Cmple

A Simple Programming Language based on C

**Submitted by:**

Canja, Jason

Cruzat, Joshua

Galindez, Elijah

Rivera, Ron

Sales, Chian

**Submitted to:**

Prof. Ria A. Sagum

**February 2021**

**Name:** Cmple

**Introduction:**

Cmple (Simple) is based on C which aims to develop a programming language for beginners who are not yet experienced and used to the different technical terms in a programming environment. Since C is probably one of the first programming languages taught to a beginner, Cmple’s structure is based on the C language. The members believe that building an easy but good foundation in terms of programming will make it easier for the “newbies” to process the technicalities of programming with ease.

After brainstorming, the members realized that most, if not all, languages use many difficult technical words that might scare a beginner. Terms that are not familiar were always used in many languages. The goal in mind in making this programming language is to make it easy and simple as much as possible because it is again, intended for beginners. This programming language will also help teach them the basics of coding without difficulty. The name Cmple (stylized with C and Simple in mind) is based on the word Simple, which means easily understood or done.

**Syntactic Elements:**

1. **Character Set**

These are the accepted characters of Cmple, which is composed of: Alphabets, Digits, and Symbols.

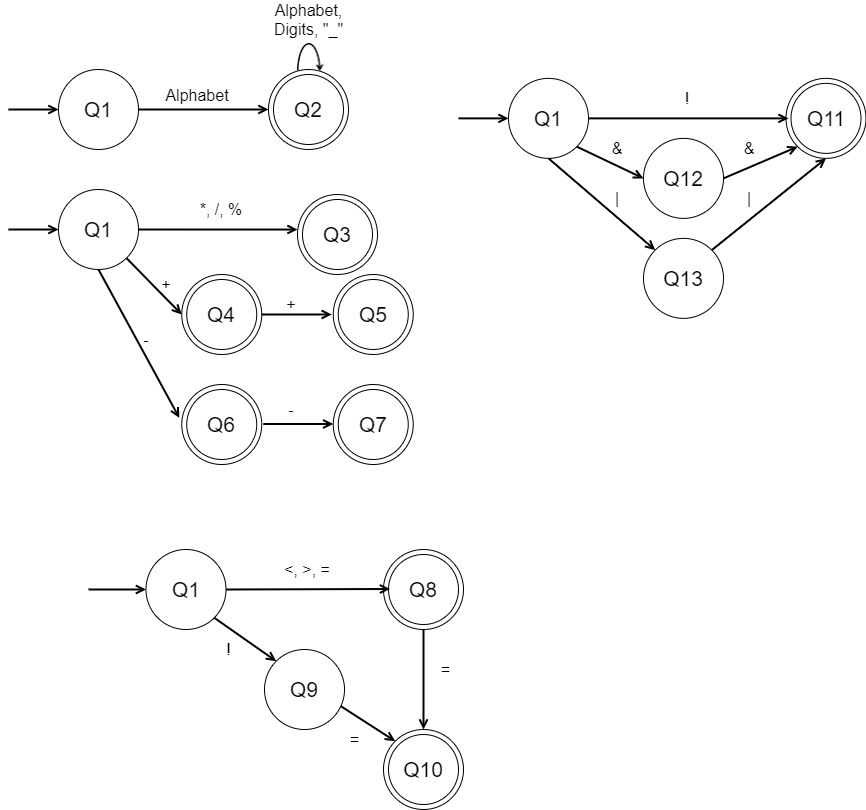
* Character\_Set = {Alphabet, Digits, Symbols}
* Alphabet = {Upper, Lower}
* Upper = {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}
* Lower = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}
* Digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0}
* Symbols = {+, -, \*, /, %, <, >, |, =, &, !, (, ), {, }, [, ], :, ;, “, ‘, ., \_, ~}

1. **Identifier**

Rules for writing an identifier:

* An identifier must start with an Alphabet.
* It should not contain any Symbols except for the character “\_” (underscore).
* Keywords and reserved words must not be used as an identifier.

Machine for Identifier:



Regular Expression for Identifier: Alphabet (Alphabet **+** Digits **+** “\_”)\*

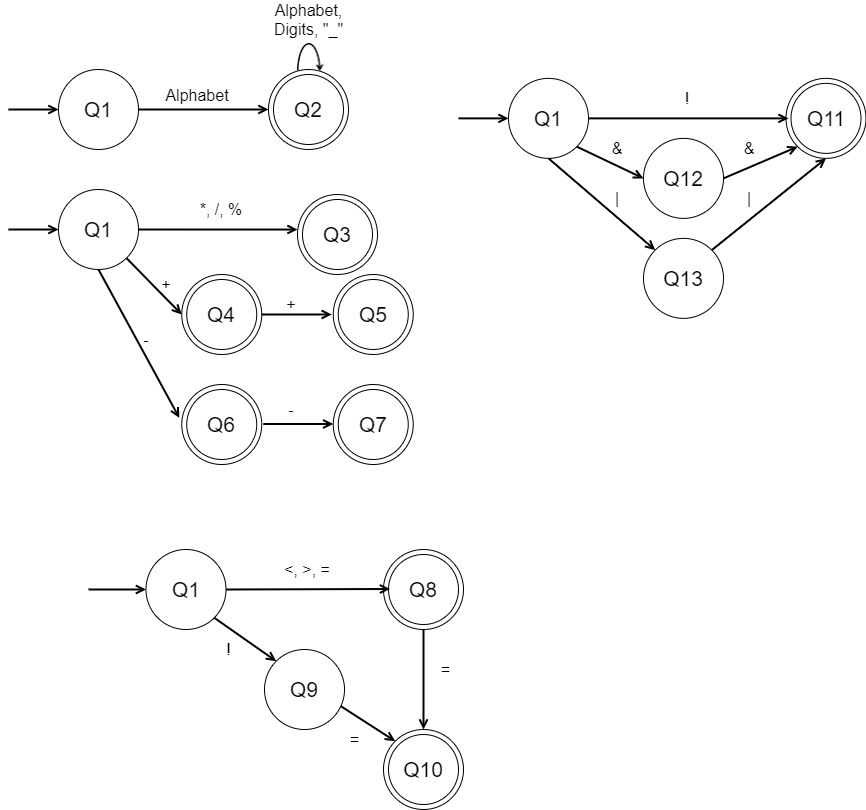
1. **Operation Symbols**

The language will accept the standard operation symbols:

* Arithmetic: +, -, \*, /, %, ++, --
* Relational: <, >, <=, >=, !=, ==, =
* Logical: &&, ||, !

|  |  |  |
| --- | --- | --- |
| **Arithmetic** | **Description** | **Example (A= 10, B = 20)** |
| + | adds two operands or unary plus | A + B = 30 |
| - | subtracts two operands or unary minus | A - B = -10 |
| / | performs division on two operands | A / B = 0.5 |
| \* | multiplies two operands | A \* B = 200 |
| % | gives the remainder of two operands | A % B = 0 |
| ++ | increases the value of the operand by 1 | A++ = 11 |
| -- | decreases the value of the operand by 1 | A-- = 9 |

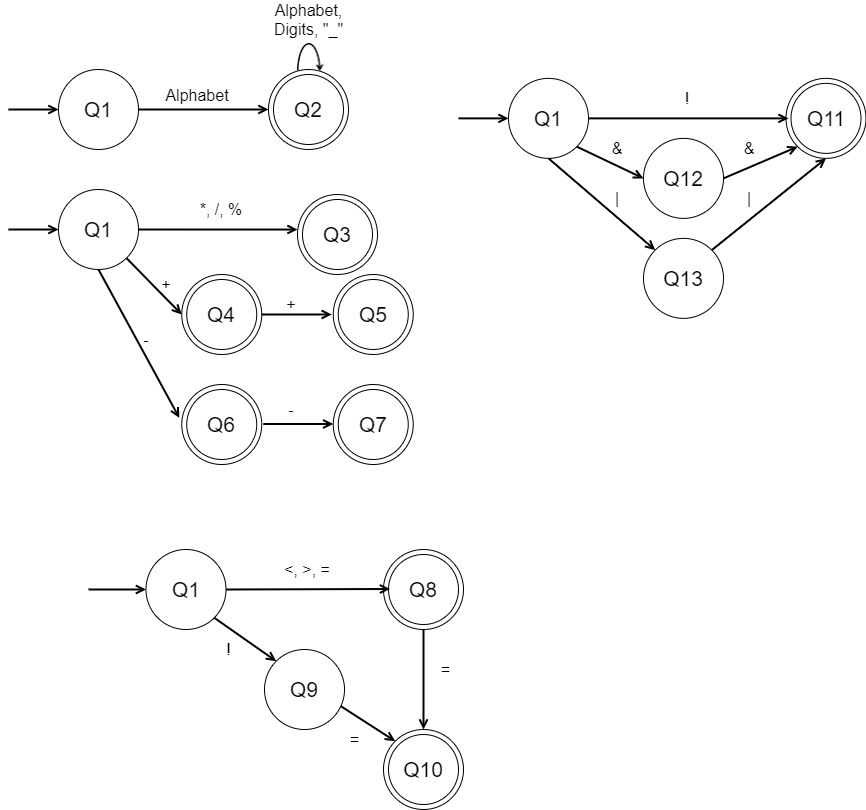
Machine for Arithmetic Operators:



Regular Expression for Arithmetic Operators: \* **+** / **+** % **+** + **+** - **+** (++) **+** (--)

|  |  |  |
| --- | --- | --- |
| **Relational** | **Description** | **Example (A= 10, B = 20)** |
| < | check operand on the left is smaller than right operand | A < B (True) |
| > | check if operand on the left is greater than operand on the right | A > B (False) |
| <= | checks if the left operand is smaller or equal than the right operand | A <= B (True) |
| >= | checks if the operand on the left is greater than or equal to the right operand | A >= B (False) |
| != | check if two operands are not equal | A != B (True) |
| == | check if the two operands are equal | A == B (False) |
| = | assigns values from right side operands to left side operand | A = B (A = 20) |

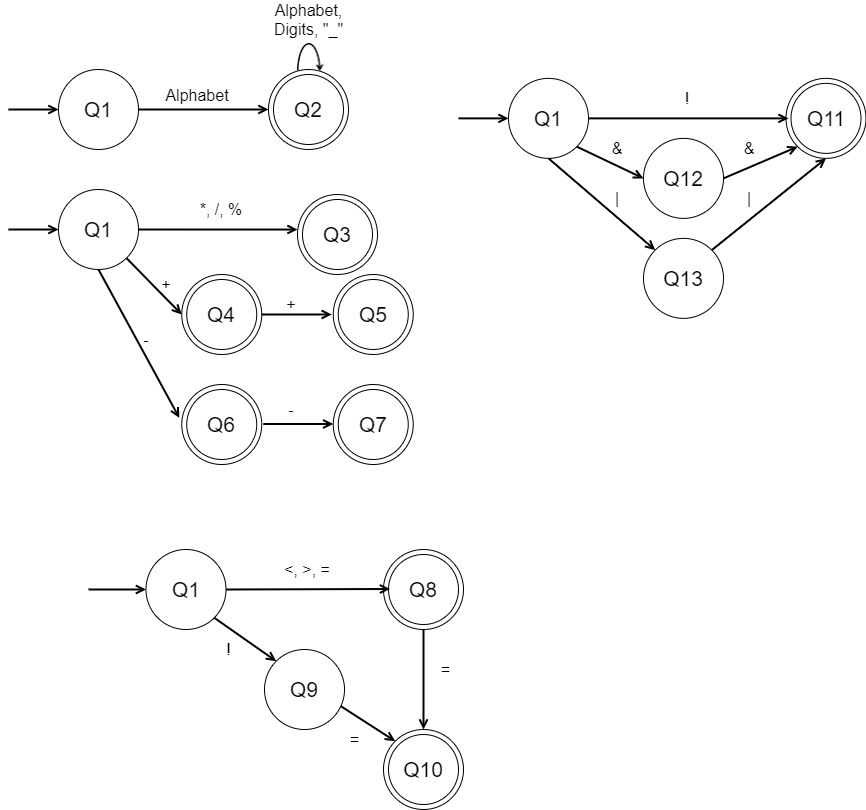
Machine for Relational Operators:



