Wonsun Ahn

Search this site

Navigation

- **Bio**
  - CV
- **Research**
- **Teaching**
  - CS 0449 Systems Software
  - CS 2210 Compiler Design
    - Project 1 FAQs
    - Project 2 FAQs
    - **Project 3 FAQs**
    - Project 4 FAQs
- **Sitemap**

Project 3 FAQs

Project 3 FAQs

Q) I don't know where to get started.

A) As to the project itself, the main purpose of project 3 is to perform binding: that is binding the uses of identifiers to their definitions. As of project 2, none of the identifiers have been bound. So, if you have the expression x = 1;, you don't know which variable x really refers to. This is reflected in the AST produced by project 2: identifiers nodes just point to entries in the string table. After project 3 is done, all symbols would have been bound. That is, all the identifier nodes in the AST would have been replaced by symbol table nodes that point to entries in the symbol table. The symbol table entry corresponds to a definition of that identifier, and will contain all relevant information (e.g. the type of the identifier, and later when we do code generation, where in memory that identifier is allocated).

So, that's the goal of project 3: to bind all identifiers. Now, in the process, you are going to perform semantic error checking, such as whether identifiers are used without being defined. To do that, you would need to call functions defined in proj3.c to manipulate the symbol table. You would not need to change anything in proj3.c though. You need not change STInit() nor the print tree function. If you modify the AST correctly, the print tree function should correctly show the modified tree.

As we learned in class, there are two ways to implement Syntax Directed Definitions: by traversing the parse tree, or through a Syntax Directed Translation Scheme. Although SDTS is more efficient, for this project, you can just traverse the parse tree to perform the semantic actions, which is more straightforward to do. You main function would look like the following:

```
int main()
{
        treelst = stdout;
        yyparse();
```

```
        STInit (); // Initialize the symbol table with built-in functions
        traversetree (); // Traverse tree while inserting and looking up symbol
        STPrint (); // Print the symbol table
        printtree (root, 0); // Print the parse tree with IDNodes converted to
        return 0;
}
```

Q) How would you change an IDNode to a STNode inside traversetree()?

A) There are two ways of doing it. You can directly modify the IDNode to transform it to a STNode by fiddling with its NodeKind and such, or you can unlink the old IDNode and link in the new STNode like the below:

```
free( LeftChild(node) ); // unlink IDNode
SetLeftChild(node, MakeLeaf( STNode, symbol_table_index) ); // link STNode
```

Q) What do you mean when you say we have to 'create a symbol table' in the assignment sheet? Do we have to implement our own symbol table?

A) No, of course not. Proj3.c already contains an implementation of a symbol table using a stack. All you have to do is call the correct APIs to generate the symbol table using that implementation.

Q) When I use try to compile and use proj3.h and proj3.c, I get warnings about illegal type conversions to int, returning a local variable from a function, etc. Is this a problem?

A) I have uploaded corrected versions of proj3.h and proj3.c to the website. Now functions like SetAttr and GetAttr correctly take uintptr_t types when attribute values are passed. In the former version, int types were used for attribute values, which can be a problem when a type tree pointer is passed as the attribute value. In 64 bit systems, the size of a pointer may be larger than the size of int, which can cause problems. Please use the new versions of proj3.*.

Q) What do you mean by 'class object visibility' and 'nested class variable visibility' on the assignment worksheet?

A) Class object visibility means you should be able to access class members through the class object such as:

```
Point p;
p.x = 10;
p.y = 20;
```

Nested class variable visibility means you can access variables in nested class scopes such as (see src9):

```
p0.p1.x = 10;
p0.p1.y = 10;
```

Q) Some test cases did not include all the system functions such as readln and println.. only the one used. Is it okay if I included everything regardless of use?

A) Yes, it's okay to include all the system functions in the symbol table and the string table. And of course that would causes differences in the indices of the respective tables which is also fine.

Q) How can we implement searching for class method calls using LookUp?

A) Suppose you have the following code:

```
class A {
  method void foo() { ... }
}
class B {
  method void main() {
    A.foo();
    B.foo(); // error
  }
}
```

You cannot use LookUp() to search for function foo() because LookUp() only searchs through the current active stack (which is the sum of all currently active scopes). Function foo() is not part of the current active stack, and is within the scope of A. So what you need to do is search directly in the symbol table for the entry for class A.. and search in the following entries which are members of A within its scope to look for foo(). Now, if you try to do B.foo() then it would be an error because you will not find foo() within the scope of B.

Q) What is the loc_str() function used in proj3.c? Do we have to implement it?

A) This is the code fragment inside proj3.c: nStrInd = loc_str("system"); /* return string index of string "system" */ The function loc_str should do exactly what the comment says: return the index of "system" in the string table. You should have already implemented this function (perhaps with a different name) when you did project 1. Now since STInit() is called at the beginning before even reading the input file, you would not have "system" in your string table at this point, unless you manually insert it. And that's exactly what you should do: you should insert system, readln, and println into the string table before calling STInit(). Then, STInit() will

insert the pre-defined class system and its member functions readln and println for you. I recommend you study the STInit() code carefully since it teaches you how to correctly insert symbols into the symbol table.

Comments

You do not have permission to add comments.