Definition of the mini language:

Reserved Words (Keywords)

These are special words that have predefined meanings and cannot be used as identifiers:

- int – keyword for integer data type
- float – keyword for float data type
- struct – keyword for defining user-defined types (structures)
- if – keyword for conditional statements
- else – keyword for alternative branch in conditionals
- while – keyword for loops
- input – keyword for reading input
- output – keyword for printing output

Operators

The mini language supports the following operators:

- **Arithmetic Operators**:
  - + – addition
  - - – subtraction
  - * – multiplication
  - / – division
  - = – assignment operator
- **Relational Operators**:
  - == – equal to
  - != – not equal to
  - < – less than
  - > – greater than
  - <= – less than or equal to
  - >= – greater than or equal to
- **Logical Operators**:
  - && – logical AND

    - || – logical OR
    - ! – logical NOT

Separators (Delimiters)

The mini language uses the following separators:

- ; – to terminate statements
- , – to separate parameters in function or variable declarations

Identifiers

Identifiers are names used for variables, functions, and structures. Identifiers must follow these rules:

- Begin with a letter (uppercase or lowercase) or an underscore (_)
- Followed by letters, digits, or underscores
- Identifiers are case-sensitive
- Examples: x, var1, count_, _myVar

Constants

The mini language supports:

- **Integer Constants**:
    - A sequence of digits (e.g., 0, 123, 4567)
- **Float Constants**:
    - A sequence of digits with a decimal point (e.g., 3.14, 0.001, 2.0)

BNF Syntax of Mini Laguage

1. Program Structure

```
<program> ::= <declaration_list> <statement_list>

<declaration_list> ::= <declaration> | <declaration> <declaration_list>

<statement_list> ::= <statement> | <statement> <statement_list>
```

2. Declarations

```
<declaration> ::= <type> <identifier> ";"

            | "struct" <identifier> "{" <declaration_list> "}" ";"

<type> ::= "int" | "float"


<struct_declaration> ::= "struct" <identifier> "{" <variable_declarations> "}"


<variable_declarations> ::= <variable_declaration> { <variable_declaration> }


<variable_declaration> ::= <type> <identifier> ";"


<type> ::= "int"

      | "float"

      | <identifier>   // For user-defined types or struct types
```

3.Statemens

```
<statement> ::= <assignment_statement>

          | <conditional_statement>

          | <loop_statement>

          | <input_statement>
          | <output_statement>

<assignment_statement> ::= <identifier> "=" <expression> ";"

<conditional_statement> ::= "if" "(" <expression> ")" "{" <statement_list> "}"
                    | "if" "(" <expression> ")" "{" <statement_list> "}" "else" "{" <statement_list>
"}"

<loop_statement> ::= "while" "(" <expression> ")" "{" <statement_list> "}"

<input_statement> ::= "cin>>" "(" <identifier> ")" ";"

<output_statement> ::= "cout<<" "(" <expression> ")" ";"
```

4.Expressions

```
<expression> ::= <simple_expression>
          | <simple_expression> <relational_operator> <simple_expression>

<simple_expression> ::= <term>
               | <simple_expression> <add_operator> <term>

<term> ::= <factor>
      | <term> <mul_operator> <factor>

<factor> ::= <identifier>
       | <constant>
       | "(" <expression> ")"
```

<relational_operator> ::= "==" | "!=" | "<" | ">" | "<=" | ">="

<add_operator> ::= "+" | "-"

<mul_operator> ::= "*" | "/"

5. Identifiers and Constants

<identifier> ::= <letter> { <letter> | <digit> | "_" }

<constant> ::= <integer_constant> | <float_constant>

<integer_constant> ::= <digit> { <digit> }

<float_constant> ::= <digit> { <digit> } "." <digit> { <digit> }

<letter> ::= "a" | "b" | "c" | ... | "z" | "A" | "B" | ... | "Z"

<digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

```
int k;

int count;
int num;
int i;
int isPrime;

cin>>k;     // Get the value of k from the user
count = 0;
num = 2;

while (count < =k) {
  isPrime = 1;

  i = 2;
  while (i < num) {
    if (num % i == 0) {
      isPrime = 0;
    }
    i = i + 1;
  }

  if (isPrime == 1) {
    cout<<num;   // Print the prime number
    count = count + 1;
  }

  num = num + 1;     }
```