Bohan Chen
Yarrick Dillard
William Vipperman

Example Inputs / Command Lines / Outputs

Source file.	Command.	Output.

`func fac(n)`
`[`
`  write n;`
`  if n < 2 [`
`  return 1;`
`  ]`
`  else`
`  [`
`  return n * fac(n-1);`
`  ]`
`]`
 	
`func main()`
`[`
`  n = fac(3);`
`  write n;`
`]`
	`execute example-1.txt`	`Execute given file.`
		`Running parser to tokenize & parse a file example-1.txt:`
		`parse_program()`
		`  parse_func()`
		`  expect(FUNC)`
		`  expect(ID(""))`
		`  parse_params()`
		`  expect(PAREN_L)`
		`  expect(ID(""))`
		
		`  expect(PAREN_R)`
		`  parse_block()`
		
		`  expect(BRACE_L)`
		`  parse_stmt()`
		`  parse_write_stmt()`
		
		`  expect(SEMICOLON)`
		`  parse_stmt()`
		`  parse_if_stmt()`
		`  expect(ID)`
		`  parse_block()`
		
		`  expect(BRACE_L)`
		`  parse_stmt()`
		`  parse_return_stmt()`
		`  expect(RET)`
		`RET`
		`  VAL(I64(1))`
		
		`  expect(SEMICOLON)`
		
		`  expect(BRACE_R)`
		
		`  parse_block()`
		
		`  expect(BRACE_L)`
		`  parse_stmt()`
		`  parse_return_stmt()`
		`  expect(RET)`
		`  expect(ID(""))`
		`  expect(PAREN_L)`
		`  expect(PAREN_R)`
		`RET`
		`  MULT`
		`  ID("n")`
		`  CALL`
		`  ID("fac")`
		`  SUB`
		`  ID("n")`
		`  VAL(I64(1))`
		
		`  expect(SEMICOLON)`
		
		`  expect(BRACE_R)`
		
		`  expect(BRACE_R)`
		`  parse_func()`
		`  expect(FUNC)`
		`  expect(ID(""))`
		`  parse_params()`
		`  expect(PAREN_L)`
		
		`  parse_block()`
		
		`  expect(BRACE_L)`
		`  parse_stmt()`
		`  parse_assign_stmt()`
		`  expect(ID(""))`
		`  expect(:=)`
		`  expect(ID(""))`
		`  expect(PAREN_L)`
		`  expect(PAREN_R)`
		
		`  expect(SEMICOLON)`
		`  parse_stmt()`
		`  parse_write_stmt()`
		
		`  expect(SEMICOLON)`
		`  expect(BRACE_R)`
		`expect(EOI)`
		`----------PARSER----------`
		
		`MTree:`
		`BLOCK`
		`  FUNC`
		`  ID("fac")`
		`  PARAMS`
		`  ID("n")`
		`  BLOCK`
		`  WRITE`
		`  ID("n")`
		`  ID`
		`  LT`
		`  ID("n")`
		`  VAL(I64(2))`
		`  BLOCK`

```
          RET
            VAL(I64(1))
        BLOCK
          RET
            MULT
              ID("n")
              CALL
                ID("fac")
                SUB
                  ID("n")
                  VAL(I64(1))
  FUNC
    ID("main")
    PARAMS
    BLOCK
      :=
        ID("n")
        CALL
          ID("fac")
          VAL(I64(5))
      WRITE
        ID("n")
  CALL
    ID("main")
----------------------ANALYZER--------------------------------------
BLOCK
  FUNC
    ID("fac")
    PARAMS
      ID("n")
    BLOCK
      WRITE
        ID("n")
      ID
        LT
          ID("n")
          VAL(I64(2))
        BLOCK
          RET
            VAL(I64(1))
        BLOCK
          RET
            MULT
              ID("n")
              CALL
                ID("fac")
                SUB
                  ID("n")
                  VAL(I64(1))
  FUNC
    ID("main")
    PARAMS
    BLOCK
      :=
        ID("n")
        CALL
          ID("fac")
          VAL(I64(5))
      WRITE
        ID("n")
  CALL
    ID("main")
analyze_func() 'fac'
BLOCK
  WRITE
    ID("n")
  ID
    LT
      ID("n")
      VAL(I64(2))
    BLOCK
      RET
        VAL(I64(1))
    BLOCK
      RET
        MULT
          ID("n")
          CALL
            ID("fac")
            SUB
              ID("n")
              VAL(I64(1))
WRITE
  ID("n")
BLOCK
  RET
    VAL(I64(1))
BLOCK
  RET
    MULT
      ID("n")
      CALL
        ID("fac")
        SUB
          ID("n")
          VAL(I64(1))
analyze_func() 'main'
BLOCK
  :=
    ID("n")
    CALL
      ID("fac")
      VAL(I64(5))
  WRITE
    ID("n")
WRITE
  ID("n")
```

