Advanced Techniques in Symbol Tables Implementation

Petr Stodůlka (xstodu05)
Martin Kubíček (xkubic34)

E-mail: xstodu05@stud.fit.vutbr.cz
E-mail: xkubic34@stud.fit.vutbr.cz

For purposes of transformations of human readable names of variables and constants inside source code into effective representation for machines a wrapper is needed during compilation, which will map strings into symbols. This structure is oftenly called Symbol Table.

Symbol table (ST) contains information about all symbols which are part of a program (variables, constants, functions, etc.). Each record usually contains name of symbol defined in source code, type of symbol and additional information, like for example value of constant or address in memory. Data inside ST are usually used for semantic analysis, but can be used even for purposes of optimisation or debugging.

ST could be represented by various structures, like lists, trees or hash-tables. However, still new ways of representation of symbol table are searched, to improve various properties. One of them are Abstract Symbol Tables. Concept of this AST originally came from ST, but implements simple abstraction with memory. Records contain abstract information of every variable, including name, abstract address, and value.

Another approach was chosen in GNU C Compiler (GCC), which mixes several types of structures into hierarchic ST, where space of each symbol table is shared for various kinds of data, which are completely independent on the others - including own mapping for each of them.

At the end, we will briefly look at LLVM, which minimizes dependence on symbol table and significant part of compilation is done without symbol table.