Regular Expression for Relational Operators: (< **+** > **+** =) **+** (< **+** > **+** = **+** !)=

|  |  |  |
| --- | --- | --- |
| **Logical** | **Description** | **Example (A= True, B = False)** |
| && | a set of operands will be true if and only if all its operands are true | A && A = True  A && B = False |
| || | a set of operands will be true if and only of one or more operands are true | A || A = True  A|| B = True  B || B = False |
| ! | returns opposite logical state of its operand | !A = False  !B = True |

Machine for Logical Operators:



Regular Expression for Logical Operators: ! **+** (&&) **+** (||)

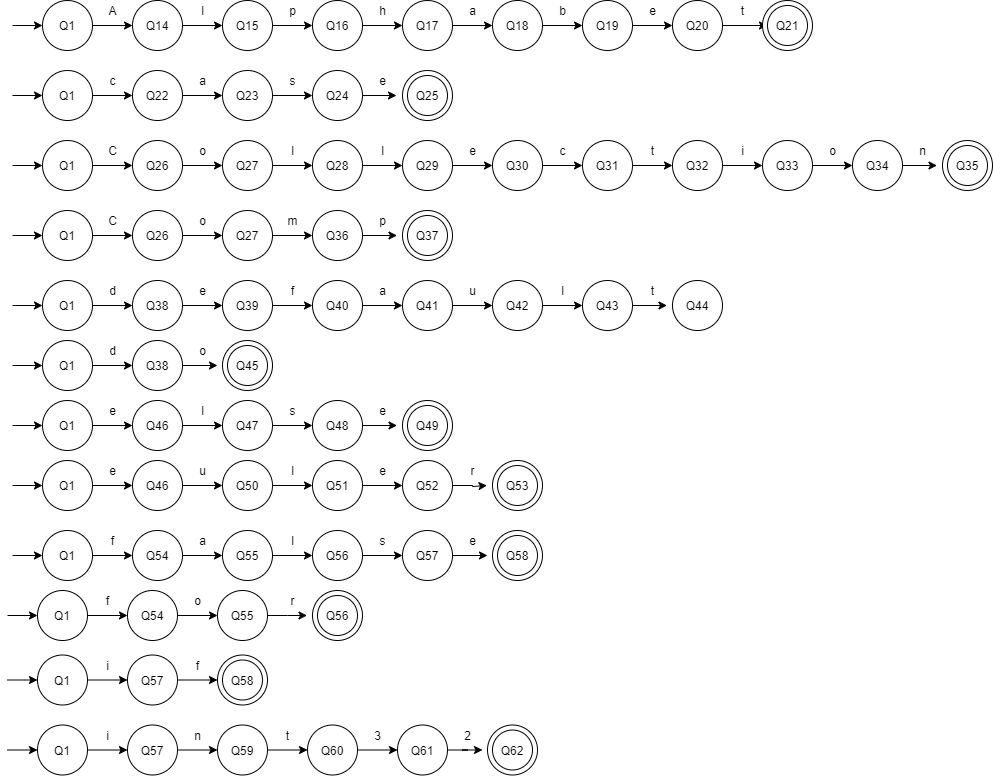
1. **Keywords and Reserved Words**

|  |  |
| --- | --- |
| **Keywords** | **Description** |
| case | conditional statement |
| default | It is the end of switch statement; executed if no case constant-expression value is equal to the value of expression |
| do | It is used for iteration of specific codes if the condition for the while loop is TRUE |
| else | Conditional statement where another task is performed if the first condition is FALSE |
| for | It is used for iteration of codes until the condition it met |
| if | Conditional statement wherein it performs a task if it is TRUE |
| otherwise | conditional statement |
| resume | a statement that instructs a program to leave the subroutine and go back to the return address |
| return | a statement that instructs a program to leave the subroutine and go back to the return address |
| stop | the loop is immediately terminated, and the program control resumes at the next statement following the loop |
| switch | A conditional statement used to test on a list of values |
| while | It is used for iteration of specific instructions if the condition is TRUE |

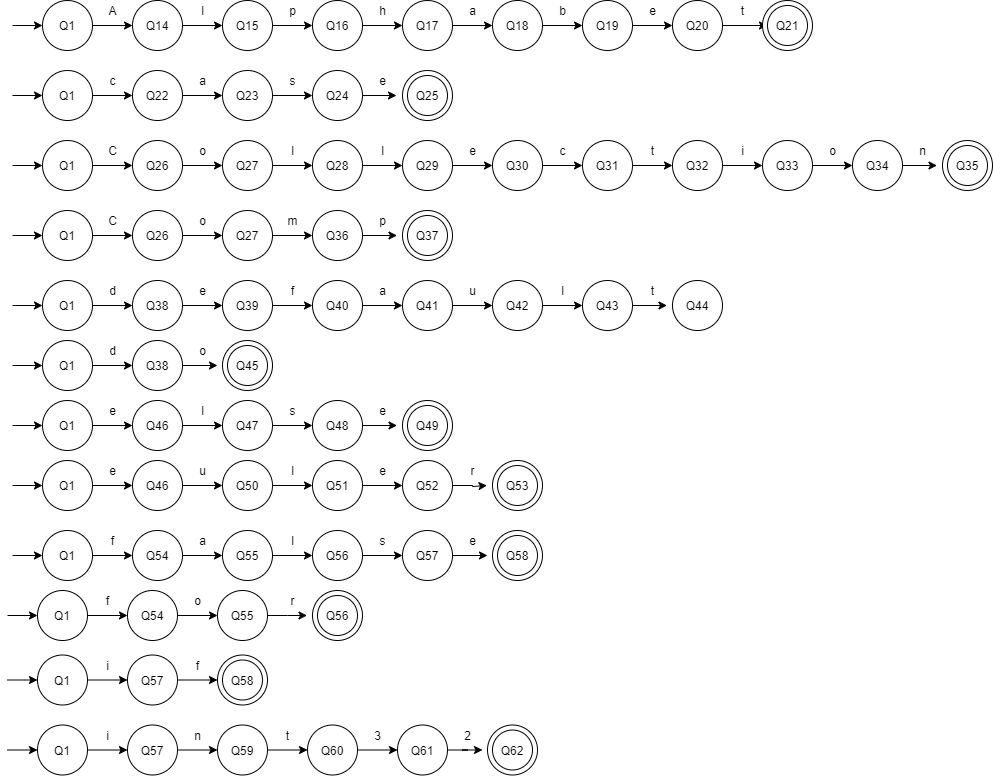
|  |  |
| --- | --- |
| **Reserved Words** | **Description** |
| Alphabet | returns ABCDEFGHIJKLMNOPQRSTUVWXYZ |
| Collection | It is used for Array identifier |
| Comp | It is used for Structures identifier |
| euler | returns a constant value of 2.71828 |
| false | returns when the logical statement is false based on the condition given |
| int32 | returns a constant value of 2147483647 |
| Item | It is used for Enum identifier |
| kelvin | returns a constant value of 273 |
| None | null, void, nothing |
| Number | It is used for numeric identifier |
| odd | returns the first five odd numbers 1,3,5,7,9 |
| pi | returns a constant value of 3.14 |
| Sentence | It is used for string identifier |
| read | Used to read or scan an input |
| then | noise word after if statement |
| Tralse | It is used for Boolean identifier |
| true | returns when the logical statement is true based on the condition given |
| write | Used to print or display the output |

Machine for Keywords and Reserved Words:

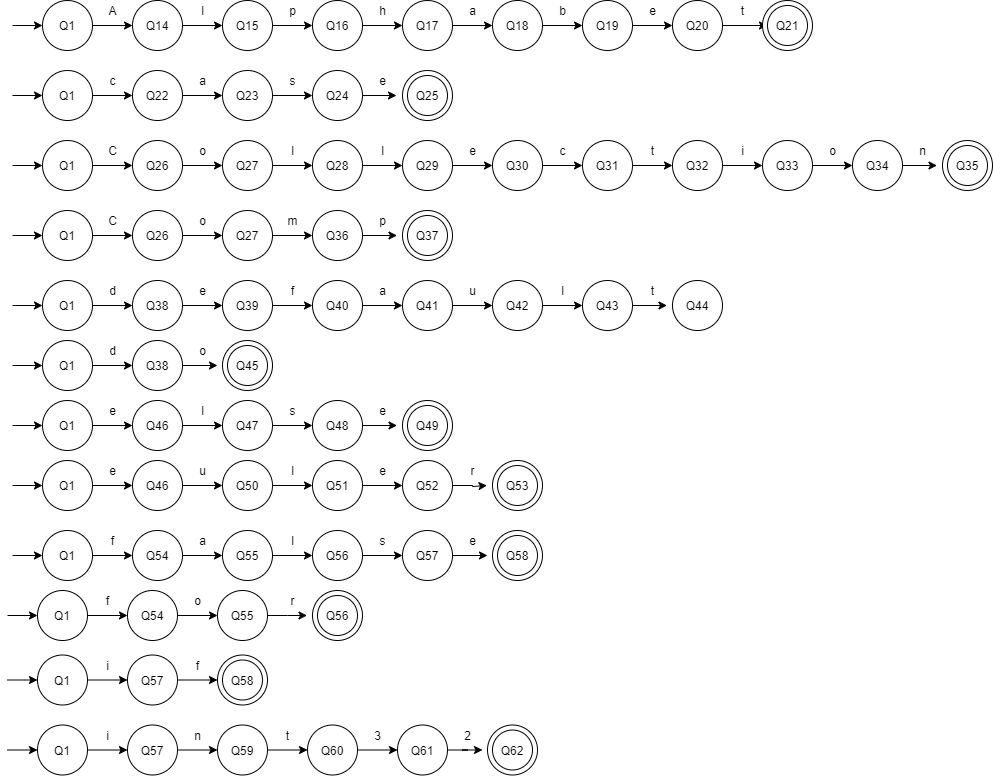
**Alphabet**



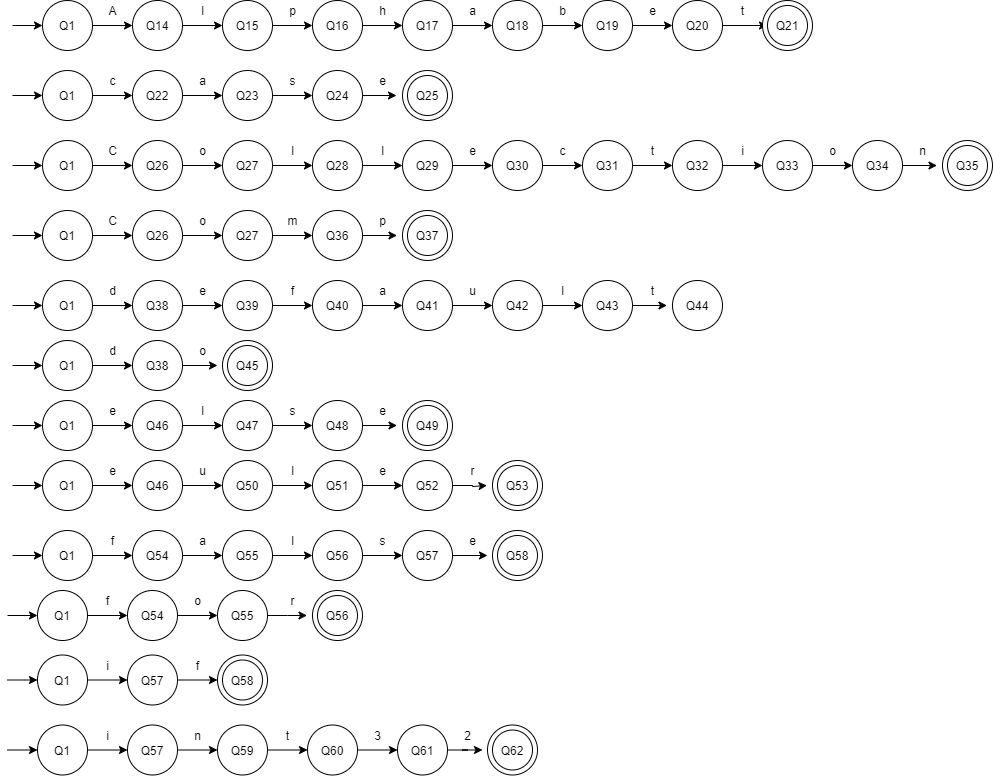
**case**



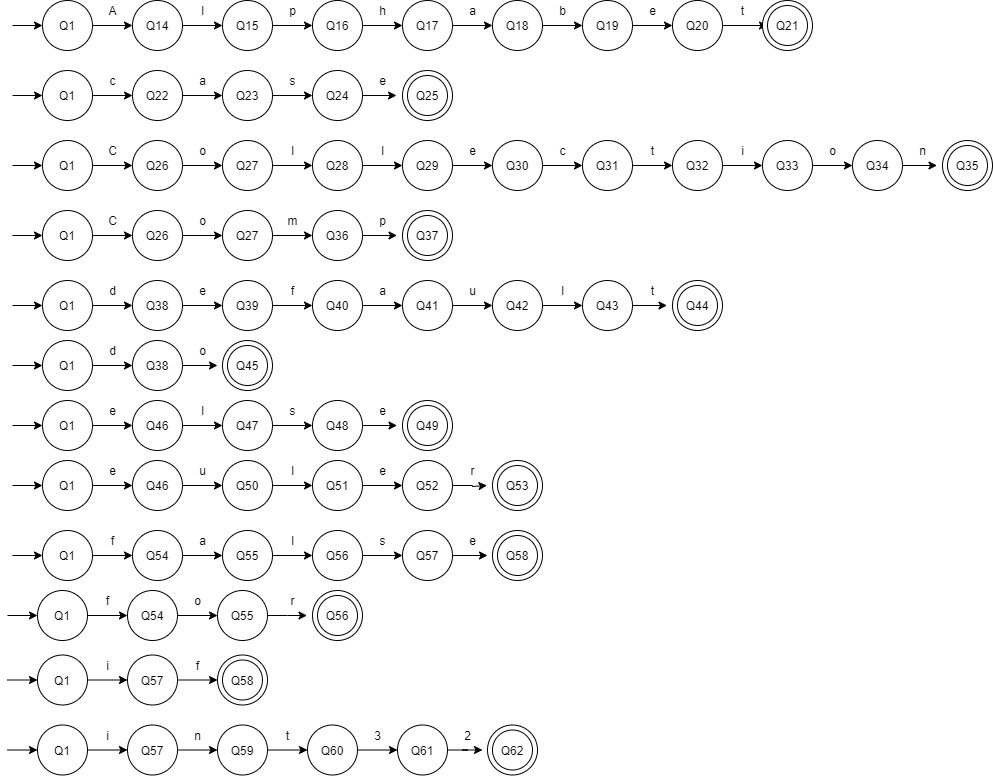
**Collection**



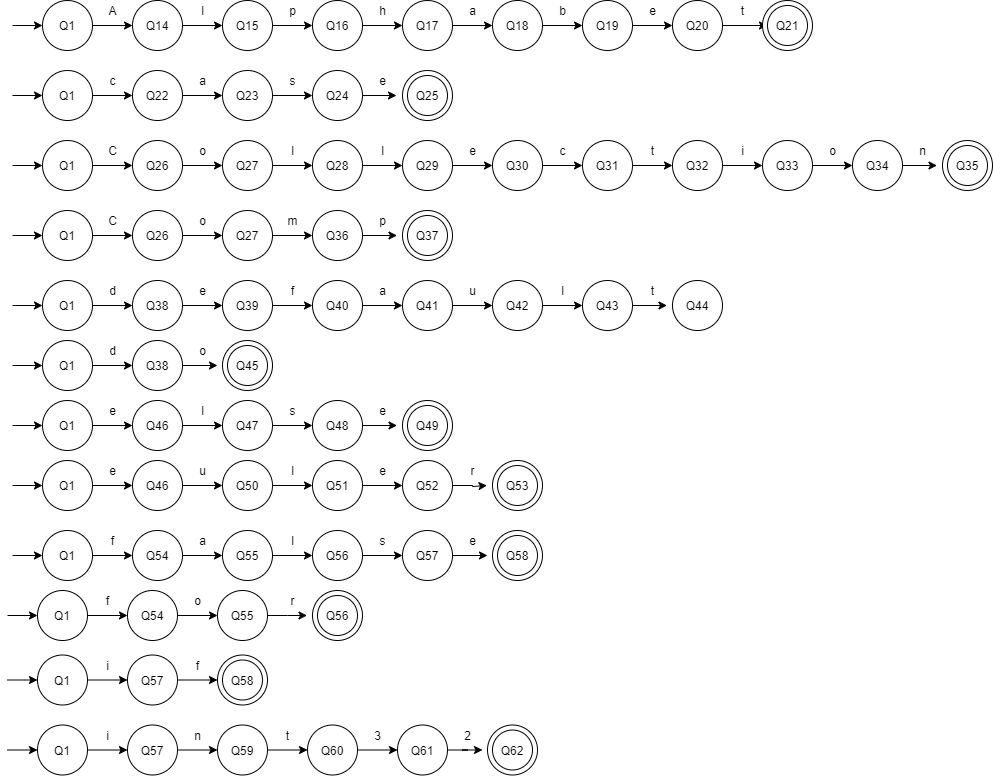
**Comp**

****

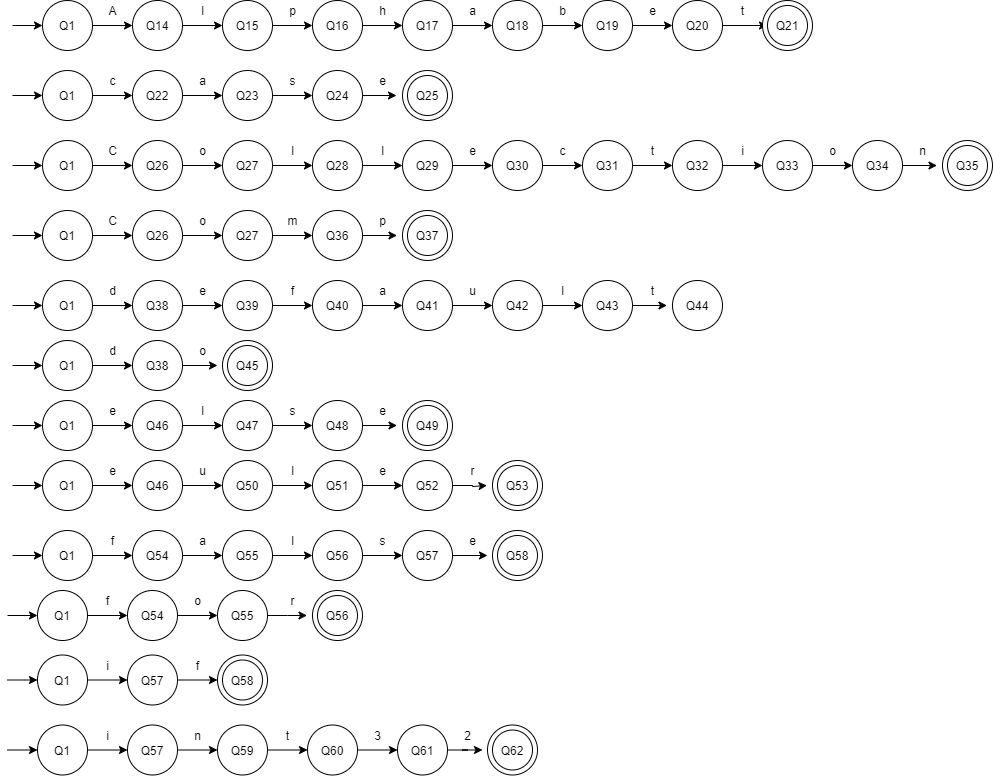
**default**

****

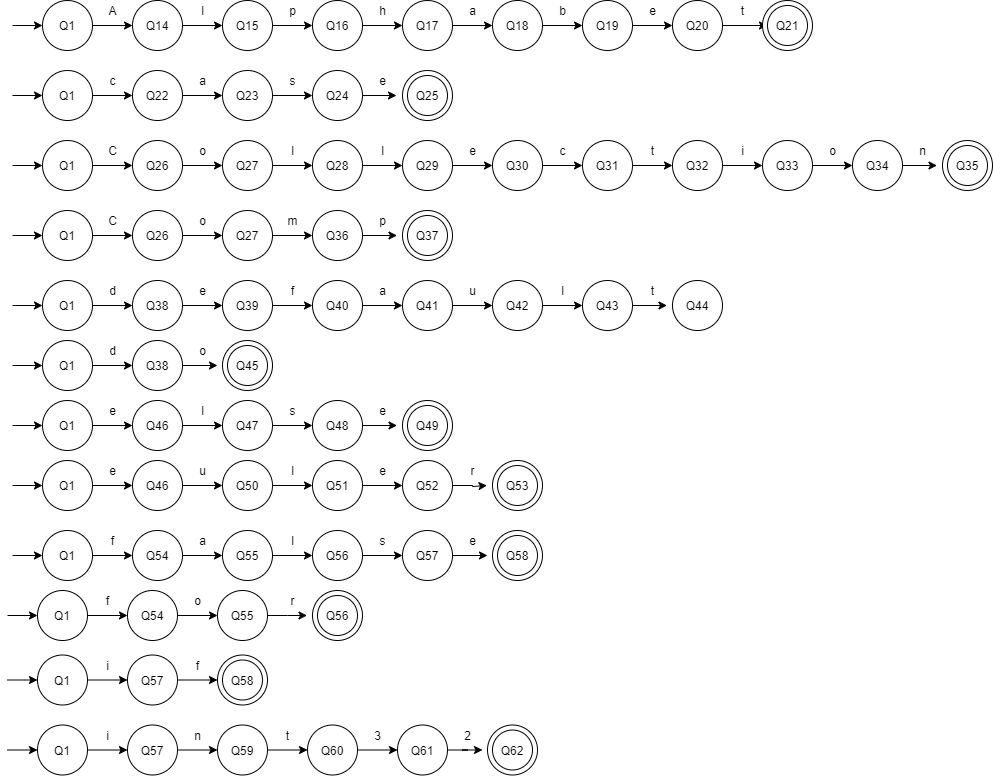
**do**

****

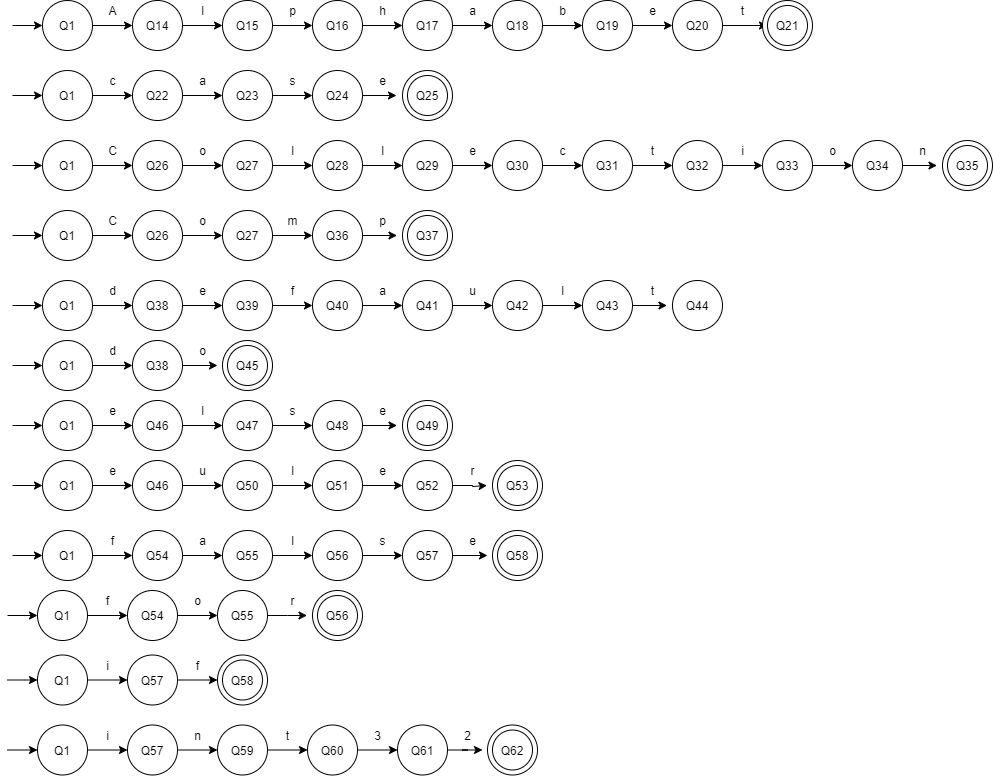
**else**

****

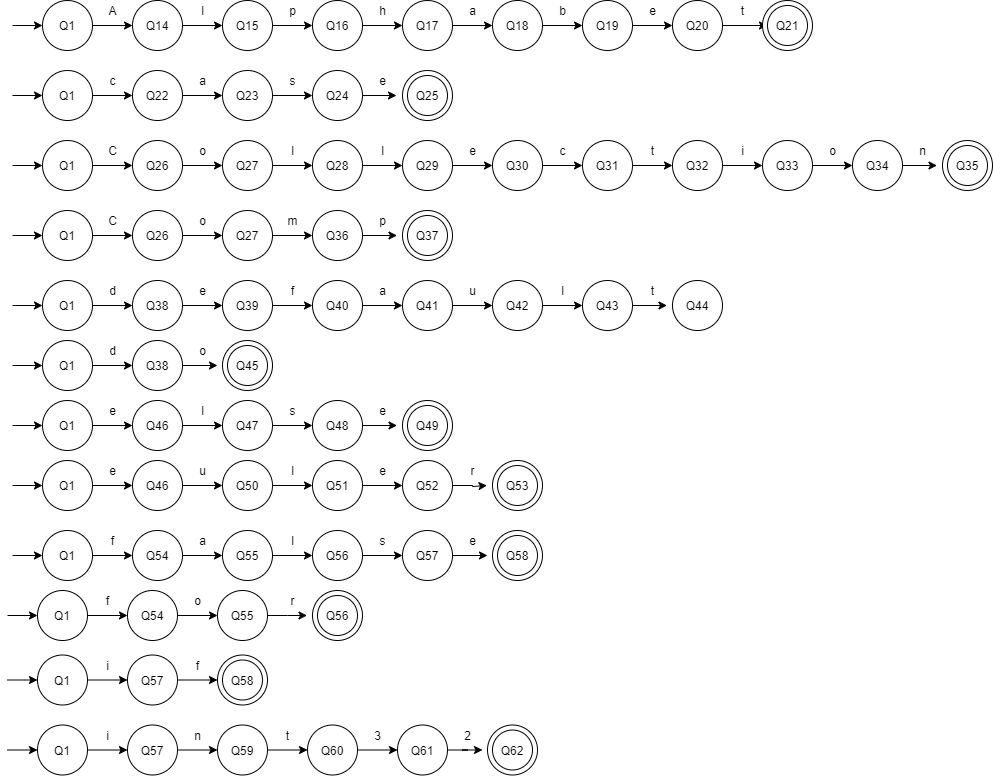
**euler**

****

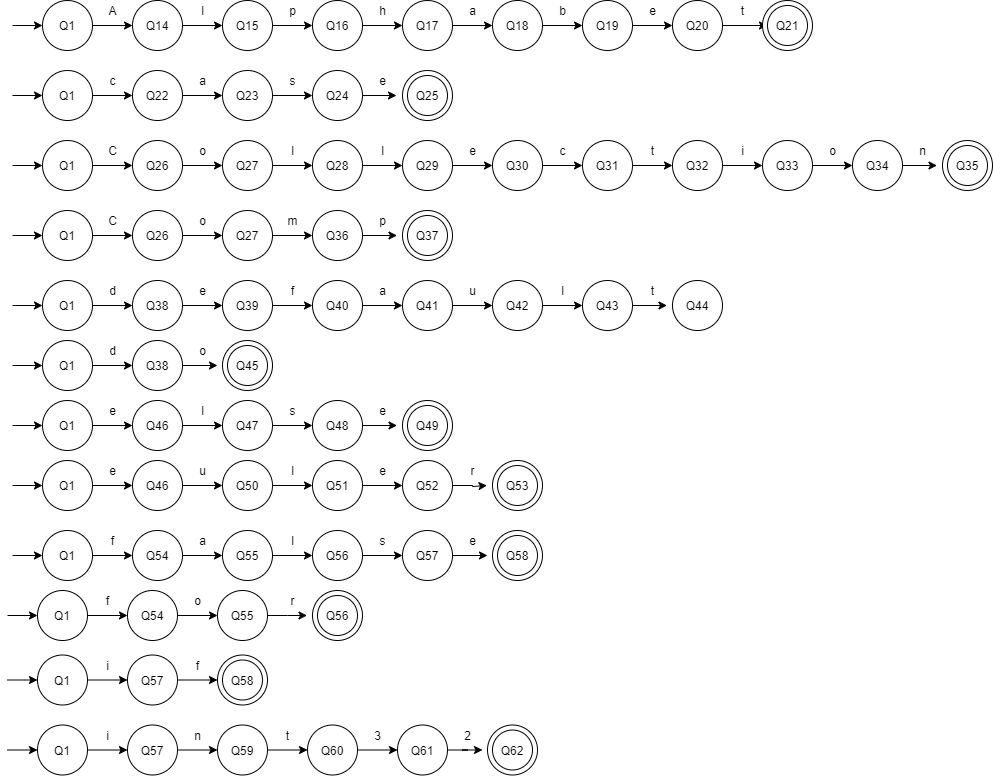
