ST. XAVIER'S COLLEGE

MAITIGHAR, KATHMANDU

Compiler Design and Construction

Lab Assignment #1

Submitted by:

Aashish Raj Shrestha

013BSCCSIT002

Submitted to:

Mr. Ramesh Shahi Lecturer, Department of Computer Science St. Xavier's College	

Date of Submission: 30th August, 2016

Table of Contents

1. Write a report on comparison of compilers.

Compiler: It is a program that translates source code into object code[4] i.e. it is a computer program that translates a program written in a high-level language into another language, usually machine language [1].

It is a special program that processes statements written in a particular programming language and turns them into machine language or "code" that a computer's processor uses [2]. It translates an entire set of instructions written in a higher-level symbolic language (as C) into machine language before the instructions can be executed [3].

The following is a conceptual example of source code being converted to assembly language and machine code by the compiler [5]:

```
Source Code:-
IF COUNT=10
GOTO END-OF-JOB
 El SE
GOTO COMPUTE-AGAIN
ENDIF

Assembly l anguage  Machine l anguage:-
Compare A to B      Compare 3477 2883
If equal go to C    If = go to 23732
Go to D             Go to 23119

Machine Code:-
10010101001010001010100
10101010010101001001010
10100101010001010010010
```

Workings of Compiler:

l et us take an example to understand how the compiler works [6]:

```
#include<stdio.h>
Void main()
{
        printf("Hello, world!");
}
```

For the given program written in C language, program prints or displays the words "Hello, world!" on the computer screen. The computer cannot understand the commands/instructions contained in a source file (helloworld.c), because C is a high-level language which means, it contains various characters, symbols, and words that represent

complex, numbers-based instructions for e.g. printf, main, header files etc. The only instructions a computer can execute are those written in machine language, consisting entirely of numbers that is the binary language in terms of 0 and 1. Before our computer can run our C program, our compiler should convert our helloworld.c into an object file; then a program called a linker should convert the object file into an executable file.

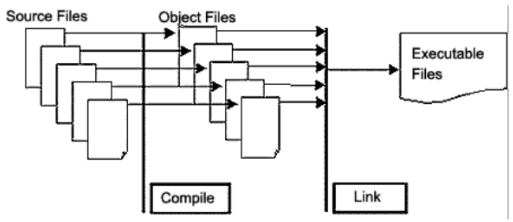

Figure 1: The process of compiling

Definition:

A compiler in real context takes a string and outputs another string [6]. This definition covers all manner of software which converts one string to anther such as text formatters which convert an input language into a printable output, programs which tend to convert among various file formats or different programming languages and also web browsers.

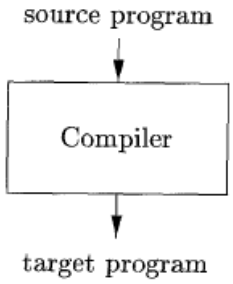

Figure 2: A Compiler

A compiler is a program that can read a program in one language - the source language - and translate it into an equivalent program in another language - the target language [7]

Types of compilers:

The main types of computer compilers are single pass compilers, multi pass compilers, cross compilers and optimizing compilers. A compiler takes one computer language, called a source code, and converts it into the target language. It enables a computer to be able to read different source codes. A compiler makes software to faster and use less memory [8].

A compiler may produce code intended to run on the same type of computer and operating system ("platform") as the compiler itself runs on. This is sometimes called a native-code compiler. Alternatively, it might produce code designed to run on a different platform. This is known as a cross compiler [9].

A cross compiler is one that can run on a computer's operating system that is different from the operating system that the program ordinarily uses. It breaks down binary codes, understands them and allows computer programmers to gain access to the codes [8].

Cross compilers are very useful when bringing up a new hardware platform for the first time (e.g. use in bootstrapping). A "*source to source compiler*" is a type of compiler that takes a high level language as its input and outputs a high level language. For example, an automatic parallelizing compiler will frequently take in a high level language program as an input and then transform the code and annotate it with parallel code annotations (e.g. OpenMP) or language constructs (e.g. Fortran's DOALL statements) [9].

