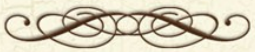
COMPILER DESIGN

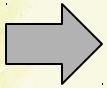


By:

Akhil Kaushik,

Asstt. Prof. T.I.T&S Bhiwani

Table of Contents



Introduction

- A **compiler** is a computer program (or set of programs) that transforms source code written in a high level language (the source language) into low level language (the target language may be assembly language or machine language (0.1)).

Introduction

```
sum = 0;
for (x = 3; x < 5; x++)
 { cout << "x is " << x;
   cout << endl;
   sum += x;
   a *= b / 2;
```

```
10011101
01011011
10111100
01100110
10101100
00011011
```

Source Language

Target Language

Introduction

Compiler basically does two things:

- Analysis: Compiling source code & detecting errors in it.
- Synthesis: Translating the source code into object code depending upon the type of machine.

Introduction

- Program errors are difficult to track if the compiler is wrongly designed.
- Hence compiler construction is a very tedious and long process.

Introduction

- One-pass compiler: It compiles the that passes through the source code of each compilation unit only once.
- They are faster than multi-pass compiler.
- Their efficiency is limited because they don’t produce intermediate codes which can be refined easily.
- Also known as ‘Narrow Compiler’.

Introduction

- Multi-pass compiler: It processes the source code or abstract syntax tree of a program several times.
- It may create one or more intermediate codes (easy to refine).
- Each pass takes output of previous phase as input; hence requires less memory.
- Also known as ‘Wide Compiler’.

History

- Software for early computers were written in assembly language.
- The need of reusability of code gave birth to programming languages.
- This need grow huge to overcome the cost restriction of compiler.

History

- The concept of machine independent programming gave birth to the need of compilers in 1950s.
- The 1st compiler was written by Grace Hopper in 1952 for A-0 programming language.

History

- The 1st complete compiler was developed by FORTRAN team lead by John Backus @ IBM in 1957.
- COBOL was the 1st language to be compiled on multiple platforms in 1960.

History

- Earlier compilers were written in assembly languages.
- The 1st compiler in HLL was created for LISP by Tim Hart & Mike Levin @ MIT, USA in 1962, which was a self-hosting Compiler.

History

- Most compilers are made in C or Pascal languages.
- However the trend is changing to self-hosting compilers, which can compile the source code of the same language in which they are created.

Why Study Compiler?

- Its essential to understand the heart of programming by computer engineering students.
- There may be need of designing a compiler for any software language in the profession.

Why Study Compiler?

- Increases understanding of language semantics.
- Helps to handle language performance issues.
- Opportunity for non-trivial programming project

Translators

- It is important to understand that compiler is a kind of translator, which translates high level language (human understandable) to low-level language(machine understandable).

Translators

- Interpreter: This software converts the high-level language into low-level language line by line.
- It takes less memory than compiler.
- It takes more time than compiler.
- It is more efficient in error detection than compiler.
- It takes more time to build.

Translators

- Assembler: This software converts the assembly language (assembly instruction mnemonics) into machine level language (opcodes i.e.0,1).
- It offers reusability of assembly codes on different machine platforms.

Translators

- Cross-Compiler: If the compiled program can run on a computer whose C.P.U or O.S is different from the one on which the compiler runs, the compiler is known as a cross-complier.

Translators

- Language Translator / Source to source translator / Language Converter: It converts programs in one high-level language to another high-level language.

- Ex: From Java code to ASP.Net code

Translators

- Language Rewriter: It is a program that changes form of expression of the same language.

- It does not changes the language of source code.

Translators

- Decompiler: It is a piece of software that converts the low-level language to high-level language .

- It is not as famous, but may prove a useful tool sometimes.

Translators

- Compiler-Compiler: It is a tool that creates a compiler, interpreter or parser from the information provided on formal description of any language.
- The earliest & most common type is parser-generator.

Translators

- Linker: It is a program that combines object modules(object programs) to form executable program.
- Each module of software is compiled separately to produce object programs.
- Linker will combine all object modules & give it to loader for loading it in memory.