		MTree (Analyzed) 'global':
		
		A_BLOCK
		| SYMBOL PROGRAM #1 "main"
		| SYMBOL PROGRAM #0 "fac"
		  FUNC
		  A_BLOCK
		| SYMBOL FUNCTION #0 "n"
		  WRITE
		  REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
		  ID
		  LT
		  REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
		  VAL(I64(2))
		  A_BLOCK
		  VAL(I64(1))
		  A_BLOCK
		  MULT
		  REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
		  CALL
		  REF CellLoc { typ: PROGRAM, idx_frame: 2, idx_cell: 0 }
		  SUB
		  REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
		  VAL(I64(1))
		  FUNC
		  A_BLOCK
		| SYMBOL FUNCTION #0 "n"
		  :=
		  REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
		  CALL
		  REF CellLoc { typ: PROGRAM, idx_frame: 1, idx_cell: 0 }
		  VAL(I64(5))
		  WRITE
		  REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
		  CALL
		  REF CellLoc { typ: PROGRAM, idx_frame: 0, idx_cell: 1 }
		----------EVALUATE----------
		
		EVALUATE MTree (Analyzed) 'global' :
		
		> 5
		> 4
		> 3
		> 2
		> 1
		> 120

func main()
[
  let n = 0;
 	
  while n < 10
  [
  n = n + 1;
  ]
 	
  write n;
]
	execute example-2.txt	Execute given file.
		Running parser to tokenize & parse a file example-2.txt:
		parse_program()
		  parse_func()
		  expect(FUNC)
		  expect(ID(""))
		  parse_params()
		  expect(PAREN_L)
		
		  parse_block()
		
		  expect(BRACE_L)
		  parse_stmt()
		  parse_assign_stmt()
		  expect(ID(""))
		  expect(:=)
		
		  expect(SEMICOLON)
		  parse_stmt()
		  parse_while_stmt()
		  expect(WHILE)
		
		  parse_block()
		
		  expect(BRACE_L)
		  parse_stmt()
		  parse_assign_stmt()
		  expect(ID(""))
		  expect(:=)
		
		  expect(SEMICOLON)
		
		  expect(BRACE_R)
		  parse_stmt()
		  parse_write_stmt()
		
		  expect(SEMICOLON)
		  expect(BRACE_R)
		expect(EOI)
		----------PARSER----------
		
		MTree:
		BLOCK
		  FUNC
		  ID("main")
		  PARAMS
		  BLOCK
		  :=
		  ID("n")
		  VAL(I64(0))
		  WHILE
		  LT
		  ID("n")
		  VAL(I64(10))
		  BLOCK
		  :=
		  ID("n")
		  ADD

		ID("n") VAL(I64(1)) WRITE ID("n") CALL ID("main") ----------ANALYZER---------- BLOCK FUNC ID("main") PARAMS BLOCK := ID("n") VAL(I64(0)) WHILE LT ID("n") VAL(I64(10)) BLOCK := ID("n") ADD ID("n") VAL(I64(1)) WRITE ID("n") CALL ID("main") analyze_func() 'main' BLOCK := ID("n") VAL(I64(0)) WHILE LT ID("n") VAL(I64(10)) BLOCK := ID("n") ADD ID("n") VAL(I64(1)) WRITE ID("n") BLOCK := ID("n") ADD ID("n") VAL(I64(1)) WRITE ID("n") MTree (Analyzed) 'global': A_BLOCK | SYMBOL PROGRAM #0 "main" FUNC A_BLOCK | SYMBOL FUNCTION #0 "n" := REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 } VAL(I64(0)) WHILE LT REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 } VAL(I64(10)) A_BLOCK := REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 } ADD REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 } VAL(I64(1)) WRITE REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 } CALL REF CellLoc { typ: PROGRAM, idx_frame: 0, idx_cell: 0 } ----------EVALUATE---------- EVALUATE MTree (Analyzed) 'global' : > 10