**false**

****

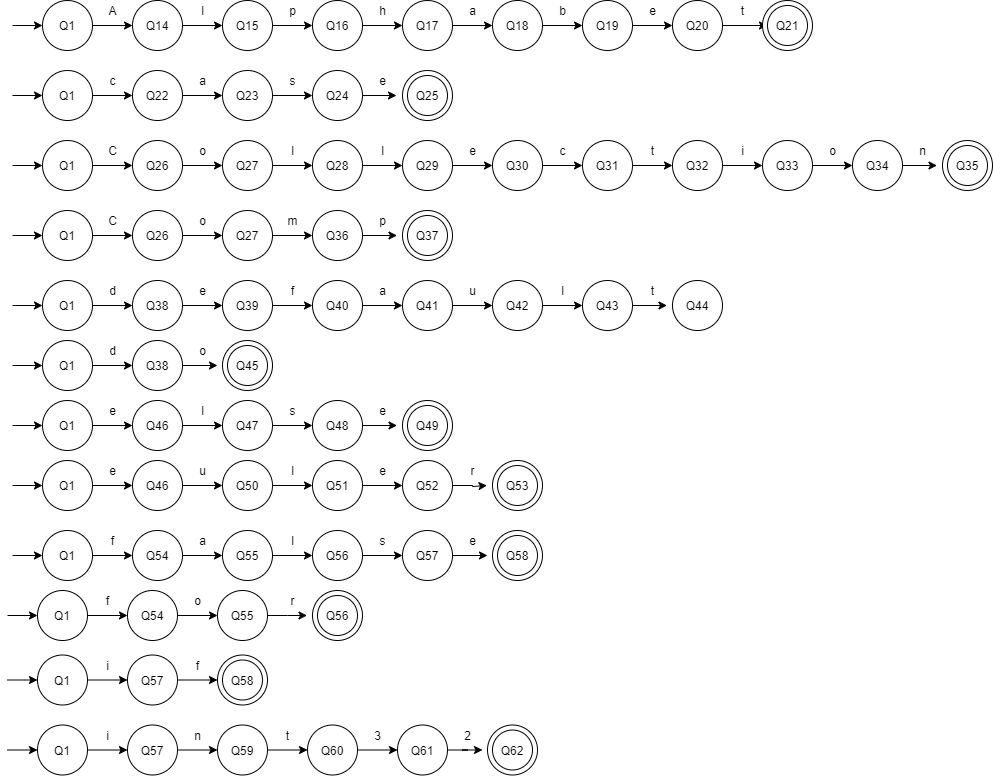
**for**

****

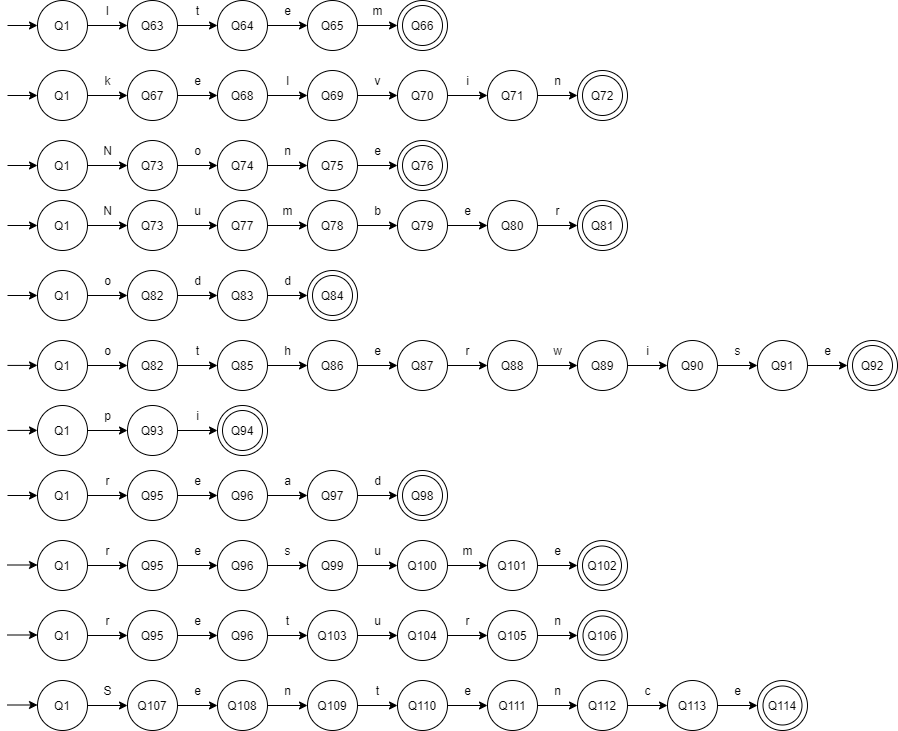
**if**

****

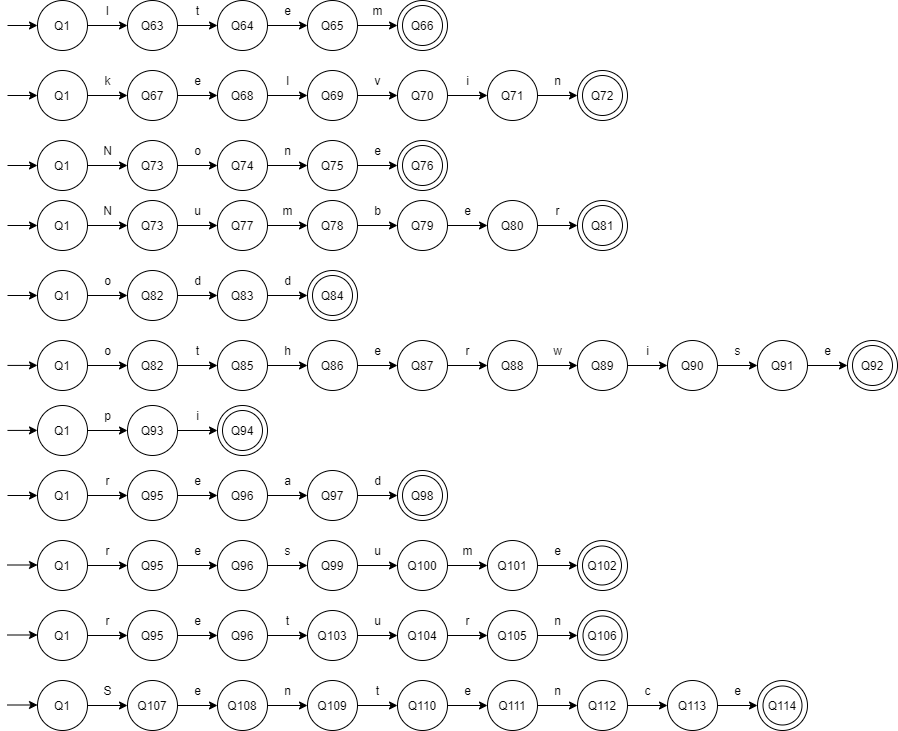
**int23**

****

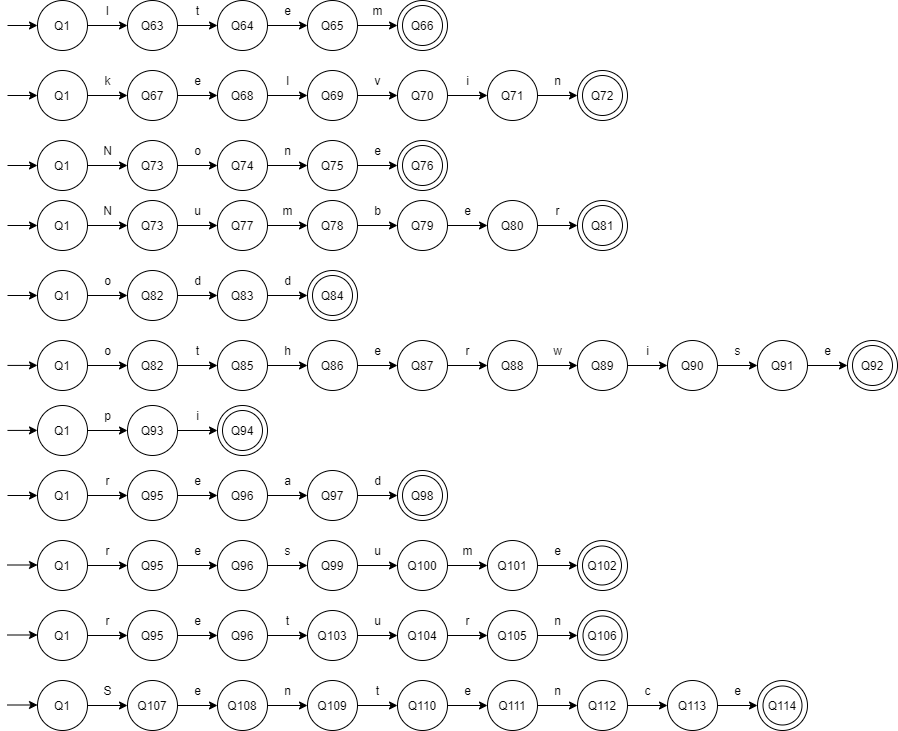
**Item**

****

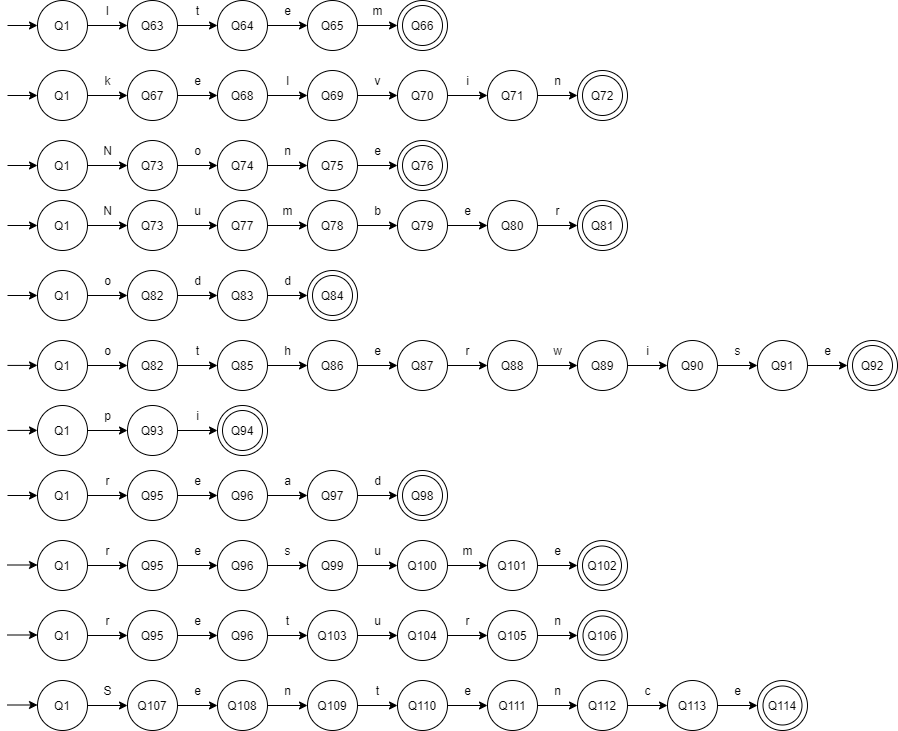
**kelvin**

****

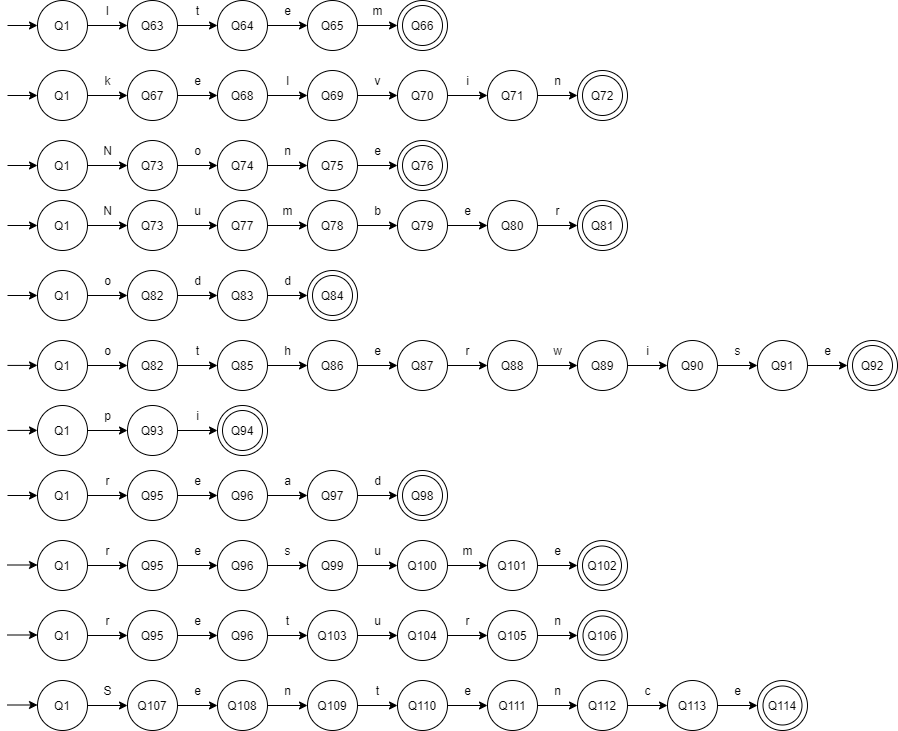
**None**

****

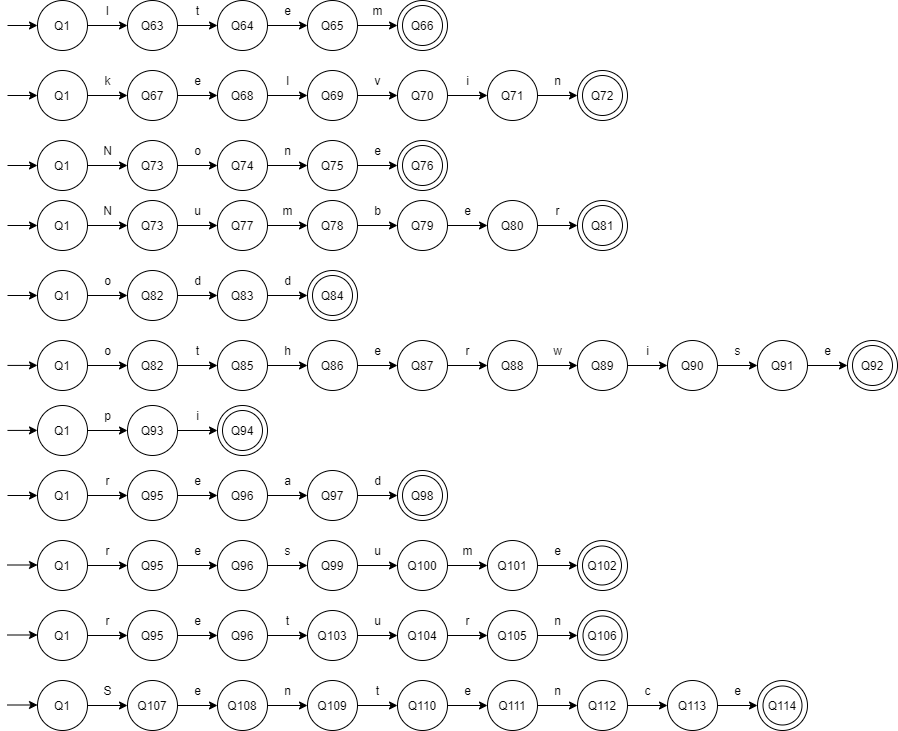
**Number**

****

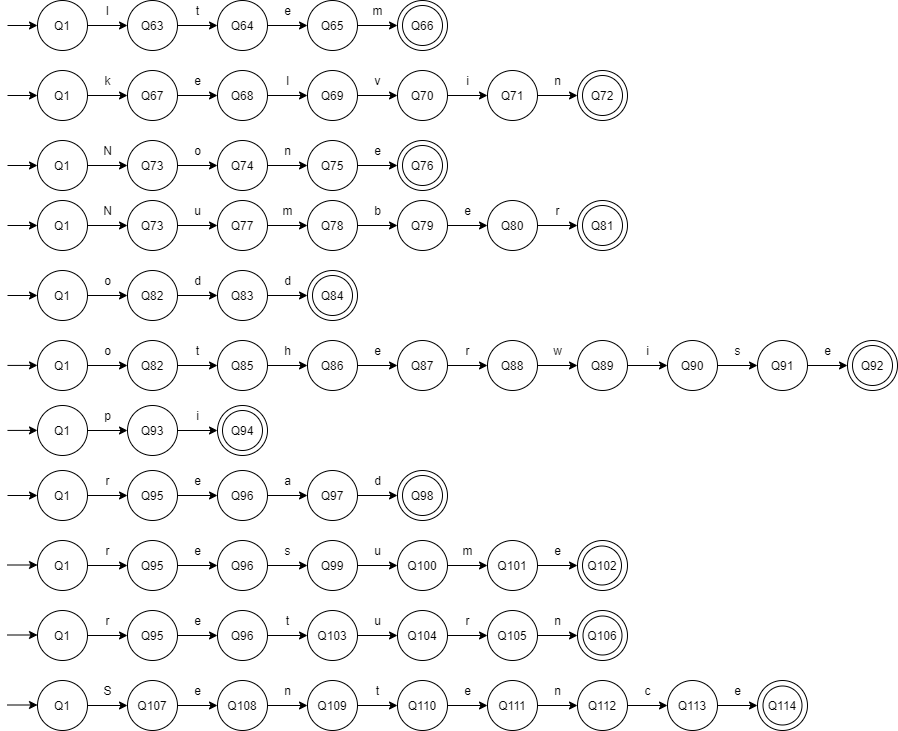
**odd**

****

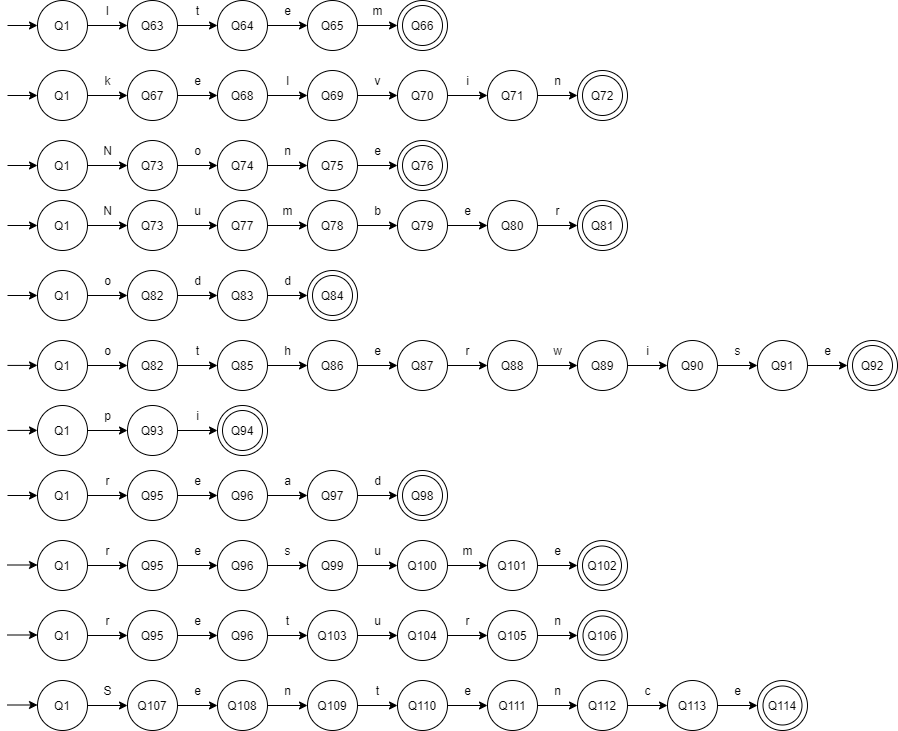
