CSE 131 – Compiler Construction

Discussion 1: Getting Started

01/08/2010

01/11/2010

Introduction

- Contact Info:

  See Lab Hours link for lab hours and TAs/Tutors scheduled times in B240

- Availability:

  We are available at our designated lab hour times, or by appointment. Feel free to e-mail us.

Overview

- Format and purpose of our discussions
- What's new in this class
- Starter Code
- STOs and Types
- Error Reporting
- What to do Next
- Topics/Questions you may have

What's new in this class

- We will be building a Translator
  - Just takes the source code (reduced-C) and compiles it (translation) into the target code (SPARC assembly). Then input will be fed to the target program to provide the output results.
  - This is different from Interpreters, which take some input and a source program, and compute and provide the appropriate output (think HTML).

Phases of Compilation

- Lexical Analysis (Lexer.java)
  - Parsing for tokens – not a major point of this course
- Syntax & Semantic Analysis (rc.cup/MyParser.java)
  - Project 1 will focus on the Semantics. In other words, does some syntactically correct piece of code make sense.
- Code Generation
  - Project 2 will focus on this

Starter Code

- The Starter Code is located in:
  /home/solaris/ieng9/cs131w/public/starterCode/
- Look through the files and get familiar with each of them.
- Also the files GETTING_STARTED and CUP_Overview provided with the Starter Code are helpful.

Important Files

- rc.cup
  - Contains the Parser's rules and actions (defines the grammar)
  - Example:

```
Designator2 ::=    Designator2:_1 T_DOT T_ID:_3
                   {:
                     RESULT = ((MyParser) parser).DoDesignator2_Dot (_1, _3);
                   :}
    |              Designator2:_1 T_LBRACKET ExprList T_RBRACKET
                   {:
                     RESULT = ((MyParser) parser).DoDesignator2_Array (_1);
                   :}
    ;
```

Important Files

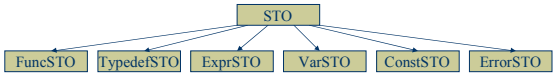
- MyParser.java
  - Contains methods for semantic analysis.
  - Example:

```
STO DoDesignator3_ID (String strID)
{
    STO sto;
    if ((sto = m_symtab.access (strID)) == null)
    {
       m_nNumErrors++;
       m_errors.print (Formatter.toString(ErrorMsg.undeclared_id, strID));
       sto = new ErrorSTO (strID);
    }
    return (sto);
}
```

Important Files

- SymbolTable.java
  - Contains the functions that support Scopes.
    - insert(STO), access(String), accessGlobal(String), accessLocal(String), openScope(), closeScope(), etc.
  - This file contains one of the issues that will be resolved in Phase 0.

STO Hierarchy



- You can change this around!
- Look at the methods in each and how they are overloaded (i.e., isVar(), isConst())

STO Hierarchy

- All STOs have internal fields to indicate if the STO is modifiable and/or addressable. Whenever you create a new STO, you should set these fields appropriately.
- Additionally, ConstSTO includes a value field, since only constants will be folded at compile time (more on this coming soon).

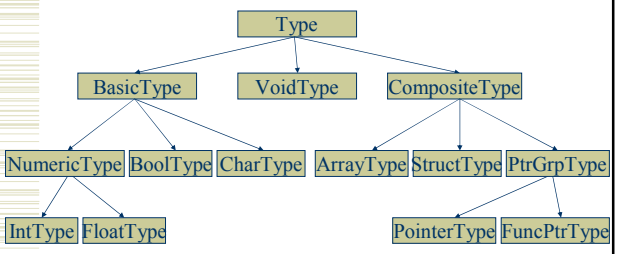
Types

```
Type ::=  SubType OptModifierList OptArrayDef
|         T_FUNCPTR T_COLON ReturnType T_LPAREN OptParamList:_3 T_RPAREN
;

SubType ::=  QualIdent
|            BasicType
;

BasicType ::=   T_INT
|               T_FLOAT
|               T_BOOL
|               T_CHAR
;
```

Types

- We will need to create objects for Basic Types, Array Types, Struct Types (only can be done in structdef), Pointer Types, and Function Pointer Types (function pointers will behave differently than normal pointers).
- How can these objects be organized to make our lives easier?
- What methods and fields should we provide within each?

One Possible Type Hierarchy



What Methods Are Useful

- Look at how the STO.java and *STO.java files are written.
- Consider making methods like isNumeric(), isFloat(), isInt(), isChar(), isArray(), etc. available in your Type Hierarchy.
- Alternatively, use Java's *instanceof* operator (e.g. obj instanceof NumericType)

What Else Is Useful?