- One-pass compiler, like early compilers for Pascal

The compilation is done in one pass, hence it is very fast.

- Threaded code compiler (or interpreter), like most implementations of FORTH

This kind of compiler can be thought of as a database lookup program. It just replaces given strings in the source with given binary code. The level of this binary code can vary; in fact, some FORTH compilers can compile programs that don't even need an operating system.

- Incremental compiler, like many Lisp systems

Individual functions can be compiled in a run-time environment that also includes interpreted functions. Incremental compilation dates back to 1962 and the first Lisp compiler, and is still used in Common Lisp systems.

- Stage compiler that compiles to assembly language of a theoretical machine, like some Prolog implementations

This Prolog machine is also known as the Warren Abstract Machine (or WAM). Byte-code compilers for Java, Python (and many more) are also a subtype of this.

- Just-in-time compiler, used by Smalltalk and Java systems

Applications are delivered in byte-code, which is compiled to native machine code just prior to execution

- A re-targetable compiler is a compiler that can relatively easily be modified to generate code for different CPU architectures. The object code produced by these is frequently of lesser quality than that produced by a compiler developed specifically for a processor. Re-targetable compilers are often also cross compilers. GCC is an example of a re-targetable compiler.

- A parallelizing compiler converts a serial input program into a form suitable for efficient execution on a parallel computer architecture.

2. Write a report on a symbol table.

Symbol Table: Symbol table is a data structure for storing information about systems in the program [10]. Symbol tables are used by both analysis and synthesis parts of the compiler. This data structure is created and maintained by the compiler in order to store information about the occurrence of various entities such as variable names, function names, objects, classes, interfaces, etc.

A symbol table may serve the following purposes depending upon the language in hand:

- 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).

A symbol table is simply a table which can be either linear or a hash table. It maintains an entry for each name in the following format [11]:

<symbol name, type, attribute>

For example, if a symbol table has to store information about the following variable declaration:

static int interest;

then it should store the entry such as:

<interest, int, static>

The attribute clause contains the entries related to the name.

Implementation

If a compiler is to handle a small amount of data, then the symbol table can be implemented as an unordered list, which is easy to code, but it is only suitable for small tables only. A symbol table can be implemented in one of the following ways:

- l inear (sorted or unsorted) list
- Binary Search Tree
- Hash table

Among all, symbol tables are mostly implemented as hash tables, where the source code symbol itself is treated as a key for the hash function and the return value is the information about the symbol.

Operations

A symbol table, either linear or hash, should provide the following operations.

insert()

This operation is more frequently used by analysis phase, i.e., the first half of the compiler where tokens are identified and names are stored in the table. This operation is used to add information in the symbol table about unique names occurring in the source code. The format or structure in which the names are stored depends upon the compiler in hand.

An attribute for a symbol in the source code is the information associated with that symbol. This information contains the value, state, scope, and type about the symbol. The insert() function takes the symbol and its attributes as arguments and stores the information in the symbol table.

For example:

int a;

should be processed by the compiler as:

insert(a, int);

lookup()

lookup() operation is used to search a name in the symbol table to determine:

- if the symbol exists in the table.
- if it is declared before it is being used.
- if the name is used in the scope.
- if the symbol is initialized.
- if the symbol declared multiple times.

The format of lookup() function varies according to the programming language. The basic format should match the following:

lookup(symbol)

This method returns 0 (zero) if the symbol does not exist in the symbol table. If the symbol exists in the symbol table, it returns its attributes stored in the table.

Scope Management

A compiler maintains two types of symbol tables: a global symbol table which can be accessed by all the procedures and scope symbol tables that are created for each scope in the program.

