Compiler Construction

Text Book:

Compiler Construction: Principles and Practice

By: Kenneth C. Louden
(San Jose State University, USA)

- Book can be used for background reading.
- Book can also be used for your personal lecture preparation.
- Further reading/learning must be accomplished using the sources description provided on the course web page.

Compiler Construction

Course Web Site:

http://www.cs.aue.auc.dk/~akbar/2006/complierconst06.html

Sources details, schedule and all necessary information is provided here.

What is to be achieved

Basic principles of compiler construction and tools so that one can utilize these concepts may be to implement a compiler project or utilized the acquired knowledge for more general software engineering problems.

How:

- Discussion of the theory.
- Discussion on the various aspects of the theory.
- Examples.
- Solutions to selected Exercises with your participation.

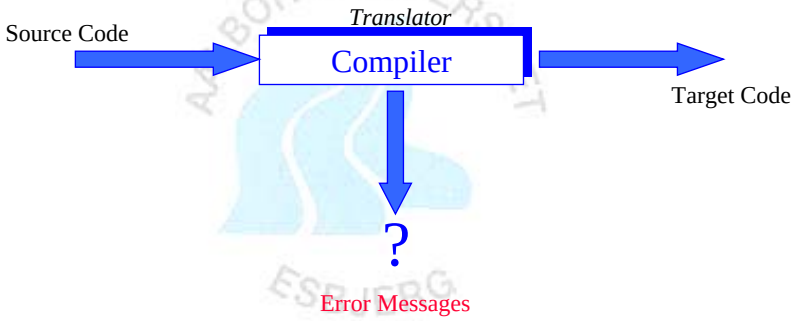
Un-expected:

- *Modification (Normally additional stuff) of the text provided on the course web page.*

Introduction

Compiler is tool: which translate notations from one system to another, usually from source code (high level code) to machine code (object code, target code, low level code).

Compiler



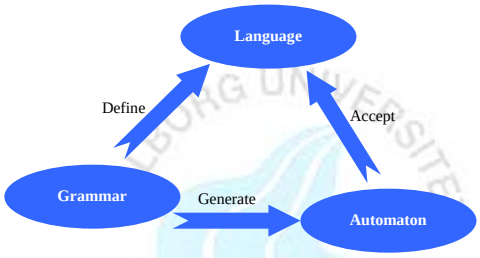
What is Involved

- Programming Languages
- Formal Languages
- Regular Expressions & Automata Theory
- Applications

Programming Languages

- We use natural languages to communicate
- We use programming languages to speak with computers

Formal Languages



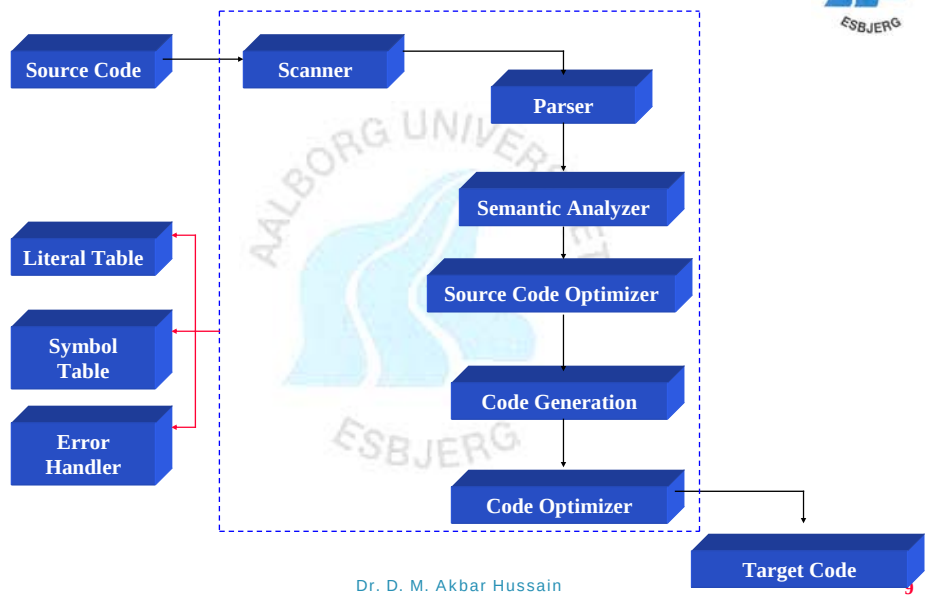
Regular Expressions & Finite Automata

Applications

Editors

Word-processors

Translation Flow



Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
9

Components of a Compiler

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis

- Intermediate Code Generation
- Code Optimization
- Code Generation

Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
10

Lexical Analysis (LA)

```
Maradona kicks the ball
```

Token Generation:

- `Maradona`
- `kicks`
- `the`
- `ball`

Who performs this ?