- All Types would benefit from methods like:
  - isAssignable(Type t) – coercible type (ie, int → float)
  - isEquivalent(Type t) – same type (e.g. this.getClass() == t.getClass())
- Some types will need to store more information:
  - ArrayType may need to store dimensions
  - StructType may need to store a Vector of fields (or better yet, an entire Scope)
  - You will need the size of these types also, for sizeof

Setting Types

- Must ensure that STO's all have some Type field within them and that this Type field is set when the type becomes known.
- What changes need to be made to the CUP and MyParser files?

Example of Setting Type

- CUP rule currently states:

```
VarDecl ::= OptStatic Type IdentListWOptInit:_3 T_SEMI
{:
    ((MyParser) parser).DoVarDecl (_3);
:}
```

- We now want to incorporate the Type, so we pass the Type to the MyParser method as well.

Example of Setting Type

- Now, in MyParser.java, want to add Type:

```
void DoVarDecl (Vector lstIDs, Type t)
{
    for (int i = 0; i < lstIDs.size (); i++)
    {
        String id = (String) lstIDs.elementAt (i);
        if (m_symtab.accessLocal (id) != null)
        {
            m_nNumErrors++;
            m_errors.print (Formatter.toString(ErrorMsg.redeclared_id, id));
        }
        VarSTO      sto = new VarSTO (id);
        // Add code here to set sto's type field!!!
        m_symtab.insert (sto);
    }
}
```

Type Checking

- Now that we have associated types with our STOs, how do we check them?

Type Checking Example

- Consider the following code:

```
int x;  // VarSTO x gets created and Type set to int
float y;  // VarSTO y gets created and Type set to float
function : void main() {
   x = 5;  // OK
   y = x + 12.5;  // OK
   x = y;  // Error
}
```

- Let's focus on the statement y = x + 12.5

Type Checking Example

y = x + 12.5;

Currently CUP has:

```
Expr7 ::= Expr7:_1 AddOp:_2 Expr8:_3
{:
   RESULT = _1;
:}
```

- What needs to be done?
  - Based on AddOp (+, -), we need to check the types of _1 and _3
  - Based on the types of _1 and _3, we need to create a new STO (an ExprSTO) to return as the result.

Type Checking Example

- Getting a Type out of the STO
  - You have some STO variable "a" and you want to check if it is Equivalent to STO variable "b":
    - a.getType().isEquivalent(b.getType())
    - The isEquivalent method should return a boolean

Organization is your friend!

- Don't just try to throw code into the files.
- Think about the current problem at hand.
- Also think about upcoming tasks.
- Try to make your code as general as possible.
- Remember that in Project 2 we will be generating assembly code, so the more forethought that occurs in Project 1, the easier Project 2 will be!

An Idea

- Define a new function inside MyParser to check expressions
- Define Operator classes to assist type checking
- Ex:
  - In MyParser:

```
STO DoBinaryExpr(STO a, Operator o, STO b) {
        STO sto = o.checkOperands(a, b);
        if (sto.isError()) {
            …
        }
        return sto;
    }
```

An Idea

- In each Operator class, we can do something like this:
-

```
STO checkOperands(STO a, STO b) {
        if (!a.getType().isNumeric() || !b.getType().isNumeric()) {
                // Error
                return (new ErrorSTO(…));
        }
        else if (a.getType().isInteger() && b.getType().isInteger()) {
                // return ExprSTO of int type
        }
        else
                // return ExprSTO of float type
    }
```

Error Reporting

- Now that we can check types, we will find errors!
- Once we find them, we want to print them out.
  - Use only the provided error messages in ErrorMsg.java – these correspond nicely with each check.

Error Reporting

- Only report the first error found in each statement.
  - Once an error is found in a statement, suppress all further errors that may occur in that statement.
- If you want to see the line number where the error occurred (for debugging purposes), do a "make debug"

ErrorSTO

- ErrorSTO is made such that it will appear to be any other STO. Once you find an error, you may want to make your result be an ErrorSTO so the error does not propagate throughout the program.

What to do Next!

1. Find a partner and setup a UnixGroup (see broadcast message for details)!
2. Use tools like CVS, Subversion, and Eclipse to make your life easier (see "Useful Links" on website).
3. Understand the Starter Code. Look through all the files!
4. Complete Phase 0 (if you haven't already!).
5. Implement a basic type structure (see Type.java).
6. Attempt Checks 1 - 4.
7. Come to lab hours and ask questions (We're here for you!)
8. Check the Moodle discussion board – Often people have the same question as you and it may already have been answered.

Topics/Questions you may have

- Anything else you would like me to go over now?
- Anything in particular you would like to see next week?
- Deep thoughts?