Define our own language!

compiler

Homework #11

TeamName: ThinkTank

student number:

201228174: 안대희 **201424492:** 여종현

1. Syntex EBNF

1. Maintains basic Python syntax. But we have Notation specifier.

single_input: NEWLINE | notation_specifier simple_stmt
| simple_stmt
| notation_specifier compound_stmt NEWLINE | compound_stmt NEWLINE

file_input: (NEWLINE | notation_specifier stmt | stmt)* ENDMARKER

eval_input: testlist NEWLINE* ENDMARKER

suite: (notation_specifier simple_stmt | simple_stmt)| NEWLINE INDENT (notation_specifier stmt | stmt)+ DEDENT (This syntex is one of the most basic and essential grammar of python)

notation_specifier: 'pre:' | 'in:' | 'post:'

1. Add type_specifier with adjusting typeSun.

Numeric Types — int, float, complex

Sequence Types — list, tuple, range

Text Sequence Type - str

Binary Sequence Types - bytes, bytearray, memoryview

Set Types - set, frozenset

Mapping Types

classdef: 'class' NAME ['(' [arglist] ')'] ':' suite

arglist: argument (',' argument)* [',']

argument: (test [comp_for] | test '=' test | '**' test | '*' test)

funcdef: 'def' type_specifier NAME parameters ['->' test] ':' suite

parameters: '(' [typedargslist] ')'

typedargslist: (tfpdefVal ['=' test] (',' tfpdefVal ['=' test])* [',' ['' *[tfpdefVal] (',' tfpdefVal ['=' test])* [',' [**'' tfpdefVal [',']]] | ''** tfpdefVal [',']]] | '' *[tfpdefVal] (',' tfpdefVal ['=' test])* [',' [**'' tfpdefVal [',']]] | ''** tfpdefVal [','])

Point!: TypeSun explicitly check the type by adding tfpedfVal in the existing Python syntax

tfpdefVal: type_specifier NAME [':' test]

type_specifier : INT | FLOAT
| COMPLEX_SPECIFIER
| LIST
| TUPLE
| RANGE
| BYTES
| BYTEARRAY
| MEMORYVIEW
| SET
| DICT

TEST_COMPLEX_SPECIFIER : (INT | FLOAT)+ (INT | FLOAT)+ 'j' ;

EXPRLIST: (expr|star_expr) (',' (expr|star_expr))* [',']

TESTLIST: test (',' test)* [',']

EXPRTUPLE: (expr|star_expr) (',' (expr|star_expr))* [',']

TESTTUPLE: test (',' test)* [',']

And so on.

3. Add C like type_specifier with adjusting typeSun.

or_test: and_test (('or' | '||') and_test)*

and_test: not_test (('and' | '&&') not_test)*

not_test: ('not'| '!') not_test | comparison

comparison: expr (comp_op expr)*

comp_op: '<'|'>'|'=='|'>='|'<='|'<>'|'!='|'in'|'not' 'in'|'is'|'is' 'not'

star_expr: '*' expr

expr: xor_expr ('|' xor_expr)*

xor_expr: and_expr ('^' and_expr)*

and_expr: shift_expr ('&' shift_expr)*

shift_expr: arith_expr (('<<'|'>>') arith_expr)*

arith_expr: term (('+'|'-') term)*

term: factor (('*'|'@'|'/'|'%'|'//') factor)*

factor: ('+'|'-'|'~') factor | power

power: atom_expr ['**' factor]

atom_expr: ['await'] atom trailer*

atom: ('(' [yield_expr|testlist_comp] ')' | '[' [testlist_comp] ']' | '{' [dictorsetmaker] '}' | NAME | NUMBER | STRING+ | '...' | 'None' | 'True' | 'False')

testlist_comp: (test|star_expr) (comp_for | (',' (test|star_expr))* [','])

trailer: '(' [arglist] ')' | '[' subscriptlist ']' | '.' NAME

subscriptlist: subscript (',' subscript)* [',']

subscript: test | [test] ':' [test] [sliceop]

sliceop: ':' [test]