Translators

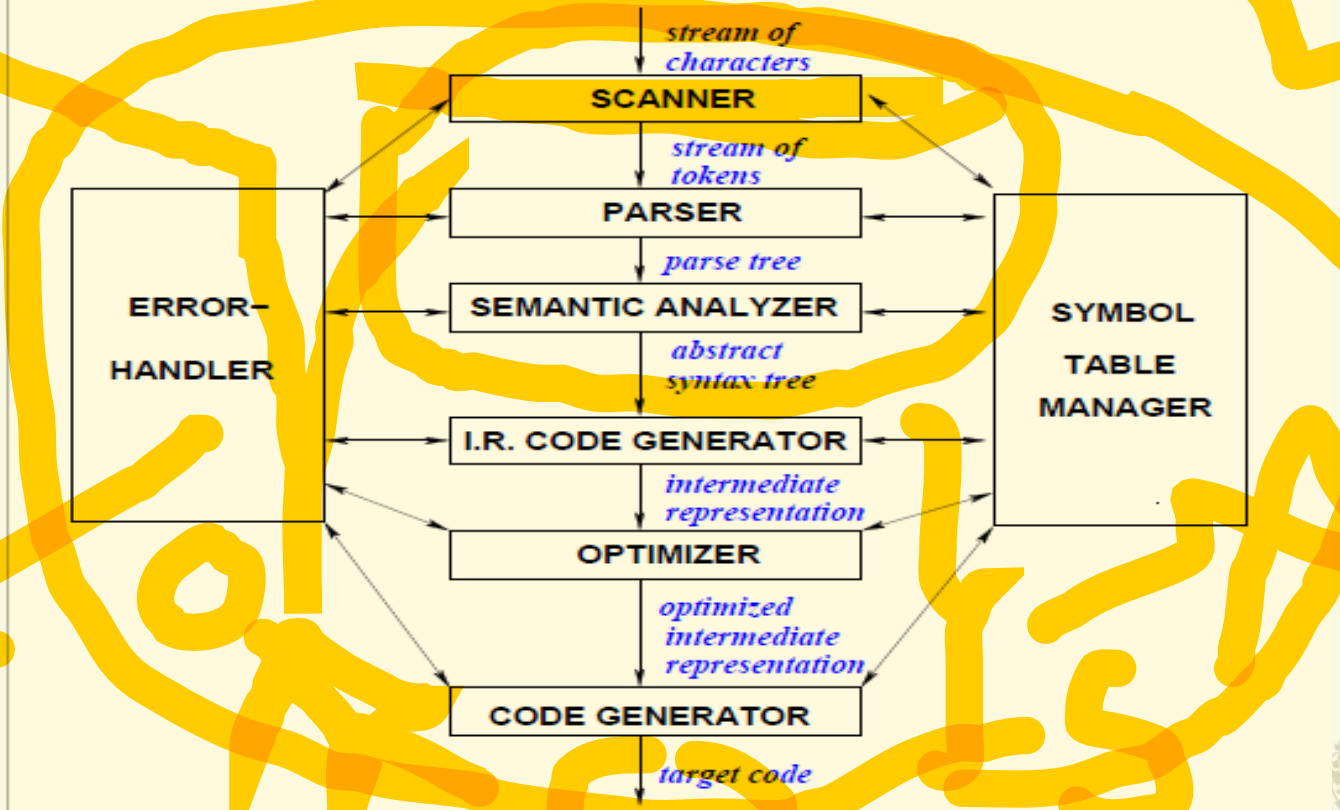
- Loader: It is a program which accepts input as linked modules & loads them into main memory for execution.
- It copies modules from secondary memory to main memory.
- It may also replace virtual addresses with physical addresses.
- Linker & Loader may overlap.

Structure of Compiler

- Compilers bridge the gap b/w high level language & machine hardware.
- Compiler requires:

1. Finding errors in syntax of program.
2. Generating correct & efficient object code.
3. Run-time organization.
4. Formatting o/p acc. to linker/ assembler.

Structure of Compiler



Structure of Compiler

Compilation- Front End

(also known as **Analysis Part)**

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- Intermediate Code Generation

* => Front end is machine independent.

