tests

Justin Chin

May 1, 2018

Test1

```
! this is comment for this sample code which converts  Fahrenheit into Celcius !

function convert$ [fahr:int]
{
  return 5 * (fahr -32) / 9;
}

%%
int    low,  high, step$;      ! declarations !

put (9.0);
get (low, high, step$);
while (low  <  high )

{  put (low);
  put (convert$ (low));
  low =  low  +  step$;
}
```

Results

<Rat18S> ::= <OptFunctionDefinitions> %% <OptDeclarationlist> <StatementList>
<OptFunctionDefinitions ::= <FunctionDefinitions> | <Empty>
<FunctionDefitions> ::= <Function> <FDPrime>
Token: Keyword Lexeme: "function"
<Function> ::= function <Identifier> [<OptParameterList>] <OptDeclarationList> <Body>
Token: Identifier Lexeme: "convert$"
Token: LBracket Lexeme: "["
<OptParameterList> ::= <ParamaterList> | <Empty>
<ParamaterList> ::= <Parameter> <ParameterListPrime>
<Parameter> ::= <IDs> : <Qualifier>
Token: Identifier Lexeme: "fahr"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Colon Lexeme: ":"
Token: Keyword Lexeme: "int"

<Qualifier> ::= int
<ParameterListPrime> ::= <Empty>
Token: RBracket Lexeme: "]"
<OptDeclarationList> ::= <Empty>
Token: LBrace Lexeme: "{"
<Body> ::= { <StatementList> }
<StatementList> ::= <Statement> <StatementListPrime>
<Statement> ::= <Return>
Token: Keyword Lexeme: "return"
<Return> ::= return <ReturnPrime>
<Expression> ::= <Term> <ExpressionPrime>
<Term> ::= <Factor> <TermPrime>
<Factor> ::= - <Primary> | <Primary>
Token: Int Lexeme: "5"
<Primary> ::= <Integer>
Token: Times Lexeme: "*"
<TermPrime> ::= * <Factor> <TermPrime>
<Primary> ::= (<Expression>)
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "fahr"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
Token: Minus Lexeme: "-"
<ExpressionPrime> ::= - <Term> <ExpressionPrime>
Token: Int Lexeme: "32"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Div Lexeme: "/"
<TermPrime> ::= / <Factor> <TermPrime>
Token: Int Lexeme: "9"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
<ReturnPrime> ::= <Expression> ;
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<FDPrime> ::= <Empty>
Token: EndOfDefs Lexeme: "%%"
<OptDeclarationList> ::= <DeclarationList>
<DeclarationList> ::= <Declaration> <DeclarationListPrime>
<Declaration> ::= <Qualifier> <IDs>
Token: Keyword Lexeme: "int"
<Qualifier> ::= int
Token: Identifier Lexeme: "low"
<IDs> :: = <Identifier> <IDsPrime>

Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "high"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "step$"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"
<DeclarationListPrime> ::= <Empty>
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
<Print> ::= put (<Expression>) ;
Token: LParen Lexeme: "("
Token: Real Lexeme: "9.0"
<Primary> ::= <Real>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Scan>
Token: Keyword Lexeme: "get"
<Scan> ::= get (<IDs>) ;
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "low"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "high"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "step$"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <While>
Token: Keyword Lexeme: "while"
<While> ::= while (<Condition>) <Statement>
Token: LParen Lexeme: "("
<Condition> ::= <Expression> <Relop> <Expression>
Token: Identifier Lexeme: "low"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>

<ExpressionPrime> ::= <Empty>
Token: Less Lexeme: "<"
<Relop> ::= <
Token: Identifier Lexeme: "high"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<Statement> ::= <Compound>
Token: LBrace Lexeme: "{"
<Compound> ::= { <StatementList> }
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "low"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
Token: LParen Lexeme: "("
<Primary> ::= <Identifier> (<IDs>)
Token: Identifier Lexeme: "convert$"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "low"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Assign>
Token: Identifier Lexeme: "low"
<Assign> ::= <Identifier> = <Expression> ;
Token: Assign Lexeme: "="
Token: Identifier Lexeme: "low"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
Token: Plus Lexeme: "+"
<ExpressionPrime> ::= + <Term> <ExpressionPrime>
Token: Identifier Lexeme: "step$"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>

<ExpressionPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<StatementListPrime> ::= <Empty>

