Csci1631 Assignment 4
Semantic Analysis: A Type Checker
(Posted on 11/20, Due 12/15 10:59PM)

In this assignment, you are required to conduct semantic checks (i.e. type checking) based on your parser program developed in assignment#3. You may use the sample template files (parser.y and lexer3.l) provided in the source directory as a starting point. You are encouraged to use your own assignment#3 code to continue building up your type checker.

You can assume the following for the test programs of this assignment:

(1) Except for potential violations of a list of C-- type rules (specified below), the syntax of the test programs are always correct. Your parser should find all violations of type rules in a given test program. It should not stop at the first violation found.

(2) To simplify the handling of function calls in this assignment, we assume that if a function is declared in the program (we have a single C-- source file), the function header always appear before any call to the function.

In the case of recursive calls, assumption (2) means the type of a function and its parameters have already been declared before the recursive calls are made.

Output: If there are no type violations, print the following message at the end of compilation:

"**Parsing completed. No errors found.**"

If any violation of the listed type rules is found, print the line number in which the error is found and print the additional information (to be specified later), in the following format:

"**Error found in line#** "
<Additional error message>

In this assignment, we provide two type of additional error messages. One is mimicking GCC compiler, which in red colors, and one for mimicking LLVM/Clang compiler, which is in green colors. You may choose to adopt one of them, but not both.

C— does not support explicit type casting. However, it does support the following automatic type conversion (type coercion) rules in ANSI C.

a. Assignment Conversions

In a simple assignment expression, the types of the expressions on the left and right sides of the assignment operator should be the same. If they are not, an attempt will be made to convert the value on the right side of the assignment to the type on the left side. In this assignment, you can simply assume such conversion will be enforced in code generation (i.e. inserting RISC-V FCVT.W.D or FCVT.L.D) instructions. Do remember to enforce this rule later in your code generator assignments).

b. Binary Conversions

When an integer value operates with a float value, the integer value must be converted to the float type first.

c. Function Arguments Conversions

When an expression appears as an argument in a function call that is not the type specified in the argument list of the function declaration, the value of the expression is converted before being passed to the function.

The above type conversions will be enforced in the code generation phase. In this assignment, you only need to focus on the following type rules:

(1) Variable declarations:

1.a

Every variable must be declared explicitly.

Additional error message for a violation:

(gcc: '<name>' was not declared in this scope)
(clang: use of undeclared identifier '<name>')

1.b

A name (a variable, a function or a type) cannot be declared more than once in the same scope (global or local).

Additional error message for a violation:

(gcc: redeclaration of '<type> <name>')
(clang: redefinition of '<name>')

(2) Functions and function calls:

2.a

A call to a function must use the correct number of parameters.

Additional error message for a violation found:

(gcc: too few arguments to function '<function signature>')
(clang: no matching function for call '<name>') "

(gcc: too many arguments to function '<function signature>')
(clang: no matching function for call '<name>')

2.b

A function must return a value of the correct type.

Additional error message:

Note: This should be a warning message rather than an error message since C supports type coercion for function returns. In our assignment, we only support Int and Float.

(clang: implicit conversion turns floating-point number into integer: '<type1>' to '<type2>')

(gcc: no warning generated)

Array references:

3.a

Array references, except for actual parameters in function calls, must match their declared dimensions.

Additional error message:

When the number of subscripts is larger than the array dimension:

(gcc: subscripted value is neither array nor pointer nor vector)

(clang: subscripted value is not an array, pointer, or vector)

When the number of subscripts is smaller than the array dimension: this depends on the type of operation to the reference. For example, the following code snippet int a[2][2]; a[0] = 1; results in error messages:

(gcc: assignment to expression with array type)

(clang: array type 'int [2]' is not assignable)

3.b

There is no need to check for array bounds. However, the array index expression must be integer.

Additional error message:

(gcc: array subscript is not an integer)

(clang: array subscript is not an integer)

3.c

An array name cannot be passed to a scalar formal parameter, and a scalar cannot be passed to an array formal parameter.

Additional error message:

(gcc: invalid conversion from '<array type>' to '<scalar type>')

(clang: no matching function for call '<function name>')

or

(gcc: invalid conversion from '<scalar type>' to '<array type>')

(clang: no matching function for call '<function name>')

For example,

```
int funct1(int a[][10]) { …}
/* formal parameter a is declared as an array */

int funct2(int a) { …}
/* formal parameter a is declared as a scalar variable */

int i,j,k,a[10][10][10];
..
i = funct1(a[0][1]);
/* passing an array slice to funct1(), this is OK */

j = funct1(a[0][1][1]);
/* passing an array element (i.e. a scalar) to funct1(). Since funct1() is expecting an
array parameter, this is a type mismatch and an ERROR */

k = funct2(a[0][1][1]);
/* passing an array element (i.e. a scalar) to funct2(). This is correct */

int x[10], b[10][10][10];

i = funct1(x[j][5]);          /* Array reference x[j][5] is invalid */
k = funct1(b[1]);             /* Passing an array slice b[1] is OK  */
```

You are required to enforce the above specified type checking rules. However, you are not limited to the above rules.

The following are a list of semantic checks supported by GCC/Clang, yet not required by our project. If you perform such tests and report errors correctly, each item will be awarded for two extra credits.

1) Negative array dimensions (e.g., int a[-1];)
   (gcc: size of array '<name>' is negative)
   (clang: '<name>' declared as an array with a negative size)
2) Calling non-callable objects
   (e.g., int a; int b = a(0);)
   (gcc: called object '<name>' is not a function or function pointer)
   (clang: called object type '<type name>' is not a function or function pointer)
3) Non-constant global declarations
   (e.g., int b = a; in global)
   (gcc: initializer element is not constant)
   (clang: initializer element is not a compile-time constant)
4) Re-declaration of typedefs
   (e.g., typedef int Type; typedef float Type;)

(gcc: conflicting types for '<Type>')
(clang: typedef redefinition with different types ('<old-type>' vs '<new-type>'))

Additional Notes:

a) You may assume the identifier names will not exceed 32 characters. However, the number of distinct identifiers should not be limited.
b) In the hw4 directory you may find the following files:

1) lexer3.l	the sample lex program that you may start with
2) header.h	contains AST data structures
3) Makefile	
4) parser.y	template YACC/Bison file
5) functions.c	functions
6) symbolTable.*	symbol table related files
7) semanticAnalysis.c	just a reference for what analysis functions you may need.

Submission requirements:

1) DO NOT change the executable name (parser).
2) Use the script file "tar.sh" to wrap up your assignment works into a single file. Then upload your packaged file to CEIBA.

Usage: ./tar.sh source_directory studentID1_studentID2 (all student IDs in your team) version_number
Example: ./tar.sh hw 12345_12346 ver1
Output: 12345_12346_ver1.tar.bz2 (submit this file)

3) We grade the assignments on the linux1 server. Before summiting your assignment, you should make sure your version works correctly on linux1.