# COSMOS

Shashank Shanbhag - CS20BTECH11061
Ayush Jha-CS20BTECH11006
Avula Dinesh - CS20BTECH11005
Yuvraj Singh Shekhawat- CS20BTECH11057
Rohan Atkurkar - CS20BTECH11041

November 25, 2022

## Contents

# 1 Reference manual:-

## 1.1 Introduction

This maual contains the details of the lexical conventions and parser rules of the COSMOS language.

## 1.2 lexical conventions

The program is translated in multiple phanses mainly

- lexical analysis
- Parsing and grammar checking
- Code generation(transpiling)

## 1.3 Tokens

We have mainly six classes of tokens -

- identifiers
- keywords
- constants
- string literals
- operators
- other separators like comments and newlines

## 1.4 Comments:-

The characters /* introduce a comment, which terminates with the characters */ . Comments do not nest, and they do not occur within a string or character literals.

| **Constant** | **Unit & How to use the Constant in the language** |
|---|---|
| Solar mass | $1.989\times10^{30}$ kg |
| $R_e$ Radius of Earth | $6.371\ \times10^{3}$m |
| $M_e$ | $5.972\ \times10^{24}$m |
| $R_s$ | $1.989\times10^{30}$m |
| Astronomical Unit | $1.496\times10^{11}$m |
| Light year | $9.460\times10^{15}$m |
| Gravitational Constant(G) | $6.674\text{e}\times10^{-11}$ |
| speed of light | $2.997\ ^{8}$ m/s |

Table 1: Constants

### 1.5 Keywords used in the language:-

| **Keyword** | | | | |
|---|---|---|---|---|
| bool | char | short | int | float |
| string | auto | astrobject | mass | density |
| dist | time | speed | acc | energy |
| force | freq | ly | AU | solar_mass |
| parsec | temp | arcsec | repeat | break |
| continue | if | else | return | void |
| struct | proc | | | |

### 1.6 Constants

*constants* *Intcosnt - and integer number used in the loop variable*

*Charcosnt - used for storing character constants*

*SN_cosnt - used to strore the scientific notation number in the format aeb̂ where a and b are real numbers*

*Boolcosnt - used to store true/false*

*String literal - used to store the string in the format ". . ."*

### 1.7 Operators:-

| Operators | Use of operator | Associativity |
|---|---|---|
| + | addition | left-right |
| - | subtraction | |
| * | multiplication | |
| / | division | |
| ∧ | power | |
| > | Greater than | |
| < | Less than | |
| >= | Greater than or equal to | |
| <= | Less than or equal to | |
| == | Test equivalence | |
| ! = | Test inequality | |
| && | AND Left | |
| \|\| | OR Left | |
| % | modulo | |
| . | access (used for structs) | |
| = | Assignment | right-left |
| ! | logical not | |

Table 2: Operators used in the language

### 1.8 Operator Precedence :

| Order Of precedence: |
|---|
| . ( ) [ ] |
| ! |
| ∧ |
| * / % |
| + - |
| < <= > >= |
| == != |
| && |
| \|\| |
| = |

Table 3: Operator precedence

### 1.9 Data Types:-

1. Boolean(as bool)
   A boolean data type can only take 2 values: true(1) or false(0).

2. Character(as char)
   A character variable can take all ASCII values ranging from 0-127.

3. int
   Int data type is used to represent integer numbers.

4. Floating point(as float) This is used to represent decimal numbers or floating point numbers. It will be used to represent mantissa in scientific notation.

   **Note:**
   All data types below will be represented in the format of scientific notation . Format:
   Scientific notation- A pair consisting of $< float > e < shortint >$
   With precision upto 6 decimals

5. mass(in kg)
   mass data type represents the mass of any planet, star or astronomical object.

6. density(in kg/m3)
   density data type represents the density of any planet, star or astronomical object.

7. dist(in m)
   dist data type represents the distance between astronomical objects.

8. speed(in m/s)
   speed data type represents the speed of the astronomical object.It might be used to represent escape velocity, orbital velocity, etc.

9. acc(in m/s2)
   acc data type represents the acceleration of the astronomical object. It might be used to represent acceleration due to gravity(g).

10. time(in s)
    time data type represents the time taken into consideration while solving the problem.