Test2

```
  ! squares and returns the argument !

function square [x:int] {
         return x * x;
}

function timesFiveMinusTwo$ [x:int] {
         return 5 * x - 2;
}

%%

int x, y, z, x2$, y2$;

get (x, y, z);


put (timesFiveMinusTwo$ (x));
while (y < z)
      { put (y);
        put (square (y));
        y = y + 1;
        }

x2$ = square(x);
y2$ = square(y);

if (x2$ > y2$)

   return true;

endif
```

Results

<OptFunctionDefinitions ::= <FunctionDefinitions> | <Empty>
<FunctionDefitions> ::= <Function> <FDPrime>
Token: Keyword Lexeme: "function"
<Function> ::= function <Identifier> [<OptParameterList>] <OptDeclarationList> <Body>

Token: Identifier Lexeme: "square"
Token: LBracket Lexeme: "["
<OptParameterList> ::= <ParamaterList> | <Empty>
<ParamaterList> ::= <Parameter> <ParameterListPrime>
<Parameter> ::= <IDs> : <Qualifier>
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Colon Lexeme: ":"
Token: Keyword Lexeme: "int"
<Qualifier> ::= int
<ParameterListPrime> ::= <Empty>
Token: RBracket Lexeme: "]"
<OptDeclarationList> ::= <Empty>
Token: LBrace Lexeme: "{"
<Body> ::= { <StatementList> }
<StatementList> ::= <Statement> <StatementListPrime>
<Statement> ::= <Return>
Token: Keyword Lexeme: "return"
<Return> ::= return <ReturnPrime>
<Expression> ::= <Term> <ExpressionPrime>
<Term> ::= <Factor> <TermPrime>
<Factor> ::= - <Primary> | <Primary>
Token: Identifier Lexeme: "x"
<Primary> ::= <Identifier>
Token: Times Lexeme: "*"
<TermPrime> ::= * <Factor> <TermPrime>
Token: Identifier Lexeme: "x"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
<ReturnPrime> ::= <Expression> ;
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<FDPrime> ::= <FunctionDefinitions>
Token: Keyword Lexeme: "function"
Token: Identifier Lexeme: "timesFiveMinusTwo$"
Token: LBracket Lexeme: "["
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Colon Lexeme: ":"
Token: Keyword Lexeme: "int"
<Qualifier> ::= int
<ParameterListPrime> ::= <Empty>
Token: RBracket Lexeme: "]"
<OptDeclarationList> ::= <Empty>

Token: LBrace Lexeme: "{"
<Body> ::= { <StatementList> }
<Statement> ::= <Return>
Token: Keyword Lexeme: "return"
Token: Int Lexeme: "5"
<Primary> ::= <Integer>
Token: Times Lexeme: "*"
<TermPrime> ::= * <Factor> <TermPrime>
Token: Identifier Lexeme: "x"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
Token: Minus Lexeme: "-"
<ExpressionPrime> ::= - <Term> <ExpressionPrime>
Token: Int Lexeme: "2"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
<ReturnPrime> ::= <Expression> ;
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<FDPrime> ::= <Empty>
Token: EndOfDefs Lexeme: "%%"
<OptDeclarationList> ::= <DeclarationList>
<DeclarationList> ::= <Declaration> <DeclarationListPrime>
<Declaration> ::= <Qualifier> <IDs>
Token: Keyword Lexeme: "int"
<Qualifier> ::= int
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "y"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "z"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "x2$"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "y2$"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"

<DeclarationListPrime> ::= <Empty>
<Statement> ::= <Scan>
Token: Keyword Lexeme: "get"
<Scan> ::= get (<IDs>) ;
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "y"
<IDs> :: = <Identifier> <IDsPrime>
Token: Comma Lexeme: ","
<IDsPrime> ::= , <IDs>
Token: Identifier Lexeme: "z"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
<Print> ::= put (<Expression>) ;
Token: LParen Lexeme: "("
<Primary> ::= <Identifier> (<IDs>)
Token: Identifier Lexeme: "timesFiveMinusTwo$"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <While>
Token: Keyword Lexeme: "while"
<While> ::= while (<Condition>) <Statement>
Token: LParen Lexeme: "("
<Condition> ::= <Expression> <Relop> <Expression>
Token: Identifier Lexeme: "y"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: Less Lexeme: "<"
<Relop> ::= <
Token: Identifier Lexeme: "z"
<Primary> ::= <Identifier>

