CSCI-GA.2130-001 – Compiler Construction, Spring 2019

Project 2

Name: Yusen Su
NetID: ys3547
University ID (UID): N15412725
Collaborator Name: Yijian Xie
NetID: yx1891

Work Log

1. Project plan
    - Read the requirement and remarked the implementation hard point
    - Started generating attributes for expressions and adding scheme
    - Defined type checking based on each expression requirement
    - Defined evaluation rules for statements and statement
    - Defined declaration and declarations
    - Extended rules for predefined functions and scoping
    - Debugged and fixed issues of the parser
    - Addressed main, sizeof and other issues
    - Fixed part of those issues
    - Wrote the document
2. Project implementation
    - Using hacs and referenced by example files
    - Discussed issues with partners
        - i. How to extend each parameter into environment of statements
        - ii. How to synthesize the function type to global scoping
        - iii. How to solve check type of main function by parser
        - iv. How to solve requirement of distinguishing the same function name
        - v. How to add predefined functions into global scoping
    - Shared the implementation idea with my partners
        - i. Extension type declaration of parameter into statement
        - ii. Type check main function and duplicate function name
        - iii. Propagate predefined and function type

Remark

1. Expression type checker
    - Function calls
        - Allow return type of the function that is called
        - For each argument, check either it matches the parameter in the corresponded place
        - Match the right number of parameters is provided
    - Pointer and reference
        - Allow a pointer value use star to dereference for assignment

      - E.g. var *int a; *a = 1;
    - Allow a pointer to reference the address of its correspond type
      - E.g. var *int a; a = &1;
  - Binary operator specification
    - All the rules followed the requirement on the requirement lists
2. Statement type checker
   - Assignment
     - Check assigned expression and given expression have the same type, error will be raised necessary
   - Test expression
     - For if and while statement, the test expression only allow to have a type of int or pointer (There is an issue in .MC file, where *string should be string)
   - Return
     - For each return statement, it should only allow the return expression has the same type as declared
3. Declaration type checker
   - Function only define at once
     - For each function, it allows only defined by once (Except main function)
4. Global predefined function
   - Allow all predefined functions used in the parser
5. Propagate function declaration
   - For each function, it knows types of other functions (including predefined)
6. Main
   - The requirement needs a program with a main function that
     - It checks the main function whether it satisfies the return type of int and parameter only char pointer
     - However, it does not raise an error when a program contains multiple main functions. I could not find what is wrong in my code. The checking procedure works for functions other than main

<u>**Testing**</u>

1. Test performance
   - The following requirements will parse succeed
     - $2.1, 1 \sim 13$
     - $2.2, 1 \sim 3$
     - 2.3
     - 2.4
   - The following requirements will parse failed
     - Main function only declared once

    - Parsing sizeof or null expressions

2. Test limitation
   - Parsing “sizeof, null” expressions will not work
     - The parser will not accept any expressions contained null of sizeof (issue should be claimed on project 1)
   - Checking rename of the main function
     - The parser will accept program that defines multiple accepted main functions

<u>Code</u>

Please find Pr2Type.hx file for more details.

<u>SDD Template</u>

Production	**Semantics Rules**
$Program \rightarrow Declaratons$	$\mathrm{DS}.e = \{\}$
$Declaratons \rightarrow Declaraton_1 \ldots\ Declaraton_n$	$\mathrm{D}_1.e = \cdots = \mathrm{D}_n.e = Trans(NoAttr) + \sum_{i=1}^{n} \mathrm{D_i}.fd$
$Declaraton \rightarrow function\ Type\ ID\ (Type_1\ ID_1, \ldots, Type_1\ ID_1)\ \{\ Statementlist\ \}$	$\mathrm{D}.fd = \{ID \rightarrow Type\ (Type_1, \ldots, Type_n)\}$ $Sl.e = D.e + \{ID_1 \rightarrow Type_1, \ldots, ID_n \rightarrow Type_n\}$ $Sl.rt = Type$
$Statementlist \rightarrow Statement\ Statementlist_1$	$S.e = Sl.e$ $Sl_1.e = S.e'(After) + Sl.e$ $S.rt = Sl_1.rt = Sl.rt$
$Statementlist \rightarrow Statement$	$S.e = Sl.e$ $S.rt = Sl.rt$
$Statementlist \rightarrow \varepsilon$	
$Statement \rightarrow var\ Type\ ID;$	$S.e = S.e + \{ID \rightarrow Type\}$
$Statement \rightarrow Expression_1 = Expression_2$	$E_1.e = E_2.e = S.e$ $S.t = CheckTypeSame(E_1.t, E_2.t)$
$Statement \rightarrow if\ (Expression)\ Statement_1\ IfTail$	$E.e = S_1.e = IfTail.e = S.e$ $S_1.rt = IfTail.rt$ $S.t = Bool(\mathrm{E}.t)$
$Statement \rightarrow while\ (Expression)\ Statement_1$	$E.e = S_1.e = S.e$ $S_1.rt = S.rt$ $S.t = Bool(\mathrm{E}.t)$

