CSE4061– Compiler Design Project
Final Assignment

Grammar Rules for Our Language:

st_delimiters -> ST_DELIMITER st_delimiters | ε

stn_delimiters -> ST_DELIMITER stn_delimiters | N-DELIMITER stn_delimiters | ε

single_new_line -> st_delimiters N-DELIMITER st_delimiters | ε

stmts -> n_stmt stmts | func_init_stmt stmts | $ (EOF)

n_stmts -> stn_delimiters n_stmt stn_delimiters n_stmts | ε

n_stmt -> decl_stmt | compound_stmt | if_stmt | for_stmt | loop_stmt | func_call_stmt | assign_stmt | unary_stmt

ufa_common_stmt -> **ID** ufa_common_stmt_continue

ufa_common_stmt_continue -> unary_continue | func_call_continue | assign_stmt_wo_id

decl_stmt -> variable_type st_delimiters decl_stmt_end

variable_type -> INT | DOUBLE | STRING

decl_stmt_end -> **ID** st_delimiters decl_stmt_body st_delimiters SEMICOLON

decl_stmt_body -> assignment_operators st_delimiters simple_expr | ε

assign_stmt_start -> **ID** st_delimiters assignment_operators st_delimiters arithm_expr

assign_stmt_wo_id -> st_delimiters assignment_operators st_delimiters arithm_expr

assignment_operators -> N-ASSIGN-OPT | A-ASSIGN-OPT | S-ASSIGN-OPT | M-ASSIGN-OPT | D-ASSIGN-OPT (with token classes explanation)

compound_stmt -> OPEN-B stn_delimiters compound_stmt_body CLOSE-B

compound_stmt_body -> stmts | ε

unary_continue -> unary_operators st_delimiters SEMICOLON

if_stmt -> IF stn_delimiters OPEN-P stn_delimiters bool_expr stn_delimiters CLOSE-P st_delimiters single_new_line n_stmt stn_delimiters if_stmt_end

if_stmt_end -> ELSE if_stmt_end_variations n_stmt

if_stmt_end_variations -> st_delimiters | single_new_line
for_stmt → FOR-LOOP stn_delimiters OPEN-P stn_delimiters for_decl stn_delimiters bool_expr stn_delimiters SEMICOLON stn_delimiters iter_spec stn_delimiters CLOSE-P st_delimiters single_new_line n_stmt

for_decl -> for_decl_start st_delimiters SEMICOLON | assign_stmt_start st_delimiters SEMICOLON

num_variable_type -> INT | DOUBLE

for_decl_start -> num_variable_type st_delimiters **ID** st_delimiters assignment_operators st_delimiters arithm_expr

iter_spec -> **ID** iter_spec_end

iter_spec_end -> st_delimiters iter_spec_assign_variations st_delimiters arithm_expr | unary_operators

iter_spec_assign_variations -> A-ASSIGN_OPT | S-ASSIGN_OPT | M-ASSIGN_OPT | D-ASSIGN_OPT

loop_stmt -> WHILE-LOOP stn_delimiters OPEN-P stn_delimiters bool_expr stn_delimiters CLOSE-P st_delimiters single_new_line n_stmt

bool_expr -> bool_expr_start_variations stn_delimiters bool_expr_continue

bool_expr_start_variations -> OPEN-P st_delimiters simple_bool_expr st_delimiters CLOSE-P | simple_bool_expr

bool_expr_continue -> logical_operators st_delimiters simple_bool_expr st_delimiters bool_expr_continue | ε

simple_bool_expr -> simple_expr st_delimiters relational_operators st_delimiters simple_expr

logical_operators -> AND-LOGICAL | OR-LOGICAL | AND-BINARY | OR-BINARY

relational_operators -> EQ | NE | GT | GE | LT | LE

unary_operators -> INCREMENT | DECREMENT

func_init_stmt -> stn_delimiters FUNCTION stn_delimiters **ID** stn_delimiters OPEN-P stn_delimiters params_list stn_delimiters CLOSE-P stn_delimiters compound_stmt stn_delimiters

params_list -> param_decl st_delimiters params_list_continue | ε

params_list_continue -> COMMA st_delimiters param_decl st_delimiters params_list_continue | ε

param_decl -> variable_type st_delimiters **ID**

func_call_continue -> **ID** stn_delimiters OPEN-P stn_delimiters params_values_list stn_delimiters CLOSE-P st_delimiters SEMICOLON

params_values_list -> simple_expr st_delimiters params_value_list_continue | ε

params_value_list_continue -> COMMA st_delimiters simple_expr st_delimiters params_value_list_continue | ε

arithm_simple_expr -> **ID** | INT-VAR | DOUBLE-VAR | OPEN-P stn_delimiters arithm_expr stn_delimiters CLOSE-P

arithm_expr -> arithm_simple_expr stn_delimiters arithm_expr_end

arithm_expr_end -> arithm_operators st_delimiters arithm_expr | ε

simple_expr -> arithm_expr | STRING-VAR

arithm_operators -> PLUS | MINUS | MULT | DIV | REM

space_delimiter → “ “

tab_delimiter → “\t“

new_line_delimiter → “\n“

variable_type → **int** | **double** | **string**

Lexical Structure of Our Language:

Comments: Comments starts with # character and ends with the same (#) character.

Keywords: The keywords in grammar of our programming language :

- int, double, string, if, else, func, loop, for

Identifiers: An identifier starts with a letter and continues with a letter or digit or an underscore character. An identifier can not be the same with a keyword exactly but it can contain it. Length of an identifier should not exceed 64 characters.

Delimiters: whitespace, tab and new line supported by our language.

Operators:

- **Assignment operators:**
  - **= | += | -=**
- **Logical Operators:**
  - && | || | !
- **Relational Operators:**
  - **< | > | <= | >= | == | !=**
- **Unary Operators:**
  - **++** | --
- **Arithmetic Operators:**
  - **+** | - | * | / | %

Numbers:

- **digit** : 0 - 9 → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- **INT-VAR (int) :** digit+
- **DOUBLE-VAR (double) :** digit+ exponent | digit+ fraction (exponent | ε)
- **exponent :** e (**+** | - | ε) digit+ (the maximum exponent value is 128)
- **fraction :** .digit+

Texts (strings):

- **STRING-VAR (string) :** Strings can contain any characters that are supported by UTF-8 format. It consists of quote characters (“) at the start and at the end.

Group Members :

İlker FENER 150115024
Cem GÜLEÇ 150117828
Büşra GÖKMEN 150116027