CS 3513

Programming Language Project

RPAL INTERPRETER

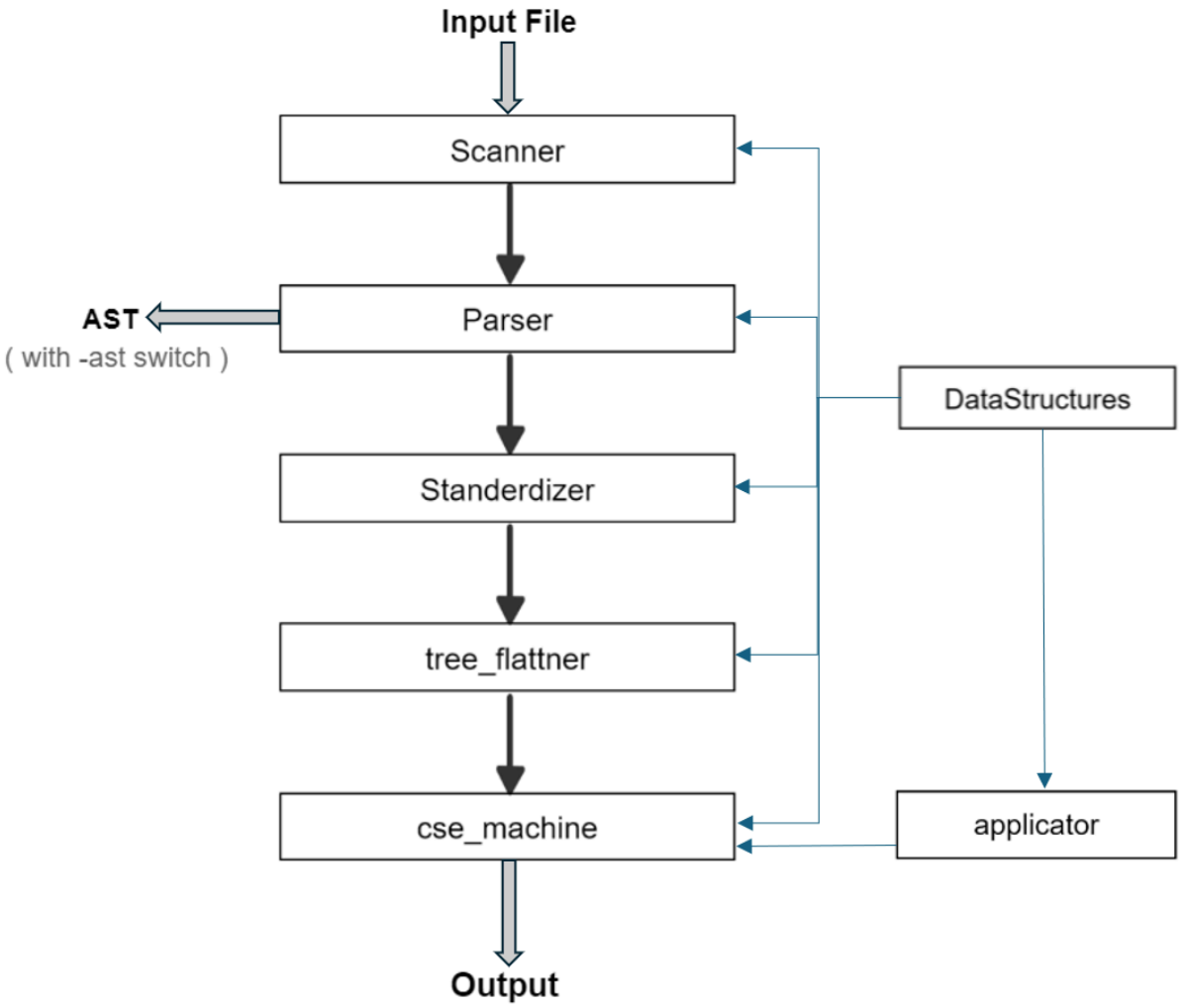
Group - 29

210302P - Kulasekara K.M.S.N

210306G - Kulathunga K.A.J.T

Overall Structure

This program mainly consists of **Scanner**, **Parser**, **Standerdizer**, **tree_flattener** and **cse_machine** programmes.
Read a given file and output the evaluated answer.
With **-ast** switch it gives an Abstract Syntax Tree.



- Scanner mostly deals with token and Python lists.
- Parser converts the list of tokens to a tree data structure.
- Parser and Standerdizer deals with tree data structure.
- Tree flattener converts tree to one or many Stacks.
- Cse_machine mostly deals with Stacks.

Function Prototypes

Scanner

This will read the given file and convert that to a list of **Token**s.

scan (input file) - return list of Tokens after scanning the content of the given file.

Parser

This will build the Abstract Syntax Tree from the list of tokens.

Parse (tokens)	-	Get Token list as input and build the Abstract Syntax Tree (AST). Then return the root node of AST.
Build (s, n, Stack)	-	Pop n items from Stack and append them as children of s and push back node s to Stack.
Check (s, Tokens)	-	Check whether s equals top of the Tokens and remove(consume) the s(top of Tokens) from the Tokens.
Is_keyword (value)	-	Check whether value is a keyword or not.