$Statement \rightarrow return\ Expression;$	$E.e = S.e$ $S.t = CheckReturnType(S.rt, E.t)$
$Statement \rightarrow \{Statementlist\}$	$Sl.e = S.e$ $Sl.rt = S.rt$
$Expression \rightarrow ID$	$E.t = Look\_up(ID)$
$Expression \rightarrow String$	$E.t =* char$
$Expression \rightarrow Int$	$E.t = int$
$Expression \rightarrow$ $Expression_0\ (Expression_1, \dots, Expression_n)$	$E_0.e = E_1.e = E_n.e = E.e$ $E.t$ $= CheckExps\big(E_0.t, (E_1.t, \dots, E_n.t)\big)$
$Expression \rightarrow sizeof(Type)$	$E.t = int$
$Expression \rightarrow null(Type)$	$E.t = Type$
$Expression \rightarrow !\, Expression_1$ $\mid -\, Expression_1$ $\mid +\, Expression_1$	$E_1.e = E.e$ $E.t = Bool(E_1.t)$
$Expression \rightarrow * \ Expression_1$	$E_1.e = E.e$ $E.t = Star(E_1.t)$
$Expression \rightarrow Expression_1$	$E_1.e = E.e$ $E.t = AddStar(E_1.t)$
$Expression \rightarrow Expression_1 * Expression_2$ $\mid Expression_1 / Expression_2$ $\mid Expression_1\ \%\ Expression_2$	$E_1.e = E_2.e = E.e$ $E.t = CheckInt(E_1.t, E_2.t)$
$Expression \rightarrow Expression_1 + Expression_2$ $\mid Expression_1 - Expression_2$	$E_1.e = E_2.e = E.e$ $E.t = LeftOp(E_1.t, E_2.t)$
$Expression \rightarrow Expression_1 < Expression_2$ $\mid Expression_1 > Expression_2$ $\mid Expression_1 \leq Expression_2$ $\mid Expression_1 \geq Expression_2$	$E_1.e = E_2.e = E.e$ $E.t = CheckInt(E_1.t, E_2.t)$
$Expression \rightarrow Expression_1 == Expression_2$ $\mid Expression_1 != Expression_2$	$E_1.e = E_2.e = E.e$ $E.t = CheckEqual(E_1.t, E_2.t)$
$Expression \rightarrow Expression_1 Expression_2$ $\mid Expression_1 \mid\mid Expression_2$	$E_1.e = E_2.e = E.e$ $E.t = LogicCheck(E_1.t, E_2.t)$

Special Operator

Name	**Functionality**
$Trans(NoAttr)$	It will automatically add all predefined functions into fd.
$CheckTypeSame(Type_1, Type_2)$	It will check types of two expressions for assignment. If they are the same, return the same type. Otherwise, it will raise an error “expected two same types for assignment”.
$CheckReturnType(ReturnType, Type)$	It will check type of return is the same as declared. If they are the same, return the same type. Otherwise, it will raise an error “return type should be same as declared”.
$Look\_up(ID)$	Same functionality mentioned on HACS manual
$CheckExps(Exp, Exps)$	It will check type matching of function and its arguments. If the declared function passed arguments have the same types of its parameter declared, it will return the type of what that function return. Otherwise, it will raise an error “expected argument types same”.
$Bool(\mathrm{Type})$	It will check whether the type is int or a pointer type. If not, raise an error “expected given type of int (Boolean)”.
$Star(Type)$	Check whether the Type is a pointer type or not. If it is, dereference that type. Otherwise, it will raise an error “type could not be dereferenced”.
$AddStar(Type)$	Add a star to the current type (it will be a pointer of that type).
$CheckInt(Type_1, Type_2)$	Check two types are int or not.
$CheckEqual(Type_1, Type_2)$	Check two types are either int or the same pointer type. If not, raise an error “expected two same types for equal or not”.
$LogicCheck(Type_1, Type_2)$	Type check for logical operator.

*Special operators have the same name in the code. Please find Pr2Type.hx file for more details.