To determine the scope of a name, symbol tables are arranged in hierarchical structure as shown in the example below:

```
int value=10;
void pro_one() {
  int one_1;
  int one_2;
    {           \
    int one_3;    |_ inner scope 1
    int one_4;    |
    }           /
  int one_5;
    {           \
    int one_6;    |_ inner scope 2
    int one_7;    |
    }           /
  }
  void pro_two() {
  int two_1;
  int two_2;
    {           \
    int two_3;    |_ inner scope 3
    int two_4;    |
    }           /
   int two_5;
  }
```

The above program can be represented in a hierarchical structure of symbol tables:

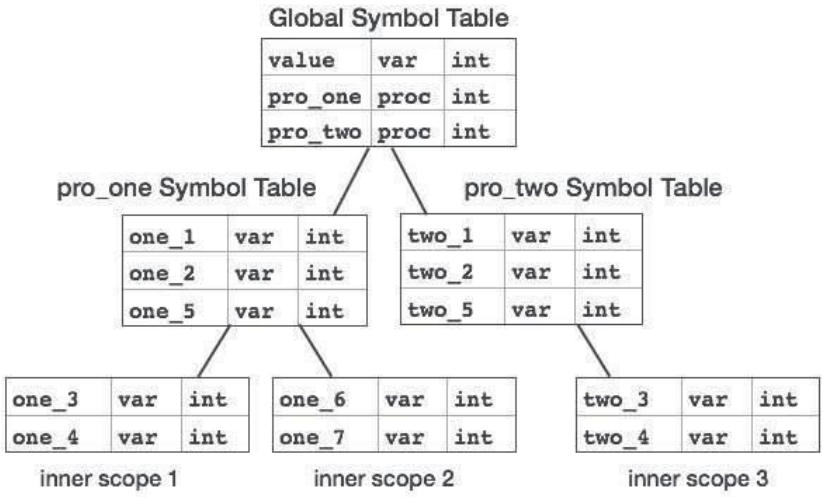

Figure 3: Hierarchical structure of symbol tables:

The global symbol table contains names for one global variable (int value) and two procedure names, which should be available to all the child nodes shown above. The names mentioned in the pro_one symbol table (and all its child tables) are not available for pro_two symbols and its child tables.

This symbol table data structure hierarchy is stored in the semantic analyzer and whenever a name needs to be searched in a symbol table, it is searched using the following algorithm:

- first a symbol will be searched in the current scope, i.e. current symbol table.
- if a name is found, then search is completed, else it will be searched in the parent symbol table until,
- either the name is found or global symbol table has been searched for the name.

3. Write a report on one-pass compiler and multi-pass compiler.

One-pass compiler: A one-pass compiler is a software compiler that processes the source code only once. One-pass compilers are fast, but the programs they generate may not be as efficient [12].

Multi-pass compiler: Software compiler that may pass through source code multiple times. Multi-pass compilers are slower, but much more efficient when compiling [13].

A multi-pass compiler is one that separates compilation into multiple passes, where each pass would continue with the result of the previous pass. Such passes could include parsing, type checking, intermediate code generation, various optimization passes and finally code generation. So for example the parser might create a parse tree, which the type checker would then check for type errors and the intermediate code generator could translate into some form of intermediate code. The optimization passes would then each take the current intermediate code and turn it into a more optimized form. Finally the code generation pass would take the optimized intermediate code and produce the target code from it [14].

In a single-pass compiler all of the steps happen in one pass. So it reads some of the source code, analyses it, typechecks it, optimizes it and generates code for it, and only then moves on to the next bit of code. Single pass compilers consume less memory (because they don't hold the whole AST and/or intermediate code in memory) and generally run faster.

Some languages, like C, are designed to be compilable in a single pass, but others are not. For example functions in C need to be declared before their first use, so the compiler has already seen the function's type signature before it reads the function call. It can then use that information for type checking. In more modern languages, like Java or C# for example, functions can be called before their definition (and forward-declarations don't exist). Such languages can't be compiled in a single pass because the type checker might know nothing about a function's signature when it encounters the function call, making it impossible to typecheck the program without having first processed the whole file.