**otherwise**

****

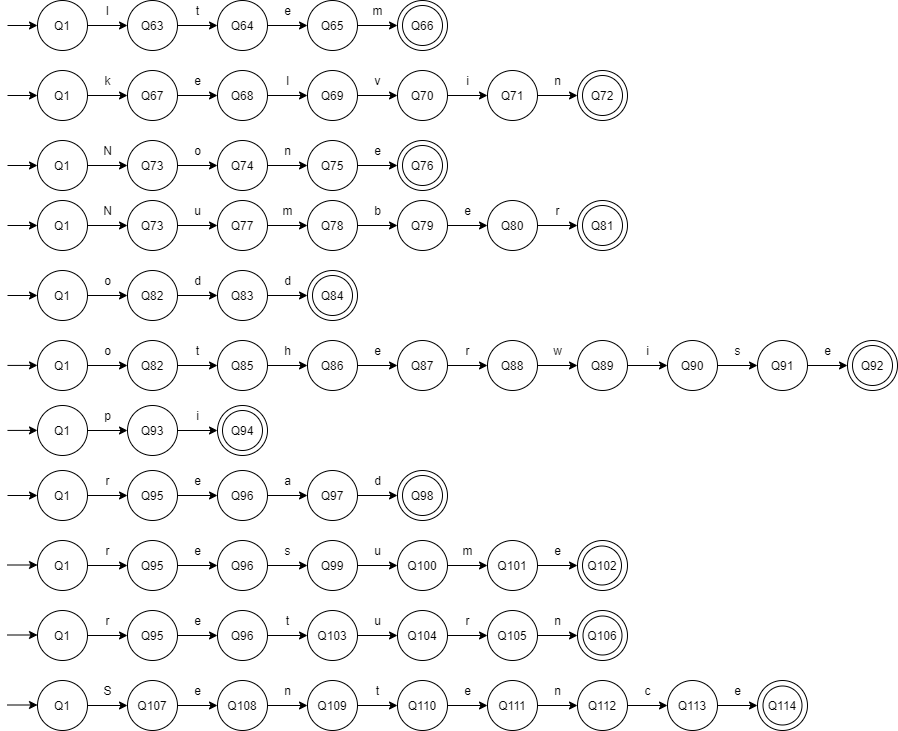
**pi**

****

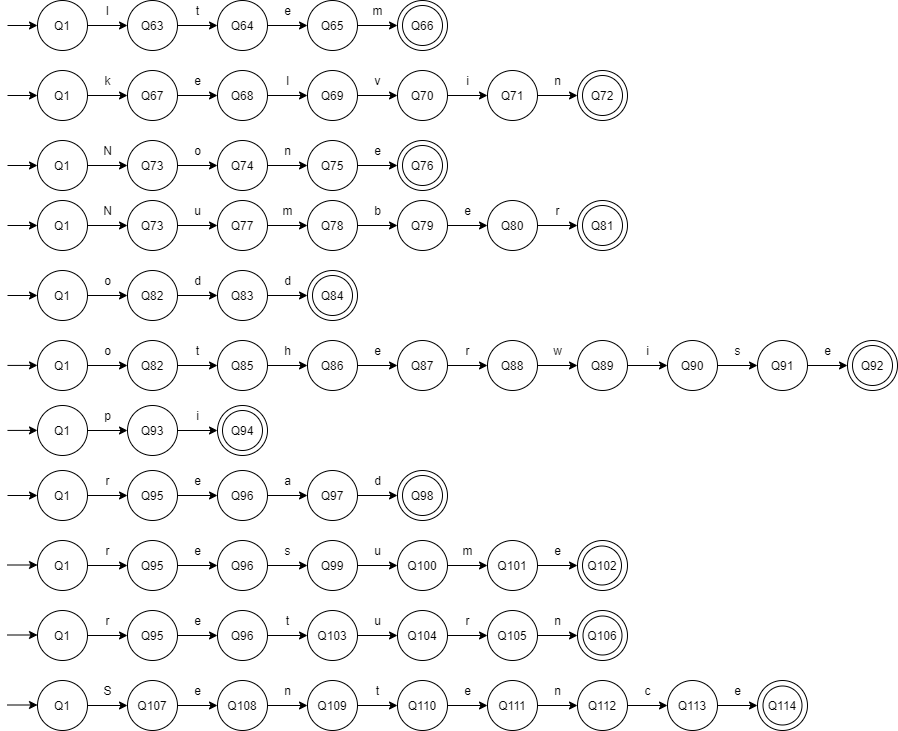
**read**

****

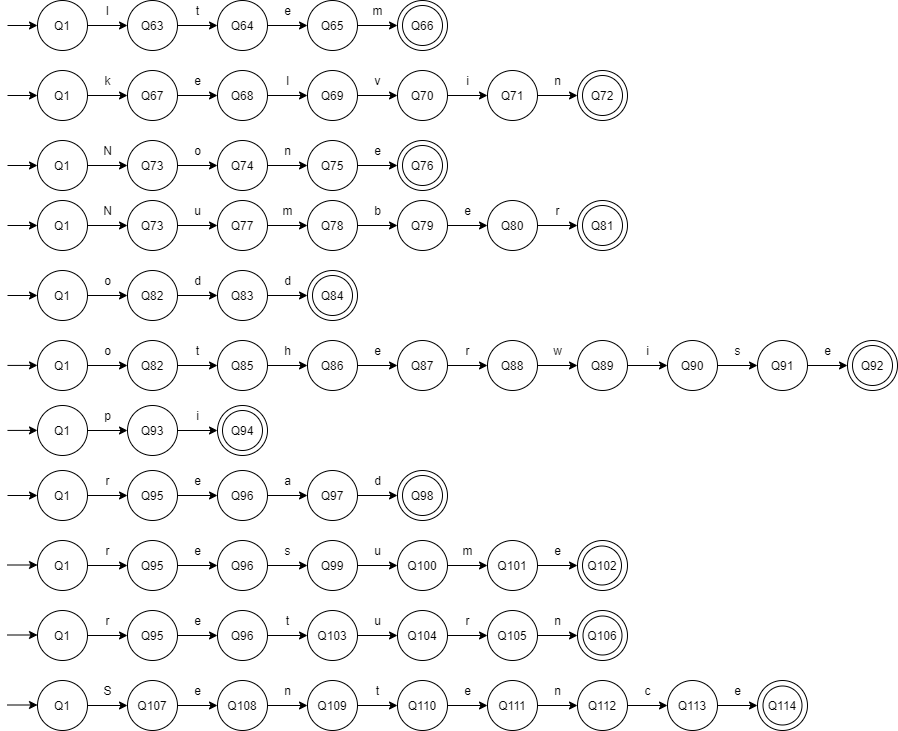
**resume**

****

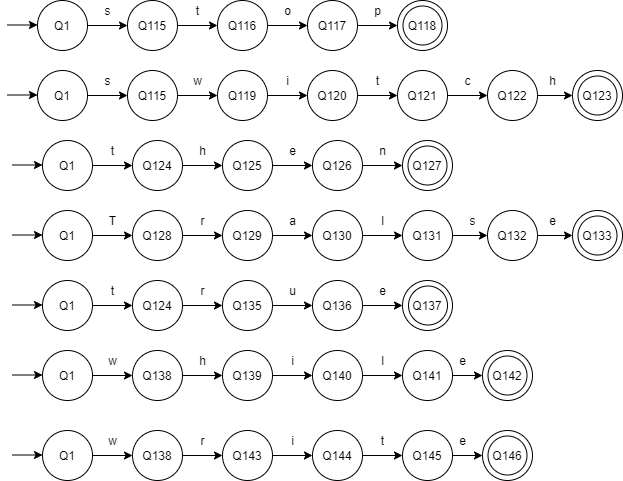
**return**

****

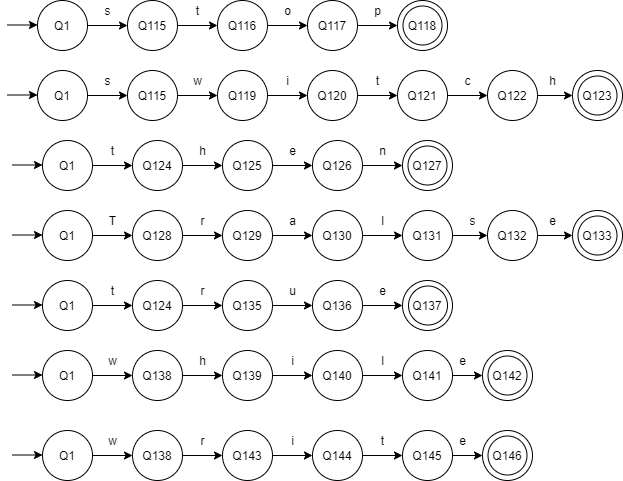
**Sentence**

****

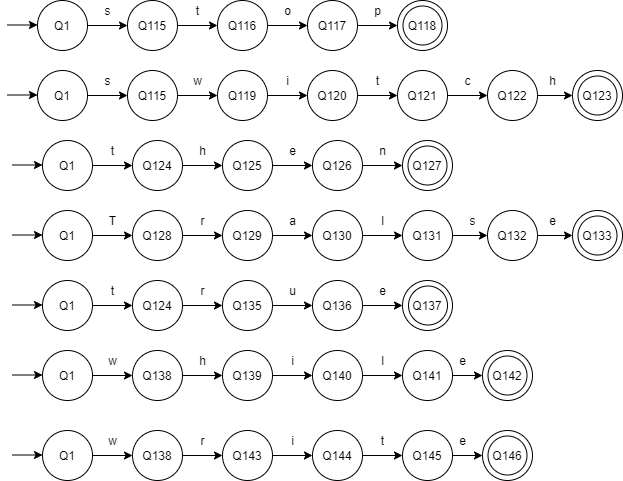
**stop**

****

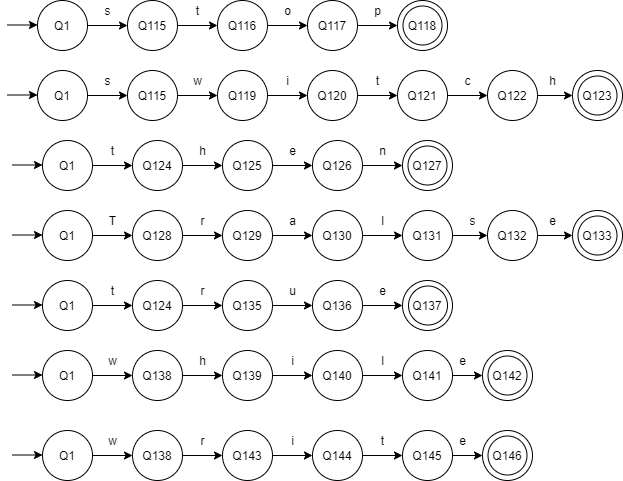
**switch**

****

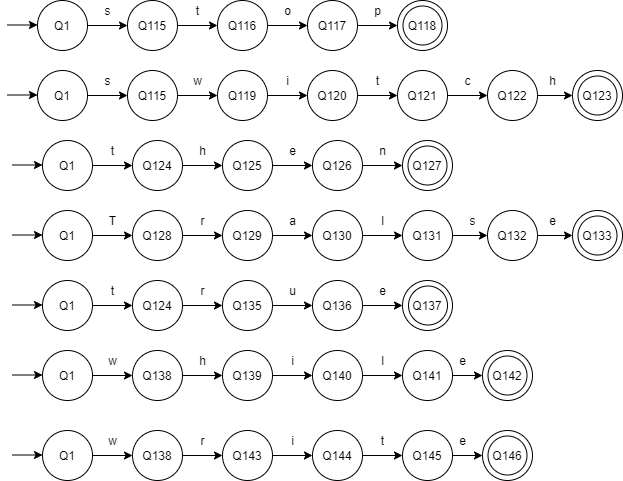
**then**

****

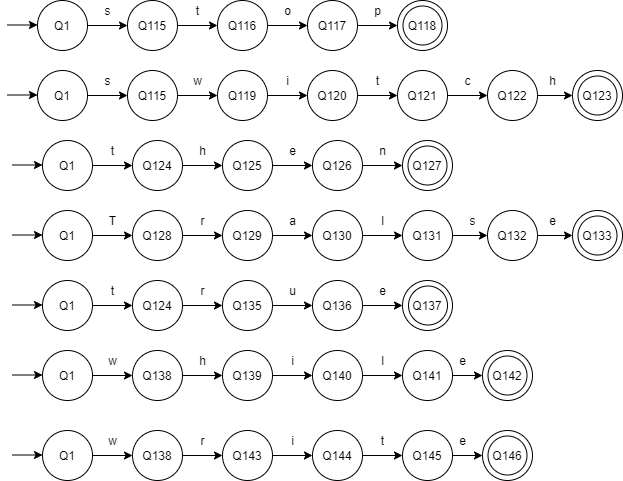