11. force(in N)
    force data type usually represents the gravitational force of attraction between astronomical objects.

12. energy(in J)
    energy data type represents kinetic energy, potential energy, binding energy, etc. for astronomical objects.

13. ly
    ly(Light year) is the distance light travels in one year. We will use this data type to represent extremely large distances.

14. AU
    AU(Astronomical unit) is roughly the distance from earth and sun. We will use this data type to represent extremely large distances.

15. solar_mass
    solar_mass(Stellar mass) is a phrase that is used by astronomers to describe the mass of a star. We will use this data type to represent large masses.

16. parsec
    parsec is equal to about 3.26 light years . One parsec corresponds to the distance at which the mean radius of the earth's orbit subtends an angle of one second of arc. We will use this

data type to represent extremely large distances.

17. temp(in K)
    temp data type will be used to represent the temperature of any star, or the average temperature of empty space between celestial bodies.

18. arcsec
    arcsec is the data type that will be used to represent angle. arcsec equals 1/3600 of a degree.
    Data type formed from combination of primitive data types

19. string
    string data type will be used to represent a combination of characters.

    Struct data type

20. astrobject
    astroject(Astronomical_object) data type is used to represent spherical astronomical objects. It will be a struct with mass and radius of the astronomical object as its members.

### 1.10 Syntax:-

Examples of syntax are

1. syntax for repeat
   repeat statement is used for looping.
   Syntax:
   repeat( initialize; condition; update){
   #statements for execution#
   }

2. syntax for if-else
   If-else is used for creating conditional statements.
   Syntax:
   if( expression )
   {
   #statements for execution if expression is true#
   }
   else
   {
   #statements for execution if expression is false#
   }

3. syntax of defining function
   keyword ”proc” is used before declaring the function.
   Syntax:-
   Proc data_type func_name(data_type1 arg1,data_type arg2....)

{
    # Function body #
}

# 2 Expressions:

## 2.1 primary_expression

*primary_expression*
    *IDENTIFIER*
    *CONSTANT*
    *'(' expression ')'*
    *;*

A primary expression can be an *identifier* or a *constant* or a ' parentheseised expression'

## 2.2 postfix_expression

*postfix_expression*
    *primary_expression*
    *postfix_expression '(' ')'*
    *postfix_expression '(' argument_list ')'*
    *postfix_expression '.' IDENTIFIER*
    *postfix_expression INC_OP*
    *postfix_expression DEC_OP*
    *;*

A postfix expression can be a chain of 'parenthesis' , 'parentheseised argument list' , 'access of struct using '.' ' ,*increment operator* , *decrement operator*

## 2.3 argument_list

*argument_list*
    *assignment_expression*
    *argument_expression_list ',' assignment_expression*
    *;*

An argument list is a bunch of *arguments* seperated by ','

## 2.4 unary_expression

*unary_expression*
    *postfix_expression*
    *INC_OP unary_expression*
    *DEC_OP unary_expression*
    *;*

Unary expression is a bunch of postfix expressions seperated by either *incrementor operator* or *decrement operator*

### 2.5 arthmetic_expression

*arthmetic_expression*
    *unary_expression*
    *arthmetic_expression '^' unary_expression*
    *arthmetic_expression '*' unary_expression*
    *arthmetic_expression '/' unary_expression*
    *arthmetic_expression '+' unary_expression*
    *arthmetic_expression '-' unary_expression*
    *;*

It is just a bunch of arthemetic operations like *power*, *addition*,*multiplication*,*division*,*substraction* on a bunch of unary expressions

### 2.6 logical_expression

*logical_expression*
    *arthmetic_expression*
    *arthmetic_expression '<' arthemetic_expression*
    *arthmetic_expression '>' arthmetic_expression*
    *arthmetic_expression LE_OP arthmetic_expression*
    *arthmetic_expression GE_OP arthmetic_expression*
    *arthmetic_expression EQ_OP arthmetic_expression*
    *arthmetic_expression NE_OP arthmetic_expression*
    *arthmetic_expression AND_OP arthmetic_expression*
    *arthmetic_expression OR_OP arthmetic_expression*
    *;*

It is a bunch of logical operations like *and* ,*or* and comparisions like *less than*,*greater than* ,*less than equal to*,*greater than equal to*,*equal to*,*not equal to*

### 2.7 assignment_expression

*assignment_expression*
    *logical_expression*
    *unary_expression '=' assignment_expression*
    *;*

It assigns the value/output of a logical expression to a unary expression in a sequential manner.