Lexical Analysis

```
X = Y + 30
```

Token Generation:

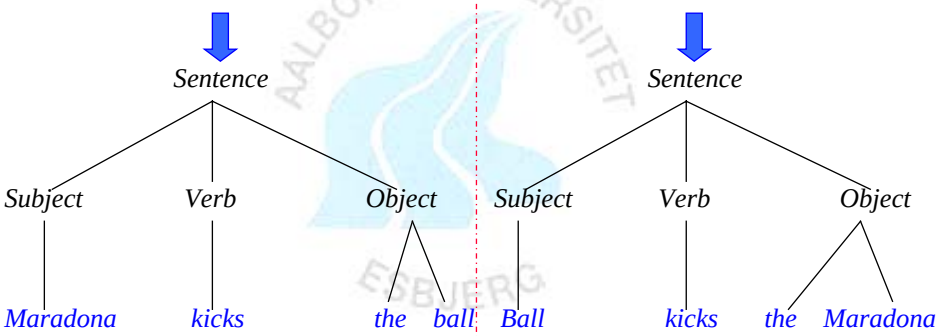
- `X` **id_1**
- `=` **operator**
- `Y` **id_2**
- `+` **operator**
- `30` **literal/constant**

What other functions Scanner can perform ?

Syntax Analysis (SA)

Structure of the program is determined by SA. Some thing similar to grammatical analysis.

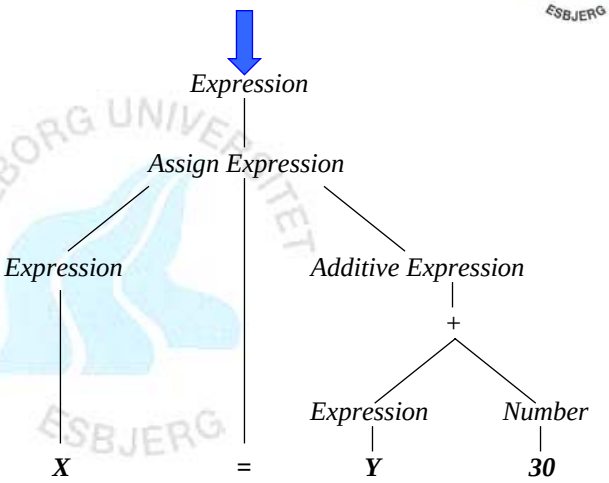
Maradona kicks the ball **Ball kicks the Maradona**



Syntax Analysis (SA)

X = Y + 30

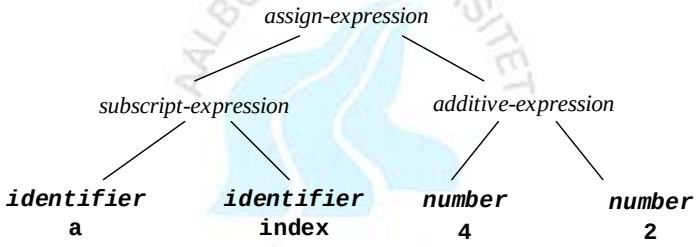
(Syntax Tree)



Who performs that ?

Syntax Analysis (SA)

Some time syntax tree is also called as Abstract Syntax tree and could be a "trimmed" version of the parse tree with only essential information:
For Example: a[index] = 4 + 2

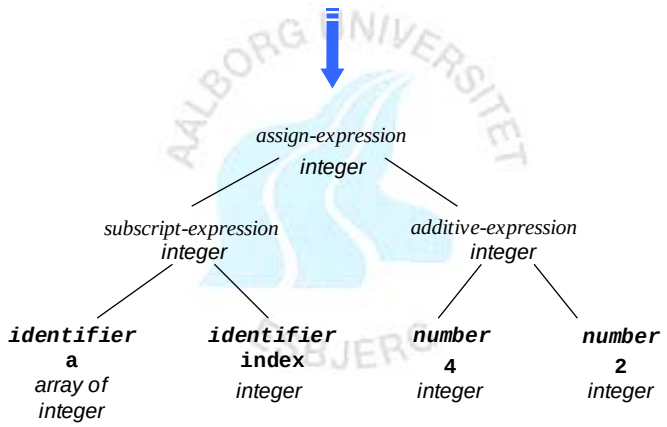


Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
15

Semantic Analyzer

This attaches meaning to tokens; For example to the same expression

a[index] = 4 + 2



Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
16

Intermediate Code Generation

Same example: X = Y + 30

Temp1 = 30

Temp2 = Y

Temp3 = Temp2 + Temp1

X = Temp3

Three Address Code

P-Code

Code Optimization

Same example: X = Y + 30

Temp1 = 30

Temp2 = Y

X = Temp2 + Temp1

Propagation ?

Constant folding ?

Common Sub-expression Elimination ?

Strength of Operation ?