**Tralse**

****

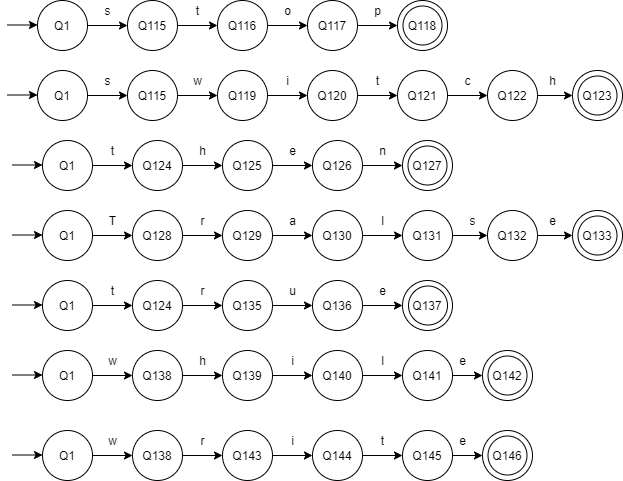
**true**

****

**while**

****

**write**

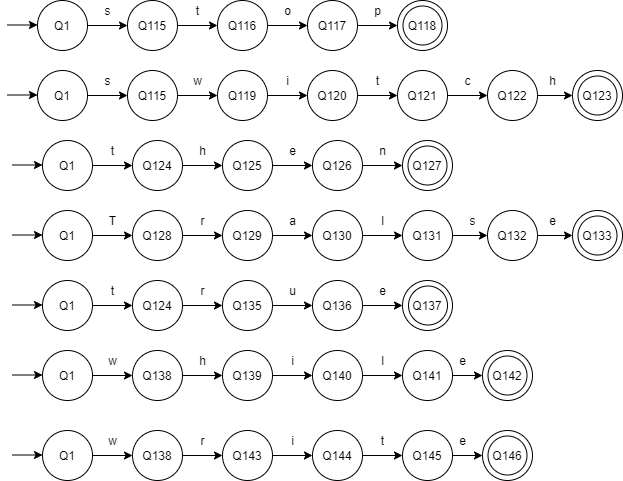
****

1. **Noise Words**

Noise words will help the analyzer to determine a syntax error and improve code readability.

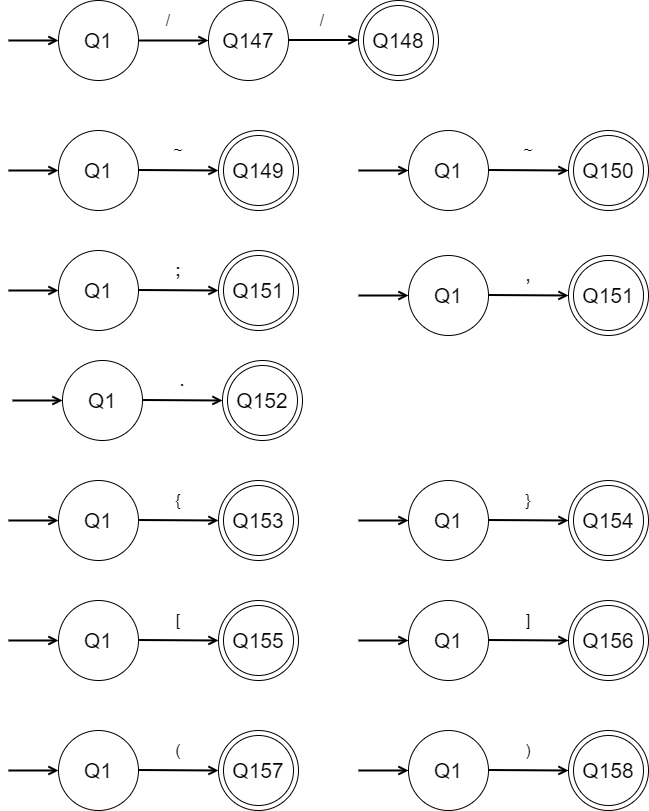
|  |  |  |
| --- | --- | --- |
| **Noise Words** | **Syntax** | **Description** |
| then | if (statement) then … | If statements contain the comparison/conditional statement and the statement itself. The