### 2.8 expression

*expression*
    *assignment_expression*
    *expression ',' assignment_expression*
    *;*

It is a bunch of assignment expressions seperated by a ',' .
This is used when declaring multiple variables in a single line

### 2.9 declaration

*declaration*
    *declaration_specifiers init_declarator_list ';'*

*struct_specifier*
*;*

This is used to declare a variable or a struct

### 2.10 init_declarator_list

*init_declarator_list*
*init_declarator*
*init_declarator_list ',' init_declarator*
*;*

This contains a bunch of initiation declarations seperated by a ','

### 2.11 init_declarator

*init_declarator*
*declarator*
*declarator '=' initializer*
*;*

This allows us to either define and assign value to a variable simultaneously or just define the variable.

### 2.12 type_specifier

*type_specifier*
*VOID*
*CHAR*
*STRING*
*INT*
*BOOL*
*STRUCT IDENTIFIER*
*DIST*
*LY*
*AU*
*MASS*
*SOLAR_MASS*
*DENSITY*
*TIME*
*SPEED*
*ACC*
*ENERGY*
*FORCE*
*FREQ*
*PARSEC*
*TEMP*
*ARCSEC*
*;*

This checks for all the data types available in the language

### 2.13 struct_specifier

*struct_specifier*
*STRUCT IDENTIFIER '' struct_declaration_list ''*
*;*
This allows us to define a struct by using the keyword *STRUCT* and give a *name/identifier* to the struct in the format of *struct_declaration_list*

### 2.14 struct_declaration_list

*struct_declaration_list*
*type_specifier struct_declarator_list*
*struct_declarator_list type_specifier struct_declarator_list*
*;*
This contains a bunch of declaration statemts that are to be stored the struct datatype

### 2.15 struct_declarator_list

*struct_declarator_list*
*declarator*
*struct_declarator_list ',' declarator*
*;*
This contains the variables and functions that are to be stored in the struct

### 2.16 declarator

*declarator*
*IDENTIFIER*
*declarator'(' ')'*
*declarator'(' parameter_list ')'*
*;*
A declarator can be a *IDENTIFIER* or a *function with arguments* or a *function with no arguments*

### 2.17 parameter_list

*parameter_list*
*parameter_declaration*
*parameter_list ',' parameter_declaration*
*;*
This is a bunch of *parameter declarations* seperated by ','

### 2.18 parameter_declaration

*parameter_declaration*
*type_specifier declarator*
*;*
This is used to declare a *variable* by just specifying the data type and name of the na==variable

### 2.19 statement

*statement*
*compound_statement*
*expression_statement*
*selection_statement*
*iteration_statement*
*jump_statement*
*;*
A statement can be one of *compound* or *expression* or *selection* or *iteration* or *jump* statement

### 2.20 statement_list

*statement_list*
*statement*
*declaration*
*statement_list statement*
*statement_list declaration*
*;*
This is used to write a bunch of declaration and normal statements in any order

### 2.21 compound_statement

*compound_statement*
*'' ''*
*'' statement_list ''*
*;*
This is used to define multiple statements in a parenthesis.

### 2.22 expression_statement

*expression_statement*
*';'*
*expression ';'*
*;*
This is used to define a *empty statement* or a *expression*

### 2.23 selection_statement

*selection_statement*
*IF '(' expression ')' statement*
*IF '(' expression ')' statement ELSE statement*
*;*
This is used to define if-else statement

### 2.24 iteration_statement

*iteration_statement*
*REPEAT '(' expression_statement expression_statement ')' statement*
*REPEAT'(' expression_statement expression_statement expression ')' statement*

;

This is used to write a loop with keyword *REPEAT* and defining the statements that needs to be done in the parenthesis

### 2.25 jump_statement

*jump_statement*
*CONTINUE ';'*
*BREAK ';'*
*RETURN ';'*
*RETURN expression ';'*
;

This is used to identify the jump statemnt that is used to jump to pointed statement

### 2.26 root_unit

*root_unit*
*external_declaration*
*root_unit external_declaration*
;

