Team Dynamic

We did not assign task explicitly to anyone. However, we are always keeping a backlog of task remaining and the dependencies between the tasks. Therefore, when a person has time, he can simply pick a task that does not have any open dependencies and work on it. We also try to have very descriptive git commit messages so that every team member is always aware of the current state of the project.

Design Decision for the Symbol Table and the Type Checker

Symbol Table

- We used a linked list of HashMap instead of a stack of HashMap
- The first element of the linked list is always the current scope
- HashMap is structured as HashMap<String, Type> where String is the name of a variable and Type is the type of that variable

Type

- We created a Type class which encapsulates all the types, such as int, float64, function, slice, etc.
- We have implemented "is(Type t)" method in the Type class to check if the two Types are the same Type.
- We also have implemented "assign(Type t)" method in each of basic type classes to check if t can be assigned to each of the basic type.
- AliasType
    - It has a Type attribute, which is the real type of the alias
- ArrayType
    - It has a Type attribute, which is the Type of element
    - It has an int attribute, which is the size of the array
- FunctionType
    - It has an ArrayList<Type> attribute, which contain the Types of parameters
    - It has a Type attribute, which is the Type of return value
- SliceType
    - It has a Type attribute, which is the Type of element
- StructType
    - It has a HashMap<String, Type> attribute, which contain the name and the Type of the fields
    - For the fields of the struct, instead of putting "x.a" into the symbol table,
      we put the Struct type with the linked list of attributes into the symbol table.

Stuff for code generation
- We have decided not to change alias type casting from function call node to type casting node.
We do return the correct type for the type cast. The decision was made because we find changing,
adding and deleting nodes quite messy and decided to do the same checks during the code generation phase.

Enumeration of the type checks

==========
Declaration
==========
Variable declaration
checks variables with same name cannot be repeatedly declared in same scope
basic types: var_decl_exp.go, var_decl_type_exp.go, var_decl_type.go
struct: type_decl_struct.go
array: type_decl_struct.go
Function declaration
check function with same name cannot be repeatedly declared in same scope
function: func_decl.go, func_param.go
Type declaration
checks alias types are well defined
type_decl_badAlias.go

==========
Statement
==========
Return
*for a function of non-void return type, check the return type on every execution path is assignable
to the function signature.
test file: return_nonvoid.go
*for a function of void return type, check the return type on every execution path is void(i.e. just "return").
test file: return_void.go

Assignment capability:

lhs\rhs	int	float64	string	bool	rune
int	YES	NO	NO	NO	YES
float64	YES	YES	NO	NO	YES
string	NO	NO	YES	NO	NO
bool	NO	NO	NO	YES	NO
rune	YES	NO	NO	NO	YES

test files: assign_floatToInt.go, assign_stringToFloat.go, assign_stringToInt.go

For loop
    check condition is type bool
    test file: for_condition.go

If statement
    check condition is type bool
    test file: if_condition.go, elseif_condition.go

Switch statement
    check expression in cases has same type as in the beginning of the switch statement
    test file: switch_type_match.go

==========
Expression
==========
Casting capability:

to\from	int	float64	string	bool	rune
int	YES	YES	NO	NO	YES
float64	YES	YES	NO	NO	YES
string	NO	NO	YES	NO	NO
bool	NO	NO	NO	YES	NO
rune	YES	YES	NO	NO	YES

    test files:cast_booToRune.go, cast_intToBool.go, cast_stringToInt.go

Binary operators:

+=,+

lhs\rhs	int	float64	string	bool	rune
int	YES	NO	NO	NO	YES
float64	YES	YES	NO	NO	YES
string	NO	NO	YES	NO	NO
bool	NO	NO	NO	NO	NO
rune	YES	NO	NO	NO	YES

test files: op_assign_plus.go, binary_intAddString.go

=-,*=,/=,%=, -, *, /, %

lhs\rhs	int	float64	string	bool	rune
int	YES	NO	NO	NO	YES
float64	YES	YES	NO	NO	YES
string	NO	NO	NO	NO	NO
bool	NO	NO	NO	NO	NO
rune	YES	NO	NO	NO	YES

test files: op_assign_sub_stringToString.go

|=,^=,&=, ||, &&
both lhs and rhs must be bool
test files:

<<=,>>=,&^=
both lhs and rhs must be int

Unary operator
+: expr must be numeric (int, float64, rune)
test file: unary_plus_string.go
-:expr must be numeric (int, float64, rune)
test file: unary_neg_string.go
!:expr must be a bool
test file: unary_not_int.go, unary_not_string.go

Indexing
check index is well-typed and has type int.
test file:index_float.go
check the result of the indexing expression is the type of the array/slice
test file:index_expressionMatch.go

Field selection (expr.id)
check expr is of type struct and has field named id
test file:fieldSelect_notStruct.go, fieldSelect_hasField.go

Append append(id, expr)
check id is found in the symbol table and maps to a Slice<T>
test file:append_notSlice.go
check type of expr is compatible with type of the slice
test file:append_incompatibleType.go