noise word “then” helps distinguish the two parts of the if statement. |

**then**

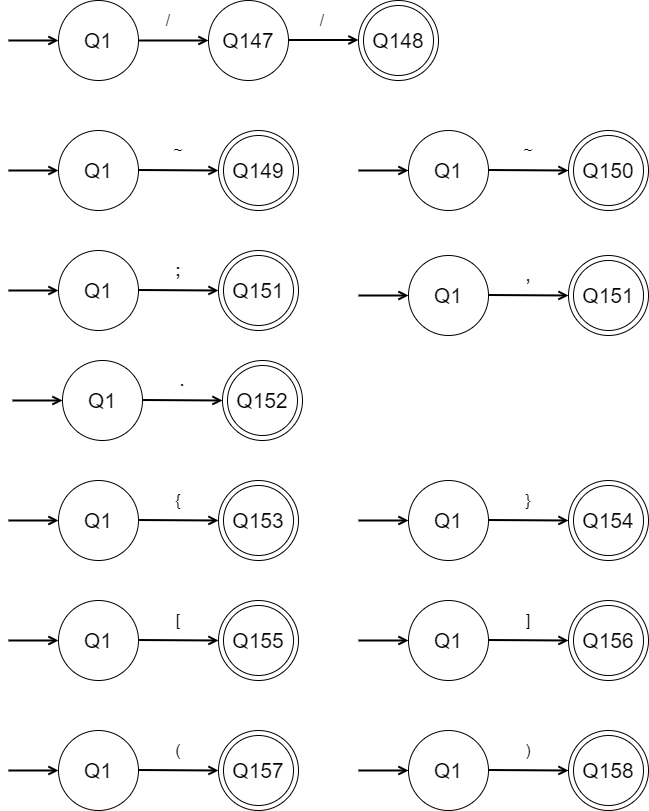
****

1. **Comments**

* Single-line comment: //this is a single-line comment



* Multi-line comment: ~ this is a multi-line comment ~

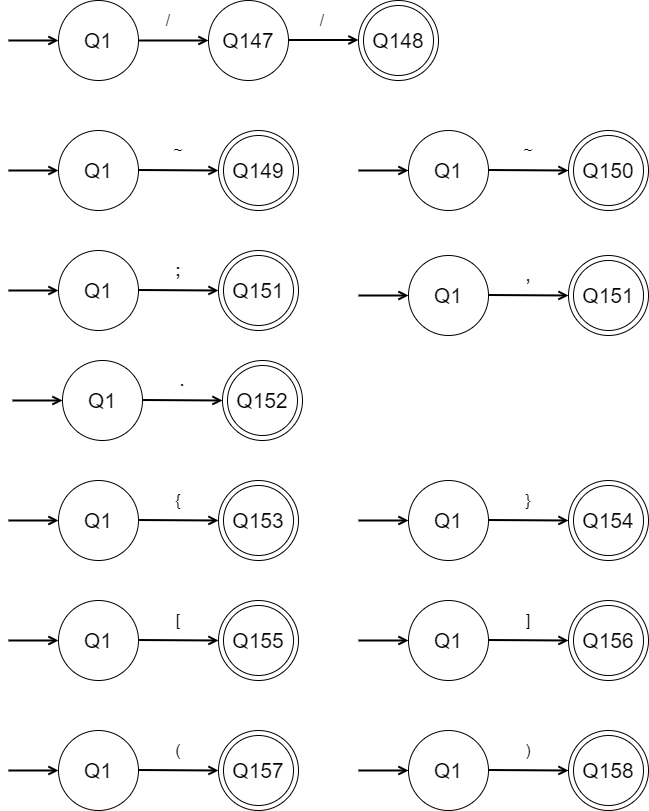


1. **Blanks (spaces)**

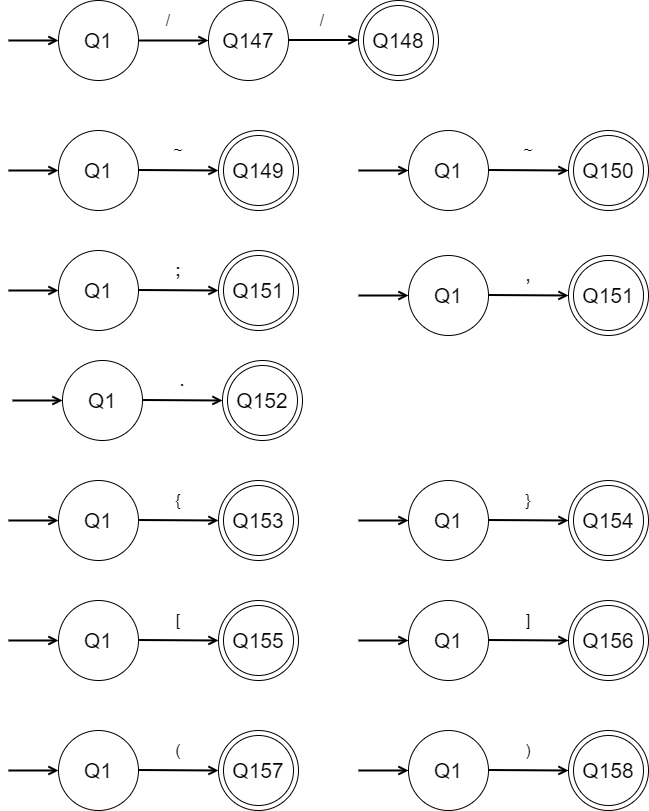
Cmple does not read whitespaces but it is used in separating read lexemes, but it is accepted if it is used inside a Sentence literal.