This is the start unit of the TM which expands to sequence of *external_declaration*

### 2.27 external_declaration

*external_declaration*
*PROC function_definition*
*declaration*
;

This is used to define a function(like main) or a *declaration*

### 2.28 function_definition

*function_definition*
*type_specifiers declarator compound_statement*
;

This is used to define the structure of the function

## 3 Grammar:-

*primary_expression*
*IDENTIFIER*
*CONSTANT*
*'(' expression ')'*
;
*postfix_expression*
*primary_expression*
*postfix_expression '(' ')'*
*postfix_expression '(' argument_list ')'*
*postfix_expression '.' IDENTIFIER*

*postfix_expression INC_OP*
*postfix_expression DEC_OP*
*;*

*argument_list*
*assignment_expression*
*argument_expression_list ',' assignment_expression*
*;*

*unary_expression*
*postfix_expression*
*INC_OP unary_expression*
*DEC_OP unary_expression*
*;*

*arthmetic_expression*
*unary_expression*
*arthmetic_expression '^' unary_expression*
*arthmetic_expression '*' unary_expression*
*arthmetic_expression '/' unary_expression*
*arthmetic_expression '+' unary_expression*
*arthmetic_expression '-' unary_expression*
*;*

*logical_expression*
*arthmetic_expression*
*arthmetic_expression '<' arthemetic_expression*
*arthmetic_expression '>' arthmetic_expression*
*arthmetic_expression LE_OP arthmetic_expression*
*arthmetic_expression GE_OP arthmetic_expression*
*arthmetic_expression EQ_OP arthmetic_expression*
*arthmetic_expression NE_OP arthmetic_expression*
*arthmetic_expression AND_OP arthmetic_expression*
*arthmetic_expression OR_OP arthmetic_expression*
*;*

*assignment_expression*
*logical_expression*
*unary_expression '=' assignment_expression*
*;*

*expression*
*assignment_expression*
*expression ',' assignment_expression*
*;*

*declaration*
*declaration_specifiers init_declarator_list ';'*
*struct_specifier*
*;*

*init_declarator_list*
*init_declarator*
*init_declarator_list ',' init_declarator*
*;*

*init_declarator*

*declarator*
*declarator '=' initializer*
*;*

*type_specifier*
*VOID*
*CHAR*
*STRING*
*INT*
*BOOL*
*STRUCT IDENTIFIER*
*DIST*
*LY*
*AU*
*MASS*
*SOLAR_MASS*
*DENSITY*
*TIME*
*SPEED*
*ACC*
*ENERGY*
*FORCE*
*FREQ*
*PARSEC*
*TEMP*
*ARCSEC*
*;*

*struct_specifier*
*struct_or_union '' struct_declaration_list ''*
*;*

*struct_declaration_list*
*type_specifier struct_declarator_list*
*struct_declarator_list type_specifier struct_declarator_list*
*;*

*struct_declarator_list*
*declarator*
*struct_declarator_list ',' declarator*
*;*

*declarator*
*IDENTIFIER*
*declarator'(' ')'*
*declarator'(' parameter_list ')'*
*;*

*parameter_list*
*parameter_declaration*
*parameter_list ',' parameter_declaration*
*;*

*parameter_declaration*
*type_specifier declarator*

;

*statement*

*compound_statement*

*expression_statement*

*selection_statement*

*iteration_statement*

*jump_statement*

;

*statement_list*

*statement*

*declaration*

*statement_list statement*

*statement_list declaration*

;

*compound_statement*

*'' ''*

*'' statement_list ''*

;

*expression_statement*

*';'*

*expression ';'*

;

*selection_statement*

*IF '(' expression ')' statement %prec LOWER_THAN_ELSE*

*IF '(' expression ')' statement ELSE statement*

;

*iteration_statement*

*REPEAT '(' expression_statement expression_statement ')' statement*

*REPEAT'(' expression_statement expression_statement expression ')' statement*

;

*jump_statement*

*CONTINUE ';'*

*BREAK ';'*

*RETURN ';'*

*RETURN expression ';'*

;

*root_unit*

*external_declaration*

*root_unit external_declaration*

;

*external_declaration*

*PROC function_definition*

*declaration*

;

*function_definition*

*type_specifiers declarator compound_statement*

;