Project for CPSC323

The programming assignments are based on a language called "Rat18F" which is described as
follows. The Rat18F language is designed to be an easy to understand. It has a short grammar and relatively
clean semantics.
You will need to form a group of 2 or 3 for the duration of the entire project (i.e., all three assignments).

RAT18F

1) Lexical Conventions:
The lexical units of a program are identifiers, keywords, integers, reals, operators and other
separators. Blanks, tabs and newlines (collectively, "white space") as described below
are ignored except as they serve to separate tokens.
Some white space is required to separate otherwise adjacent identifiers, keywords, reals and integers.
<Identifier> is a sequence of letters or digits however, the first character must be a letter and last char must be a
letter. Upper and lower cases are same.
<Integer> is an unsigned decimal integer i.e., a sequence of decimal digits.
<Real> is integer followed by "." and Integer, e.g., 123.00
Some identifiers are reserved for use as *keywords*, and may not be used otherwise:
> **e.g., int, if, else, ifend, while, return, get, put   etc.**

 Comments are enclosed in   **[*   *]**

2) Syntax rules : The following BNF describes the Rat18F.

R1. <Rat18F> ::= <Opt Function Definitions> \$\$ <Opt Declaration List> <Statement List> \$\$
R2. <Opt Function Definitions> ::= <Function Definitions> | <Empty>
R3. <Function Definitions> ::= <Function> | <Function> <Function Definitions>
R4. <Function> ::= function <Identifier>   (<Opt Parameter List>) <Opt Declaration List> <Body>
R5. <Opt Parameter List> ::= <Parameter List> | <Empty>
R6. <Parameter List> ::= <Parameter> | <Parameter> , <Parameter List>
R7. <Parameter> ::= <IDs > : <Qualifier>
R8. <Qualifier> ::= int | boolean | real
R9. <Body> ::= { < Statement List> }
R10. <Opt Declaration List> ::= <Declaration List> | <Empty>
R11. <Declaration List> := <Declaration> ; | <Declaration> ; <Declaration List>
R12. <Declaration> ::=   <Qualifier > <IDs>
R13. <IDs> ::=   <Identifier> | <Identifier>, <IDs>
R14. <Statement List> ::=   <Statement> | <Statement> <Statement List>
R15. <Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>
R16. <Compound> ::=   { <Statement List> }
R17. <Assign> ::=   <Identifier> = <Expression> ;
R18. <If> ::=   if (<Condition>) <Statement> ifend |
                   if (<Condition>) <Statement> else <Statement> ifend
R19. <Return> ::= return ; | return <Expression> ;
R20. <Print> ::=   put (<Expression>);
R21. <Scan> ::=   get (<IDs>);
R22. <While> ::= while (<Condition>) <Statement> whileend // should be <StatementList>?
R23. <Condition> ::=   <Expression> <Relop> <Expression>

R24. <Relop> ::=   == | ^= | > | < | => | =<
R25. <Expression> ::=   <Expression> + <Term> | <Expression> - <Term> | <Term>
R26. <Term>   ::=   <Term> * <Factor> | <Term> / <Factor> | <Factor>
R27. <Factor> ::=   - <Primary> | <Primary>
R28. <Primary> ::=   <Identifier> | <Integer> | <Identifier> (<IDs>) | (<Expression>) |
                  <Real> | true | false

R29. <Empty> ::= □

3) Some Semantics

- Rat18F is a conventional imperative programming language. A Rat18F program consists of a sequence of functions followed by the "main body" where the program executes.
- **All variables and functions must be declared before use.**
- Function arguments are passed by value.
- There is an implied expressionless return at the end of all functions; the value returned by expressionless return statement is undefined.
- Arithmetic expressions have their conventional meanings.
- Integer division ignores any remainder.
- Type casting is not allowed (e.g., assigning an integer to a real variable)
- No arithmetic operations are allowed with booleans (e.g., true + false)
- Others as we will define during the semester

4) A sample Rat18F Program

```
[* this is comment for this sample code which
   converts  Fahrenheit into Celcius *]

function convert1x (fahr : int)
{
      return 5 * (fahr -32) / 9;
}

$$
   int   low,  high, step;     [* declarations *]

   get (low, high, step);
   while (low  <  high )
     {  put (low);
        put (convert1x  (low));
        low =  low  +  step;
     }
    whileend

 $$
```