LabReport

on

Symbol Table Manager

Course: Compiler Design Lab

Course Code:CSE 331

December 9, 2015

Submitted To:

Ahmed Al Marouf
Lecturer
Daffodil International University
.

Submitted By:

1. Md. Sakhawat Hossain— ID-131 − 15 − 2406
2. Md. Din-Islam—ID-131 − 15 − 2243
3. Md. Shahadat Hossain— ID-131 − 15 − 2313
4. Md. Ibrahim Faisal—ID:-131 − 15 − 2262
5. Md. Al-Amin Khan—ID:-131 − 15 − 2412

Daffodil International University
.

0.1 Introduction

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler.

- To store the names of all entities in a structured form at one place.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.
- To determine the scope of a name (scope resolution).

0.2 Who Creates Symbol Table

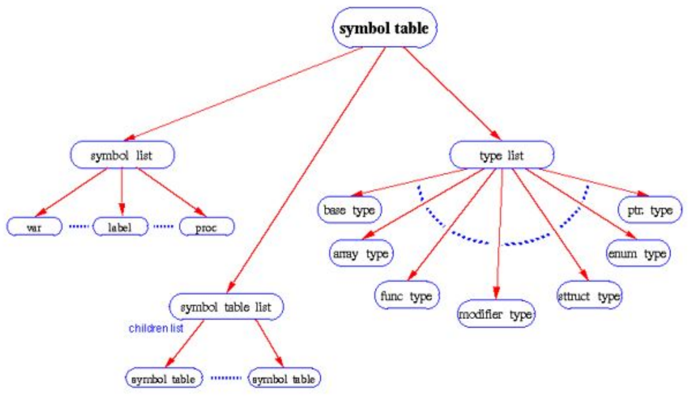
Identifiers and attributes are entered by the analysis phases when processing a definition (declaration) of an identifier. In simple languages with only global variables and implicit declarations. The scanner can enter an identifier into a symbol table if it is not already there In block-structured languages with scopes and explicit declarations. The parser and/or semantic analyzer enter identifiers and corresponding attributes.

0.3 Purpose of symbol table

The purpose of symbol table is following:

- To store the names of all entities in a structured form at one place.
- Symbol table information is used by the analysis and synthesis phases. To verify that expressions and assignments are semantically correct type checking.
- To verify if a variable has been declared.
- To implement type checking, by verifying assignments and expressions in the source code are semantically correct.

- To determine the scope of a name (scope resolution).
- To generate intermediate or target code.



0.4 Implementation

A common implementation technique is to use a hash table. A compiler may use one large symbol table for all symbols or use separated, hierarchical symbol tables for different scopes. There are also trees, linear lists and self-organizing lists which can be used to implement a symbol table. It also simplifies the classification of literals in tabular format. The symbol table is accessed by most phases of a compiler, beginning with the lexical analysis to optimization.

0.5 Use

An object file will contain a symbol table of the identifiers it contains that are externally visible. During the linking of different object files, a linker will use these symbol tables to resolve any unresolved references.

A symbol table may only exist during the translation process, or it may be embedded in the output of that process for later exploitation, for example, during an interactive debugging session, or as a resource for formatting a diagnostic report during or after execution of a program.

While reverse engineering an executable, many tools refer to the symbol table to check what addresses have been assigned to global variables and known functions. If the symbol table has been stripped or cleaned out before being converted into an executable, tools will find it harder to determine addresses or understand anything about the program.

At that time of accessing variables and allocating memory dynamically, a compiler should perform many works and as such the extended stack model requires the symbol table.

0.6 Conclution

A common implementation technique is to use a hash table. A compiler may use one large symbol table for all symbols or use separated, hierarchical symbol tables for different scopes. There are also trees, linear lists and self-organizing lists which can be used to implement a symbol table. It also simplifies the classification of literals in tabular format. The symbol table is accessed by most phases of a compiler, beginning with the lexical analysis to optimization.