Code Generation

Same example: X = Y + 30

Movei Y, r1

Addi 30, r1

Movei r1, X

Algorithmic Tools

- **Token:**
  - Using Regular Expressions.
- **Scanner:**
  - Implementation of finite state machine to recognize tokens.
- **Parser:**
  - An Automaton (i.e. uses a stack), based on grammar rules in a standard format (BNF -- Backus Naur Form).
- **Semantic Analyzer and Code Generator:**
  - Recursive evaluators based on semantic rules for attributes (properties of language constructs).

Portability Consideration

- **Front End: Scanner, Parser, Semantic Analyzer and source code optimizer depend primarily on source language.**
- **Back End: Code generator and target code optimizer depend primarily on target language (machine architecture).**
- **Passes ?**

Data structures

- **Syntax tree: Kind of a link list structure**
- **Literal table: "Hello, world!", 3.141592653589793, etc.**
- **Symbol table: Names for variables, functions, classes, typedefs, constants.**

Error handling

- One of the difficult part of a compiler to design.
- Must handle a wide range of errors
- Must handle multiple errors.
- Must not get stuck.
- Must not get into an infinite loop.

Kinds of errors

- **Syntax:**

```
if (x == 0) y + = z + r; }
```

- **Semantic:**

```
int x = "Hello, world!";
```

- **Runtime:**

```
int x = 2;
...
double y = 3.14159 / (x - 2);
```

Error Handling Requirements

- A compiler must handle syntax and semantic errors, but not runtime errors (whether a runtime error will occur is a million dollar question).
- Sometimes a compiler is required to generate code to catch runtime errors and handle them in some graceful way (either with or without exception handling). This, too, is often difficult.

Sample compilers

- **TINY:** A 4-pass compiler for the TINY language, based on Pascal (Text pages 22-26).
- **C-Minus:** Based on C, Text, pages 26-27 and Appendix A.

TINY Example

```
read x;
if x > 0 then
  fact := 1;
  repeat
    fact := fact * x;
    x := x - 1
  until x = 0;
  write fact
end
```


C-Minus Example

```
int fact( int x )
{ if (x > 1)
    return x * fact(x-1);
  else
    return 1;
}

void main( void )
{ int x;
  x = read();
  if (x > 0) write( fact(x) );
}
```

Structure of the TINY Compiler

globals.h	main.c
util.h	util.c
scan.h	scan.c
parse.h	parse.c
symtab.h	symtab.c
analyze.h	analyze.c
code.h	code.c
cgen.h	cgen.c

Conditional Compilation Options

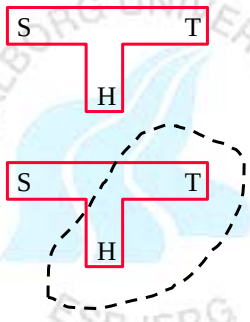
- **NO_PARSE**: Builds a scanner-only compiler.
- **NO_ANALYZE**: Builds a compiler that parses and scans only.
- **NO_CODE**: Builds a compiler that performs semantic analysis, but generates no code.

Listing Options (built in - not flags)

- **`EchoSource`**: Echoes the TINY source program to the listing, together with line numbers.
- **`TraceScan`**: Displays information on each token as the scanner recognizes it.
- **`TraceParse`**: Displays the syntax tree in a linearlized format.
- **`TraceAnalyze`**: Displays summary information on the symbol table and type checking.
- **`TraceCode`**: Prints code generation-tracing comments to the code file.

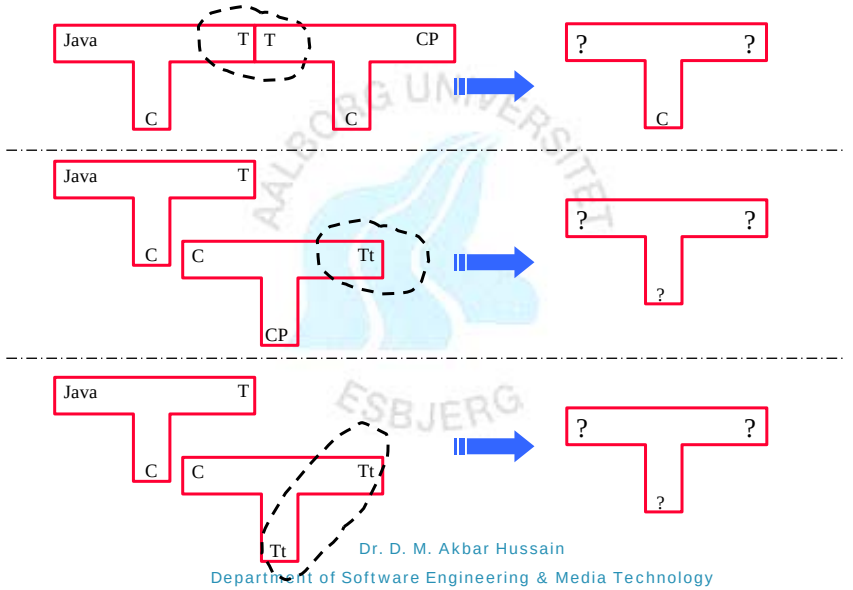
Porting & Bootstrapping

- Tool Chain:
- T-Diagram:



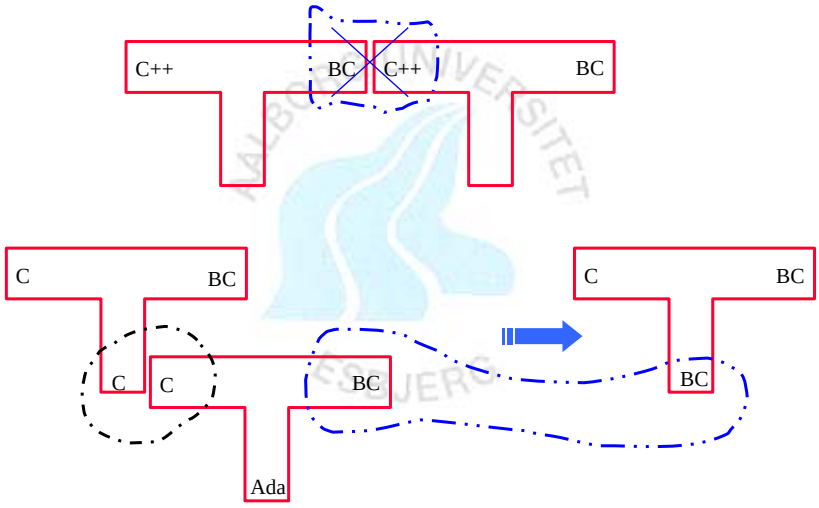
Porting & Bootstrapping

Two Compilers: one converts Java to T & second from T to C++



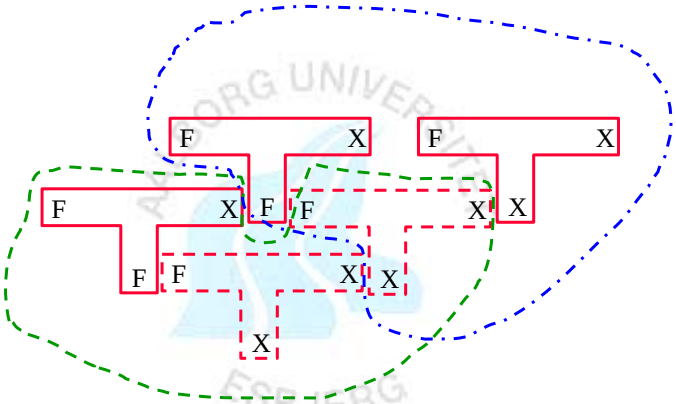
Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
33

Rules

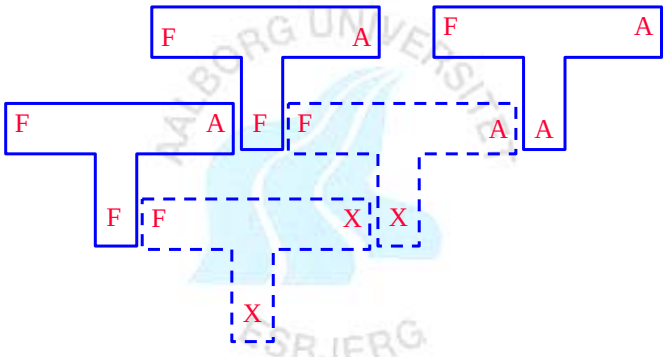


Dr. D. M. Akbar Hussain
Department of Software Engineering & Media Technology
34

Steps of Bootstrapping



Another Example



Compiler Construction books at E-Books Directory: files with free access on the Internet. These books are made freely available by their respective authors and publishers.Â e-books in Compiler Construction category. Crafting Interpreters: A handbook for making programming languages by Robert Nystrom - craftinginterpreters.com , 2015 This book contains everything you need to implement a full-featured, efficient scripting language. Compiler Construction. Mayer Goldberg \ Ben-Gurion University October 20, 2018. Mayer Goldberg \ Ben-Gurion University. Compiler Construction. October 20, 2018 1 / 115. Week 1.Â â–¶ Introduction to compiler construction â–¶ Introduction to the ocaml programming language. Mayer Goldberg \ Ben-Gurion University. Compiler Construction. October 20, 2018 2 / 115. Abstraction. Compiler construction. This is a Wikipedia book, a collection of Wikipedia articles that can be easily saved, imported by an external electronic rendering service, and ordered as a printed book. Edit this book: Book Creator Â· Wikitext.