Structure of Compiler

Compilation- Front End

- Determines operations implied by the source program which are recorded in a tree structure called the Syntax Tree.
- Breaks up the source code into basic pieces, while storing info. in the symbol table

Structure of Compiler

Compilation- Back End

(also known as **Synthesis Part)**

- Code Optimization
- Code Generation

* => Back end is machine dependent.

Structure of Compiler

Compilation- Back End

- Constructs the target code from the syntax tree, and from the information in the symbol table.
- Here, code optimization offers efficiency of code generation with least use of resources.

Structure of Compiler

Lexical Analysis

- Initial part of reading and analyzing the program text.
- Text is read and divided into tokens, each of which corresponds to a symbol in the programming language.
- Ex: Variable, keyword, delimiters or digits.

Structure of Compiler

Lexical Analysis

Ex: a = b + 5 – (c * d)

Token Type	Value
Identifier	a, b, c, d
Operator	+, -, *
Constant	5
Delimiter	(,)

Structure of Compiler

Syntax Analysis

- It takes list of tokens produced by lexical analysis.
- Then, these tokens are arranged in a tree like structure (Syntax tree), which reflects program structure.
- Also known as Parsing.

Structure of Compiler

Semantic Analysis

- It validates the syntax tree by applying rules & regulations of the target language.
- It does type checking, scope resolution, variable declaration, etc.
- It decorates the syntax tree by putting data types, values, etc.

Structure of Compiler

Intermediate Code Generation

- The program is translated to a simple machine independent intermediate language.
- Register allocation of variables is done in this phase.

Structure of Compiler

Code Optimization

- It aims to reduce process timings of any program.
- It produces efficient programming code.
- It is an optional phase.

Structure of Compiler

Code Optimization

- Removing unreachable code.
- Getting rid of unused variables
- Eliminating multiplication by 1 and addition by 0
- Removing statements that are not modified from the loop
- Common sub-expression elimination.

Structure of Compiler

Code Generation

- Target program is generated in the machine language of the target architecture.
- Memory locations are selected for each variable.
- Instructions are chosen for each operation
- Individual tree nodes are translated into sequence of m/c language instructions.

Structure of Compiler

Symbol Table

- It stores identifiers identified in lexical analysis.
- It adds type and scope information during syntactical and semantical analysis.
- Also used for ‘Live analysis’ in optimization.
- This info is used in code generation to find which instructions to use.

Structure of Compiler

Error Handler

- It handles error handling & reporting during many phases.
- Ex: Invalid character sequence in scanning, invalid token sequences in parsing, type & scope errors in semantic analysis.

Compiler Construction Tools

- Compiler construction tools were introduced after widespread of computers.
- Also known as compiler- compilers, compiler-generators or translator writing systems.
- These tools may use sophisticated algo. or specified languages for specifying & implementing the component.

Compiler Construction Tools

- Scanner Generators: These generate lexical analyzers.
- The basic lexical analyzer is produced by Finite Automata, which takes input in form of regular expressions.
- Ex: LEX for Unix O.S.

Compiler Construction Tools

- Parser Generators: These software produce syntax analyzers which takes input based on context-free grammar.

- Earlier, used to be most difficult to develop but now, easier to develop & implement.

Compiler Construction Tools

- Syntax-directed Translation Engines: These s/w products produce intermediate code with the help of parse tree.
- Main idea is associating 1 or more translations with each node of parse tree.
- Each node is defined in terms of translations at its neighboring nodes in the tree.

Compiler Construction Tools

- Automatic Code Generator: This software basically take intermediate code as input & produce machine language as output.
- It is capable of fetching data from various storage locations like registers, static memory, stack, etc.
- Basic technique here is ‘template matching’.

Compiler Construction Tools

- Data Flow Engines: It is a tool used for code optimization.
- Info is supplied by user & intermediate code is compared to analyze the relation.
- It also does data-flow analysis i.e. finding out how values are transmitted from one part to another part of the program.

Further Contact Details

For any queries, please contact me at:

akhil1681@gmail.com

Or

http://akhil1681.blogspot.com