Bučko Peter xbucko00@stud.fit.vutbr.cz
Gajdušek Radek xgajdu07@stud.fit.vutbr.cz

Compiler Design in C – Symbol Table (abstract)

A compiler declaration system consists of set of data structures collectively called the *symbol table*. In fact, the symbol table is a database that contains information about subroutines, variables, and so forth. The database is indexed by a key field - a subroutine or variable name and each record contains information about that item such as the variable type or subroutine return value.

Symbol table managment must fulfill some requirements. The time complexity of look-up for symbols in the table must be as fast as possible. The symbol table should be able to grow as symbols are added to it. In case that source language supports arbitraty complex type, the symbol table must be also able to represent it. Duplicate entries must be supported for different nesting levels.

Structures Used for Symbol Table

The simplest possible structure is a linear array organized as a stack. New symbols are added to the top of the stack and it is searched from the top to the bottom. A stack-based symbol table is simple to implement and one real advantage is that it is very easy to delete a block of declaration. A significant disadvantage is the linear searching time. In addition, the number of variables that can be handled by the table is limited.

The search-time and limited-size problems can be solved by using a binary tree. The binary tree has disadvantages such as collisions – situations where the same name is used for variables at different scope levels – are difficult to resolve.

The best data structure for most symbol table applications is a hash table. An ideal hash table is indexed directly by the key field of the object in the table. They are more efficient than binary trees and more appropriate for those applications in which several elements of the table might have the same key (same name of local and global variable).

Symbol Table Implementation

The implementation uses two data structures: the hash table itself is an array of pointers to "buckets". Each bucket is a single database record. The buckets are organized as a double-linked list.

The implementation contains also the additional flags and fields, that are used for symbol table maintenance. For example, field called *level*, that holds the declaration level for the symbol. It helps to detect a duplicate declaration. The *type* and *etype* fields in the symbol structure point to yet another data structure that describes the object type.

Representing Types

A system with limited number of possible types is called *constrained* system. In such case, types can be represented with a simple numeric coding in the symbol structure. The situation is more complex with *unconstrained* typing system that permits virtually unlimited complexity in a variable declaration.

Variable declarations have two parts: a specifier part which is a list of various keywords (int, long, struct, and so forth) and declarator part that is made up of the variable's name and an arbitrary number of stars, array-size specifiers and parentheses.