1. **Delimiters**

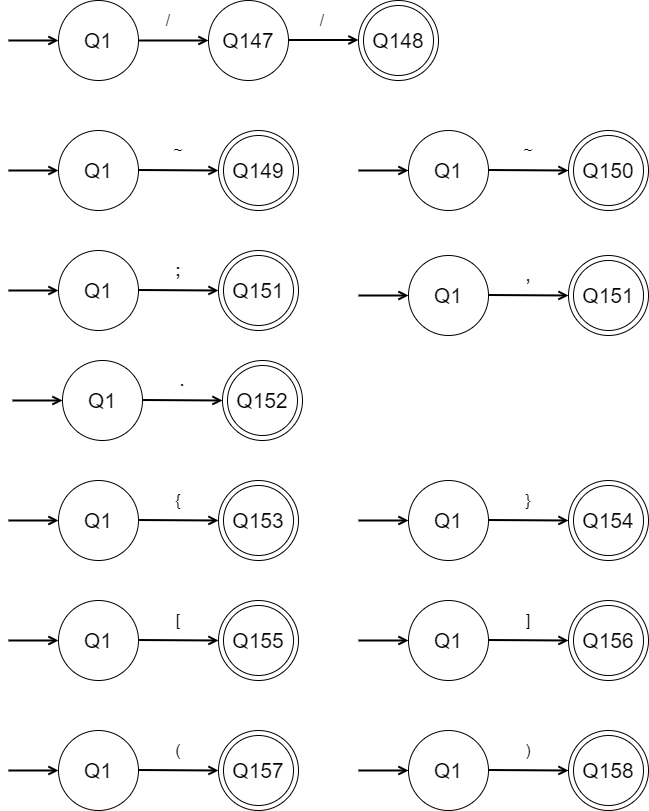
* Semicolon – used to terminate a line of code**.**



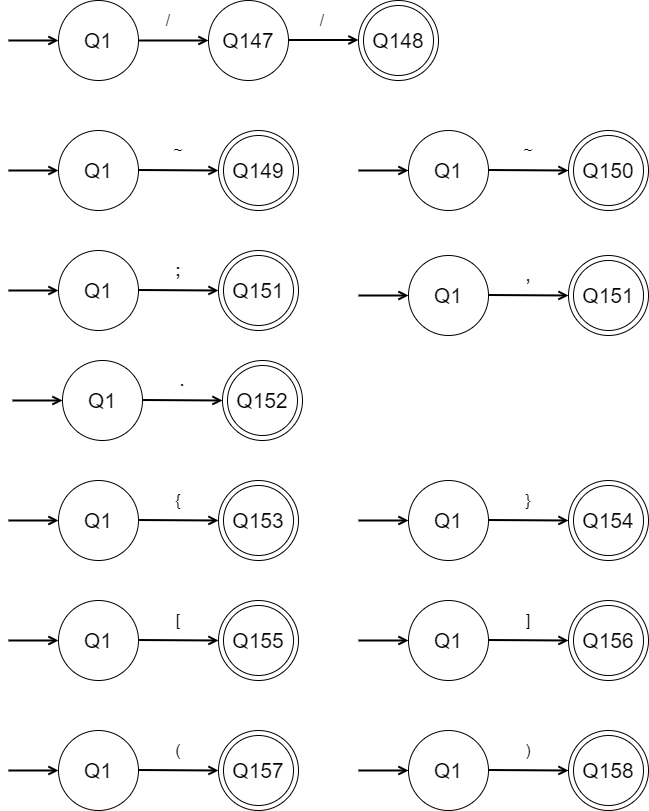
* Comma – used to separate values.



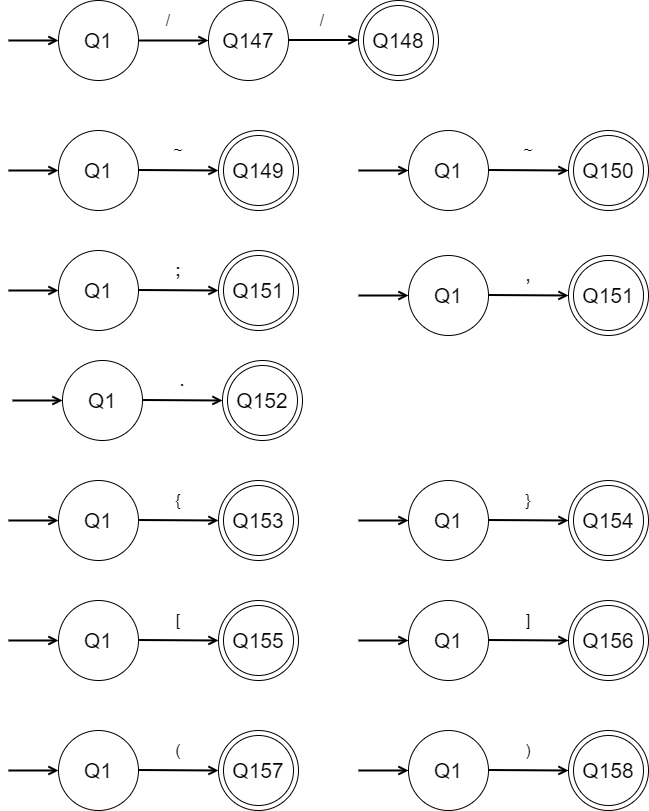
* Period – used as a decimal separator.



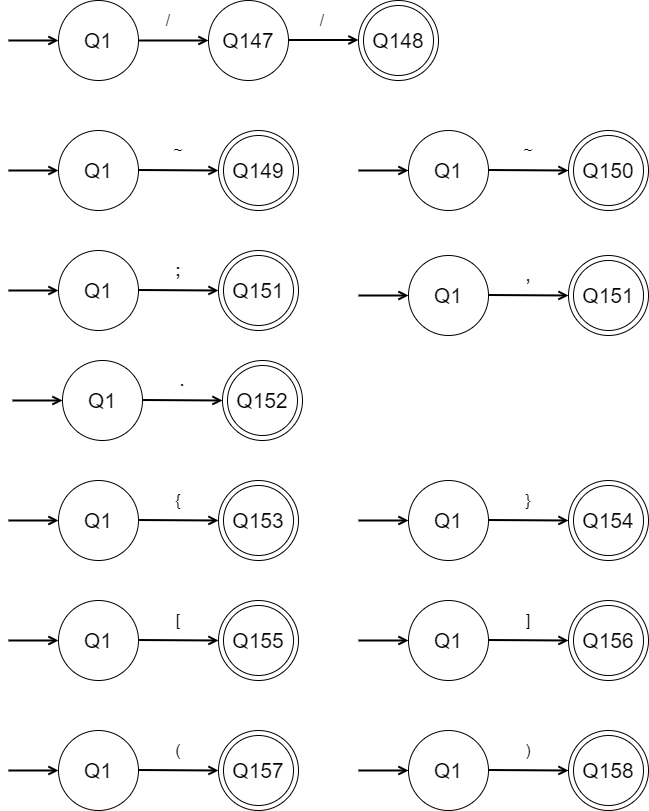
* Curly Brackets – used to enclose statements.



* Square Brackets – used to enclose data types in declaring statements.



* Parentheses – used to group expressions and statements.



1. **Free-and-fixed-fields Formats**

This language is a free format type of language with some statements such as a comment that needs to be in a fixed-field format for it to work.

1. **Expression**

Rules for evaluating expressions:

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Arithmetic** | **Description** | **Associativity** |
| 1 | ++, -- | Postfix Increment/Decrement | Left-to-Right |
| 2 | \*, /, % | Multiplication, Division, Modulus | Left-to-Right |
| 3 | +, - | Addition, Subtraction | Left-to-Right |

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Relational** | **Description** | **Associativity** |
| 1 | >, >=, <, <= | Greater than, Greater than or equal to, Less than, Less than or equal to | Left-to-Right |
| 2 | ==, != | Equal to, Not Equal to | Left-to-Right |

|  |  |  |  |
| --- | --- | --- | --- |
| **Precedence** | **Logical** | **Description** | **Associativity** |
| 1 | ! | Logical NOT | Left-to-Right |
| 2 | && | Logical AND | Left-to-Right |
| 3 | || | Logical OR | Left-to-Right |

1. **Statements**

* Declaration Statement

Data\_Types = {Number, Sentence, Tralse, Collection, Comp, Item}

Gen\_DataTypes = {Number, Sentence, Tralse}

Spec\_DataTypes = {Comp, Item}

Coll\_DataType = {Collection}

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Variations** | **Output** |
| <Gen\_DataTypes> <Identifier>; | Number x;  Sentence intro = “hello!”;  Number value = 89.2; | x is a place holder  hello!  89.2 |
| <Spec\_DataTypes> <Identifiers> {<Declaration\_Statements>} | Comp ID {Sentence name; Number age;}  Item size {“small”, “med”, “large”}; | ID {Qwerty, 20}  size.small = 0 |
| <Coll\_DataType> [<Gen\_DataTypes>]<Identifier>; | Collection [Sentence] words;  Collection [Number] num\_id = {32, 88, 91} ; | words is a place holder for an array of strings  32, 88, 91 |

**Source Code:**

Sentence name;

Number age;

Number x = age + 10;

read(name);

read(age);

write(name + “ will be ” + x + “ in 10 years.”);

**Input:**

Mary

19

**Output:**

Mary will be 29 in 10 years.

* Input Statement

**Syntax:** read(<expression>);

**Source Code:**

write(“Insert two numbers: ”);

Number opr\_1, opr\_2;

read(opr\_1);

read(opr\_2);

Number result = opr\_1 + opr\_2;

write(opr\_1 + “ + ” opr\_2 “ = ” + result);

**Input:**

88

90

**Output:**

88 + 90 = 178

* Output Statement

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Examples** | **Output** |
| write(“<Character\_Set>”); | write(“X”) | X |
| write(<identifier>); | Number qwe = 23;  write(qwe) | 23 |
| write(<identifier> + “<Character\_Set>” + ….); | Number qwe = 23;  write(23 + “ Hello”) | 23 Hello |

* Assignment Statement

Arithmetic\_Operators = {+, -, \*, /, %}

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Examples** | **Output** |
| <identifier> “=” literal; | Sentence Word = “hello”  write(Word); | hello |
| <identifier> “=” <identifier>; | Tralse ans, check = false;  ans = check;  write(ans); | false |
| <identifier> “=” <identifier> <Arithmetic\_Operators> <identifier>; | Number x = 1, y = 2, z;  z = x + y;  write(z); | 3 |
| <identifier> “=” value <Arithmetic\_Operators>  <identifier>; | Number op1 = 72, result;  result = op1 – 10;  write(result); | 62 |

* Conditional Statement
  + if Statement

**Syntax:**

if(conditional statement)

{

//statements to be performed

}

**Source Code:**

Number value;

read(value);

if(value > 10)

{

write(“Invalid Input!”);

}

**Input:**

98

**Output:**

Invalid Input!

* + if Else Statement

**Syntax:**

if(conditional statement)  
{  
// statements   
}  
else  
{  
// statements  
}

**Source Code:**

Number x = 9, y = 3;

if(x == y)  
{  
write(“both numbers are not equal”);  
}  
else  
{  
Number res = x – y;

write(res);  
}

**Output:**

6

* + If Otherwise Else Statement

**Syntax:**

if(conditional statement)  
{  
// statements   
}  
otherwise(conditional statement)

{

//statements

}  
else  
{  
// statements  
}

**Source Code:**

Sentence keyword;

Tralse check;

if(keyword == “pi”)  
{  
write(pi);  
}  
otherwise(keyword == “kelvin”)

{

write(kelvin);

}  
else  
{  
write(“not a keyword”);  
}

**Input:**

kelvin

**Output:**

273

* + Switch Statement

**Syntax:**

switch(identifier) {

case literal:

//statements

stop;

case literal:

//statements

stop;

case literal:

//statements

stop;

...

}

**Input:**

x = 0

**Source Code:**

Number password;

read(password);

switch(password) {

case 0012:

write(“Log in Successful!”);

default:

write(“Incorrect Password!”);

stop;

}

**Input:**

0012

**Output:**

Log in Successful!

* Iteration Statement
  + For Statement

**Syntax:**

for(initialization; conditional statement; iteration)

{

// Statements

}

**Source Code:**

for(Number x = 0; x< 3; x++)

{

write(x+” ”);

}

**Output:**

0 1 2

* + Do While Statement

**Syntax:**

do

{

//Statements

}

while(condition);

**Source Code:**

do

{

write(x);

x++;

}

while(x<2);

**Output:**

0 1

* + While Statement

**Syntax:**

while(condition)

{

//statements

}

**Source Code:**

Number x = 0;

while( x < 1)

{

x++;

write(x);

}

**Output:**

1

**Sample Input:**

//WEEEEEEEE

then

if (two && two)

{}[]()+++---/%\*

<=>=<>

== =

while do if otherwise

stop

else

Alphabet

case

comp

Collection

default

else

euler

false

for

Item

if

int32

kelvin

None

Number

resume

odd

otherwise

pi

read

resume

return

Sentence

stop

switch

then

true

Tralse

while

write

two

"qweqwe"

qwe23\_eqw

qweqwewq//qweqweqweqw

qwe~ewqwe

||

&&

!

!=

~Apparently this is

true LMAO ~

Number x = 23.423;

if ( x >= 23 )

{

x = 40;

}

otherwise ( x <= 12)

{

x = 0;

}

else

{

x = 4;

}

for (Number i = 0; i <= 23; i++)

{

write(i);

}