func factorial_recursion(n)
[
if (n < 2)
[
return 1;
]
else
[
return n *
factorial_recursion(n-1);
]
]
 	
func factorial_loop(n)
[
let p;
p = n;
 	
while n > 0
[
n = n - 1;
p = p * n;
]
	execute example-3.txt	Execute given file.
		Running parser to tokenize & parse a file example-3.txt:
		parse_program()
		parse_func()
		expect(FUNC)
		expect(ID(""))
		parse_params()
		expect(PAREN_L)
		expect(ID(""))
		
		expect(PAREN_R)
		parse_block()
		
		expect(BRACE_L)
		parse_stmt()
		parse_if_stmt()
		expect(ID)
		
		expect(PAREN_R)
		parse_block()
		
		expect(BRACE_L)
		parse_stmt()

```
    return p;
]

func main()
[
    let n;
    n = 5;
    write factorial_loop(n);
    write factorial_recursion(n);
]
```

```

```

```
              parse_return_stmt()
                expect(RET)
RET
  VAL(I64(1))

                expect(SEMICOLON)

            expect(BRACE_R)

          parse_block()

            expect(BRACE_L)
            parse_stmt()
              parse_return_stmt()
                expect(RET)
                expect(ID(""))
                expect(PAREN_L)
                expect(PAREN_R)
RET
  MULT
    ID("n")
    CALL
      ID("factorial_recursion")
      SUB
        ID("n")
        VAL(I64(1))

                expect(SEMICOLON)

            expect(BRACE_R)


      expect(BRACE_R)
  parse_func()
    expect(FUNC)
    expect(ID(""))
    parse_params()
      expect(PAREN_L)
      expect(ID(""))

      expect(PAREN_R)
    parse_block()

      expect(BRACE_L)
      parse_stmt()
        parse_assign_stmt()
          expect(ID(""))
          expect(:=)


          expect(SEMICOLON)
      parse_stmt()
        parse_while_stmt()
          expect(WHILE)

          parse_block()

            expect(BRACE_L)
            parse_stmt()
              parse_assign_stmt()
                expect(ID(""))
                expect(:=)

                expect(SEMICOLON)
            parse_stmt()
              parse_assign_stmt()
                expect(ID(""))
                expect(:=)

                expect(SEMICOLON)

            expect(BRACE_R)
      parse_stmt()
        parse_return_stmt()
          expect(RET)
RET
  ID("p")

          expect(SEMICOLON)


      expect(BRACE_R)
  parse_func()
    expect(FUNC)
    expect(ID(""))
    parse_params()
      expect(PAREN_L)

    parse_block()

      expect(BRACE_L)
      parse_stmt()
        parse_assign_stmt()
          expect(ID(""))
          expect(:=)

          expect(SEMICOLON)
      parse_stmt()
        parse_write_stmt()
          expect(ID(""))
          expect(PAREN_L)
          expect(PAREN_R)

          expect(SEMICOLON)
      parse_stmt()
        parse_write_stmt()
          expect(ID(""))
```

```
          expect(PAREN_L)
          expect(PAREN_R)

          expect(SEMICOLON)
      expect(BRACE_R)
expect(EOI)
-----------------------PARSER---------------------------------------

MTree:
BLOCK
  FUNC
    ID("factorial_recursion")
    PARAMS
      ID("n")
    BLOCK
      ID
        LT
          ID("n")
          VAL(I64(2))
        BLOCK
          RET
            VAL(I64(1))
        BLOCK
          RET
            MULT
              ID("n")
              CALL
                ID("factorial_recursion")
                SUB
                  ID("n")
                  VAL(I64(1))
  FUNC
    ID("factorial_loop")
    PARAMS
      ID("n")
    BLOCK
      :=
        ID("p")
        ID("n")
      WHILE
        GT
          ID("n")
          VAL(I64(0))
        BLOCK
          :=
            ID("n")
            SUB
              ID("n")
              VAL(I64(1))
          :=
            ID("p")
            MULT
              ID("p")
              ID("n")
      RET
        ID("p")
  FUNC
    ID("main")
    PARAMS
    BLOCK
      :=
        ID("n")
        VAL(I64(5))
      WRITE
        CALL
          ID("factorial_loop")
          ID("n")
      WRITE
        CALL
          ID("factorial_recursion")
          ID("n")
  CALL
    ID("main")
-----------------------ANALYZER-------------------------------------
BLOCK
  FUNC
    ID("factorial_recursion")
    PARAMS
      ID("n")
    BLOCK
      ID
        LT
          ID("n")
          VAL(I64(2))
        BLOCK
          RET