<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<Statement> ::= <Compound>
Token: LBrace Lexeme: "{"
<Compound> ::= { <StatementList> }
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "y"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Print>
Token: Keyword Lexeme: "put"
Token: LParen Lexeme: "("
<Primary> ::= <Identifier> (<IDs>)
Token: Identifier Lexeme: "square"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "y"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Assign>
Token: Identifier Lexeme: "y"
<Assign> ::= <Identifier> = <Expression> ;
Token: Assign Lexeme: "="
Token: Identifier Lexeme: "y"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
Token: Plus Lexeme: "+"
<ExpressionPrime> ::= + <Term> <ExpressionPrime>
Token: Int Lexeme: "1"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<StatementListPrime> ::= <StatementList>

<Statement> ::= <Assign>
Token: Identifier Lexeme: "x2$"
<Assign> ::= <Identifier> = <Expression> ;
Token: Assign Lexeme: "="
<Primary> ::= <Identifier> (<IDs>)
Token: Identifier Lexeme: "square"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "x"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <Assign>
Token: Identifier Lexeme: "y2$"
<Assign> ::= <Identifier> = <Expression> ;
Token: Assign Lexeme: "="
<Primary> ::= <Identifier> (<IDs>)
Token: Identifier Lexeme: "square"
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "y"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <StatementList>
<Statement> ::= <If>
Token: Keyword Lexeme: "if"
<If> ::= if (<Condition>) <Statement> endif | if (<Condition>) else <Statement> endif
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "x2$"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: Greater Lexeme: ">"
<Relop> ::= >
Token: Identifier Lexeme: "y2$"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
<Statement> ::= <Return>
Token: Keyword Lexeme: "return"
Token: Keyword Lexeme: "true"

<Primary> ::= true
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
<ReturnPrime> ::= <Expression> ;
Token: Semicolon Lexeme: ";"
Token: Keyword Lexeme: "endif"
<StatementListPrime> ::= <Empty>

Test3

```
  function convert$ [fahr:int]

{

        return 5*(fahr -32)/9;

}

int low, high step$ !declarations!


     put (9.0);
     get (low, high, step$);
     while (low  <  high )
         {  put (low);
             put (convert$ (low));
             low =  low  +  step$;
         }
```

Results

<Rat18S> ::= <OptFunctionDefinitions> %% <OptDeclarationlist> <StatementList>
<OptFunctionDefinitions ::= <FunctionDefinitions> | <Empty>
<FunctionDefitions> ::= <Function> <FDPrime>
Token: Keyword Lexeme: "function"
<Function> ::= function <Identifier> [<OptParameterList>] <OptDeclarationList> <Body>
Token: Identifier Lexeme: "convert$"
Token: LBracket Lexeme: "["
<OptParameterList> ::= <ParamaterList> | <Empty>
<ParamaterList> ::= <Parameter> <ParameterListPrime>
<Parameter> ::= <IDs> : <Qualifier>
Token: Identifier Lexeme: "fahr"
<IDs> :: = <Identifier> <IDsPrime>
<IDsPrime> ::= <Empty>
Token: Colon Lexeme: ":"
Token: Keyword Lexeme: "int"
<Qualifier> ::= int

<ParameterListPrime> ::= <Empty>
Token: RBracket Lexeme: "]"
<OptDeclarationList> ::= <Empty>
Token: LBrace Lexeme: "{"
<Body> ::= { <StatementList> }
<StatementList> ::= <Statement> <StatementListPrime>
<Statement> ::= <Return>
Token: Keyword Lexeme: "return"
<Return> ::= return <ReturnPrime>
<Expression> ::= <Term> <ExpressionPrime>
<Term> ::= <Factor> <TermPrime>
<Factor> ::= - <Primary> | <Primary>
Token: Int Lexeme: "5"
<Primary> ::= <Integer>
Token: Times Lexeme: "*"
<TermPrime> ::= * <Factor> <TermPrime>
<Primary> ::= (<Expression>)
Token: LParen Lexeme: "("
Token: Identifier Lexeme: "fahr"
<Primary> ::= <Identifier>
<TermPrime> ::= <Empty>
Token: Minus Lexeme: "-"
<ExpressionPrime> ::= - <Term> <ExpressionPrime>
Token: Int Lexeme: "32"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
Token: RParen Lexeme: ")"
Token: Div Lexeme: "/"
<TermPrime> ::= / <Factor> <TermPrime>
Token: Int Lexeme: "9"
<Primary> ::= <Integer>
<TermPrime> ::= <Empty>
<ExpressionPrime> ::= <Empty>
<ReturnPrime> ::= <Expression> ;
Token: Semicolon Lexeme: ";"
<StatementListPrime> ::= <Empty>
Token: RBrace Lexeme: "}"
<FDPrime> ::= <Empty>
    line 9
Error at int: Expecting '%%' after function definitions.
main: Maybe.fromJust: Nothing