Standerdizer

standardize_tree (Node)	-	This convert AST to a Standard Tree (ST) by calling this recursively. This is done in depth first manner.
standardize_node (Node)	-	Standardise the Node. This calls the suitable standardise function for Node.
standardize_let (Node)	-	Standardise the **let** node.
standardize_where (Node)	-	Standardise the **where** node.
standardize_fcn_form (Node)	-	Standardise the **fcn_form** node.
standardize_multi_parameters_function (Node)	-	Standardise the **multi parameter**

		function (lambda) node.
standardize_within (Node)	-	Standardise the **within** node.
standardize_at (Node)	-	Standardise the **@** node.
standardize_and (Node)	-	Standardise the **and** node.
standardize_rec (Node)	-	Standardise the **rec** node.

Tree Flattener

This will convert the standardized tree into control structures.

CS (ST) - This will take the standardized tree as the input and return the converted control structures as a list.

extract_string(string) - This is used to split a string containing colon in to 2 parts as the type and the value (ex : “IDENTIFIER : x” will return type=IDENTFIFIER and value=x)

tree_flattener(Node,control_structures,current_cs) - for every node in the standardized tree, flatten the node in to a control structure.

flattener_lambda(Node,control_structures,current_cs) - flatten the “lambda” node to a control structure.

flattener_arrow(Node,control_structures,current_cs) -flatten the “->” node to a control structure.

flattener_tau(Node,control_structures,current_cs) - flatten the “tau” node to a control structure.

CSE machine

This evaluated the program with the help of control, stack and environment.

extract_environment(control) - return the current environment of the given control.

lookup(current_element,current_environment,environment_list) - used to get the value of a variable from the environments. Here the variable is inputted as current_element. If the value cannot be found in the current environment, search in the parent environment. Repeat

this procedure until the value is found. If the value cannot be found even in the primitive environment, return an error.

evaluate (structures) - takes the control structures list as the input and evaluates them by pushing the primitive environment and the first control structure into the control and pushing the primitive environment into the stack.

rule_1(current_element,Stack,environment_list,control) - pop the variable from the control and find the value from the environment and push the value into the stack.

rule_2(current_element,Stack,environment_list,control) - pop the lambda object from the control and assign the index of the current environment as the E value of lambda and push it to the stack.

rule_3(current_element,Stack,environment_list,control) - pop the 'gamma' object from the control and pop operator and operand from the stack. Apply operator to the operand and push the result to the stack.

rule_4(current_element,Stack,environment_list,control,control_structures,var_list) - pop the 'gamma' object from the control and pop the lambda object and the value of the variable fro the stack. Create a new environment by assigning the variable of the 'lambda' object and make the current environment as the parent of the new environment. Push the new environment to both control and the stack and push the control structure of the 'lambda' object into the control.

rule_5(current_element,Stack,environment_list,control) - pop the environment object from control and value,environment object from the stack and push the value back into the stack.

rule_6(current_element,Stack,environment_list,control) - pop the 2 operands from the stack, apply the operation as in the 'current_element.value' and push the result to the stack.

rule_7(current_element,Stack,environment_list,control) - pop the operand from the stack, apply the operation as in the 'current_element.value' and push the result to the stack.

rule_8(current_element,Stack,environment_list,control,control_structures) - pop the 'beta' value from the control and pop the 'delta-else' and 'delta-then' objects from the control respectively. Pop the top from the stack if it's 'true' push the control structure of 'delta-then' to the control. If it's 'false' push the control structure of 'delta-else' to the control.

rule_9(current_element,Stack,environment_list,control) - pop the tau object from the control. Pop 'tau.value' number of elements from the stack, make a tuple from popped elements and push it to the stack.

rule_10(current_element,Stack,environment_list,control) - pop the 'gamma' object from the control. Pop the tuple and the index from the stack respectively. Push the element of the tuple at the index to the stack.

rule_11(current_element,Stack,environment_list,control,control_structures,var_list) - pop the ‘gamma’ object from the control and pop the lambda object and the values of the variable fro the stack. Create a new environment by assigning the variables of the ‘lambda’ object and make the current environment as the parent of the new environment. Push the new environment to both control and the stack and push the control structure of the ‘lambda’ object into the control.

rule_12(current_element,Stack,environment_list,control) - pop ‘gamma’ object from the control and pop ‘y_star’ object and ‘lambda’ object from the stack respectively. Create a ‘eta’ object with the same parameters of the ‘lambda’ object and push it to the stack.

rule_13(current_element,Stack,environment_list,control) - pop ‘eta’ object from the stack, make a ‘lambda’ object with the same parameters and push the ‘eta’ object and ‘lambda’ object respectively. Then push a ‘gamma’ object to the control.

Applicator

This defines the default functions and operations.

Is_binary_operation(op) - checks whether the operation is a binary operator or not.

is_unary_operation(op) - checks whether the operation is a unary operator or not.

is_default_function(op) - check whether the input is a default function.

is_int_operation(op) - check whether the operation is defined only for integers.

apply_binary_operation(op, left, right) - applying the binary operator for left and right operands.

apply_unary_operation(op, operand) - applying the unary operator for the operand.

apply_default_function(op, operand) - applying the default function for the operand.

DataStructures

This includes the data structures used in the implementation.

print_tree(root, level=0) - Printing the tree in the given format.