            VAL(I64(1))
        BLOCK
          RET
            MULT
              ID("n")
              CALL
                ID("factorial_recursion")
                SUB
                  ID("n")
                  VAL(I64(1))
  FUNC
    ID("factorial_loop")
    PARAMS
      ID("n")
    BLOCK
      :=
        ID("p")
        ID("n")
      WHILE
        GT
          ID("n")
```

```
          VAL(I64(0))
        BLOCK
          :=
            ID("n")
            SUB
              ID("n")
              VAL(I64(1))
          :=
            ID("p")
            MULT
              ID("p")
              ID("n")
      RET
        ID("p")
  FUNC
    ID("main")
    PARAMS
    BLOCK
      :=
        ID("n")
        VAL(I64(5))
      WRITE
        CALL
          ID("factorial_loop")
          ID("n")
      WRITE
        CALL
          ID("factorial_recursion")
          ID("n")
  CALL
    ID("main")
analyze_func() 'factorial_recursion'
BLOCK
  ID
    LT
      ID("n")
      VAL(I64(2))
    BLOCK
      RET
        VAL(I64(1))
    BLOCK
      RET
        MULT
          ID("n")
          CALL
            ID("factorial_recursion")
            SUB
              ID("n")
              VAL(I64(1))
BLOCK
  RET
    VAL(I64(1))
BLOCK
  RET
    MULT
      ID("n")
      CALL
        ID("factorial_recursion")
        SUB
          ID("n")
          VAL(I64(1))
analyze_func() 'factorial_loop'
BLOCK
  :=
    ID("p")
    ID("n")
  WHILE
    GT
      ID("n")
      VAL(I64(0))
    BLOCK
      :=
        ID("n")
        SUB
          ID("n")
          VAL(I64(1))
      :=
        ID("p")
        MULT
          ID("p")
          ID("n")
  RET
    ID("p")
BLOCK
  :=
    ID("n")
    SUB
      ID("n")
      VAL(I64(1))
  :=
    ID("p")
    MULT
      ID("p")
      ID("n")
analyze_func() 'main'
BLOCK
  :=
    ID("n")
    VAL(I64(5))
  WRITE
    CALL
      ID("factorial_loop")
      ID("n")
  WRITE
    CALL
      ID("factorial_recursion")
      ID("n")
```

```
WRITE
  CALL
    ID("factorial_loop")
    ID("n")
WRITE
  CALL
    ID("factorial_recursion")
    ID("n")

MTree (Analyzed) 'global':

A_BLOCK
| SYMBOL PROGRAM #1 "factorial_loop"
| SYMBOL PROGRAM #0 "factorial_recursion"
| SYMBOL PROGRAM #2 "main"
  FUNC
    A_BLOCK
| SYMBOL FUNCTION #0 "n"
      ID
        LT
          REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
          VAL(I64(2))
        A_BLOCK
          VAL(I64(1))
        A_BLOCK
          MULT
            REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
            CALL
              REF CellLoc { typ: PROGRAM, idx_frame: 2, idx_cell: 0 }
              SUB
                REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
                VAL(I64(1))
  FUNC
    A_BLOCK
| SYMBOL FUNCTION #1 "p"
| SYMBOL FUNCTION #0 "n"
      :=
        REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 1 }
        REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
      WHILE
        GT
          REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
          VAL(I64(0))
        A_BLOCK
          :=
            REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
            SUB
              REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
              VAL(I64(1))
          :=
            REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 1 }
            MULT
              REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 1 }
              REF CellLoc { typ: FUNCTION, idx_frame: 1, idx_cell: 0 }
      REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 1 }
  FUNC
    A_BLOCK
| SYMBOL FUNCTION #0 "n"
      :=
        REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
        VAL(I64(5))
      WRITE
        CALL
          REF CellLoc { typ: PROGRAM, idx_frame: 1, idx_cell: 1 }
          REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
      WRITE
        CALL
          REF CellLoc { typ: PROGRAM, idx_frame: 1, idx_cell: 0 }
          REF CellLoc { typ: FUNCTION, idx_frame: 0, idx_cell: 0 }
  CALL
    REF CellLoc { typ: PROGRAM, idx_frame: 0, idx_cell: 2 }
----------------------------EVALUATE-----------------------------------

EVALUATE MTree (Analyzed) 'global' :

> 0
> 120
```