Further a multi-pass compiler can make use of more kinds of optimizations, so even for languages that can be compiled in a single pass, modern compilers usually use multiple passes.

Differences between one-pass compiler and multi-pass compiler [15]:

1. A one-pass compiler is a compiler that passes through the source code of each compilation unit only once. A multi-pass compiler is a type of compiler that processes the source code or abstract syntax tree of a program several times.
2. A one-pass compilers is faster than multi-pass compilers
3. A one-pass compiler has limited scope of passes but multi-pass compiler has wide scope of passes.
4. Multi-pass compilers are sometimes called wide compilers where as one-pass compiler are sometimes called narrow compiler.
5. Many programming languages cannot be represented with a single pass compilers, for example Pascal can be implemented with a single pass compiler whereas languages like Java require a multi-pass compiler.

References

[1] "compiler", dictionary.com, 2016. [Online]. Available: http://www.dictionary.com/browse/compiler [Accessed: 25-August-2016].

[2] "compiler", techtarget.com, 2016. [Online]. Available: http://whatis.techtarget.com/definition/compiler [Accessed: 25-August-2016].

[3] "compiler", merriam-webster.com, 2016. [Online]. Available: http://www.merriam-webster.com/dictionary/compiler [Accessed: 25-August-2016].

[4] "compiler", webopedia.com, 2016. [Online]. Available: http://www.webopedia.com/TERM/C/compiler.html [Accessed: 25-August-2016].

[5] "Definition of: compiler", pcmag.com, 2016. [Online]. Available: http://www.pcmag.com/encyclopedia/term/40105/compiler [Accessed: 25-August-2016].

[6] "What is a Compiler", engineersgarage.com, 2016. [Online]. Available: http://www.engineersgarage.com/articles/what-is-compiler-tutorial [Accessed: 25- August- 2016].

[7] Alfred V. Aho, Monica S. l am, Ravi Sethi and Jeffery D. Ullman, "l anguage Processors," in *Compiler Design and Construction*, Boston, MA 021 16, 2nd Ed., 200?, pp. 2

[8] "What are the different types of compilers?", reference.com, 2016. [Online]. Available: https://www.reference.com/technology/different-types-compilers-944f1bc982c6c535# [Accessed: 25- August- 2016].

[9] "Compiler", compilers.net, 2016. [Online]. Available: http://www.compilers.net/paedia/compiler [Accessed: 25-August-2016].

[10] "Introduction to Compilers: Symbols Tables", cs.cornell.edu, 2016. [Online]. Available: http://www.cs.cornell.edu/courses/cs412/2008sp/lectures/lec12.pdf [Accessed: 26-August-2016].

[11] "Compiler Design – Symbol Table", tutorialspoint.com, 2016. []. Available: http://www.tutorialspoint.com/compiler_design/compiler_design_symbol_table.htm [Accessed: 26-August-2016].

[12] "One-pass compiler", computerhope.com, 2016. [Online]. Available: http://www.computerhope.com/jargon/o/onepassc.htm [Accessed: 26-August-2016].

[13] "Multi-pass compiler", computerhope.com, 2016. [Online]. Available: http://www.computerhope.com/jargon/o/onepassc.htm [Accessed: 26-August-2016].

[14] "Difference between one pass and multi pass compilers?", stackoverflow.com, 2016. [Online]. Available: http://stackoverflow.com/questions/35673818/difference-between-one-pass-and-multi-pass-compilers [Accessed: 26-August-2016].

[15] "What's the difference between one-pass compiler and multi-pass compiler?", bayt.com,2016.[Online]. Available: http://www.bayt.com/en/specialties/q/111836/what-s-the-difference-between-one-pass-compiler-and-multi-pass-compiler [Accessed: 26-August-2016].