Block Statements | **If** | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** |
| 1. Condition \_Expression→ Condition (Condition \_Expression)' 2. (Condition \_Expression)' → (Ꜫ |Condition \_Op Condition) | {ID, Number} | { ) } |
| 1. Condition \_Op → && | || | { & , | } | { ID , Number} |
| 1. Condition→   Expression Comparison \_Op Expression | { ID , Number } | { & ,| , ) } |
| 1. Comparison \_Op → == | != | > | >= | < | <= | { = , ! , > , < } | { ID , Number } |
| 1. However \_Statement → However (Condition \_Expression) Block Statements | { However } | **em**  **Ipok ,Sipok ,Craf ,Sequence ,Ipokf ,Sipokf ,Valueless ,Rational**  **if**  **However**  **when**  **Respondwith**  **Endthis Scanvalur**  **print** |
| 1. when\_Statement → when ( expression ; expression ; expression ) Block Statements | { when } | NA |
| 1. Respondwith\_Statement → Respondwith Expression ; | return ID ; | {Respondwith} | NA |
| 1. ----Endthis \_Statement→ Endthis; | { Endthis } | { Endthis } |
| 1. Expression → Term Expression\ | { ID , Number } | { ! , > , < , & , | , ) } |
| 1. (Expression)\ → Add\_Op Term Expression\ | ε | { + , - , ε } | { ; , ) , = , ! , > , < ,&,|, } |
| 1. Add\_Op → + | - | { + , - } | { ID , Number } |
| 1. Term→Factor Term\ | { ID , Number } | { + , - , ; , ) , = , ! , > , < , & , | , } |
| 1. (Term)\ → Mul\_Op Factor Term\  | ε | { \* , / , ε } | { + , - , ; , ) , = , ! , > , < , & , | } |
| 1. Mul\_Op→ \* | / | { \* , / } | { ID , Number } |
| 1. Factor→ ID| Number | { ID , Number } | { \* , / , + , - , ; , ) , + , - , = , ! , > , < , & , | } |
| 1. Comment → | \*\*\*STR | { < , \* } | NA |
| 1. Require\_command →Require(F\_name.txt); | { Require } | NA |
| 1. F\_name →STR | { STR } | { . } |
|  |  |  |

**Parser Tree**

****

****

**Abstract syntax tree**

****