

## Project Part 2

Monday, March 28, 2022 7:07 PM

Cameron S. Williamson  
MyPL to Golang

lexer → AST parser → Static checker → goVisitor → printing

run on VM ← Create bytecode

- Package main
- Imports will be required
  - `fmt.Println()`
- Weird Behavior:
  - Getting characters
    - `Get(string, int) == string[int]`
  - Reading strings
    - `Reader := bufio.NewReader(os.Stdin)`
    - `Fmt.Println("Enter text: ")`
    - `Text, _ := reader.ReadString('\n')`
    - `Fmt.Println(text)`

Program	( <b>&lt;tdecl&gt;</b>   <b>&lt;fdecl&gt;</b> )*	( <b>&lt;tdecl&gt;</b>   <b>&lt;fdecl&gt;</b> )*
Tdecl	TYPE ID LBRACE <b>&lt;vdecls&gt;</b> RBRACE	TYPE ID STRUCT LBRACE <b>&lt;vdecls&gt;</b> RBRACE
Vdecls	( <b>&lt;vdecl_stmt&gt;</b> )*	( <b>&lt;vdecl_stmt&gt;</b> )*
Fdecl	FUN ( <b>&lt;dtype&gt;</b>  VOID) ID LPAREN <b>&lt;params&gt;</b> RPAREN LBRACE <b>&lt;stmts&gt;</b> RBRACE	FUN ID LPAREN <b>&lt;params&gt;</b> RPAREN LBRACE <b>&lt;stmts&gt;</b> RBRACE
Params	<b>&lt;dtype&gt;</b> ID (COMMA <b>&lt;dtype&gt;</b> ID)* <b>ϵ</b>	ID <b>&lt;dtype&gt;</b> (COMMA <b>&lt;dtype&gt;</b> ID)* <b>ϵ</b>
Dtype	INT_TYPE DOUBLE_TYPE BOOL_TYPE CHAR_TYPE STRING_TYPE ID	INT_TYPE DOUBLE_TYPE BOOL_TYPE CHAR_TYPE STRING_TYPE ID
Stmts	( <b>&lt;stmt&gt;</b> )*	( <b>&lt;stmt&gt;</b> )*
Stmt	<b>&lt;vdecl_stmt&gt;</b>   <b>&lt;assign_stmt&gt;</b>   <b>&lt;cond_stmt&gt;</b>   <b>&lt;while_stmt&gt;</b>   <b>&lt;for_stmt&gt;</b>   <b>&lt;call_expr&gt;</b>   <b>&lt;ret_stmt&gt;</b>   <b>&lt;delete_stmt&gt;</b>	<b>&lt;vdecl_stmt&gt;</b>   <b>&lt;assign_stmt&gt;</b>   <b>&lt;cond_stmt&gt;</b>   <b>&lt;while_stmt&gt;</b>   <b>&lt;for_stmt&gt;</b>   <b>&lt;call_expr&gt;</b>   <b>&lt;ret_stmt&gt;</b>   <b>&lt;delete_stmt&gt;</b>
Vdecl_stmt	VAR ( <b>&lt;dtype&gt;</b>   <b>ϵ</b> ) ID ASSIGN <b>&lt;expr&gt;</b>	(VAR ID <b>dtype</b> ASSIGN <b>&lt;expr&gt;</b> )   (ID COLON ASSIGN <b>&lt;expr&gt;</b> )
Assign_stmt	<b>&lt;lvalue&gt;</b> ASSIGN <b>&lt;expr&gt;</b>	<b>&lt;lvalue&gt;</b> ASSIGN <b>&lt;expr&gt;</b>
Lvalue	ID (DOT ID)*	ID (DOT ID)*
Cond_stmt	IF <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE <b>&lt;condt&gt;</b>	IF <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE <b>&lt;condt&gt;</b>
Condt	ELIF <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE <b>&lt;condt&gt;</b>   ELSE LBRACE <b>&lt;stmts&gt;</b> RBRACE   <b>ϵ</b>	ELIF <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE <b>&lt;condt&gt;</b>   ELSE LBRACE <b>&lt;stmts&gt;</b> RBRACE   <b>ϵ</b>
While_stmt	WHILE <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE	FOR <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE
For_stmt	FOR ID FROM <b>&lt;expr&gt;</b> (UPTO   DOWNT0) <b>&lt;expr&gt;</b> LBRACE <b>&lt;stmts&gt;</b> RBRACE	FOR <b>&lt;vdecl_stmt&gt;</b> SEMICOLON <b>&lt;expr&gt;</b> <b>&lt;expr&gt;</b> R
Call_expr	ID LPAREN <b>&lt;args&gt;</b> RPAREN	ID LPAREN <b>&lt;args&gt;</b> RPAREN
Args	<b>&lt;expr&gt;</b> (COMMA <b>&lt;expr&gt;</b> )*   <b>ϵ</b>	<b>&lt;expr&gt;</b> (COMMA <b>&lt;expr&gt;</b> )*   <b>ϵ</b>
Ret_stmt	RETURN ( <b>&lt;expr&gt;</b>   <b>ϵ</b> )	RETURN ( <b>&lt;expr&gt;</b>   <b>ϵ</b> )
Delete_stmt	DELETE ID	DNE
Expr	( <b>&lt;rvalue&gt;</b>   NOT <b>&lt;expr&gt;</b>   LPAREN <b>&lt;expr&gt;</b> RPAREN) ( <b>&lt;operator&gt;</b> <b>&lt;expr&gt;</b>   <b>ϵ</b> )	
Operator	PLUS   MINUS   DIVIDE   MULTIPLY   MODULO   AND   OR   EQUAL   LESS_THAN   GREATER_THAN   LESS_THAN_EQUAL   GREATER_THAN_EQUAL   NOT_EQUAL	
Rvalue	<b>&lt;pval&gt;</b>   NIL   NEW ID   <b>&lt;idval&gt;</b>   <b>&lt;call_expr&gt;</b>   NEG <b>&lt;expr&gt;</b>	
Pval	INT_VAL   DOUBLE_VAL   BOOL_VAL   CHAR_VAL   STRING_VAL	
Idval	ID (DOT ID)*	

- Things I need to watch out for
  - Converting the loops
    - A while loop in MyPL is a for loop in golang.
    - A for loop in MyPL is a different type of for loop in golang.
  - Flags for imports.