Compiler Design in C – Chapter 6.3 (Code generation: Symbol Table)

Marek Kotásek (`xkotas02`), Filip Kozák (`xkozak12`)

Abstract

Symbol table is a structure similar to database which contains information about subroutines, variables, etc. The table is indexed by a *key* field (here a subroutine or variable's name) and each record contains information about that item such as the variable's type or subroutine's return value. The symbol table can also be used to communicate with the lexical analyzer (for example when we use the `typedef` keyword). In implementation, many problems such as duplicate entries support may occur.

The symbol table has two layers – *database* layer and *maintenance* layer. The database layer holds information needed for compilation and performs operations like inserting new entries in the table, finding them, deleting them, and so forth. The maintenance layer is used for managing the table at a higher level, creating systems of data structures to represent specific symbols and inserting these structures into the table using the low-level insert function.

Several data structures can by used for the symbol table: *stack*, *tree* or a *hash table*. Each of them has advantages and disadvantages and is appropriate in specific situations. In this book the author describes hash-based symbol table where the record contains about a dozen items.

A C variable's type must be represented by a system of data structures working in concert. Variable declarations have two parts: a *specifier* part which is a list od various keywords and a *declarator* part that consists of the variable's name and an arbitrary number of stars, array-size specifiers and parentheses.

A complete implementation of the symbol table (including source code) can be found in *Compiler design in C* book, chapter 6.3.